Deep Learning on Incomplete and Multi-Source Data : Application to Cancer Immunotherapy

Thomas Ranvier

THESE DE DOCTORAT DE
L'UNIVERSITE CLAUDE BERNARD LYON 1

École Doctorale N°512
Mathématiques et Informatique (InfoMaths)

Discipline : Informatique

Soutenue publiquement le 06 décembre 2023 par

Thomas Ranvier

Apprentissage Profond à Partir de Données Incomplètes et Multi-Sources : Application à l'Immunothérapie Contre le Cancer

---

Devant le jury composé de :

Mme.	**Boudjeloud-Assala Lydia**	*MCF - HDR, Université de Lorraine*	Rapporteure
M.	**Quafafou Mohamed**	*Professeur, Université d'Aix-Marseille*	Rapporteur
Mme.	**Benbernou Salima**	*Professeure, Université Paris-Descartes*	Examinatrice
M.	**Grozavu Nistor**	*Professeur, Université Cergy*	Examinateur
Mme.	**Maucort-Boulch Delphine**	*PU - PH, UCBL - HCL*	Présidente
M.	**Benabdeslem Khalid**	*MCF - HDR, Université Lyon 1*	Directeur
M.	**Coquery Emmanuel**	*MCF, Université Lyon 1*	Co-encadrant
M.	**Elghazel Haytham**	*MCF, Université Lyon 1*	Co-encadrant

DOCTORAL THESIS OF THE UNIVERSITY OF LYON

operated within

Université Claude Bernard Lyon 1

Doctoral School No. 512
Mathematics and Computer Science (InfoMaths)

Specialty/Discipline of PhD: Computer Science

Presented and publicly defended on December 6th, 2023 by

Thomas Ranvier

Deep Learning on Incomplete and Multi-Source Data: Application to Cancer Immunotherapy

---

In front of the jury composed of:

Mrs.	**BOUDJELOUD-ASSALA Lydia**	*MCF - HDR, Université de Lorraine*	Reviewer
Mr.	**QUAFAFOU Mohamed**	*Professor, Université d'Aix-Marseille*	Reviewer
Mrs.	**BENBERNOU Salima**	*Professor, Université Paris-Descartes*	Examiner
Mr.	**GROZAVU Nistor**	*Professor, Université Cergy*	Examiner
Mrs.	**MAUCORT-BOULCH Delphine**	*PU - PH, UCBL - HCL*	Examiner
Mr.	**BENABDESLEM Khalid**	*MCF - HDR, Université Lyon 1*	Advisor
Mr.	**COQUERY Emmanuel**	*MCF, Université Lyon 1*	Co-supervisor
Mr.	**ELGHAZEL Haytham**	*MCF, Université Lyon 1*	Co-supervisor

Laboratoire LIRIS
Adresse
Bâtiment Nautibus
Campus de la Doua
25 avenue Pierre de Coubertin
69622 Villeurbanne Cedex

École Doctorale InfoMaths
7, avenue Jean Capelle
69621 VILLEURBANNE Cedex

Acknowledgments

Abstract

In this thesis work, we are interested in Machine Learning, and more specifically in the Deep Learning field, based on incomplete and multi-source data. We aim to propose innovative approaches for dealing with incomplete data in Machine Learning, and more specifically when training Neural Networks. We also focus on maximizing learning performance on multi-source data, by designing a new advanced Supervised Domain Adaptation approach. The application of our work is part of the European research project QUALITOP, which aims at improving the quality of life of patients suffering of cancer and undergoing immunotherapy treatment. We aim to design predictive models that will be integrated within an open smart digital platform to offer real-time recommendations to medical experts. Our predictive models will provide a valuable help for medical experts, to personalize and optimize treatment strategies for each patient, and help identify factors influencing patients health status. Ultimately, those predictive models can lead to designing and providing more efficient and safer immunotherapy treatments, benefiting patients and healthcare providers alike.

In this work, we are interested in maximizing predictive performance in the specific context of incomplete and multi-source data. To achieve this goal, we have directed our attention towards addressing critical research challenges that are highly relevant to this learning context and to the QUALITOP project. We first introduce a new innovative attribute noise correction method, data Denoising and Imputation in One Step (DIOS). DIOS is the first approach in the Machine Learning literature that is able to impute missing values and correct erroneous ones in a tabular dataset as a unique preprocessing step. We are then interested in maximizing Neural Network learning performance when trained on completed data. In this context, we propose two frameworks that can be used to account for imputation uncertainty during Neural Network training, leading to better predictions, Single-Hotpatching (S-HOT) and Multiple-Hotpatching (M-HOT). This work is a first step towards finding better ways to deal with missing values imputation for training predictive models, in the hope that it spikes the interest of other researchers throughout the world on this matter. Then we propose a new advanced Domain Adaptation (DA) method, Weighted Multi-Source Supervised Domain Adaptation (WMSSDA). WMSSDA is able to extract valuable knowledge from several labeled source datasets, to improve learning performance on a related target dataset. Our proposed approach includes a new component that helps limiting negative transfer through an adaptive scaling of the impact of each source on the training of the model. Finally, we showcase the application of our complete work in a unified predictive pipeline, that we ap-

ply with great success in a real-world medical scenario, highlighting the pertinence of our work for medical prediction and for the QUALITOP project.

Keywords. Attribute Noise, Imputation Uncertainty, Domain Adaptation, Machine Learning, Deep Learning, Medical Prediction.

Résumé

Dans ce travail de thèse, nous nous intéressons à l'apprentissage automatique, et plus particulièrement au domaine de l'apprentissage profond, à partir de données incomplètes et multi-sources. Nous visons à proposer des approches innovantes pour le traitement des données incomplètes en apprentissage automatique, et plus spécifiquement lors de l'entraînement de réseaux de neurones. Nous nous concentrons également sur l'optimisation des performances d'apprentissage sur des données multi-sources, en concevant une nouvelle approche avancée d'adaptation de domaines supervisée. L'application de notre travail fait partie du projet de recherche européen QUALITOP, visant à améliorer la qualité de vie de patients souffrant de cancer et suivant un traitement par immunothérapie. Nous visons à concevoir des modèles prédictifs, qui seront intégrés au sein d'une plateforme numérique intelligente ouverte, afin d'offrir des recommandations en temps réel à des experts médicaux. Nos modèles prédictifs fourniront une aide précieuse aux experts, les aidant à personnaliser et à optimiser les stratégies de traitement pour chaque patient, ainsi qu'à identifier les facteurs influençant leur état de santé. En fin de compte, ces modèles prédictifs peuvent conduire à la conception et à la prescription de traitements par immunothérapie plus efficaces et plus sûrs, bénéficiant ainsi aux patients ainsi qu'aux médecins.

Dans ce travail, nous nous intéressons à maximiser la performance prédictive dans le contexte spécifique des données incomplètes et multi-sources. Pour atteindre cet objectif, nous avons orienté notre attention vers la résolution de problèmes de recherche critiques et pertinents pour notre contexte d'apprentissage et pour le projet QUALITOP. Nous introduisons d'abord une nouvelle méthode innovante de correction de bruit d'attributs, data Denoising and Imputation in One Step (DIOS). DIOS est la première approche dans la littérature de l'Apprentissage Automatique capable d'imputer les valeurs manquantes et de corriger les valeurs erronées dans un jeu de données tabulaires en une unique étape de prétraitement. Nous nous intéressons ensuite à maximiser la performance d'apprentissage des réseaux de neurones lorsqu'ils sont entraînés sur des données complétées. Dans ce contexte, nous proposons deux frameworks pouvant être utilisés pour tenir compte de l'incertitude d'imputation lors de l'entraînement des réseaux de neurones, conduisant à de meilleures prédictions, Single-Hotpatching (S-HOT) et Multiple-Hotpatching (M-HOT). Ce travail est une première étape vers la recherche de meilleures manières de traiter l'imputation de valeurs manquantes pour l'entraînement de modèles prédictifs, dans l'espoir de susciter l'intérêt d'autres chercheurs autour du monde sur cette question. Par la suite, nous proposons une nouvelle méthode avancée d'adaptation de domaine, Weighted Multi-Source

Supervised Domain Adaptation (WMSSDA). WMSSDA est capable d'extraire des connaissances précieuses de plusieurs jeux de données labellisés, afin d'améliorer la performance d'apprentissage sur un jeu de données similaire. Notre approche proposée inclut un nouveau composant aidant à limiter le transfert négatif grâce à une pondération adaptative de l'impact de chaque source sur l'entraînement du modèle. Enfin, nous présentons l'application de notre travail complet, unifié au sein d'un pipeline prédictif, que nous appliquons avec grand succès dans un scénario médical réel, soulignant la pertinence de notre travail pour la prédiction médicale.

Mots Clés. Bruit d'Attributs, Incertitude d'Imputation, Adaptation de Domaine, Apprentissage Automatique, Apprentissage Profond, Prédiction Médicale.

Contents

List of Figures

List of Tables

Acronyms

AE Auto-Encoder. 21, 24, 25, 49–55, 87, 89, 90, 96–98, 164, 179
AI Artificial Intelligence. 1, 7, 8, X
API Application Programming Interface. 175, 176, 178
AUC Area Under the Curve. 57, 64, 94, 95, 116, 117, 119, 121–123, 148, 149, 151, 152, 168, 170, 216–220, XIII, XV, XVI, XVIII

CGAN Conditional Generative Adversarial Network. 52
CNN Convolutional Neural Network. 23, 24, 27, 55, 91, 99

DA Domain Adaptation. 5, 8, 26, 66–82, 129–131, 133, 134, 139, 141–145, 148–151, 155, 168–170, 180, 182, II, XI, XVI, XVII
DAE Denoising Auto-Encoder. 49–51, 55, 89, 90, 127, 181
DIOS data Denoising and Imputation in One Step. 5, 6, 87, 96, 97, 99–102, 109–119, 126, 127, 163, 164, 167–170, 177, 179–181, II, IV, XIII, XVII
DL Deep Learning. 2, 4, 7, 8, 11, 12, 26, 27, 38, 44, 48, 50, 53–55, 58, 71, 73, 89, 90, 94, 130, 157, 166, II, X
DNN Deep Neural Network. 11, 12, 48, 76, 91, 96, 97, 99, 100

GAN Generative Adversarial Network. 26, 52–55
GNN Graph Neural Network. 53, 54

irAE Immune-Related Adverse Event. 1, 2, 10, 11, 127, 131, 157, 158, 166, 179

KL Kullback-Leibler. 72, 81, 82

LSTM Long Short-Term Memory. 27, 54

M-HOT Multiple-Hotpatching. 5, 87, 96, 103, 106, 107, 119, 121–126, 128, 180, 181, 216–225, II, IV, XIII, XV, XVIII
MAE Mean Absolute Error. 56, 64
MAR Missing At Random. 31–33, 38, 39, 93, 109, 111–113, 119, 204–206, XI, XIII, XV, XVII
MCAR Missing Completely At Random. 31–33, 38, 39, 93, 109, 111–113, 119, 204–206, XI, XIII, XV, XVII
MD Moment Distance. 76, 77, 135, 137–139, 141, 154, 168, 180, XIV
MI Multiple-Imputation. 35–37, 50, 56, 85, 87–90, 102, 103, 105, 106, 119, 121–125, 216–225, XI, XIII, XV, XVIII

ML Machine Learning. 1–11, 14, 27, 28, 30, 34, 37, 38, 40, 42, 44, 45, 47, 48, 58, 65, 66, 71, 72, 85–87, 89, 90, 94–96, 126–128, 130, 135, 137, 156–159, 163, 165, 166, 171, 175–177, 179–181, II, X, XV, XXI
MLP Multi-Layer Perceptron. 14–16, 22, 23, 94, X, XI
MMD Maximum Mean Discrepancy. 72–77, 141, 146, 147, 154, XII
MNAR Missing Not At Random. 31–33, 38, 39, 52, 93, 94, 109, 111–113, 119, 126, 160, 204–206, XI, XIII, XV–XVII
MSDA Multi-Source Domain Adaptation. 4, 8, 67, 68, 70, 74, 76, 77, 81, 129, 130, 133–135, 138, 141, 145–148, 150, 154, 155, 158, 160, 163, 169, 170, XVI, XVII
MSSDA Multi-Source Supervised Domain Adaptation. 4, 5, 130, 132, 134, 154, 165, 168, 170, 177, 180

NLP Natural Language Processing. 27, 54
NN Neural Network. 3–5, 11–17, 19, 20, 22–27, 48, 49, 53–55, 72–74, 76, 77, 84, 85, 87, 89, 91, 95, 96, 99, 102–107, 118–120, 126, 128, 137, 147, 150, 164–170, 177, 179–181, II, X, XIII, XVII

RMSE Root-Mean-Square Error. 56, 63, 64, 90, 109, 111–116, 211, 212, XIII, XIV
RNN Recurrent Neural Network. 27, 54
ROC Receiver Operating Characteristic. 57

S-HOT Single-Hotpatching. 5, 6, 87, 96, 102–107, 119, 121–126, 128, 163, 165, 168–170, 177, 180, 181, 216–225, II, IV, XII, XIII, XV, XVII, XVIII
SDA Supervised Domain Adaptation. 67, 68, 70, 71, 74, 82, 83, 130–132, 134, 137, 141, 146, 148, II
SGD Stochastic Gradient Descent. 19, 22, 23, X, XI
SI Single-Imputation. 3, 35–37, 85, 87–89, 103, 105, 119, 121–125, 128, 216–225, XI, XIII, XV, XVIII
SSDA Single-Source Domain Adaptation. 67, 68, 70, 72–74, 77, 80, 130

UDA Unsupervised Domain Adaptation. 67, 68, 70, 71, 78–80, 82, 130, 133, 134, 146, 148

VAE Variational Auto-Encoder. 51, 52, 55, 72, 89, 90

WMSSDA Weighted Multi-Source Supervised Domain Adaptation. 5, 6, 130–132, 137, 138, 141–143, 146, 148–155, 163, 165, 168–170, 172, 180, 181, II, IV, V, XIV, XVII

Table of Notations

Notation	Description
X	Data sample
Y	Labels sample
x	Data instance
y	Label instance
$\mathcal{X}$	Feature space
$\mathcal{Y}$	Label space
$\tilde{X}$	Corrupted data sample
$\mathring{X}$	Corrected data sample
$\hat{y}$	Prediction output
$\mathbb{D}$	Domain
$\mathbb{S}$	Source domain
$\mathbb{T}$	Target domain
$P(\cdot)$	Distribution
$f(\cdot)$	Labeling function
s	Number of source domains
c	Number of classes
n	Number of instances
$\odot$	Hadamard product
$\mathcal{L}(\cdot)$	Loss function
$\mathcal{J}(\cdot)$	Objective function
$f_\theta(\cdot)$	Machine Learning model
θ	Machine Learning model trainable parameters
∇	Gradient
∂	Partial derivative
λ	Hyper-parameter
$\{\cdot\}$	Set
$[\cdot]$	Range
$(\cdot)$	List

Table 1: Main notations used throughout this manuscript.

List of Publications

International Conferences

- **T. Ranvier**, H. Elghazel, E. Coquery, K. Benabdeslem. *Accounting for Imputation Uncertainty During Neural Network Training*. DOI: 10.1007/978-3-031-39831-5_24. **DaWaK 2023**, 28-30 august 2023, Penang, Malaysia.
- **T. Ranvier**, H. Elghazel, E. Coquery, K. Benabdeslem. *Autoencoder-based Attribute Noise Handling Method for Medical Data*. DOI: 10.1007/978-981-99-1645-0_18. **ICONIP 2022**, 23-26 november 2022, New Delhi, India, (Online).

National Conferences

- **T. Ranvier**, E. Coquery, H. Elghazel, K. Benabdeslem. *Considération de l'Incertitude d'Imputation pour l'Apprentissage des Réseaux de Neurones*. **SFC 2023**: Société Francophone de Classification, 6-7 juillet 2023, Strasbourg, France.

Journals

- **T. Ranvier**, H. Elghazel, E. Coquery, K. Benabdeslem. *Deep Multi-Source Supervised Domain Adaptation with Class Imbalance*. DOI: 10.21203/rs.3.rs-3160713/v1. **KAIS**: Knowledge and Information Systems, **(Under Review)**.
- **T. Ranvier**, H. Elghazel, E. Coquery, K. Benabdeslem. *DIOS: data Denoising and Imputation in One Step*. **IJDSA**: International Journal of Data Science and Analytics, **(Under Review)**.

Publications on Unrelated Topics

Three other papers have been published during this PhD, but are not directly linked to the topic of this thesis:

- M. Hennequin, K. Benabdeslem, H. Elghazel, **T. Ranvier**, E. Michoux. *Multi-view Self-attention for Regression Domain Adaptation with Feature Selection*. DOI: 10.1007/978-3-031-30105-6_15. **ICONIP 2022**, 23-26 november 2022, New Delhi, India, (Online).
- **T. Ranvier**, K. Benabdeslem, K. Bourhis, B. Canitia. *Deep Multi-View Learning for Tire Recommendation*. DOI: 10.1109/ijcnn52387.2021.9534318. **IJCNN 2021**, 18-22 july 2021, Shenzhen, China, (Online).
- **T. Ranvier**, K. Benabdeslem, K. Bourhis, B. Canitia. *Apprentissage multi-vues pour la recommandation dans le domaine du pneumatique*. DOI: 1002655. **EGC 2021**, 25-29 january 2021, Montpellier, (Online).

Introduction

In our modern world, Machine Learning shines as a powerful force of change. In an era of data abundance, it acts as a guiding light that helps us explore new areas of knowledge. Its potential is especially exciting in the field of medical studies. Machine Learning can analyze complex patterns in large datasets, revealing new insights into human health that were previously unknown and inaccessible. It holds the key to revolutionizing disease diagnosis and personalized treatments. In this era of information, Machine Learning gives us the ability to better understand the complexities of human health, bringing us closer to a future where the advancements of Artificial Intelligence propel medical progress, enhancing the quality of life for all.

Context and Motivations

This thesis is part of the European research project QUALITOP: Monitoring multidimensional aspects of QUAlity of Life after cancer ImmunoTherapy - an Open smart digital Platform for personalized prevention and patient management[1] [2]. This project aims at improving the quality of life of patients suffering of cancer and undergoing immunotherapy treatment. Cancer immunotherapy has significantly progressed in the treatment of cancer, showing high efficacy in certain cancers, such as achieving up to a 60% objective response rate in melanoma and an 80% complete response rate in acute lymphoblastic leukemia. Despite those successes, immunotherapy is responsible for lots of Immune-Related Adverse Events (irAEs), spanning from benign reactions, such as fatigue or skin rash, to very severe and serious reactions, such as strokes or myocarditis. Those adverse reactions can drastically impact the quality of life of cancer patients, to the point where it might be better to stop the immunotherapy treatment. In the worst cases, when irAEs are too severe, the patient survival may be at risk due to the treatment. The QUALITOP project aims to design a European immunotherapy-specific open smart digital platform, that will offer real-time recommendations to enhance patient care based on patient profiles, and help identify the factors influencing patients health status.

An official goal of the QUALITOP project is the following: "Using Machine Learning approaches, QUALITOP will provide real-time recommendations stemming from patient profiles and feedbacks via the Smart Digital Platform." This thesis work aims at designing such

[1] https://cordis.europa.eu/project/id/875171

[2] https://h2020qualitop.liris.cnrs.fr/wordpress/index.php/project

Machine Learning and Deep Learning tools, to predict the risks that a patient may face when undergoing an immunotherapy treatment. In such a context, there are two main predictive tasks of interest:

- Survival outcome prediction. The ability to evaluate the survival outcome of a cancer patient that would undergo immunotherapy is of critical importance. As it allows medical experts to make informed and well-founded decisions regarding the patient treatment and care. By accurately predicting the potential outcomes and risks associated with immunotherapy for each individual patient, healthcare professionals can choose the best suited treatment depending on each patient specific needs, maximizing the chances of positive outcome and maximizing the overall patient quality of life. This information helps medical experts optimize treatment strategies, provide appropriate support, and improve as much as possible the patient quality of life throughout their cancer journey.
- Immune-Related Adverse Events prediction. Similarly to survival outcome prediction, the ability to predict the severity of adverse reactions that might occur because of an immunotherapy treatment is extremely valuable. Indeed, at the moment, there are no identified bio-markers that can help in predicting irAEs. Designing a Machine Learning model capable of predicting irAEs before they occur will not only help medical experts take better and more informed decisions, but will also show that crucial bio-markers actually exist within patients health data and can be exploited to anticipate immunotherapy adverse events. Such a predictive model helps to take proactive measures to mitigate potential irAEs and significantly improve patient outcomes.

If we predict that a patient is likely to experience severe adverse reactions from immunotherapy treatment, or has a very low chance of survival, it is crucial to make optimal decisions that prioritize and maximize the patient quality of life. In such cases, it may be more beneficial to avoid treatments such as immunotherapy, that could potentially cause additional discomfort and suffering, with limited potential for a positive outcome. Instead, it might be more pertinent to provide care, and supportive measures aimed at reducing pain, and improving comfort during the remaining time of the patient life. Such predictive models can uncover hidden patterns and bio-markers that offer crucial insights into a patient response to immunotherapy, which can help medical experts to personalize and optimize treatment strategies for each individual patient profile. Ultimately, those predictive models can lead to designing and providing more efficient and safer immunotherapy treatments, benefiting patients and healthcare providers alike.

Research Issues

Within this specific applicative context, we identified three main research issues that must be tackled. In our work, we addressed each of those research issues with the proposal of several innovative scientific solutions.

Attribute Noise Must be Corrected. Missing and erroneous values are a common issue that should be tackled as one, they are sometimes referred in the literature as attribute noise (Zhu and Wu, 2004). The presence of missing values in tabular data is a significant challenge for data analysis and the application of Machine Learning algorithms. From a practical point of view, missing values are a nuisance, as most Machine Learning algorithms require complete datasets for both learning and inference processes. Missing values are a recognized issue in data science, as they are nearly inevitable in real-world scenarios, extensive efforts have been made to address this issue in the Machine Learning literature. Erroneous values in tabular data are similarly problematic and nearly unavoidable. Research suggests that standard real-world datasets often contain around 5%, or more, erroneous values (Zhu and Wu, 2004). Such errors can arise during manual data entry or manipulation, as well as due to measurement tool inaccuracies caused by improper calibration, machine degradation, and other factors. These erroneous values can significantly degrade the quality of inferences, much like missing values. Erroneous and missing values share similarities, they are both forms of data corruptions, and need to be handled in order to maximize inference quality. While methods to handle missing values, which are only a specific subset of attribute noise, have been, and are still, extensively researched, very limited research has been conducted on designing erroneous values handling methods. More specifically, to the best of our knowledge, there is currently an absence of method in the literature capable of handling attribute noise in its entirety. Current literature often addresses missing values while neglecting erroneous ones or assuming the absence of errors. In the rare cases where both erroneous and missing values are considered, they are typically managed as distinct steps. To fill this significant gap in the Machine Learning literature, we aim to develop an approach that can correct attribute noise in tabular data as a unique preprocessing step.

Neural Network Training on Completed Data Leads to Biased Models. When dealing with incomplete data, a naive and overly used framework is Single-Imputation (Rubin, 2004). That is, to arbitrarily choose an imputation method, use it to impute the dataset, and then treating the filled dataset as if it were the actual complete data for future analyses. When training a Neural Network, or similarly strong learners, on completed data, these models can usually generalize well enough to mitigate the bias introduced by imputation uncertainty. This leads to good enough results so that researchers have not yet looked for better ways to deal with missing values when training a Neural Network. Research even indicates that, when employing strong inference models, almost any imputation approach tends to asymptotically lead to optimal predictions (Le Morvan et al., 2021). This probably contributes to the lack of investigations for accounting for imputation uncertainty to improve generalization of Machine Learning models, and thus, prediction results. Nevertheless, even in situa-

tions where strong models achieve good prediction results, their predictions are still biased by imputation uncertainty, meaning that their results can be improved ever so slightly. In our work, we aim to show that accounting for this uncertainty during the training phase of a Neural Network helps to reach even better prediction results.

Data Split Across Multiple Labeled Domains. The specific applicative context of QUALITOP involves health data being collected from various hospitals across several European countries. When dealing with several similar but distinct datasets, a naive way of training a prediction model would be by concatenating all datasets into one and train the model in a standard way. But, this would lead to poor predictive results for all hospitals. Indeed, when data is collected from distinct sources, we need to account for the domain shift that exists between all sources in order to maximize learning performance. This is the specific learning scenario of Multi-Source Domain Adaptation, where knowledge must be carefully transferred from source domains to the selected target domain during training to maximize learning performance. A well-known issue in Multi-Source Domain Adaptation is Negative Transfer, where transfer of knowledge from a source domain negatively impacts the learning performance for the target domain. This critical issue is still an open research question, and it must be tackled when designing a Multi-Source Domain Adaptation approach to reach the best possible results. In our work, we focus on designing such a Multi-Source Domain Adaptation approach that includes a component carefully designed to avoid Negative Transfer during training.

Manuscript Outline

During this thesis, we were interested in addressing the research challenges previously introduced, within our specific application context. Our work has led to several notable scientific contributions to the field of Machine Learning, which are comprehensively detailed in the following four main chapters of this manuscript.

Chapter 1: Background and Related Works. The opening chapter establishes the theoretical foundations necessary for the understanding of the research work presented in this thesis, where we aim at contributing to the field of Machine Learning within our specific context of medical application. We first explore fundamental concepts of Machine Learning and Deep Learning, to better understand their evolution, core principles, and applications. A global overview of missing values imputation techniques follows, exploring the various approaches used to address incomplete datasets to improve predictive results. We are then interested in the more generic field of attribute noise correction, where we explore methodologies for identifying and correcting both erroneous and missing data attributes. Finally, we explore the field of Multi-Source Supervised Domain Adaptation, where we are interested in maximizing the learning performance of a predictive model on a target dataset by transferring useful information from related source datasets. Through an extensive survey of literature across these fields, this chapter establishes a global understanding of the current state-

of-the-art in each area, while preparing the reader for the novel contributions presented in next chapters.

Chapter 2: Dealing with Attribute Noise. In this next chapter, we are interested in exploring and finding better ways for preprocessing datasets afflicted by both missing and erroneous values, that is, datasets afflicted by attribute noise. We find that the Machine Learning literature is missing a method that would be able to both impute missing values, and correct erroneous ones, simultaneously. We introduce a novel and straightforward approach to correct attribute noise in one preprocessing step, called data Denoising and Imputation in One Step (DIOS). We evaluate this new approach and compare its experimental results to those of other state-of-the-art imputation and correction approaches. Our experiments show that, not only does DIOS compete against existing approaches, but it also outperforms them in many instances. To the best of our knowledge, it is the first attempt to perform both erroneous values correction and missing values imputation in one preprocessing step in the literature. Moreover, we are interested in improving the training process of Neural Networks (NNs) when learning on completed data. We theorize that training a Neural Network while taking account of imputation uncertainty should lead to a less biased model, that is, a NN with a better generalization capacity. We design two new frameworks, Single-Hotpatching (S-HOT) and Multiple-Hotpatching (M-HOT), that can be used to improve the training of Neural Networks when learning on completed data, and experimentally show that they lead to better generalized NNs, and so, to better inference results.

Chapter 3: Learning with Multiple Labeled and Imbalanced Domains. This next chapter focuses on searching and proposing a Domain Adaptation approach that can be used to maximize the learning performance of a predictive model on a target dataset in our multi-source and imbalanced data application context. We introduce an innovative Multi-Source Supervised Domain Adaptation approach, named Weighted Multi-Source Supervised Domain Adaptation (WMSSDA). Our proposed approach improves the learning performance on a target domain by exploiting valuable information from related source domains during its training. WMSSDA includes a component that helps preventing the transfer of detrimental information from source domains, which would negatively impact the learning performance of the model. We experimentally show that our approach performs well under a setting of limited and imbalanced data. Our experiments show that WMSSDA outperforms other state-of-the-art Domain Adaptation approaches in most scenarios. An ablation study is performed to evaluate the individual contributions of each component of WMSSDA.

Chapter 4: Application for Survival Outcome Prediction for Covid Patients. The final chapter showcases the entire application of our work, unified within a predictive Machine Learning pipeline, applied on a real-world medical dataset. We design an advanced Machine Learning pipeline, composed of several approaches proposed in this thesis: DIOS, S-HOT, and WMSSDA. In addition to our proposed component, we add a final calibration component, which drastically improves the pertinence and reliability of the obtained predictions. This final element provides the assurance that medical professionals can place their trust in the model output probabilities, so that they can use them as valuable guidance to inform their decisions. We show that the unification of our work leads to a pipeline that is applicable and leads to very high quality results in a real-world medical survival outcome prediction context. The application of this pipeline on this dataset shows the pertinence of the proposed approaches in this thesis for our application context.

Chapter 1

Background and Related Works

Contents

In this thesis we are interested in contributing to the advancement of the Machine Learning (ML) scientific field in our specific medical context. This chapter introduces the various fundamental concepts necessary to understand this manuscript. We first give a brief overview of Machine Learning, introducing key concepts for the understanding of next sections and situating the Machine Learning and Deep Learning (DL) fields in relation to Artificial Intelligence (AI). Section 1.2 gives a detailed overview of the imputation literature on tabular data, from statistical to Deep Learning methods, with a main focus on Machine Learning approaches. We introduce the concept of attribute noise and the ways to deal with it in a Machine Learning context in section 1.3. Finally, section 1.4 gives a global overview of

the Domain Adaptation (DA) literature, with a particular interest for Multi-Source Domain Adaptation (MSDA) on tabular data.

1.1 Machine and Deep Learning

Machine Learning is a sub-field of the Artificial Intelligence field. An Artificial Intelligence can be defined as a system that is able to perform a task that is commonly thought to require human-like intelligence. That is, any system able to demonstrate any intelligence. Note that an intelligent system does not necessarily mean a system that learns. Some examples of AI systems that do not rely on learning are: path-finding algorithms, expert systems, fuzzy logic, etc. Those are rule-based systems, they rely on manually specified rules, for long they have been used to solve problems and automate tasks that could only be solved by humans beforehand. Defining the rules of such a system is hard, laborious and sometimes almost impossible. For example, defining a set of rules precise enough to perform image classification quickly becomes impossible as any variance in the image set must be taken under account. Machine Learning aims to automate the manual definition of those rules by learning them automatically, it exclusively focuses on algorithms that are able to improve from experience. A ML algorithm automatically learns common patterns in data and defines generalized rules from them, it is then able to exploit its learned rules to make prediction on new data. ML algorithms can extract more complex and abstract knowledge from data than can be manually defined by humans, tasks that rule-based systems struggled with can be solved with ML. For example, classifying images of handwritten digits is a task that is easy to solve with ML, but very hard to solve with manually defined rules. In the following sections we will focus on, and describe more precisely, the Deep Learning (DL) field, which is a sub-field of ML. Figure 1.1 illustrates where are ML and DL situated in relation to AI.

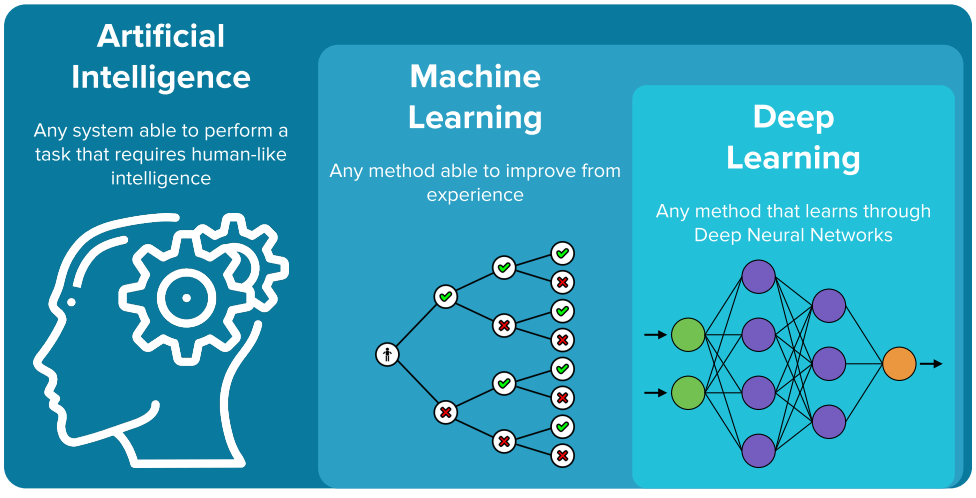


Figure 1.1: Fields of Artificial Intelligence, Machine Learning and Deep Learning.

1.1.1 Machine Learning

In his book "Machine Learning", (Mitchell, 1997) proposed the following Machine Learning definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". In our work we refer to those learning computer programs as ML algorithms or models.

A ML model is trained to perform a task on observed data, its goal being to obtain the best possible results on this known dataset, to then be able to perform the same task on never observed data. In Mitchell's definition the experience E corresponds to the observed data, which we call a training set. The training set is composed of examples that are assumed to originate from a same distribution. It can be seen as a discrete sample of an underlying distribution. We want the model to learn this distribution, as we assume that non-observed data originate from the same distribution. We call the non-observed data the test set, on which we want to perform the same task that the model learned on the training set. Since the distribution between training and test data should be the same, by learning to perform a task on training data, the model should be able to perform the same task on test data. Simply put, the objective of a standard ML model is to learn generalizations from a training set in order to maximize predictions quality on a test set composed of samples following the same distribution but never observed by the model. Technical details on the learning process of a ML model are given in the section 1.1.2 of this chapter.

There are three main learning settings in ML, they are divided depending on the nature of the data (Bishop, 2006):

- **Supervised Learning:** In a supervised learning context, the training set is composed of inputs associated to their output, this is called a labeled dataset, since the output is known given the input. In this training set, the inputs are features, and their associated outputs are the targets we aim to predict given the features. On the other hand, the test set is composed of inputs for which the outputs are unknown. The goal is to learn the mapping from input to output on the training data to predict outputs on test data, assuming that test data originates from the same distribution as training data.

  For example, we want to predict the survival outcome of a Covid patient given its health data, figure 1.2 illustrates a possible training set in this context. In this example, a training instance is composed of the patient's health features (age, sex, weight, etc.) as the input, and the binary survival outcome of this patient as the output. The training set is composed of an important amount of such data points. In this case the test set is composed of new Covid patients that are being treated, we aim to predict their survival outcome given their health features. A ML algorithm is trained to learn the mapping health features to survival outcome on the observed training data. Once trained, the generalization capacity of the algorithm is exploited to predict survival outcome of new Covid patients.

- **Unsupervised Learning:** In an unsupervised learning context, the dataset is only composed of inputs without any corresponding target, this is called an unlabeled dataset.

Age	Sex	Weight	...	Survival
71	Male	86	...	False
58	Female	54	...	True
26	Male	72	...	True
42	Female	65	...	False

Figure 1.2: Example of a supervised training set on fake Covid patients, patients health features are age, sex, weight, etc. and the target output is the survival column.

The ML algorithm must automatically find patterns and correlations in the data without relying on known targets. There are several possible goals in this learning setting (Bishop, 2006), it might be to automatically organize the data in distinct clusters (Jain and Dubes, 1988), to learn the underlying distribution (Knuth, 2006), or to dimensionally reduce the data while retaining as much knowledge as possible (Maaten and Hinton, 2008).

- **Reinforcement Learning:** In reinforcement learning, an agent learns what the best action to perform is by observing the consequences of its own actions in its environment and learning through trial and error. The agent is given a reward in accordance to the quality of its actions, the algorithm learns to maximize this reward by discovering the association between its last observation and the best action to perform (Sutton and Barto, 2018).

In this work, we almost exclusively focus on the supervised learning setting, although unsupervised and reinforcement learning share key concepts that are important to be aware of. In section 1.1.2 we formulate in greater details and more formally the supervised learning setting.

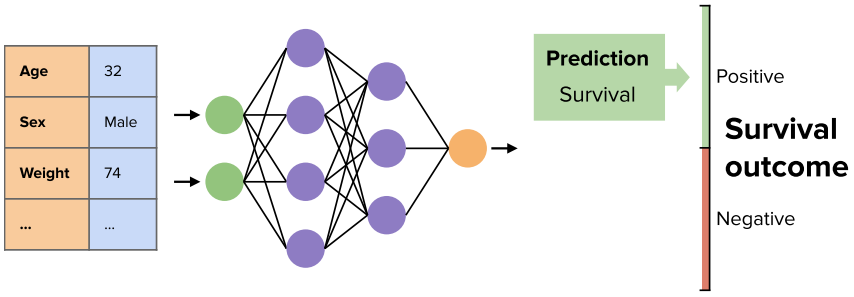
In the remainder of this manuscript, we deal with two classic supervised learning tasks, namely classification and regression, which are illustrated through examples in figure 1.3:

- **Classification:** Classification aims at predicting a discrete target output given an input, which is its category. In this case, data samples are classified in at least two separate categories, the learning algorithm must learn the mapping from inputs to their class. As this task is supervised, the training set is composed of labeled instances that the algorithm learns from.

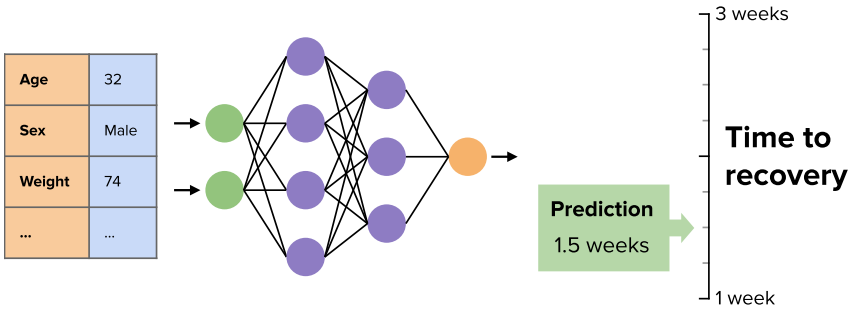
  In our work, we ultimately want to predict survival outcome and Immune-Related Adverse Events (irAEs) in cancer patients treated with immunotherapy, which we identified as a classification task (i.e. we aim to predict which adverse event(s) the patient is most likely to develop). We present our classification system in detail in chapter 3 and its application to our real-world medical data in chapter 4.

- **Regression:** Regression aims at predicting a continuous target output given an input, which can be a score, a scale, etc. Here, training samples are labeled with the continuous output value corresponding to their input and the algorithm learns to estimate its value.

In our work, before being able to perform prediction of survival outcome and Immune-Related Adverse Events, we must first clean the data from missing and noisy values. We propose to solve this problem through a regression task, we present our preprocessing method in detail in chapter 2, and its application to our real-world medical data in chapter 4.



(a) Classification task on the Covid example, prediction of survival is discrete.



(b) Regression task on the Covid example, estimation of time to recovery is continuous.

Figure 1.3: Example of a classification task on top and a regression task on the bottom.

Similarly, ML algorithms are not trained or evaluated in the exact same way depending on the task, in the following section we explain in more details how Neural Networks (NNs) are trained in both classification and regression contexts.

1.1.2 Deep Learning

In this section, we describe in a more general way the field of Deep Learning and the required concepts for the understanding of the rest of this thesis manuscript.

Deep Learning relies on Deep Neural Networks to extract and exploit progressively higher level features from raw data to perform a given task (LeCun et al., 2015). An advantage of DL models over other ML ones are their capacity to handle raw data without requiring much/any feature engineering. Deep Neural Networks (DNNs) are able to learn and extract low to high level features from raw data, automatically learning to perform simple feature engineering tasks such as feature selection, linear transformations, etc. DNNs lower layers

are known to learn low level features, such as angles or edges in images, while higher layers identify complex and abstract concepts that are useful for the task to solve. Thanks to their high abstraction potential, DNNs can learn from highly dimensional data.

A Neural Network (NN) is a function approximator, it learns a continuous function given a discrete sample from that function. In practice, we assume that our given data is a discrete sample that originates from a unique distribution, every instance in the dataset is drawn from this distribution, plus a certain amount of noise. We employ a Neural Network to approximately learn this underlying distribution from the accessible part of the data, that is, our training set, and apply the trained NN to new data instances, that is, the test set.

The term "Deep" in Deep Learning refers to the number of layers used in DNNs through which raw data is transformed. That is, DNNs apply a substantial amount of successive transformations from input data to output. For a feed-forward Neural Network, that is, a model in which information flows forward through layers from the input layer and through each hidden layer until reaching the final output layer, this depth corresponds to the number of hidden layers plus the output layer. There is not a precisely defined depth that differentiates DNNs from shallow NNs, but a general consensus in the DL literature is to talk about a DNN when the network depth is higher than 2. DNNs are known to reach better inference results than shallow NNs by extracting higher level and more complex features from the data thanks to their extra layers.

It has been mathematically proved that feed-forward NNs are universal approximators of any continuous or discontinuous multivariate functions. The first pioneer paper in NN theory is (Hecht-Nielsen, 1987), that proves that for any given continuous multivariate function, there exists a three layer NN that can be constructed to model the function exactly. Soon after, (Hornik et al., 1989) extends the previous work and proves that a two layer feed-forward NN with finite number of neurons is capable of approximating any continuous function to any desired degree of accuracy. Very recently, (Ismailov, 2023) proved that a three layer NN can represent not only any continuous multivariate function, but also any discontinuous multivariate function. Other works have proved that NNs are universal approximators of any multivariate function using various types of activation functions, such as with the formerly often used Sigmoid (Cybenko, 1989), or with modernly used unbounded activation functions, such as ReLU (Sonoda and Murata, 2017). Though no universal rule exist that dictates how to construct and train such Neural Network, those works prove that for any given function there exists, theoretically, a feed-forward Neural Network that models it exactly. As stated in (Hornik et al., 1989), this implies that failure in applying a NN necessarily comes from inadequate learning, insufficient number of hidden neurons or a lack of deterministic relationship between the input data and the target to predict.

1.1.2.1 Problem Formulation

We formally describe the common problem of classification in a supervised learning context, that will serve as a basis for the rest of this section and manuscript.

Let $\mathcal{X} \in \mathbb{R}^d$ denote the input feature space, with d the number of features, and $\mathcal{Y} \in \{1, \ldots, c\}$ the multi-class output label space, with c the amount of classes. We note $X = \{x_i \in \mathcal{X}\}_{i=1}^n$ the data sample, with n the number of instances in the dataset. We assume the associated labels are constructed given the labeling function $f : \mathcal{X} \to \mathcal{Y}$ mapping from feature space to label space. Thus, we note $Y = \{f(x_i)\}_{i=1}^n$ the associated label sample. We assume that data instances are drawn from the underlying marginal distribution noted $P(X)$. Finally, we note $\mathbb{D} = (P(X), f)$ the dataset from which instances and corresponding labels are drawn.

An instance $x \in X$ can be any type of data, such as tabular data, time-series, images, etc. In a classification context the label $y = f(x)$ corresponds to the index of a class corresponding to the instance x, in a regression context y would be a continuous value.

In a supervised learning context, the goal is to learn the labeling function f, so that we can predict the associated label of any data instance $x \in \mathcal{X}$. This is based on the assumption that test instances are drawn from the same marginal distribution as training instances, $P(X_{test}) = P(X_{train})$, and that the labeling function f remains identical between training and test samples. A prediction obtained on a data instance x given the learned labeling function f_θ is noted $\hat{y}$:

$$\hat{y} = f_\theta(x) \tag{1.1}$$

Common practice is to note the learned labeling function f_θ, with θ the parameters of the model used for predictions. The model parameters θ are learned during training based on training data instances and associated labels drawn from the dataset $\mathbb{D}$. The loss function $\mathcal{L}(\hat{y}, y)$ is used to measure the prediction error of the model between the true label y and the predicted one $\hat{y}$. The objective function $\mathcal{J}$ is the mean prediction error over the whole training set.

$$\mathcal{J}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f_\theta(x_i), y_i) \tag{1.2}$$

Model parameters are learned by minimizing the objective function $\mathcal{J}$ over the labeled training dataset composed of n instances $\{(x_i, y_i)\}_{i=1}^n$, where $\{x_1, \ldots, x_n\} \sim P(X_{train})$ and $y_i = f(x_i)$. Optimal parameters are noted θ^*.

$$\theta^* = \arg\min_\theta \mathcal{J}(\theta) \tag{1.3}$$

Where argmin is the function that returns the parameters for which the minimal value is obtained.

The rest of this section will describe more precisely what are Neural Networks, how the training phase leading to learned model parameters takes place, how loss functions are defined, what kind of NNs can be used, etc.

1.1.2.2 Neural Networks Architecture

Artificial Neural Networks are ML models inspired by the biological neural networks in our brains. They are composed of units called neurons organized as a network, with several layers composed of multiple neurons. The input of the model is fed through the network, traveling from the input layer to the output layer. The obtained output corresponds to the prediction of the model for the given input.

The Neuron: Unit Piece of a Neural Network. The unit piece of a Neural Network is the artificial neuron, also called a perceptron, inspired by the biological neuron and originally proposed in (McCulloch and Pitts, 1943). An artificial neuron is given a vector as input and produces a single scalar as output, it operates a linear transformation of its input, followed by an activation function, which adds a non-linearity to the output. Typically, artificial neurons are organized as a network composed of multiple layers, that is, a feed-forward fully-connected Neural Network, also called Multi-Layer Perceptron (MLP). Each neuron in one layer is connected to all neurons from the previous layer and all neurons of the next layer (fully-connected). Data instances are given as input vectors to the first layer, which is called the input layer. The layer that produces the final result is called the output layer. All layers in-between are called hidden layers. Information is fed through the input layer and is passed from one layer to the next until the final result is obtained from the output layer (feed-forward). This is the most classic way of organizing the architecture of a NN, other types of NN architectures exist and will be described in section 1.1.2.4.

We use the same notations as above, we consider a vector input noted $x \in \mathbb{R}^d$, with d the number of features. The neuron applies a linear transformation composed of a scaling with a dot product between the input vector $x \in \mathbb{R}^d$ and a weight vector $w \in \mathbb{R}^d$, and a translation with a bias $b \in \mathbb{R}$ added to the previous result. The result of this linear transformation is then fed through an non-linear activation function σ, also called a non-linearity. We note the neuron f_θ, with $\theta = \{w, b\}$ the set of parameters of the neuron. The obtained output is a single scalar $\hat{y} \in \mathbb{R}$.

$$\hat{y} = f_\theta(x) = \sigma(x \cdot w^T + b) \tag{1.4}$$

Going from a single neuron to a layer of neurons, and from a vector input to a batch of instances, is mathematically trivial. The weights of the layer neurons are noted $W \in \mathbb{R}^{m \times d}$ and the biases are noted $b \in \mathbb{R}^m$, with m the number of neurons in the layer. Layer parameters are $\theta = \{W, b\}$. The input batch matrix is noted $X \in \mathbb{R}^{n \times d}$, with n the number of instances in the batch and d the number of features. Through the same equation 1.4, the obtained output is a vector of m real values $\hat{y} \in \mathbb{R}^m$. Section 1.1.2.3 describes how parameters θ are found. Figure 1.4 shows a visual representation of the formal artificial neuron.

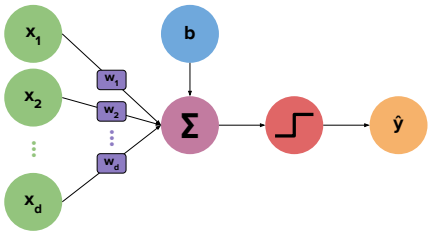


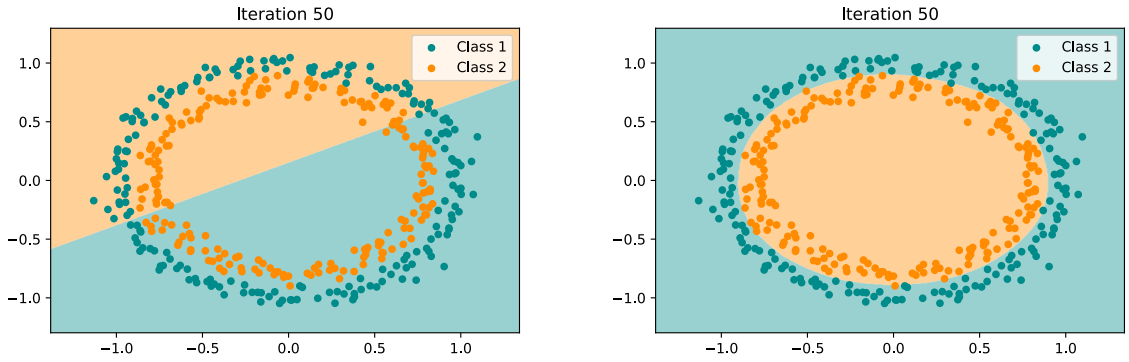
Figure 1.4: Representation of the computational graph of an artificial neuron.

Activation Functions. In the previous section, we introduced the concept of non-linear activation functions, in this section, we detail why activation functions are important and give examples of the most commonly used ones.

Let's consider a binary classification problem with bivariate data $X \in \mathbb{R}^{n\times 2}$, that is, a problem where we want to discriminate data instances between two possible classes $\mathcal{Y} = \{0,1\}$. In such a context, a linearly separable problem is one in which it is possible to discriminate between the two classes by simply drawing a straight line between instances of one class and others. Respectively, a non-linearly separable problem is one where it is not possible to discriminate between the two classes using just a straight line. In this non-linear context, a linear model will not be able to properly solve the problem. As seen in the previous section, without activation functions, neurons apply only a linear transformation to their input. A succession of linear transformations of an input necessarily leading to a simple linear transformation of the input, a Neural Network without activation function cannot solve non-linearly separable problems. By adding a non-linear transformation to neurons outputs, non-linear activation functions allow Neural Networks to solve non-linearly separable problems. Figure 1.5 shows the decision boundaries computed using a MLP on a non-linearly separable binary classification problem. On the left of the figure the NN uses no activation function, on the right a non-linear activation function is used (ReLU). As can be seen, the NN is able to construct a non-linear decision boundary that correctly discriminates between the two classes when using a non-linear activation function. This is what makes Neural Networks a tool that is so powerful, as almost any complex non-linear problem can be solved using a properly dimensioned and trained Neural Network.

The choice of an activation function relies on some important points and empirical results. As seen previously, it is primordial for the function to be non-linear. We want the activation function to be monotonic, that is, a function that only increases or decreases. It is also important for the chosen function to be continuously derivable, or at least piece-wise derivable, as the derivative of the activation function is used in the back-propagation phase, which will be described in next section. We list some known and used activation functions, their equations are given in Table 1.1 and Figure 1.6 shows their visual representation:

- **Sigmoid:** The sigmoid is one of the most known activation functions, it has been widely used in NNs. It has several advantages, the sigmoid function is non-linear, monotonically increasing, differentiable in all points, and bounded, output values are within the range $[0,1]$. It is less used in practice now that better results are reached in most cases using the ReLU function.



(a) No activation function

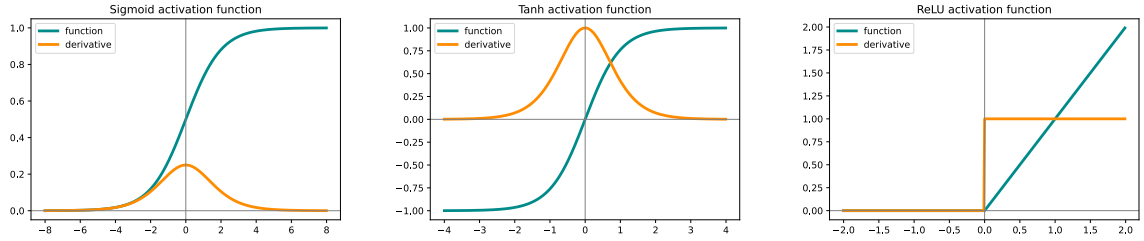
(b) Non-linear activation function

Figure 1.5: Decision boundaries computed using a MLP on a non-linearly separable binary classification problem. On the left the NN uses no activation function, leading to a linear decision boundary. On the right a non-linear activation function is used (ReLU), leading to a non-linear decision boundary. The discrimination of the NN is symbolized by the color of the corresponding class.

- **Tanh:** The hyperbolic tangent is similar to the sigmoid function, it has the same advantages with the addition of being bounded in the range $[-1,1]$ and centered around 0. It is most often used on the last layer of generative models, which are used to construct synthetic data.
- **ReLU:** ReLU stands for Rectified Linear Unit, it is defined as the positive part of its argument. This function is also non-linear and monotonically increasing, but it is not bounded and non-differentiable in 0. As it is piece-wise derivable, a common practice is to arbitrarily chose its derivative at 0 to be either 0 or 1. It is now the most popular and widely used activation function.

Name	Function	Derivative
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	$\sigma'(x) = \sigma(x)(1-\sigma(x))$
Tanh	$\sigma(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$\sigma'(x) = 1 - \sigma(x)^2$
ReLU	$\sigma(x) = \max(0,x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$	$\sigma'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$

Table 1.1: Commonly encountered Sigmoid, Tanh and ReLU activation functions equations with their corresponding derivative



(a) Sigmoid (b) Tanh (c) ReLU

Figure 1.6: Commonly encountered Sigmoid, Tanh and ReLU activation functions visual representation with their corresponding derivative

1.1.2.3 Training a Neural Network

As formalized in section 1.1.2.1 and precised in section 1.1.2.2, a feed-forward Neural Network f_θ takes an input vector x and information flows forward through the network to produce its output $\hat{y}$. Information is initially provided by the input x, information is then propagated from the input layer to the first hidden layer, and successively until the output layer is reached and outputs $\hat{y}$. This process is called forward-propagation, this is how we draw inference from the model. During training, a loss function $\mathcal{L}$ is used to compare the predictions of the model $\hat{y}$, to the target y. We note $\mathcal{J}(\theta)$ the objective function of the Neural Network defined in equation 1.2 which computes the mean prediction error of the model over the training data. In order to train the model, we aim to find the model parameters θ^* that lead to the lowest possible value of $\mathcal{J}(\theta)$ by solving equation 1.3. In practice, it is unfeasible to find an exact solution to this equation, since there are as many unknown variables as there are model parameters in the set θ. Instead, we aim to find parameters θ^* that minimize as much as possible the value of $\mathcal{J}(\theta)$, in this section, we describe how such optimization problem is solved using back-propagation and a gradient descent algorithm.

Back-Propagation. We aim to minimize the objective function $\mathcal{J}(\theta)$ value by optimizing the model parameters θ. A way to solve this optimization problem is by computing the gradient of the objective in regard to model parameters, noted $\nabla_\theta \mathcal{J}(\theta)$, and then using a gradient descent algorithm to fit parameters θ in accordance. The gradient is computed using the back-propagation algorithm (Rumelhart et al., 1986). The term back-propagation only refers to the algorithm used to compute the gradient, but is often confused with the entire learning algorithm, which would be the association of back-propagation and gradient descent (Goodfellow et al., 2016).

The loss between the feed-forward Neural Network output $\hat{y}$, given a data instance x, and its associated target y, is computed as:

$$\mathcal{L}(\hat{y}, y) = \mathcal{L}(f_\theta(x), y) = \mathcal{L}(\sigma_L(\sigma_{L-1}(\dots\sigma_1(xW_1^T + b_1)\dots W_{L-1}^T + b_{L-1})W_L^T + b_L), y) \tag{1.5}$$

With L the total number of layers in the Neural Network, σ_L the activation function at layer L and $\theta = \{W_1, \dots, W_L, b_1, \dots, b_L\}$ the set of trainable parameters of the model. We note $z_l =$

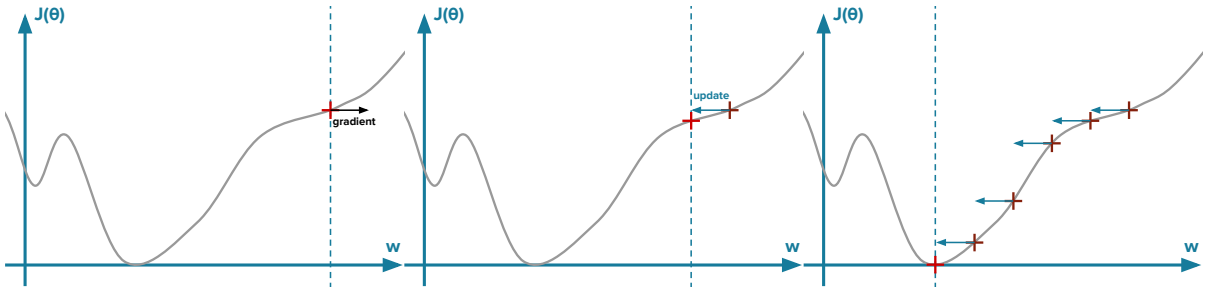
$z_{l-1}W_l^T + b_l$ and $a_l = \sigma(z_l)$ the intermediary results obtained at layer l in equation 1.5.

The back-propagation algorithm works by computing the gradient of the objective function in regard to each model parameter using the chain rule. The gradient $\nabla_\theta \mathcal{J}(\theta)$ is the Jacobian matrix composed of the partial derivatives of $\mathcal{J}(\theta)$ with regard to each parameter. The gradient for a given parameter, say $w_{1,j,k}$ for the weight at position j,k in layer 1, is noted $\frac{\partial \mathcal{J}(\theta)}{\partial w_{1,j,k}}$ and is computed in the following way:

$$\frac{\partial \mathcal{J}(\theta)}{\partial w_{1,j,k}} = \frac{\partial \mathcal{J}(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_L} \frac{\partial a_L}{\partial z_L} \frac{\partial z_L}{\partial a_{L-1}} \cdots \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial w_{1,j,k}} \tag{1.6}$$

Once the entire gradient $\nabla_\theta \mathcal{J}(\theta)$ is computed we use a gradient descent algorithm to update parameters in accordance.

Gradient Descent Algorithms. As we aim to minimize $\mathcal{J}(\theta)$, computing its gradient in regard to model parameters $\nabla_\theta \mathcal{J}(\theta)$ gives the direction in which the function increases in regard to each parameter. Necessarily, by updating each parameter in the opposite direction than the one given by the gradient computed in regard to this specific parameter, we minimize the global $\mathcal{J}(\theta)$ value. Figure 1.7 shows a simple visual representation of the optimization process in regard to one parameter w.



(a) Gradient $\frac{\partial \mathcal{J}(\theta)}{\partial w}$ is computed (b) Update of w in opposite direction (c) Optimization of w until convergence

Figure 1.7: Visual representation of $\mathcal{J}(\theta)$ in regard to parameter w, current value of w is symbolized with the blue dotted vertical line, the corresponding value of $\mathcal{J}(\theta)$ is visualized as the red cross. On (a) the gradient which points in the direction the $\mathcal{J}(\theta)$ curve increases is visualized as the black arrow pointing to the right. On (b) the optimization process decreases the value of w which in return minimizes the value of $\mathcal{J}(\theta)$. On (c) the same process is repeated until convergence, that is, until a minimum is reached.

The gradient descent algorithm is an iterative algorithm that update each parameter given the gradient $\nabla_\theta \mathcal{J}(\theta)$ computed using the back-propagation algorithm. Trainable parameters of the model θ are initially randomly defined. Parameters are updated in the following way:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{J}(\theta) \tag{1.7}$$

The algorithm iteratively updates parameters until convergence, that is, until the objective function $\mathcal{J}(\theta)$ stops decreasing or reaches a defined threshold value. The variable η is called

the learning rate, this value defines the size of the corrective steps at which parameters are updated. If η is set too high, it can results in too large updates, which facilitates fast learning but may lead to instability and poor prediction quality. Conversely, if η is set too low, learning may be slower, and there is a risk of becoming trapped in a sub-optimal local minimum, which can also lead to poor prediction quality. To achieve the best possible inference results, it is primordial to establish an appropriate value for the learning rate. The learning rate is a hyper-parameter, a kind of parameter that is manually set before the training phase starts. Hyper-parameters such as the learning rate and the number of neurons in hidden layers are not learned during training, unlike model trainable parameters.

To apply back-propagation and gradient descent on the entire objective function at once, the model is fed the complete training set and the gradient is computed in its entirety. This process is highly impractical in practice as it quickly becomes too heavy when dealing with large training sets and leads to overfitting issues (detailed in paragraph 1.1.2.3). In practice, a commonly used approach is to use batch training, where small batches of randomly selected training instances are used to compute a partial gradient and back-propagation and gradient descent are performed on the loss computed from those instances only. The most basic optimization algorithm that implements this principle is Stochastic Gradient Descent (SGD), the process remains identical than with standard gradient descent but the gradient is only computed on the randomly drawn instances. Using such algorithm is faster and yields better results by encouraging generalization, avoiding overfitting.

SGD is the most basic gradient descent algorithm that is currently used when training NNs. Much improvements and variations of this algorithm have been proposed in the literature to improve NNs learning performance. A popular practice is to linearly decay the learning rate value until a determined iteration τ (Goodfellow et al., 2016). This allows a fast convergence at the beginning of the training, avoiding small local minimums, and a more accurate and slow convergence at the end of the training, leading to more accurate results. Another practice is to introduce momentum (Polyak, 1964) to accelerate training. Momentum is accumulated in the average direction of past gradients and continues to move in their direction, the momentum is added to the gradient when applying the parameters update. The effect is a faster convergence when gradients remain in the same direction for several iterations, and slower otherwise, improving the overall convergence by finding better quality local minimums. An update of this concept have been proposed in (Sutskever et al., 2013), as Nesterov momentum, which is commonly used in practice. Instead of adding momentum to the gradient on the update, Nesterov momentum first applies an update in direction of the momentum and then computes the gradient from this new point to update again the parameters, which is slightly different and seems to lead to faster convergence overall. A default practice is to initialize trainable parameters on a random uniform distribution, but there are other possibilities, such as initializing parameters on a normal distribution. Initialization of the parameters is an important step that is not fully understood and still under active research. Nowadays, the most commonly used optimization algorithms aim at automatically adapting the learning rate during training in order to maximize convergence and minimize the impact the initial η value has on training. The AdaGrad algorithm (Duchi and Hazan, 2011) individually adapt the learning rate to each parameter by scaling the learn-

ing rate value to the average gradient value of the corresponding parameter. A parameter leading to high gradients will have a decrease in its learning rate, and inversely. RMSProp (Hinton, 2012) is an updated version of AdaGrad designed to converge rapidly when applied to a convex objective function. It is currently one of the most used optimization algorithms. The Adam algorithm (Kingma and Ba, 2014) can be seen as a combination of RMSProp and momentum, it is known as not being too sensitive in the set learning rate value, which simplifies its usage. It is currently the most popular and used optimization algorithm. There is currently no consensus in the literature as to how to choose the right optimization algorithm (Goodfellow et al., 2016). Previously presented algorithms are all commonly used, with the most popular one being Adam. The choice of the optimization algorithm largely depend on empirical testing and personal preferences.

Loss Functions. In order to train a Neural Network to solve a supervised task, it is important to use a well-adapted loss function $\mathcal{L}$, that is, a function that will evaluate consistently the NN capacity to succeed at the task on training data. As the partial derivative $\frac{\partial \mathcal{J}(\theta)}{\partial \hat{y}}$ is required to be computed for training it is important to choose a loss function that is continuously differentiable. As for activation functions, it is possible to use a loss function that is piecewise differentiable, in this case, values at discontinuities are to be determined arbitrarily.

In the case of a binary classification task, we wish to discriminate data instances between two classes. Thus, we wish to evaluate how close the model probability output $\hat{y} \in [0,1]$ is from the target label $y \in \{0,1\}$. In this case the most commonly used loss function is the Binary Cross-Entropy loss.

$$\mathcal{L}_{BCE}(\hat{y}, y) = -(y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y})) \tag{1.8}$$

In the more general case of a multi-class classification task, we wish to evaluate how close the model probability vector output $\hat{y} \in [0,1]^c$, is from the target vector $y \in \{0,1\}^c$. In this case, the most commonly used loss function is the general Cross-Entropy loss, the Binary Cross-Entropy loss is in fact a special case of the general Cross-Entropy loss.

$$\mathcal{L}_{CE}(\hat{y}, y) = -\sum_{i=1}^{c} y_i \cdot \log(\hat{y}_i) \tag{1.9}$$

In the case of a standard regression task, we wish to evaluate how close the output of the model is from the target output, in this case, both model output and target have the same format, and it is possible to simply evaluate the distance between the two. Let's take the example of a generative model that we train to construct target images of 28×28 pixels, in this case we wish to evaluate how close the model generated output $\hat{y} \in \mathbb{R}^{28,28}$ is from the target image $y \in \mathbb{R}^{28,28}$. In this case, the commonly used loss function is the Mean Squared Error, that is, the square of the Euclidean distance (also called L_2-norm).

$$\mathcal{L}_{MSE}(\hat{y}, y) = (\hat{y} - y)^2 \tag{1.10}$$

Loss Regularization. The loss is what we use to control and point the learning in the right direction. Adding other terms to previously presented loss functions can help to push the model to respect a given constraint during its training. It is possible to add any regularization term to the loss at the condition that this new term is also continuously or piece-wise differentiable, and that some or all model parameters are included in the computation of the term. Solving a regularized loss is a balance game, where both the original loss term and the regularization term are minimized, common practice is to scale the regularization term with a defined hyper-parameter λ to limit its impact on learning. The number of regularization terms that can be added to the loss term is not limited, but a simpler loss will lead to faster convergence than a more complex one, and convergence might even be impossible if the loss is overly complicated. Widely used constraints are the L_2 and L_1 regularizations of the weights, their goal is to directly regulate the model parameters, to be as small possible in the first case and as sparse as possible in the second.

The L_2 regularization, also commonly called Ridge regression or weight decay, is often used to penalize the parameters of the model to get closer to 0. This has the effect of limiting the overfitting of the model during training, more information are given about overfitting in the next section. The objective function regularized with an L_2 regularization is noted $\tilde{\mathcal{J}}(\theta)$:

$$\tilde{\mathcal{J}}(\theta) = \mathcal{J}(\theta) + \frac{\lambda}{2}||\theta||_2^2 \tag{1.11}$$

And its gradient is simply:

$$\nabla_\theta \tilde{\mathcal{J}}(\theta) = \nabla_\theta \mathcal{J}(\theta) + \lambda\theta \tag{1.12}$$

The L_1 regularization is also commonly used to penalize the parameters of the model to become sparse, that is, as many parameters as possible will be set to 0 while the model learns to solve the task. This leads to a model in which some connections are disabled, which is often seen as a form of automatic feature selection. The regularized objective function is:

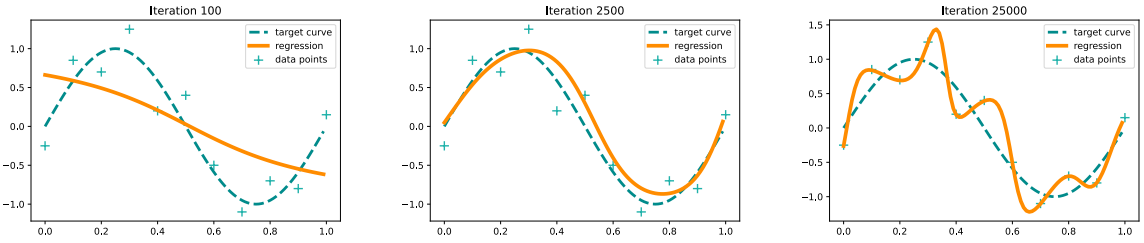
$$\tilde{\mathcal{J}}(\theta) = \mathcal{J}(\theta) + \lambda||\theta||_1 \tag{1.13}$$

With its gradient:

$$\nabla_\theta \tilde{\mathcal{J}}(\theta) = \nabla_\theta \mathcal{J}(\theta) + \lambda \text{sign}(\theta) \tag{1.14}$$

Other regularizations are often used to force a model to respect some given constraints during its training, such as preventing the model from choosing a trivial solution that would nullify the usefulness of the model, ensuring that a mathematical property is respected, etc. When training several models it is also possible to include the loss term of other models in one common loss to minimize, a common example are Auto-Encoders (AEs), such situation will be encountered on many occasions in the following work.

Under and Overfitting. A problem that is common to the training of any Neural Network is finding a balance between under and overfitting. Simply put, underfitting is when the NN has not yet or not been able to learn enough generalization from training data to be applied successfully to test data, while overfitting is when the NN has learned over-specific concepts on training data that are not true for test data. Underfitting usually occurs either when the model has not been trained enough or if the model is not well dimensioned and simply cannot learn interesting generalization about the data because of a limited learning potential. Overfitting usually occurs when the model is trained for too many iterations or trained on a limited dataset. Figure 1.8 shows a representation of under, good and overfitting on a simple toy experiment where a MLP is trained on a simple 2D regression task using SGD.

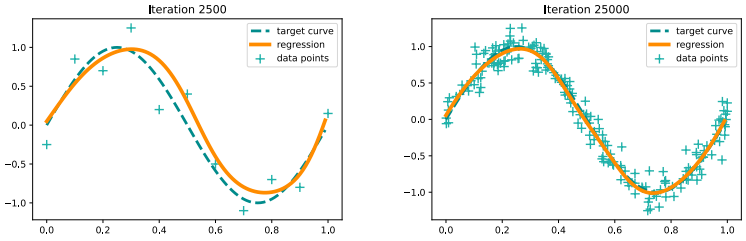


(a) Underfit (b) Good fit (c) Overfit

Figure 1.8: Toy experiment where a MLP is trained on a regression task using SGD, blue points are the training instances, the dotted line is the target curve we want the model to approximate. Training points originate from this distribution with an added random noise. In (a) the model has not been trained enough (100 iterations), which leads to underfitting. In (b) the model has been trained for a proper amount of iterations (2500 iterations) and leads to relatively good generalization. In (c) the model has been trained too much (25000 iterations) and has overfit training data.

An obvious way of reducing the overfitting risk and improving the generalization of the model is to get access to more data. In such case the model is less sensitive to overfitting, making it easier to obtain a well generalized and more accurate model. Figure 1.9 shows the same regression example as above to illustrate the difference between the best fit obtained with a limited training set and a better fit obtained with a larger training set.

In practice it is common to divide the training set in two, the most important part of the split is used as training data, the other part is used as a validation set. The validation set is used to regularly evaluate the model during training on data that is not used during training. When using such technique it becomes possible to compare the evolution of the mean loss value on the validation set compared to the training set. Intuitively, both losses decrease during training, as the model learn generalization from training data. After a certain amount of gradient descent iterations the validation loss starts to increase while the training loss keeps decreasing, this is the moment where the model starts overfitting. The training loss keeps decreasing since the model learns over specific concepts about training data, but validation loss increases as the model loses its generalization capacity on unseen data. This is used as an indication that the training must be stopped, this is called early stopping. Early stopping is a good practice that most data scientists use when training any model.



(a) Best fit on 10 instances (b) Best fit on 200 instances

Figure 1.9: Toy experiment where a MLP is trained on a regression task using SGD, blue points are the training instances, the dotted line is the target curve we want the model to approximate. In (a) the model is trained using only 10 instances, leading to a poor fit. In (b) the model is trained using 200 instances, leading to a well fit and more accurate model.

1.1.2.4 Types of Neural Networks

Previous sections present important Neural Network concepts that are important to understand the rest of this work. Until now, we mainly focused on feed-forward fully-connected Neural Networks, also called Multi-Layer Perceptrons. MLPs are a kind of NN used to deal with tabular data in a standard feed-forward manner, in this section, we present the other mainly used network architectures in the literature.

Convolutional Neural Network. Convolutional Neural Networks (CNNs) are a kind of feed-forward Neural Network, inspired by the way animal visual cortex works, that is specialized to be applied on image data. CNNs have been first introduced in (LeCun et al., 1989) and applied on handwritten digits recognition. Since then, CNNs have been widely researched and used on many different tasks (Li et al., 2022), such as image classification, object detection, image segmentation, etc.

Main components of a CNN are the convolution operation, which replaces fully-connected neurons presented earlier in MLPs, padding and pooling. The convolution operation is the one in charge of extracting relevant features from the data, it is composed of a set of convolution kernels (or filters), a dot product is applied between the input and each kernel, the produced output is called a feature map. Kernels can be seen as windows that are slid on the input from left to right and top to bottom until the whole input has been processed, at each iteration an output is obtained by performing an element-wise product between the covered part of the input and the kernel. Convolution kernels are trainable spatial filters, those are the parameters which are learned during training. During the training phase, each kernel learns to extract one specific pattern from its input, lower layers in the model are responsible for the extraction of low-level features (edges, angles, etc.) where higher layers extract high-level features (abstract and complex concepts). Important information might lie on the edges of an image, in such case applying a convolution kernel might miss this information as it cannot be centered on an edge to compute a convolution in this place. A common approach to solve this issue is to use padding, to add pixels out of the edges of the image and allow the kernel to overlap on the edge. It is also common practice

to use a pooling operation to dimensionally reduce the feature map obtained after convolution, down-sampling the image size while preserving local image correlations. The most usual kind of pooling is max-pooling, where the input is divided in squares of a given size and only the maximum value in each square is preserved. Another common kind of pooling is average-pooling, the process is the same except that the output value of each square is the average value of all pixels inside. Figure 1.10 shows a visual representation of the succession of a padding, a convolution and a max-pooling operation.

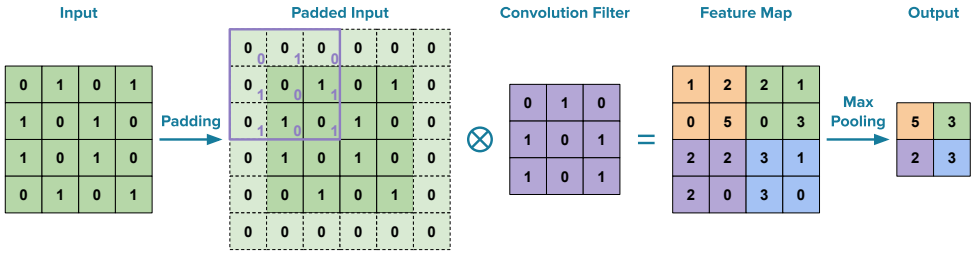


Figure 1.10: Visual representation of the succession of a padding, a convolution (noted ⊗) and a max-pooling operation.

One of the main advantages of CNNs are their capacity to model local correlations, such as neighboring pixels, which is primordial in understanding images. Since convolution kernels share the same parameters over the whole image they rely on less parameters than fully-connected NNs, making them usable on much larger images than their counterpart. Some variations of CNNs are used in other contexts than images, especially to handle time-series. Indeed, it is possible to apply convolutions on signal for example, where local neighboring values in time are relevant to be taken under account.

Auto-Encoder. AEs are often considered to be unsupervised models, indeed, they are trained to reconstruct their own input (Li et al., 2023). But the way of training an AE is as a supervised regression task, where the output target is identical to the input. The goal of an AE is to learn a dimensionally reduced latent representation of its input, from which it can be reconstructed as accurately as possible. It is composed of two parts, the Encoder part is a feed-forward NN that encodes the input into a reduced meaningful representation, the Decoder is a feed-forward NN trained to reconstruct the input from the latent representation produced by the Encoder. Those two parts are trained simultaneously by minimizing the same regression loss. As the training task to solve is a regression, the objective function of a standard Auto-Encoder is defined as:

$$\mathcal{J}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}_{MSE}(D_{\theta_D}(E_{\theta_E}(x_i)), x_i) \tag{1.15}$$

Where parameters to optimize are the set of Encoder and Decoder parameters, $\theta = \{\theta_E, \theta_D\}$. The training process remains exactly the same as with a standard feed-forward model, such as described earlier.

AEs are used for various applications in the literature (Bank et al., 2021). A common use of an AE is as a feature extractor, where the model is trained to reconstruct its input, and then,

the Encoder part is used as a feature extractor on which a classification model is plugged. The classification model is trained to classify labeled data, based on the latent representation of the Encoder which has been previously learned on both labeled and unlabeled data. This is especially useful in cases where a large part of the dataset is unlabeled, which would have usually been useless to train a classifier. In such case, the learned representation of the Encoder facilitates the classification task of the classifier. An obvious use of AEs is for dimensionality reduction, as they naturally produce a reduced representation from input data. In practice, many applications of AEs use this type of models to learn to produce an output that is different from their input. In computer vision, AEs are commonly used to learn to improve image quality, with tasks such as deblurring, upscaling, etc. In such case, the training process remains identical, with the exception that the target is not set to the input of the model, but to an improved version of the input depending on the task to learn.

One of the most famous and commonly used AE architecture is the U-Net, proposed in (Ronneberger et al., 2015a). This convolutional model has been originally applied on a biomedical image segmentation task, the model is trained to construct the segmentation map associated with the given input. The architecture is symmetrical and shaped as a "U", with the Encoder part scaling down the information down to a bottleneck layer, from which the Decoder constructs the output while dimensionnaly scaling up the information. They introduce the important concept of skip-connections in AEs. In a symmetrical AE, a skip-connection is a direct link from an intermediary layer of the Encoder to the corresponding layer of the Decoder. The advantage of skip-connections in this architecture is that the output is not solely produced from the reduced latent representation, which can be too limited. Figure 1.11 shows the architecture of a U-Net.

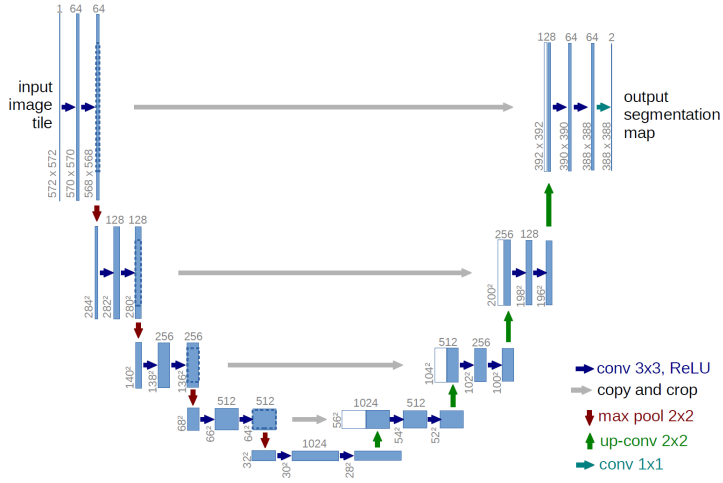


Figure 1.11: Architecture of a U-Net, from (Ronneberger et al., 2015a). The architecture is symmetrical, information is scaled down to the bottleneck layer and up to the output. Skip-connection appear in gray, they lead to less information loss and allow for a more accurate output.

In this thesis, AEs are employed to propose a new Neural Network based mixed-type tabular data imputation method.

Adversarial Networks. Generative Adversarial Networks (GANs) are a kind of generative DL model proposed in (Goodfellow et al., 2014) that rely on the simultaneous training of two models, a Generator G and a Discriminator D. During training, the Generator NN is trained to generate outputs that fool the Discriminator in failing to discriminate them. The Generator input is a random noise vector, noted z, following the chosen prior distribution, usually a normal distribution. The Discriminator is trained to discriminate between data generated by the Generator and real data from the dataset. Both models are trained as a minimax game, where G must maximize the loss of D, and D must minimize its own loss.

The objective function of the Discriminator is formalized as:

$$\mathcal{J}(\theta_D) = -\frac{1}{n}\sum_{i=1}^{n} \mathcal{L}_{BCE}(D_{\theta_D}(G_{\theta_G}(z_i)), D_{\theta_D}(x_i)) \tag{1.16}$$

Where θ_D are the parameters of the Discriminator, which are optimized by minimizing $\mathcal{J}(\theta_D)$. Note the minus sign in front of the objective function, the effect is that it inverses the gradients, applying gradient descent will then lead to ascending the gradients, maximizing the capacity of the discriminator to differentiate generated data from real data.

The objective function of the Generator is formalized as:

$$\mathcal{J}(\theta_G) = \frac{1}{n}\sum_{i=1}^{n} \log(1 - D_{\theta_D}(G_{\theta_G}(z_i)) \tag{1.17}$$

Where θ_G are the parameters of the Generator, which are optimized by minimizing $\mathcal{J}(\theta_G)$. The Generator objective function corresponds to the second part of a Binary Cross-Entropy loss. Minimizing this loss means that the Discriminator wrongfully assumes that generated data is real data.

Both objective functions are minimized in parallel using gradient descent, leading to each model pushing the limits of the other model in getting better at its own task. GANs have been successfully applied in many image related tasks, as they excel at learning to generate images which are almost indistinguishable from real images (Wang et al., 2022).

GANs are not the only possible application of adversarial learning, where two or more models are trained in a competition against each other. Research in the field of Domain Adaptation have led to the introduction of a gradient reversal layer in (Ganin and Lempitsky, 2016). The gradient reversal layer allows to train two adversarial models by minimizing a unique loss, only the gradients of the adversary model are inverted during gradient computation. Such process will be described more precisely in section 1.4.2.2, and used in the following work to solve our Domain Adaptation problem.

Neural Networks for Natural Language Processing. Some types of NNs are more useful for a specific field, it is the case of the previously presented CNNs which excel when applied to images. In the field of Natural Language Processing (NLP), the most common and best performing NNs ones are Recurrent Neural Networks (RNNs), and those last years, Transformers.

The RNN architecture is cyclic, allowing the NN to handle time-series data, by updating its current state based on past and current input data (Yu et al., 2019). For example, when a sentence is given as an input to an RNN, each word of the sentence is fed to the model in the sequential order of the sentence. An RNN model takes two inputs and produces two outputs, its outputs are the output that answers the learned task and a hidden state, its inputs are the hidden state from the previous iteration and the current word to process in the sentence. A common issue with the standard RNN model is information loss between the start of the time-series and the end. Indeed, all information from the start of the time-series to the current point is contained in the previous hidden state, as this hidden representation is limited in size, old information is lost while processing the data. This problem has been identified a long time ago, (Hochreiter and Schmidhuber, 1997) proposed the Long Short-Term Memory (LSTM) model to solve it. Their contribution was to add a system of long-term memory to the RNN in addition to the native short-term memory of such model. More recently, (Bahdanau et al., 2014) introduced the concept of an attention mechanism in a Recurrent Neural Network. This attention mechanism allows the model to pay attention to words in the input sentence that are relevant to the current word, no matter their position in the input series, which drastically improves translation results. Even more recently, (Vaswani et al., 2017) generalized the attention mechanism of (Bahdanau et al., 2014) and proposed Transformers. The architecture of a Transformer is based on the combination and succession of attention mechanisms. It is specialized in NLP, but dispenses with recurrence entirely, replacing recurrent layers with the proposed multi-headed self-attention. Transformers have now become the best performing and most commonly used and researched models in NLP, achieving state-of-the-art performance in most applications (Lin et al., 2022).

Neural Networks presented in this paragraph are not used in the rest of this manuscript, but NLP is a very active and innovative research field, that it is primordial for a DL researcher to explore to stay up to date.

1.2 Missing Values Imputation

Earliest imputation approaches have been developed in the early 1970s, initially based on statistics, more and more Machine Learning based methods have been proposed since the early 1990s. Even more recently, many Deep Learning methods have been introduced to impute missing values as accurately as possible. Imputing missing values is an eternal problem in data science, as it is usually not possible to avoid missing values in real-life, finding the optimal way to deal with them is an unsolved problem. In this section, we describe the missing values problem, and common ways of dealing with missing data in the literature.

In this section, we first set the context of missing values, and present the mechanisms and patterns of missing data. We then review the literature of imputation approaches globally,

and propose a classification of imputation approaches based on their main associated research field. We finish with a presentation and discussion of the commonly used and most pertinent ways of evaluation the quality of imputation results.

1.2.1 Why are Missing Values an Issue?

Missing values in tabular data are an issue when it comes to data analysis, and thus, when applying ML algorithms. From a practical point of view, missing values are a nuisance as most ML algorithms require a complete dataset to both learn and infer. Figure 1.12 shows an example dataset, with fake health data, that we use in the following to illustrate concepts and issues about missing values.

Age	Sex	Weight	Glucose	Insulin	Pressure	Diabetes	Survival
71	Male	86	187	304	68	True	**False**
58	Female	54	91	86	72	False	**True**
26	Male	72	89	37	76	False	**True**
87	Female	65	83	71	78	False	**False**
46	Female	92	171	240	110	True	**False**

Figure 1.12: Complete toy dataset (fake health data) used to illustrate concepts and issues about missing values. The target output is the survival column, remaining features are the patients' health data.

We can compute simple descriptive statistics about the toy data in figure 1.12. For example, we can compute the expectation (mean) of the insulin feature on patients that did not survive, we get an expectation of $\frac{304+71+240}{3} = 205$. As no values are missing in this example, computed statistics are as representative of the real data distribution as possible, given the amount of available data. Now, let's imagine that the insulin value of the patient on first line is missing, a simple and naive way of dealing with such a situation would be to compute the statistics while simply omitting the missing value. We reiterate the same calculation as before in this context and obtain an expectation of $\frac{71+240}{2} = 155.5$. As we can see, the result is drastically different from the one obtained when no value is missing, 155.5 is much lower than 205 and conclusions drawn from descriptive statistics computed on data with missing values might not be pertinent and representative of the real data distribution. ML models learn by extracting statistical patterns and correlations within the data, thus, they are sensible to such an issue. With this simple example we can understand that a model learning on incomplete data might be biased by missing values and draw incorrect assumptions and correlations, leading to a trained model that performs poorly for its application on test data.

In this example, we introduced the simplest and most naive way to deal with missing values, that is, list-wise deletion, also called complete-case analysis (Buuren, 2021). With this approach, we simply delete all data instances that contain one or more missing values. An obvious disadvantage of such an approach is that it deletes non-missing data by removing all incomplete instances, which is wasteful. In medical datasets, as in many other fields, it is not

uncommon for some features to be measured very infrequently, leading to a very high percentage of incomplete instances. In such a context, deleting all incomplete instances leads to the loss of a large part of non-missing values, which strongly limits the potential for data analysis. Another disadvantage is that list-wise deletion is simply unusable in contexts in which test data are also incomplete, which is extremely common in real-world scenarios. In such a context, we want to be able to draw inference for complete and incomplete instances alike.

Instead of deleting incomplete instances, we might be able to guess plausible values that could go in place of missing ones, based on correlations and assumptions drawn from non-missing values in the data. In our example, we can observe a correlation between the insulin feature and the glucose feature, a high glucose value seems to be correlated with a high insulin value, inversely, a low glucose value seems to be correlated with a low insulin value. Naturally, in the case of the insulin value of the first patient being missing, since we know that the glucose is high it would be most plausible for the insulin value to also be high. Intuitively, we see that detecting such correlations in the data can help making educated guesses to replace missing values with plausible ones. This is called missing values imputation.

Such linear correlations can be computed mathematically, for example, we illustrate pairwise linear correlations in the example dataset (figure 1.12) using Pearson's Correlation Coefficient (Rodgers and Nicewander, 1988). We note ρ the Pearson Correlation Coefficient, which can be used to measure the strength and direction of linear relationships between pairs of variables. It is common practice to measure correlations between each pair of features in a dataset and display the obtained results in a correlation matrix to find insights about their relationships. Computed values range from -1 to $+1$, with 0 meaning no correlation and $1/-1$ meaning perfect positive/negative correlation between the two features. For example, let's compute the Pearson's correlation coefficient between features "weight" and "glucose" in our toy example, we note the weight feature X_1 and the glucose feature X_2. The Pearson Correlation Coefficient is computed as:

$$\rho_{X_1,X_2} = \frac{cov(X_1,X_2)}{\sigma_{X_1}\sigma_{X_2}} = \frac{\mathbb{E}[(X_1-\mu_{X_1})(X_2-\mu_{X_2})]}{\sigma_{X_1}\sigma_{X_2}} \tag{1.18}$$

With $cov(X_1,X_2)$ the covariance between the pair of features, μ_X and σ_X the mean and standard deviation of the corresponding feature respectively, and $\mathbb{E}$ the expectation. We recall that the mean is computed as $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$ and standard deviation as $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2}$, with n the number of values in X. We compute $\mu_{X_1} = 73.8$, $\mu_{X_2} = 124.2$, $\sigma_{X_1} \approx 13.8$ and $\sigma_{X_2} \approx 45.11$, which gives:

$$\rho_{X_1,X_2} = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_{1,i}-73.8)(x_{2,i}-124.2)}{13.8\times 45.11} \approx 0.8678 \tag{1.19}$$

We obtain a positive correlation coefficient of about 0.87, which illustrates a high positive correlation between weight and glucose features. We compute the correlation coefficients between each pair of features in the dataset and display the results as a correlation matrix for visualization in figure 1.13.

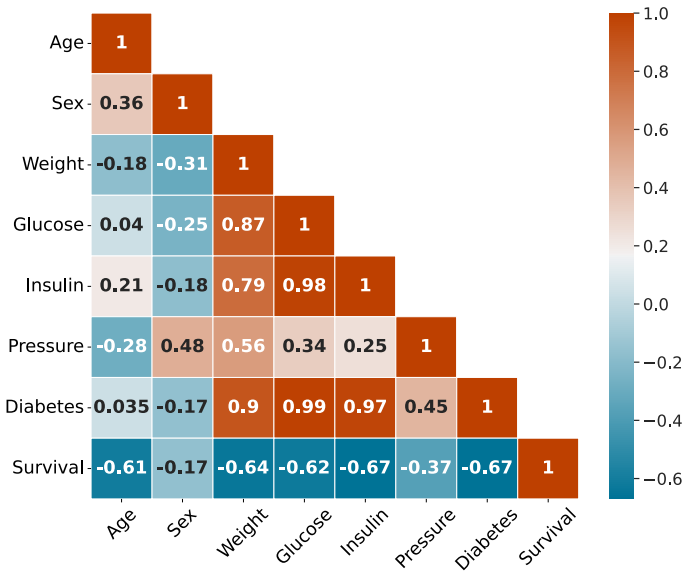


Figure 1.13: Correlation matrix computed from the toy dataset in figure 1.12.

As can be seen from this matrix, diabetes is highly positively correlated with weight, glucose, and insulin values. Which indicates that high values in those features are correlated with a positive diabetes value, or inversely. We see that the survival outcome is very negatively correlated with age, weight, glucose, insulin and diabetes on this toy dataset, which means that high values in those features are correlated with a negative survival outcome (0). Note that those coefficients only measure correlations, which do not imply causation, it is not possible to draw conclusions about the causal link between variables from this correlation matrix only. Insights can be drawn from such a correlation matrix, for example, if the insulin value of a patient is missing, and we know that this patient suffers from diabetes, we can guess with a high confidence that the missing insulin value must be high. Using other correlated features in the dataset with non-missing values for this patient, we can sharpen the value to impute to be as plausible as possible.

Advanced ways of dealing with missing values in the ML literature rely mainly on imputing missing values with plausible ones, using the learning power of ML algorithms. While the correlation matrix in figure 1.13 only represents linear correlations between each pair of features in the dataset, advanced ML algorithms are able to detect and extract non-linear correlations between features. Non-linear correlations between multiples variables in the dataset that humans cannot even apprehend can be detected by ML algorithms. Most modern missing values imputation approaches in the ML literature are based on learning such complex non-linear correlations in the data, to impute missing values as accurately as possible. In this missing values imputation background section we will present, describe, and discuss, the various categories of imputation approaches in the ML literature.

1.2.2 Missing Data Mechanisms and Patterns

Missingness mechanisms and missingness patterns are distinguished in the missing data literature. Missingness mechanisms describe the relationship between missingness and the variables within the data. While missingness patterns describe which values are missing or observed in the data. In this section, we present and discuss those mechanisms and patterns.

1.2.2.1 Missingness Mechanisms

Missing data are caused by various causes, (Rubin, 1976) classified mechanisms that cause missing data in three categories: Missing Completely At Random (MCAR), Missing At Random (MAR), Missing Not At Random (MNAR). Some authors, such as (Seaman et al., 2013), find Rubin's definitions to be not clear enough, but the vast majority of practitioners in the missing values imputation field still use the missing data mechanisms such as defined by Rubin. In Rubin's theory, each value in a dataset has a probability of being missing, we call "missing data mechanism" the process that dictates those probabilities. In this section, we are inspired by (Buuren, 2021) to define those three missingness mechanisms using our notations, we keep the toy dataset from figure 1.12 to illustrate our points and discuss each mechanism.

In the following, we note $X \in \mathbb{R}^{n \times d}$ the dataset composed of n instances and d features. We note X_{obs} the non-missing values in X and X_{mis} the missing values. We note $M \in \mathbb{R}^{n \times d}$ the binary missingness indicator such as $M_{i,j} = 0$ if $X_{i,j}$ is observed and $M_{i,j} = 1$ if $X_{i,j}$ is missing. The distribution of M is fully dependent of the missing data mechanism that caused missing values. We define as $P(M|X_{obs}, X_{mis}, \psi)$, the distribution of M given known and missing values in X, with ψ the parameters of the missing data model applying the missing data mechanism.

Definition 1 (MCAR: Missing Completely At Random)

The data are said to be MCAR if the probability for a value to be missing depends only on some parameters ψ. $P(M|X_{obs}, X_{mis}, \psi) = P(M|\psi)$

We define MCAR using our notations in definition 1. In this case, the missingness probability is the same for all values in the dataset, which implies that causes of missing values are not related to the data. An example of the MCAR mechanism in a real-life scenario would be a machine that randomly does not record some values. Such a missing data mechanism does not often appear in real-world data, as the causes of missing values are rarely unrelated to the dataset itself and truly random. Figure 1.14 illustrates the MCAR mechanism on our example dataset. As can be seen in this figure, missing values in measured health features seem to be distributed uniformly randomly, in this context, no known nor unknown information could be useful to explain missing values.

Definition 2 (MAR: Missing At Random)

The data are said to be MAR if the probability for a value to be missing depends on observed values. $P(M|X_{obs}, X_{mis}, \psi) = P(M|X_{obs}, \psi)$

Age	Sex	Weight	Glucose	Insulin	Pressure	Diabetes	Survival
71	Male	86	NA	304	68	NA	False
58	Female	54	91	NA	72	False	True
26	Male	72	89	37	NA	False	True
87	Female	65	NA	71	NA	False	False
46	Female	92	171	NA	110	NA	False

Figure 1.14: Toy dataset from figure 1.12 with missing values governed by a Missing Completely At Random (MCAR) mechanism.

We define MAR using our notations in definition 2. In this case, the missingness probability is dependent on known values in the data. Such a missingness mechanism is very common in real-life situations. Figure 1.15 illustrates the MAR mechanism on our example dataset. In this example, missing values in measured health features seem to appear more often for older patients than for younger ones. In this context, known information within the data can be useful to explain missing values.

Age	Sex	Weight	Glucose	Insulin	Pressure	Diabetes	Survival
71	Male	86	NA	NA	NA	NA	False
58	Female	54	NA	86	72	False	True
26	Male	72	89	37	76	False	True
87	Female	65	NA	71	NA	NA	False
46	Female	92	171	NA	110	True	False

Figure 1.15: Toy dataset from figure 1.12 with missing values governed by a Missing At Random (MAR) mechanism.

Definition 3 (MNAR: Missing Not At Random)

The data are said to be MNAR if the probability for a value to be missing depends on observed but also on unknown values, $P(M|X_{obs}, X_{mis}, \psi)$ cannot be simplified.

We define MNAR using our notations in definition 3. MNAR is used to refer to all cases not covered by MCAR or MAR. In this case, the missingness probability can be dependent on both known and unknown values in the data, missing values can also be caused by external factors that cannot be present in the data. This mechanism is the most commonly occurring missingness mechanism in real-life data, and is known as the most complex case of missingness (Baraldi and Enders, 2010; Buuren, 2021). Figure 1.16 illustrates the MNAR mechanism on our example dataset. In this example, missing values seem to occur more on instances that have a negative survival outcome. This could be explained by the fact that health data are less recorded on people that are too sick. In this case, missingness probability is dependent on the survival outcome, which is not known at the moment health data are last collected. In the previous example, we show an example of a missingness probability

dependent of missing values, but as stated above, in the case of MNAR, missing values can also be caused by external factors. A real-life example of such an occurrence would be when a machine generates more missing values as it gets older. Such missing values are dependent on an external factor, which is the age of the machine, but this information is not present in the data.

Age	Sex	Weight	Glucose	Insulin	Pressure	Diabetes	Survival
71	Male	86	NA	304	NA	NA	False
58	Female	54	91	86	72	False	True
26	Male	72	89	37	NA	False	True
87	Female	65	NA	NA	NA	NA	False
46	Female	92	NA	NA	110	NA	False

Figure 1.16: Toy dataset from figure 1.12 with missing values governed by a Missing Not At Random (MNAR) mechanism.

In the missing values imputation literature missing data mechanisms are often simulated on originally complete datasets to perform comparative studies of imputation approaches in each possible missingness scenario. In almost any real-world situation MCAR can be empirically rejected in favor of MAR or MNAR, as in most cases, missing values are caused for a reason (Collins et al., 2001; Rubin, 1996; Schafer, 1997). Whether this cause is explainable or not given the observed data determines if the mechanism is MAR or MNAR. Most real-world scenarios are a combination of both MAR and MNAR mechanisms. It is often not possible to determine a precise mechanism behind real-world missing values, since the distinction between MCAR and MNAR depends on unknown information.

1.2.2.2 Missingness Patterns

We have seen the three possible mechanisms that govern missing data in previous sections. Given the missingness mechanism, missing values can appear in various patterns in the dataset. The literature is not unified on this point and many authors use their own definitions, in the following we present and discuss the main recurrent patterns in the literature.

- **Univariate/Multivariate patterns:** Most authors make distinction between univariate and multivariate missingness patterns, such as (Lacerda et al., 2007), (Little and Rubin, 2019) or (Buuren, 2021). The univariate missing data pattern is a case where only one feature in the data contains missing values, whereas we talk of a multivariate pattern when more than one feature contains missing values.

- **Monotone/Non-monotone patterns:** Distinction between monotone and non-monotone patterns is also extremely common in missing data literature. As described by (Horton and Kleinman, 2007), we talk about a monotone pattern when the dataset can be rearranged in such a way that missing values are organized as a hierarchy. Such as if a value in a previous feature of an instance is missing, then all following feature

values are also missing. Inversely, the non-monotonous pattern is the case where the dataset cannot be rearranged in such a hierarchical manner.

- **Missing by design pattern:** Some authors describe the missing by design pattern, some also call it missing by logic. This pattern is described by (de Leeuw et al., 2003) as the case where some values are voluntarily not collected for some instances. In medical studies, it is common to encounter variables that are only observed in a specific category of patients and not others. For example, data related to pregnancies are usually not collected on male patients as it would be a waste of resources.

- **General or Arbitrary pattern:** The general missingness pattern, sometimes called the arbitrary missingness pattern appears frequently in the literature. It is described by (Berglund and Heeringa, 2014) as the case in which there is no particular missingness pattern that can be detected in the data. Missingness seems arbitrary, it is naturally a multivariate, non-monotone missing data pattern.

- **Other missingness patterns:** Other less common missingness patterns are also described in the literature. In questionnaires it is common to encounter what (de Leeuw et al., 2003) called partial non-response, that is, after some point in the questionnaire all following data is missing, which is explained by the fact that the person stopped answering questions. They also describe item non-response, in cases where a respondent does not answer a specific question, which can be considered a form of general missingness pattern in standard tabular data. (Buuren, 2021) defines connected and unconnected patterns, where a missingness pattern is said to be connected if any non-missing value can be reached from any other observed value given a sequence of horizontal or vertical moves.

In this work we do not aim to redefine missingness patterns in our own way, nor to try and unify all proposed terminologies. Indeed, missingness patterns are mostly relevant when developing statistical imputation approaches, but advanced Machine Learning imputation approaches are not sensible to such patterns. As stated in (Horton and Kleinman, 2007), simpler methods can be used if the missingness pattern is monotonous which could motivate the effort to identify such a pattern in used datasets, but a monotonous pattern is highly unrealistic in real data. Therefore, in the rest of this work we assume that missing values in our experiments are always organized following what is usually called a general or arbitrary missingness pattern as it is the most often occurring and important one in real-world data.

1.2.3 Imputation Frameworks and Methods

Usually in the imputation literature no distinction is made between imputation frameworks and imputation methods. We argue that this makes no sense as any imputation method can be used based on any imputation framework, and comparing two imputation methods that are not applied following the same framework is an unfair and non rigorous comparison. In the following work, we will precisely distinguish between such frameworks and methods, and will refer to both indiscriminately as approaches.

In the remaining of this section and manuscript, we note $X \in \mathbb{R}^{n \times d}$ the clean version of the dataset, that is, the version without any missing values (which might not exist in a real-world scenario). We note $\tilde{X}$ the corrupted version of X, that is, the version with missing values (which might be the only available one in a real-world scenario). Finally, we note $\mathring{X}$ the corrected version of $\tilde{X}$, that is, the version with imputations in place of missing values.

Definition 4 (Missing Values Imputation Framework)

An imputation framework is a defined procedure that makes use of an imputation method to impute missing values and exploit imputations in a formalized way.

We propose a definition of an imputation framework in definition 4. Examples of an imputation framework are Single-Imputation (SI) and Multiple-Imputation (MI), that can be applied using any imputation method to train any kind of model. We review imputation frameworks in the following section.

Definition 5 (Missing Values Imputation Method)

An imputation method is a model f_θ, that is applied to the incomplete dataset noted $\tilde{X} \in \mathbb{R}^{n \times d}$ and outputs a corresponding complete dataset $\mathring{X} = f_\theta(\tilde{X})$, where missing values in $\tilde{X}$ are imputed with plausible values, leading to $\mathring{X}$.

Definition 5 gives a formal definition of an imputation method, we review imputation methods in the following section.

In this section, we are interested in all tabular data imputation approaches, we will briefly address the topics of imputation in time-series and images at the end of the section. Figure 1.17 illustrates the classification we defined to organize the following reviewing work, we created our classification inspired by the one proposed in (Lin and Tsai, 2020).

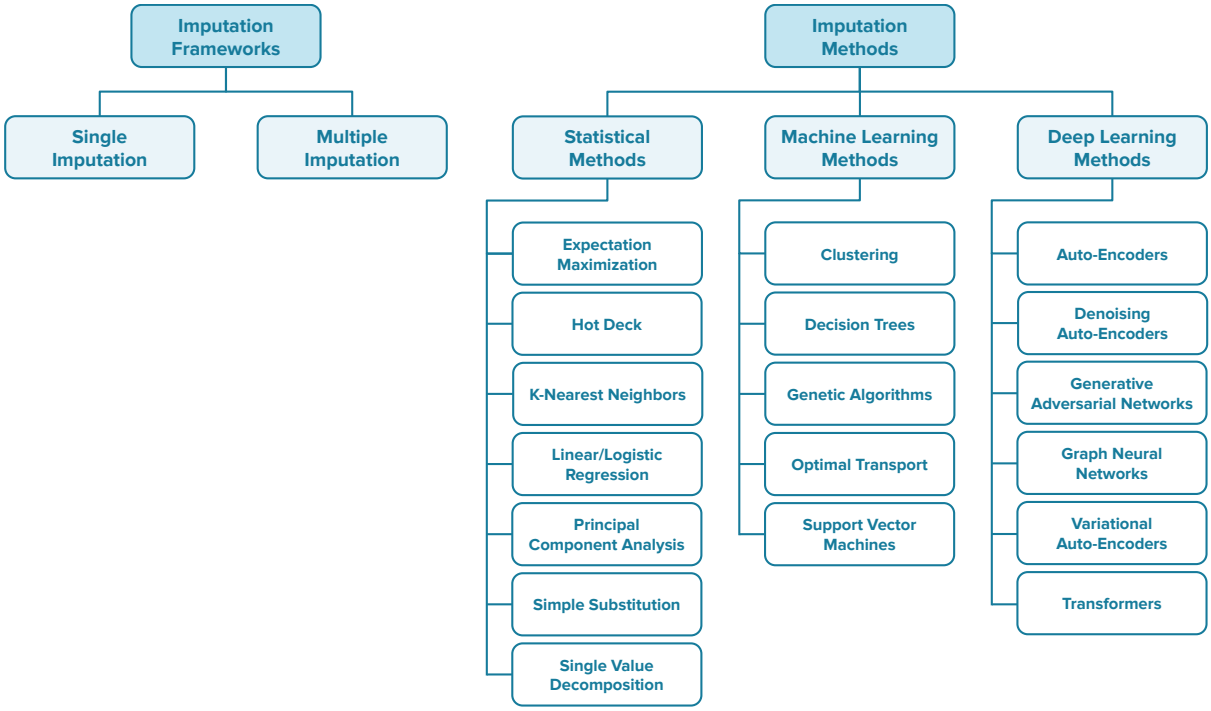


Figure 1.17: Statistical and Machine/Deep Learning Imputation Approaches Classification

1.2.3.1 Imputation Frameworks

There are very few tabular data imputation frameworks available and used in the literature, in fact, to the best of our knowledge, there are currently only two: Single-Imputation and Multiple-Imputation.

Single-Imputation. SI is the default imputation paradigm that anyone uses, it has the advantage of being extremely simple and straightforward. It simply consists in choosing an imputation method, applying it to the incomplete dataset $\tilde{X}$, which yields a completed dataset $\mathring{X}$ where missing values have been assigned with new plausible values, and using $\mathring{X}$ as one would with any complete dataset (Buuren, 2021; Rubin, 1987). Obtaining a single completed dataset is a huge advantage, as it is then possible to integrate SI into any existing pipeline or system, making them usable with missing values without any needed modification (Josse et al., 2019). However, SI presents multiple important drawbacks. Once the incomplete dataset has been imputed, the imputed dataset $\mathring{X}$ is treated as the new complete dataset, this is a problem as all methods that exploit the completed dataset will treat missing values as if they were known (Buuren, 2021; Rubin, 1987). In such case, the extra variability due to the unknown missing values cannot be taken into account by inference methods applied on $\mathring{X}$, thus, their inferences will be too sharp and overstate precision. As SI does not account for the uncertainty about obtained imputations, it results in biased inference models which lack generalization capacity. But since it outputs a unique imputed dataset that can be exploited as any complete dataset, it is extremely convenient and easy to use in any scenario. This simplicity to use and convenience explains why such framework is so widely used in practice, despite the fact that it is known not to be the best solution. Figure 1.18 illustrates

this simple framework.

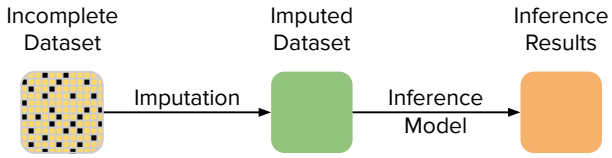


Figure 1.18: Simple illustration of the Single-Imputation framework.

Multiple-Imputation. In (Rubin, 1987), Rubin proposes Multiple-Imputation, where each missing values in $\tilde{X}$ is replaced by a set of plausible values representing a distribution of possibilities, which represents the uncertainty about the right value to impute. This procedure results in m imputed datasets, with $\mathring{X}^i$ the i-th imputed dataset. In a Machine Learning context, one learner is trained on each imputed dataset and the results from all the learners are then pooled in an ensemble manner. An advantage of MI over SI is that each missing value is represented by a sample of possible imputation values, which results in inferences that reflect the uncertainty level associated with each missing value (Buuren, 2021). Therefore, the results pooled from the ensemble of learners will be less biased compared to those of each learner taken independently. Applying the MI framework results in inferences that properly reflects the uncertainty of imputations, leading to less biased inference results. The use of the powerful ensemble paradigm helps to leads to significantly better results when using the MI framework compared to SI. An obvious disadvantage of MI compared to SI is the computational cost of such a framework, the ensemble training of the learners multiplies the required amount of calculation time. While being quite an old and well-known framework, MI is still not used in most cases, scientists and imputation methods users usually still rely on SI. This can be partially explained by the computational cost of MI, which is high since it requires computing m imputations of the same dataset and then training m inference models, which can be quite tedious and often inconceivable in practice. Figure 1.19 illustrates the MI framework.

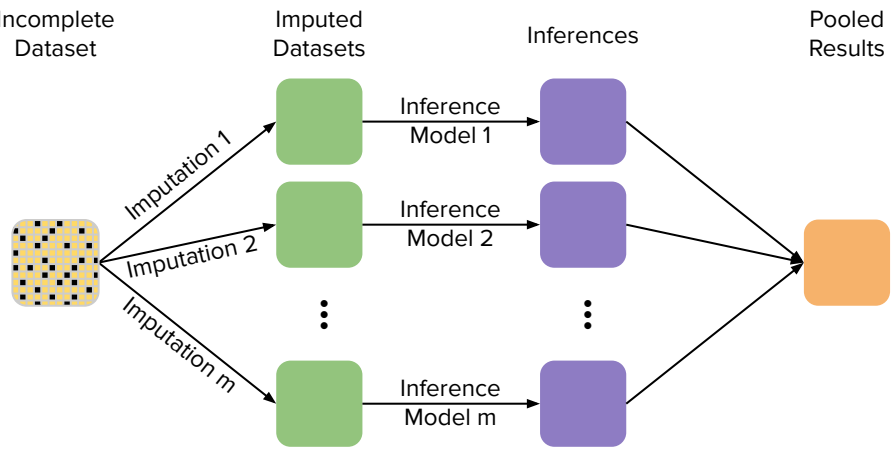


Figure 1.19: Illustration of the Multiple-Imputation framework.

1.2.3.2 Statistical Imputation Methods

In this section, we present the most popular categories of statistical imputation methods, roughly ordered by popularity and quality of results. As this thesis focuses on ML and DL approaches, we review very briefly each statistical imputation category and describe a bit more precisely ML based methods.

Simple Substitution. Mean substitution consists in replacing missing values with the mean value of the corresponding feature. Respectively median substitution is the same using the median value, and mode substitution for categorical features consists in replacing missing values with the most common value in the corresponding feature. Mean substitution is still a widely used method in practice, especially in the medical field (Armitage et al., 2015; de Souto et al., 2015). Median substitution is also sometimes used as outliers might drastically impact imputations when imputing with mean substitution. When imputing categorical features in this context it is common practice to impute missing values with the mode of the feature (Buuren, 2021; Lin and Tsai, 2020).

It has been shown that mean imputation distorts the data distribution in several ways. Mean substitution is known to produce unrealistic imputations by underestimating the variance, modifying feature correlations and biasing all other estimates than mean in an MCAR setting and all estimates in the data in MAR and MNAR settings (Buuren, 2021; Emmanuel et al., 2021). Median substitution might lead to better results than mean substitution when original data distribution is skewed, but similar drawbacks are to be reported. Despite the extreme simplicity to both implement and use such substitution methods, general consensus in the missing values field has for a long time been to avoid using them in all circumstances (Troyanskaya et al., 2001).

Hot Deck. The hot Deck imputation method is among the simplest imputation methods. It consists in replacing missing values of an instance by observed values from a random or a similar instance in the dataset with respect to commonly observed features in both instances. Several versions of this algorithm exist (Andridge and Little, 2010). The simplest form is to simply impute missing values using observed values from random instances in which they are observed. Obviously this method leads to highly unrealistic and biased imputations and should never be used. More advanced methods aim to order the dataset in regard to selected important features in the dataset, then we iterate though the dataset and missing values from each instance are imputed with observed values from the closest instance in which they are observed. This method is also used in longitudinal data, in which it makes more sense, as we can simply impute missing values in an instance using the last observed value for this same instance previously in time.

This method is known to impute highly biased values and leading to potentially false conclusions (Molnar et al., 2008). Similarly to simple substitution methods presented above, Hot Deck was widely used in practice, but it is now of general consensus that this method should not be used anymore, other more advanced imputation methods have been developed and can be used to obtain far better results.

Expectation Maximization. The Expectation Maximization algorithm to impute missing values has been proposed in (Dempster et al., 1977). The algorithm iteratively computes the maximum-likelihood estimates in place of missing values given all observed ones (the expectation step), and fits model parameters to maximize the expected likelihood of the estimates obtained from the previous step (the maximization step). This process is repeated until convergence is achieved. Hidden Markov Models training is solved using the Baum–Welch algorithm. This algorithm is generally considered superior to the naive list-wise deletion and simple substitution methods under the MCAR and MAR assumptions (Musil et al., 2002). This method is known to produce biased standard errors (Musil et al., 2002).

Many papers have used and proposed improved versions of Expectation Maximization for missing values imputation (Emmanuel et al., 2021; Lin and Tsai, 2020). The recent paper (Delalleau et al., 2018) proposes an improved version of the Expectation Maximization algorithm to largely speed-up the algorithm training time on large dimensional tabular data. They replace the originally used Hidden Markov Models by Gaussian Mixture Models and develop a more efficient algorithm. They experimentally showed that imputed data using their method lead to a large classification improvement over simple substitution methods.

This method is known to do well under MCAR and MAR settings, but is not efficient when the missing data mechanism is MNAR (D.Allison, 2002), which is an important limitation for real-world data. Overall, the Expectation Maximization algorithm provides better results than simple substition methods, but general consensus seems to indicate that better imputation methods have been proposed that can be used instead.

Single Value Decomposition. Single Value Decomposition can be seen as a method that transforms a dataset with linearly correlated features into a set of uncorrelated features that explain as much variance in the data as possible. We do not describe the mathematical process of SVD as it is not relevant to this work, in-depth information can be found in the following tutorial (Baker, 2005).

Using SVD for imputation has first been proposed in (Hastie et al., 2001). They propose two solutions, the first is to fit a SVD on the complete part of the dataset, assuming that enough instances are fully observed, the second is a more advanced variant of SVD to exploit all available data. It has also been presented and used for comparison in (Troyanskaya et al., 2001). More recently, (Mazumder et al., 2010) proposed Soft-Impute, an advanced SVD based imputation method. They use the nuclear norm as a regularization and propose a highly efficient algorithm that is able to scale to very large amounts of data. They obtain mitigated results in terms of imputation quality but have the advantage of being able to output satisfactory imputations in very reasonable running times for extremely large datasets.

One of the main limitation of such method is that it can only rely on linear correlations between features to compute imputations, which limits the relevance of obtained imputations.

Principal Component Analysis. Principal Component Analysis has been originally developed as an exploratory tool for data analysis (Jolliffe and Cadima, 2016), the process is highly related to Single Value Decomposition. The goal of a PCA is to compute principal components, which can be seen as new features, that account for as much variance as possible in the original data. Those principal components are computed as a linear combination of the data features. These linear combinations are computed in such a way that new principal components are as uncorrelated as possible from all previously computed ones and that each component accounts for as much variance as possible. This leads to a set of components where the first ones explain most of the variance, and so, most of the useful information about the data.

PCA has been used as an imputation method at several occasions. A Bayesian PCA based imputation method has been proposed in (Oba et al., 2003) and applied to gene expression profile imputation. The algorithm consists in the combination of a Bayesian PCA, which is able to automatically determine the best number of principal components to extract, and the Expectation Maximization algorithm to estimate imputations. More recently, (Josse and Husson, 2012) proposed the iterative PCA, which is able to compute principal components from incomplete data, and that imputes missing values in the process. As their algorithm is based on principal components of the data, imputations naturally take account for features linear correlations. They show that their method obtain better results than simple substitution methods.

As with Single Value Decomposition, one of the main limitation is that PCA imputation relies solely on linear correlations between features to compute imputations. Statistical literature on PCA for imputation is very limited, most proposed methods are only compared to simple substitution methods, which seems to indicate that imputation quality is limited.

k-Nearest Neighbors. The k-Nearest Neighbors method is usually considered as a ML approach in the literature. In this work we classified it as a statistical method as we consider that ML methods are based on learning, which is not the case of the KNN method. As many statistical imputation methods, KNN imputation first appeared in the medical field, applied to gene expression profile missing values imputation in (Hastie et al., 2001) and (Troyanskaya et al., 2001). Basically, missing values of an incomplete instance are imputed using the mean value from the k closest complete instances in the data. The similarity distance between two instances is computed between observed values of both instances. Various similarity functions can be used depending on the application context of the algorithm, the most common one being a simple Euclidean distance between features.

In their application to gene expression profile imputation, (Troyanskaya et al., 2001) report that KNN imputation obtains better results than simple substitution methods. They also show that KNN imputation shows less performance deterioration when the missing rate increases when compared to previously presented SVD imputation. Overall, they show that KNN imputation is more robust to the nature of data and less sensitive to parameterization. More recently, (Armitage et al., 2015) also reported better KNN results on medical data when compared to simple substitution methods which supports original claims. Limitations to this method are similar to above mentioned methods in regard to linear correlations, the

KNN algorithm is also known not to scale well as the amount of data increases. Indeed, computation of the distance between each possible pair of instances in the data naturally leads to a quadratic complexity. Despite those drawbacks KNN imputation is very commonly used in practice as it is simple to use, relatively fast, and provides good imputations in most situations.

Linear/Logistic Regression. There are many different implementations of linear or logistic regression in the missing values statistical literature (Lin and Tsai, 2020), most of them have first appeared in the medical field and then been democratized to other scientific fields, is one of the preferred statistical technique for handling missing values (Emmanuel et al., 2021). Usually statistical regression methods are divided in two main phases, first a regression model is trained on all complete instances in the dataset, then the model is applied to estimate missing values. The main limitation of such models is that they can only compute missing values imputation based on a linear combination of observed values. This assumes that missing values must be linearly explainable given non-missing values, which is not always the case.

Many linear regression methods have been proposed in the statistical literature, using different ways of performing regression. A main way of performing linear regression is using the least squares method to produce missing values estimation by minimizing the error between the measured and predicted values in observed values. In 2004, LSimpute have been proposed (Bø et al., 2004), a missing values imputation method based on least squares regression. Least squares aims at finding the best possible linear fit that minimizes the variance: the sum of squares of the errors. Such linear regression model is often noted $y = \alpha + \beta x + \varepsilon$, with y the estimate of the missing value, x the observed features, ε the error term for which the variance is minimized and α and β the model parameters. (Kim et al., 2005) proposed a method called LLSimpute: Local Least Squares imputation, applied to missing values in gene expression data, that extends the previously proposed LSimpute. Instead of performing a global regression based on all observed values as in LSimpute, this method first selects the k closest neighboring genes, either based on the L_2-norm or on Pearson Correlation Coefficients, a local regression model is then fitted on those k selected genes. Many other variants of linear regression using various kinds of least squares methods have been developed throughout the years (Lin and Tsai, 2020).

One of the most widely used and best performing statistical imputation method is MICE: Multivariate Imputation by Chained Equation, which have been proposed in (van Buuren et al., 1999). Ever since, it has been widely used in the medical field and many others (Buuren and Groothuis-Oudshoorn, 2011), it can still be considered as a state-of-the-art imputation method as it reaches excellent and highly competitive results. MICE is an iterative algorithm that works in the following way:

1. Initially, missing values are imputed using a simple substitution method, such as mean substitution.

2. One variable to complete is selected, usually base on the amount of missing values for this feature.

3. Observed values of the selected feature are used to fit a linear (if the feature is continuous) or a logistic (if it is categorical) regression model. Any type of statistical regression can be used in this method.

4. Missing values of the selected feature are replaced with the obtained predictions from the regression model.

5. Steps 2 to 4 are repeated for each feature in the dataset that has missing values. The algorithm iterates through those steps for a set number of cycles, each iteration leads to imputations that reflect more the linear correlations between all features.

Many studies apply MICE to deal with missing values, and many others propose improved versions of MICE. For example, (Khan and Hoque, 2020) proposed SICE, an extension of MICE that is more pertinent to use in presence of a large amounts of data. (Burgette and Reiter, 2010) proposed a MICE variant that uses classification and regression trees instead of a standard linear/logistic regression model. The main advantage of such a method is that it allows MICE to exploit non-linear correlations between features to perform imputation, which automatically leads to superior results. An obvious disadvantage of this extension, and of MICE in general, is that the method needs to fit one regression model per incomplete feature at each iteration, which leads to important performance issues when dealing with a dataset that has many incomplete features.

1.2.3.3 Machine Learning Imputation Methods

The main limitation of linear regression models and other previously presented statistical methods is that they can only compute missing values imputation based on a linear combination of observed values. This assumes that missing values must be linearly explainable given non-missing values, which is not always the case. Advanced Machine Learning methods aim at finding and exploiting non-linear correlations in the data, which automatically leads to improved inference results when compared to linear methods. In this section, we review the most popular imputation methods in the ML literature.

Support Vector Machines. A once popular ML classification and regression method were Support Vector Machines. Standard Support Vector Machines can be used to perform linear classification, the commonly called kernel trick allows SVM to perform non-linear classification and regression, which highly improves potential and performance of this method. Like any supervised ML method, Support Vector Machines rely on learning generalizations from a training dataset to then infer on never seen test data.

In standard linear SVM (Boser et al., 1992) we make the assumption that the data is linearly separable, that is, we can draw a line that separates all instances between two classes, identified by -1 and 1 respectively. We note $X \in \mathbb{R}^{n \times d}$ the sample of training data, where n is the amount of instances in the dataset and d the number of features, and $Y \in \{-1, 1\}^n$ the associated labels. The i-th instance in the dataset is represented as $x_i = (x_{i,1}, \ldots, x_{i,d})$ with its label y_i. In this simple binary classification context it is possible to find two hyper-planes that satisfy: $Wx - b = 1$, where every point on this boundary or on one given side is of class 1,

and $Wx - b = -1$, where every point on this boundary or on the other side is of class -1, with $W \in \mathbb{R}^d$ and $b \in \mathbb{R}$. Figure 1.20 illustrates a hyperplane that maximize margin between two classes on example data. The distance between those two hyperplans is computed as $\frac{2}{||W||}$, therefore, to maximize this margin we can minimize $||W||$ directly. Since we want to prevent any point inside the margin we also need to ensure that:

$$\begin{cases} Wx_i - b \geq 1 & \text{if } y_i = 1 \\ Wx_i - b \leq -1 & \text{if } y_i = -1 \end{cases} \forall i \in (1, \ldots, n) \tag{1.20}$$

Which can be rewritten as $y_i(Wx_i - b) \geq 1 \forall i \in (1, \ldots, n)$. In practice, in order to solve those equations it is common to formalize it as an optimization problem, to find the best possible solution given the data (Bishop, 2006):

$$\min_{W,b} ||W|| + \sum_{i=1}^{n} \max(0, 1 - y_i(Wx_i - b) \tag{1.21}$$

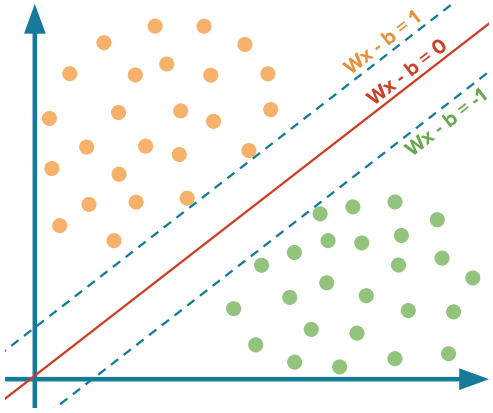


Figure 1.20: Support Vector Machines computed maximum margin hyperplane on example data

In order for Support Vector Machines to be able to handle non-linear classification or regression, (Boser et al., 1992) introduced the kernel trick, which maps the model inputs into a much higher dimensional feature space, which is hoped to make separation linear in that projection space. This trick is what makes regression possible using Support Vector Machines.

Handling missing values is not particularly close to the usual tasks performed by Support Vector Machines, but SVM regression has been successfully used on several occasions to perform missing values imputation. In their paper, (Wang et al., 2006) proposed an imputation method based on SVM regression and compared it to previously presented statistical methods KNN imputation, SVD imputation and Bayesian PCA, applied to gene expression profiles. They experimentally showed that their SVM based imputation method was able to compete and even outperform the above statistical methods in terms of imputation accuracy, they take advantage of the kernel trick to exploit non-linear correlations in the data to outperform linear methods. More recently, (Yang et al., 2012) proposed another SVM based

imputation method. Similarly to (Wang et al., 2006), they rely on SVM regression using fully observed features to impute missing ones. They evaluate the quality of their imputations with a prediction task and note an increase of 1 to 8% of accuracy.

The Support Vector Machine is one of the most basic ML algorithm, fundamental theoretical principles used in this method are quite important as they are used in most other ML methods. SVM based methods are not so popular nowadays because of their strong limitations (Boser et al., 1992), more advanced ML and DL methods are used to overcome those weaknesses. Indeed, SVM are not suitable to be applied to large datasets since the complexity of the algorithm highly depends on data dimensions. They are also known not to perform well on imbalanced datasets, where the computed hyperplane gets highly biased towards the minority class, leading to poor classification or regression performance. The main advantage of SVM over simpler statistical methods is that we can use the kernel trick to exploit non-linear correlations. Choosing the right kernel function to use is key, meaning that data analysis must be performed in order to properly decide which non-linear kernel would be the best in the given situation. Overall, those limitations make this method not obvious and not easy to use for non-expert persons.

Clustering. Clustering methods are unsupervised ML methods that aim at dividing data in separate clusters without relying on known labels. Clustering alone does not seem to be applicable to the imputation of missing values, but Clustering methods have been used several times in combination to other ML methods to perform imputation in the literature.

In their papers, (Zhang et al., 2006) and (Zhang et al., 2008) propose a clustering based imputation method. They divide the dataset complete instances in clusters using the K-Means method and associate each remaining incomplete instance to the cluster most similar to it. They then use a kernel based method to impute missing values based on complete instances in the same cluster. They compare their method with a Decision Tree imputation method and obtain competitive results. More recently, (Rahman and Islam, 2016) proposed FEMI, a fuzzy Expectation Maximization and fuzzy Clustering based missing values imputation framework. It performs imputation in incomplete instances based on instances classified in the same cluster. Clusters are obtained using a fuzzy Clustering method, and imputations are performed using the proposed fuzzy Expectation Maximum algorithm. They compare their methods to statistical imputation methods and one Decision Tree method and note superior imputation quality. As they do not compare their method to modern and advanced imputation methods it is not possible to conclude that FEMI would perform well when compared to state-of-the-art methods.

Clustering application to the imputation field is limited and does not seem to be much researched at the moment. Clustering based imputation approaches do not seem to lead to better results than most currently used statistical, ML and DL imputation methods.

Decision Trees. Decision Trees are simple ML methods that can perform supervised non-linear classification and regression learning (Twala, 2009). A Decision Tree is constructed as a tree-like structure, each node of the tree represents an attribute on which a condition is executed, depending on the condition result, the node splits in two or more branches. The first node of the tree is called the root node, from which the whole tree is constructed. Branches of the tree are composed of decision nodes, each decision node applies a condition. At the extremity of each branch of the tree we find leaf nodes, each leaf node corresponds to a decision outcome. For a given data instance we can follow the conditions through the tree, which ultimately leads to the predicted output. A big advantage of Decision Trees are their easy interpretability, a reasonably dimensioned tree can easily be followed and understood by a human. There are several possible ways to construct a Decision Tree (i.e. train the tree) given training data. Most known tree training algorithms are ID3 (Quinlan, 1986), its successors C4.5 (Salzberg, 1994) and commercially marketed C5, and CART (Breiman, 2017), which is the most known and currently used tree algorithm. Figure 1.21 shows a visual representation of an example toy Decision Tree on Covid complication risk with its main elements illustrated.

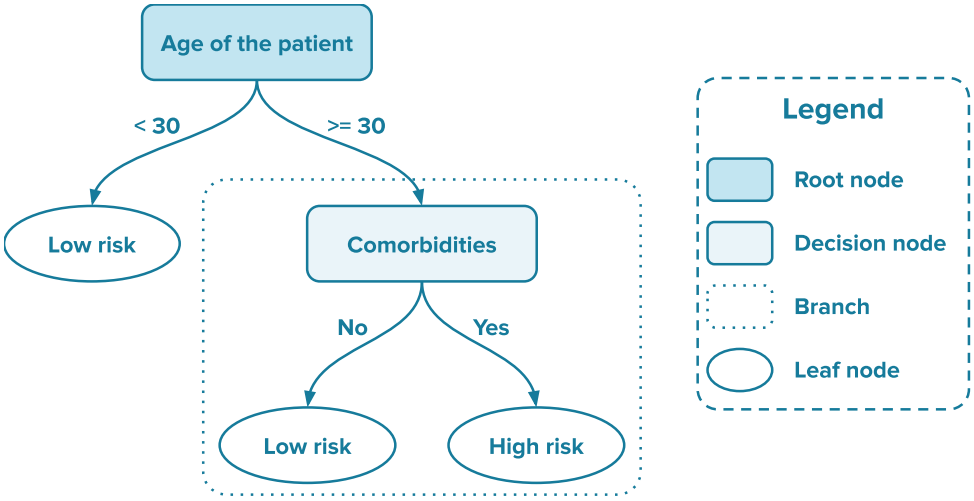


Figure 1.21: Example of a Decision Tree applied on complication risk assessment for Covid patients.

Decision Tress have been used to perform imputation at several occasions in the literature. A recent approach is DIFC, which have been proposed in (Nikfalazar et al., 2020), it is an imputation method that relies on Decision Trees and fuzzy Clustering. The dataset is split between complete and incomplete instances. One Decision Tree is trained on complete data for each feature with missing values. Values are imputed in the incomplete part of the dataset using trained trees. Imputed values are improved repeatedly based on a proposed iterative fuzzy Clustering algorithm. They only compare their DIFC to other non state-of-the-art imputation methods, it is thus impossible to assess the quality of their results.

Decision Trees can lead to rather weak inferences, a more advanced and powerful approach is to use Decision Trees in an ensemble manner, where multiple trees are trained on the same task and their results are combined through a majority vote, this is called a Ran-

dom Forest. Random Forests leads to far better prediction results than individual trees, to the cost of their interpretability.

The most known, used and performing imputation method using Random Forests is MissForest, proposed in (Stekhoven and Bühlmann, 2012). It is an iterative imputation method relying on Random Forests that has been shown to outperform most other imputation methods, including the statistical state-of-the-art MICE method, at several occasions (Kokla et al., 2019; Tang and Ishwaran, 2017; Waljee et al., 2013). In this algorithm, a Random Forest is computed for each feature with missing values to impute. The process is very similar to the MICE algorithm presented earlier. Missing values are initially imputed using a simple substitution method. Each forest is trained and imputes missing values in their associated feature, usually starting with the imputation of features with the most amount of missing values. The process of forests computation and imputation is repeated until convergence.

MissForest can still be considered a current state-of-the-art imputation method as it leads to competitive and often better results when compared to other currently state-of-the-art methods. This method is able to exploit non-linear correlations between features to perform high quality imputation, while using the powerful ensemble learning paradigm. The main drawback of MissForest is its required running time which is highly dependant of the amount of features in the dataset. Indeed, a separate forest of Decision Trees is computed for each feature with imputation performed in a round-robin manner, with the whole process being iteratively repeated until convergence (Tang and Ishwaran, 2017).

Genetic Algorithms. Genetic Algorithms are a type of exploratory algorithms inspired by the theory of evolution, they are usually used to discover high-quality and complex solutions to optimization and search problems (Katoch et al., 2021). Those algorithms are known to come up with original solutions that humans would have struggled to find. A Genetic Algorithm simulates a population and its evolution through generations towards better performing solutions, the evolution of the population is based on the concepts of natural selection and reproduction.

Key elements of Genetic Algorithms are chromosome representation, fitness evaluation function, chromosomal crossover and mutations. Typically, the procedure of a Genetic Algorithm is as follows:

1. A population of individuals/entities (candidate solutions) is initialized, each entity chromosomes are randomly generated. Those chromosomes are the characteristic of each individual, they dictate their behavior.

2. The fitness (performance measure) of each entity is evaluated based on the defined fitness evaluation function. Best candidate solutions get higher scores than worse solutions. The fitness scores of individuals is used as weighting for the selection process leading to reproduction.

3. A new population is formed through the reproduction process based on individuals fitness scores, best performing entities have more chances of being selected. Each pair of individuals chosen for reproduction generates an offspring, this offspring inherits

characteristics of both parents through a chromosomal crossover process. At the time of reproduction, mutations can occur and affect the chromosomes of the offspring, leading to new properties. Usually, the amount of entities remain constant from a generation to the next.

4. Steps 2 and 3 are repeated until a candidate solution reaches a fitness score above a wanted threshold.

Chromosomes representation and the fitness evaluation function are specific to the task the algorithm is aimed to solve, and have to be defined precisely in regard to the problem at stake. The exploratory part of such algorithm is based on the random initialization of all individuals in the population, and on the crossovers and mutations that occur during reproduction. This exploratory process is what allows Genetic Algorithms to find new interesting solutions. As best candidate solutions are selected through generations, the whole population average fitness score increases, leading to better performing solutions.

The use and research on Genetic Algorithms seems to have spiked the interest of some researchers in the field of imputation those last years with very recent applications. In their paper, (Shahzad et al., 2017) propose a straightforward implementation of a Genetic Algorithm for imputation. They propose a fitness evaluation function based on Information Gain, also commonly called Kullback–Leibler divergence, which is a measure of divergence between two distributions based on correlation. This measure is used to estimate the additional information captured in imputations throughout generations, leading to better and more pertinent imputations. Another recent proposal for Genetic Algorithms based data imputation is (Al-Helali et al., 2021). They propose a hybrid algorithm based on a Weighted KNN method and a Genetic Algorithm. Weighted KNN is employed to select the k closest instances to the incomplete instance of interest. The Genetic Algorithm is trained on those k instances, weighted with the same weights as the KNN method. Each incomplete instance in both training and test set are imputed in this manner, they show competitive results when compared to other ML methods. Very recently, (Figueroa-García et al., 2023) proposed MIGA, a Genetic Algorithm for imputation with a new multi-objective fitness evaluation function. Their fitness function is based on the Minkowski distance computed on the means, covariances and skewness between available and completed values, with the goal to impute missing values without modifying those three statistical properties. The proposed fitness function is composed of three terms, one term for each statistical property to preserve. They show that their method reaches good imputation results while preserving original statistical properties of the data. State-of-the-art results have been reported, the only method that obtained better results than theirs on one dataset is CMIM, a clustering based imputation method.

Modern imputation solutions based on Genetic Algorithms have been developed those last years, with good imputation results reported when compared to other state-of-the-art methods. Genetic Algorithms seem to be a newly active area of interest in the imputation research field.

Optimal Transport. Optimal Transport aims at solving the general problem of moving a distribution of mass to another distribution in the most efficient way possible (Villani, 2009). The usual example to explain the intuition behind Optimal Transport theory is the following. Consider trying to fill a hole using sand from a pile of the same volume, while minimizing the average distance moved. Optimal Transport aims at finding this minimal way of moving each sand grain from the pile to the hole. This simply put problem sees applications in many scientific fields, such as in mathematics, geometry, data science, etc. It is currently under very active research.

Currently, and to the best of our knowledge, only one imputation method based on Optimal Transport theory exists for tabular data imputation. We noted two other papers for imputation based on Optimal Transport but focused on time-series imputation, we talk more about them in section 1.2.3.5. In their paper, (Muzellec et al., 2020) present two new innovative imputation methods based on Optimal Transport theory. The first one is based on the Optimal Transport Sinkhorn divergence measure, it aims at minimizing the Sinkhorn distance between batches with respect to the imputed values. They formalize their theoretical problem as an optimization problem where the Sinkhorn divergence is to be minimized and solve it through gradient descent, they apply the commonly used Adam optimizer. This method is trained in a supervised manner on complete instances and imputes incomplete ones. They propose a second method where they address this limitation and train the method on both complete and incomplete instance in a round-robin manner, which consists in iteratively imputing over features in a cyclical manner as it is the case in the statistical method MICE. Their best performing method is the Sinkhorn based imputer without the round-robin implementation. They obtain extremely good results when compared to other state-of-the-art imputation methods, their Sinkhorn method can be considered a state-of-the-art imputation method.

As applying Optimal Transport theory for imputation is very innovative and leads to highly competitive and efficient methods, it is very probable that we might see more imputation methods using advances in Optimal Transport in the years to come.

1.2.3.4 Deep Learning Imputation Methods

More recently, methods that rely on Deep Neural Networks have been researched in order to improve the quality of produced imputations. As presented in section 1.1.2, Neural Networks are a type of Machine Learning model that are organized as a succession of layers composed of neurons. Deep Learning methods are able to exploit more abstract information about the data than other ML methods, which explains why most currently used learning methods are based on DNNs. The amount of proposed DL based imputation methods is still very limited at the moment, this research field seems to get more active those last years with recent interesting and promising proposals in the literature. In this section, we review the most popular DL imputation methods.

Auto-Encoder. AEs, presented in section 1.1.2.4, are a type of NN that are trained to reproduce their input. Standard AEs are limited in their usefulness for imputing missing values, but some methods have been proposed in the imputation literature (Pereira et al., 2020b), we present those works here.

In their paper, (Thinh et al., 2016) proposed U-AuCo, an imputation method based on an AE for imputing missing quality of service values of web services. They propose a multi-view AE, exploiting data from other views for training and inference, they do not describe the way missing values are imputed. The authors in (Beaulieu-Jones et al., 2017) perform imputation based on an AE trained using a modified version of a Cross-Entropy loss: $\mathcal{L} = \frac{x\log(\hat{x})m+(1-x)\log(1-\hat{x})m}{\text{sum}(m)}$. Where $\hat{x}$ is the reconstruction of x and m represents the missingness, with $m_i = 1$ if x_i is observed and 0 otherwise. This modified Cross-Entropy loss is only computed between observed values, missing values are imputed by the AE depending on other observed values in the instance. They apply their approach to health data with high amounts of missing values and show that their method performs well given any missing data mechanism. Authors of (Zhao et al., 2018b) propose an imputation scheme based on AE based fast-clustering and k-nearest neighbors. First, missing values are imputed using a distinctive value, an AE is trained to reconstruct the initially imputed data. Then, fast clustering is applied to the hidden representation obtained with the trained AE, using the high-level features extracted by the AE. In each cluster, a k-nearest neighbors algorithm is used to compute weighted imputation based on similarity between instances of the cluster. This three steps process is repeated iteratively until convergence of obtained imputations. They propose a very complete imputation approach, they compare their results to other state-of-the-art related methods and demonstrate that they outperforms those other methods. Unfortunately they did not publish their code, rendering especially difficult to reproduce their results given the complexity of the approach. Recently, (Lai et al., 2019) proposed the tracking removed Auto-Encoder (TRAE) architecture, which dynamically redesigns the structure of hidden neurons. The method first imputes naively missing values. They base their approach on the observation that in a standard AE, when value $\hat{X}_{i,j}$ is generated in the output, it largely depends on value $\tilde{X}_{i,j}$ in the input, the correspondence between those two values is said to be tracked in the model. In TRAE, to generate the prediction value $\hat{X}_{i,j}$, the value $\tilde{X}_{i,j}$ is removed from the input layer, largely weakening the role of $\tilde{X}_{i,j}$ on the estimation of $\hat{X}_{i,j}$ and enhancing the cross-correlation between $\hat{X}_{i,j}$ and other attributes. The process is performed in order to impute all missing values, and is then iteratively repeated until convergence of imputed values. They evaluate their elegant solution against other NN based imputation methods and demonstrate the capacity of their model to successfully impute incomplete tabular data.

Other approaches relying on more advanced kinds of AEs have been proposed in the literature, the next paragraphs present imputation methods based on Denoising Auto-Encoders (DAEs) and variational AEs.

Denoising Auto-Encoder. DAEs are a special kind of AE model that aim to solve the limit on the size of the bottleneck layer. Indeed, when designing an AE, hidden layers dimensions are usually set to be smaller than the number of features in the input and output, the point being to operate a dimension reduction to maximize model generalization. This comes with the disadvantage that the model is limited in its learning. A naive approach to solve this limitation would be to set the size of the hidden layers larger than the size of the input. This is an issue as the model might then be able to simply learn to perform an identity function from the input to the output, rendering the hidden representation void of any generalization and the model useless in its application to test data. The goal of a DAE is to increase the model learning potential by allowing hidden layers to be larger than the output. They prevent the model from learning an identity function by randomly masking parts of the input and training the model to reconstruct the clean input from the corrupted one. DAEs are a popular model in the DL literature as they come with the advantage of strongly limiting overfitting. The fact that they are trained to reconstruct a complete output from a partially masked input makes them extremely suitable models for missing values imputation.

Quite early-on, DAEs have been used to try to deal with missing data. In their paper, (Vincent et al., 2008) propose a way of training a DAE while corrupting its input so that it becomes robust to missing values and able to naturally perform imputation on test data. The implementation is straight-forward, the model is trained on the complete part of the dataset to reconstruct a clean input based on a corrupted one using a reconstruction loss, such as an MSE, and is then applied to the incomplete part of the data to impute missing values. A limitation of this simple approach is that it can only be trained on complete instances, which might not be a consequent part of a real-world dataset. More recently, (Lee and Lee, 2017) proposed imputation-boosted DAE (IDAE), for imputing missing values and learning on the imputed dataset to perform top-N recommendation. They apply the model to a binary rating matrix $\tilde{X} \in \{0,1\}^{n \times d}$ and use a Cross-Entropy loss to train the model. First, a standard AE model is trained to reconstruct the incomplete data, missing values in $\tilde{X}$ are imputed depending on the corresponding value in $\hat{X}$ based on a manually defined threshold. Then, improved imputations are obtained by training a DAE, original incomplete instances are used as a corrupted input and the model is trained to reconstruct the imputed output of the first model. This second model leads to better rating predictions. They compare their approach to other AE based imputation approaches applied to recommendation and show that their approach outperforms other methods. Authors of (Ning et al., 2017) propose a straight-forward implementation of a stacked DAE for imputation. A stacked AE is simply a way of training a multi-layer AE, its training process is to train the first layer alone, then the next one, and so-on until the deepest layer is trained. In their denoising version, the input of each layer is corrupted during training, improving the overall stability of the learned representation. Recently, (Gondara and Wang, 2018) proposed Multiple-Imputation using DAEs (MIDA), to perform imputation on mixed-type tabular data. The architecture of the DAE in MIDA sets larger hidden layers than the input/output size, improving the learning capacity of the model while avoiding overfitting thanks to the corruption of the input. They train the model by adding a corruption of 50% to the input and reconstructing the clean input. The model has the advantage of being able to be trained on incomplete data. Extensive experi-

ments on multiple mixed-type tabular datasets with many studied missing data mechanisms shows that this model outperforms the statistical state-of-the-art approach MICE. In their paper, (Sánchez-Morales et al., 2020) proposed SDAE, using stacked DAEs such as in (Ning et al., 2017). They propose a slightly modified loss term in order to exploit all available values in the data, the minimized reconstruction loss is $\mathcal{L} = ||(\hat{x} - x) \odot m||_2^2$, with m the missingness vector. Minimizing such loss makes use of all observed values in $\tilde{X}$ while not computing a loss value for any missing value. They then improved SDAE by proposing a complete imputation scheme called CMSDAE in (Sánchez-Morales et al., 2021). This scheme makes use of an intermediary classifier that is trained on a classification task given initially imputed data. Each DAE layer is then trained to perform both reconstruction of the input and the previous classifier output, ensuring that high-level features are extracted and present in the latent representation of each layer. They demonstrate better imputation results than MICE.

Variational Auto-Encoder. A Variational Auto-Encoder (VAE) is a kind of generative model that is composed of an Encoder and a Decoder and is trained as a standard AE. The main difference is that a VAE learns a mapping from the input to a multivariate normal distribution parameters μ and σ, from which the output is constructed. Once trained, it is possible to generate any sample on the latent normal distribution given any μ and σ parameter. Learning the multivariate normal parameters leads to the learning of a disentangled latent representation, where close instances in the latent space are also close in the original data space. Recently, several scientists have tried to exploit VAEs generative properties to perform missing values imputation in tabular data.

In their recent paper, (Mattei and Frellsen, 2019) proposed the missing data importance-weighted Auto-Encoder (MIWAE) bound, they use it to train a deep latent variable model to perform imputation. They train a VAE on the naively imputed dataset using their defined bound and generate imputations by using a Monte Carlo sampling approach. More recently, (Pereira et al., 2020a) introduced VAE Filter for Bayesian Ridge Imputation (VAE-BRIDGE). Their method uses a VAE to filter similar instances that are used to perform imputation, based on a linear regression. A VAE is trained on all complete instances in the data, learning a multivariate normal distribution parameters that represents the data. Incomplete instances are then fed to the trained VAE, their latent representation is used to determine how similar all instances are from one another. For each incomplete instance, the k most similar instances in the latent space are used to train a Bayesian linear ridge regression and perform imputation of missing values. They show in extensive experiments that their method obtains better results than other VAE and AE methods and MICE. Better results could certainly be obtained if the authors had chosen a more advanced non-linear regression model. Authors of (Nazabal et al., 2020) present HI-VAE, for handling incomplete heterogeneous data with VAEs, which can be used to both learn directly on incomplete data or impute missing values. They train their VAE model by optimizing a lower bound that is defined solely on observed data. They compare their method to other state-of-the-art imputation methods and show that their approach leads to competitive imputation results and similar classification performances. In their paper, (Qiu et al., 2020) perform imputation on medical data using a VAE approach. They propose a way of regularizing a VAE to correct the shift in distribution

that appears between complete data and incomplete data in an MNAR case. They show that VAEs easily lead to better imputation performance than standard AEs.

Variational Auto-Encoders have been recently explored and successfully used for missing values imputation in mixed-type tabular data. The generative properties of VAEs make them naturally inclined for imputation, leading to consistently better results than standard AEs.

Generative Adversarial Networks. Similarly to VAEs, Generative Adversarial Networks (GANs) are generative models that can be used to learn a data distribution and generate synthetic data samples on the same distribution. There exists many GAN based imputation methods that can be applied to images but a lot less that can be applied to tabular data. In this section, we review the few available GAN based tabular data imputation methods.

The most known and popular GAN based tabular data imputation method is GAIN, proposed in (Yoon et al., 2018). The Generator takes an incomplete instance $\tilde{x} \in \mathbb{R}^d$ and a noise vector z as input and outputs a completed instance $\mathring{x} \in \mathbb{R}^d$. The Discriminator takes the imputed instance $\mathring{x}$ and is trained to determine which of the d values is real and which is an artificial imputation from the Generator. They improve the process by providing the Discriminator with additional information about the missingness in the original instance $\tilde{x}$ in the form of a hint vector h. The hint vector h is constructed from the missingness vector m that indicates which values are missing in $\tilde{x}$, they propose a stochastic way to construct this hint vector. This hint mechanism helps the Disciminator in its task to discriminate based on the quality of the imputations. They obtain very good experimental results when compared to other state-of-the-art imputation methods such as MIDA, MICE, MissForest, Soft-Impute, etc. They evaluate their method both on the raw quality of their imputations compared to the real values and on the predictive performance obtained when training a learner on the imputed dataset. Following GAIN, (Zhang et al., 2018) extended the idea by using a Stackelberg GAN, that is, a type of GAN that contains multiple Generators. They use the input of multiple generator to obtain better imputations. Recently, (Awan et al., 2021) proposed CGAIN, a new imputation method based on a Conditional Generative Adversarial Network (CGAN). A CGAN is a variant of a GAN in which a condition is given to the Generator to generate an output following the given condition, this is ensured during training by giving the same condition to the Discriminator. The Discriminator learns to recognize if instances follow the given condition or not, which forces the Generator to produce instances properly constrained by the given condition. They base their method on the previously presented GAIN, they give the label of the incomplete instance to both the Generator and the Discriminator and propose a modified loss to train both models. They do not propose a way of imputing incomplete instances that are not labeled (test set). They compare their approach to GAIN, MICE and MissForest and show very good results in term of imputation accuracy. In parallel, authors of (Wang et al., 2021) proposed PC-GAIN a similar approach to CGAIN that has the advantage of being applicable to incomplete and fully/partially unlabeled datasets. First a standard GAIN model is trained to impute a part of the data that contain few missing values, a clustering method is used to cluster the imputed data, those clusters are used as pseudo-labels which are learned by a classifier. Then, the PC-GAIN model is trained, the conditional constraint is given by the previously trained classifier. This process makes it pos-

sible to apply PC-GAIN on both labeled and unlabeled data. They compare their method to other state-of-the-art method and show superior imputation accuracy results in most cases.

GANs tend to lead to better results when applied to image data, more diverse applications of GANs can be found in the image literature. GAIN is the most known GAN based method for imputing tabular data, some authors have contributed to try and improve this method.

Graph Neural Networks. Graph Neural Networks (GNNs) are a kind of NN that were originally introduced in (Tian et al., 2014) to learn a non-linear embedding of a graph and perform clustering on the latent representation, demonstrating that DL methods could be employed for graph clustering. Recently, innovative imputation methods based on GNNs have been proposed in the imputation literature. Those methods rely on the transformation of the tabular data to a graph, which represents the association between objects of the data. Most proposed methods are applied to the completion of ratings matrices, which associates users to items given the user rating, this kind of matrix is used in recommender systems. Some methods build a graph from tabular data where observations are linked to features, the weights of the connections are the values of each features. In this last case, GNN based imputation methods aim at predicting the weights of missing links in the graph.

Authors of (Berg et al., 2018) proposed graph convolutional matrix completion (GC-MC), a graph based Auto-Encoder to impute a rating matrix. They transform the rating matrix into a graph where users that rated certain items are linked to those items with a connection weight equal to the attributed rating. Their method learns to predict links from users to items, those predictions are used to perform imputation of missing ratings. They demonstrate that GC-MC performs better than related methods in the recommendation field. Recently, (Spinelli et al., 2020) proposed an adversarially trained GNN for tabular data imputation. Their approach resembles the previously presented GAIN, they base their method on a kind of graph Auto-Encoder, that is used as a Generator, with a Discriminator that is used to determine which values in a completed instance are real or imputed. They demonstrate good results when compared to other state-of-the-art imputation methods on well-known benchmark tabular datasets. In their paper, (You et al., 2020) proposed GRAPE, a GNN framework that is able to impute missing values while performing classification. They solve both those tasks by learning a prediction task, an edge-level prediction task to impute missing values, and a node-level prediction task to predict labels. They obtain extremely good results on both tasks.

GNN based imputation is a modern research field that will certainly see many interesting new methods in the years to come.

Transformer. Transformers are a recent kind of Neural Network initially proposed in (Vaswani et al., 2017), that is solely based on attention mechanisms. In their paper, (Vaswani et al., 2017) generalizes the attention mechanism first proposed in (Bahdanau et al., 2014) for LSTM models, to propose the dot-product attention and multi-head self-attention layer. A Transformer is composed of a succession of such layers, where learned parameters are used to pay attention to part of their input. This kind of model is very widely used in Natural Language Processing, where paying attention to a part of a sentence makes sense to perform tasks such as translation, answering questions, etc. Similarly, Transformers can naturally be used to learn on time-series data, but some researchers try to exploit the potential of Transformers and apply them on tabular data.

To the best of our knowledge only one Transformer based imputation method for non-longitudinal tabular data exists at the moment, that is (Wu et al., 2020). They propose an architecture similar to an AE that relies on a modified dot-product attention. They handle continuous and discrete variable separately, with a different loss to train the model in both cases. The dot-product attention is central and used to learn a representation of both continuous and discrete embeddings. They evaluate their approach on the quality of their imputation and compare their results to other imputation methods, they demonstrate very good performance on tabular data.

1.2.3.5 Imputation in Time-Series

Missing values in time-series data is a very common problem in real-world applications, thus, imputation in such a context is a very active research area. Time-series data are ordered by a temporal dimension, common examples are signals, which are a series of points evolving through time. As time-series are ordered through time, given enough readings, points should follow a trend through time, making it possible to try and predict future values given past points. When points are missing in the series it is possible to impute them based on the observed trend in the surrounding, given that enough surrounding points are observed. This area of research is well developed and many DL based imputation methods have been proposed to impute time-series. Many DL based methods to impute missing values in time-series have been proposed, based on RNNs, GANs, AEs and their variants, GNNs, Transformers, and others (Du et al., 2023; Miao et al., 2021; Pereira et al., 2020b). All those methods rely on the temporal dimension of the data to impute missing values in the series, despite their similarities with other tabular data imputation methods, those cannot be used on non-longitudinal data.

1.2.3.6 Imputation in Images

In the image field, imputation of missing pixels is usually referred as inpainting. Imputing missing pixels in an image is quite a different task than imputing missing values in tabular data. In an image, neighboring pixels contain much information about close pixels, this information is invaluable to impute missing values with plausible values. Usually, missing values in images are not isolated pixels, but whole areas of the image that are missing. Inpainting methods are then used to reconstruct the missing area based on the known information in the rest of the image. The term inpainting comes from the hand inpainting process, which is performed by artists to restore damaged paintings.

Research to propose automatic methods for image inpainting have been researched for a long time, with traditional methods dating from before the 2000s (Jam et al., 2021). With the development of DL, image inpainting became an extremely active research field in the last ten years, with many DL based methods proposed to solve this problem (Jam et al., 2021; Qin et al., 2021). Most inpainting methods rely on CNNs, as this type of NN is currently the best kind of model to handle images. GANs, AEs, DAEs, and VAEs are widely used and explored in this kind of work. More recently, Vision Transformers have been used to perform inpainting (Pirnay and Chai, 2022).

There exists many DL based methods to perform image inpainting, those methods cannot be used as is on tabular data, as neighboring instances and features in a dataset do not share the same useful information as neighboring pixels in an image. However, some DL methods for image inpainting can be inspiring to develop new innovative and interesting imputation methods for tabular data.

1.2.4 How to Evaluate Imputation Methods

There are two main ways to evaluate the quality of the predictions obtained when applying an imputation method. The first way is to compare imputed values with the ground-truth values. Obviously this method can only be used when new missing values have been artificially inserted to a previously complete dataset, meaning that it is only useful in an experimental setting. The second way is to impute a same incomplete dataset using each imputation method to compare, train a learner on each imputed dataset and compare the obtained inference results. Better inference results indicate that more meaningful information has been captured in imputed values. Unlike the first, this second method can be used in any real-world scenario.

1.2.4.1 Evaluating Imputations Against Ground-Truth

The most popular and used method to compare imputation methods is to evaluate their imputations against ground-truth values. This method can only be used in an experimental setting, where ground-truth values are known and missing values have been artificially introduced in the originally complete dataset.

Commonly used metrics are the Mean Absolute Error (MAE) and Root-Mean-Square Error (RMSE) (Buuren, 2021; Lin and Tsai, 2020). We note x_i the i-th instance of the original complete dataset, and $\mathring{x}_i$ the i-th instance of the imputed dataset, n is the number of instances in the whole dataset, then, those metrics are defined as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - \mathring{x}_i| \tag{1.22}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mathring{x}_i)^2} \tag{1.23}$$

Both those metrics are representative of how close imputations are from the real original value.

This evaluation method is widely used in the imputation literature, most of the previously presented papers rely on one or the other to evaluate their method and compare state-of-the-art results. However, in the second chapter of his book (Buuren, 2021), Stef van Buuren argues that using such an evaluation metric is not a proper way of comparing two imputation methods. He shows that a linear regression model that obtains lower RMSE and MAE values than the MICE method leads to more biased estimates and invalid statistical inferences. That is because a linear regression model does not account for the uncertainty in missing values, unlike MICE which performs MI iteratively to impute more relevant values. He concludes that the RMSE metric is not an informative metric for evaluating imputation methods.

1.2.4.2 Evaluating Imputations on Inference Results

Another way of evaluating imputation results is to train a chosen inference model on the imputed training data, and evaluate inference results on the test set. Better inference results indicate better imputation quality. To obtain significant results, several inference models should be trained on a same imputed dataset, with their inference metric results averaged over all runs.

Any kind of metric that is commonly used to evaluate inference models can be used in such context. When the inference model is trained on a classification task, the most popular metric to evaluate the obtained results is the Accuracy:

$$ACC = \frac{TP + TN}{TP + FP + TN + FN} \tag{1.24}$$

Where TP and FP are the number of True Positives and False Positives respectively, and TN and FN are the number of True and False Negatives. The Accuracy indicates the percentage of right classification prediction over all made predictions. The higher the metric is, the most

accurate the inference model is.

The Accuracy metric is not always perfectly representative of the quality of inference results, as it is not sensitive to the distribution of False Positives versus False Negatives. In many situations it is preferable for an inference model to lead to as few false positives or false negatives as possible, where a low amount of one is highly preferable to a high amount of the other. For example, in a medical context where we want to predict if a patient is sick, it is important to limit as much as possible the amount of False Negatives, that is, the amount of patients that are predicted as healthy while, in reality, they are sick.

A metric can be used to evaluate the capacity of an inference model to limit the amount of False Negatives, that is, the Recall:

$$REC = \frac{TP}{TP + FN} \tag{1.25}$$

Inversely, another metric can be used to evaluate the capacity of an inference model to limit the amount of False Positives, that is, the Precision:

$$PRE = \frac{TP}{TP + FP} \tag{1.26}$$

As can be seen from the above example, the Accuracy metric alone might not always be a relevant metric to evaluate the quality of an inference model. The specific situation in which the model is employed must be considered to choose an adequate combination of evaluation metrics.

A metric that is commonly employed when it is equally as important to minimize both Precision and Recall is the F_1-score. The F_1-score is computed as the harmonic mean of the Precision and Recall, it represents both those metrics in one:

$$F_1 = \left(\frac{PRE^{-1} + REC^{-1}}{2} \right)^{-1} = 2\frac{PRE \cdot REC}{PRE + REC} = \frac{2TP}{2TP + FP + FN} \tag{1.27}$$

Another widely used classification metric is the area under the Receiver Operating Characteristic (ROC) curve, that is, the Area Under the Curve (AUC). The ROC curve is obtained by plotting the Recall against the False Positive Rate $FPR = \frac{FP}{FP+TN}$ at various probability thresholds. A random inference model leads to an AUC value of about 0.5, whereas a perfect inference model gets a value of 1. In a binary classification setting, the AUC represents the probability that a random instance of the positive class is positioned to the right of a random instance of the negative class on the model output predicted probability.

There exists a lot more inference evaluation metrics, for classification and other supervised and unsupervised tasks, we only presented the most common ones that will be used in the rest of this manuscript. When imputing values in preparation for the training of an inference model, we argue that the most relevant metrics to use are those that evaluate inference quality of the model trained on the imputed training data and evaluated on imputed test data. Those are the only metrics that will be a real indication of the meaningful non-linear correlations that have been captured by the model and transferred into imputed values, which can then be exploited to maximize learning capacity.

1.2.5 Discussion

The imputation literature is an extensive and well-established research field, across the years, researchers have contributed in enriching the field with the proposal of many innovative approaches to deal with missing values in data science. Historically, imputation methods were mostly statistical methods, carefully design to exploit statistical properties of the data to replace missing values with the most pertinent and probable values. A very limiting factor of these approaches has been their limitation to only be able to impute missing values as a kind of linear combination of observed values. Nowadays, most well-performing imputation approaches are based on Machine Learning and Deep Learning, which both have the advantage of being able to extract non-linear correlations from observed values to generate more pertinent imputations compared to most statistical imputation approaches.

1.3 Attribute Noise Correction

In this thesis we are interested in improving the quality of the available data as much as possible before using it for training Machine Learning models. As seen previously, this means finding a proper solution to handle missing values. In this section, we will see that it also means finding a solution to handle erroneous values. The presence of erroneous values in real-world data is almost unavoidable, there are two main causes for this type of corruption (Zhu and Wu, 2004). Human errors during data entry or any other data manipulation task, and errors introduced by measurement tools, due to improper calibration, machine degradation, etc. Such erroneous values can have a highly negative impact on inference quality, similarly to missing values. Erroneous and missing values are very similar, as they are both corruptions of the data that need to be handled in order to maximize inference quality. In the following work we will refer to those two types of data corruptions as “noise”, such as in (Zhu and Wu, 2004).

In order to unify notations, in the rest of this work we note $X \in \mathbb{R}^{n \times d}$ the clean ground-truth dataset, without any missing or erroneous values, that is, without attribute noise. Obviously this ground-truth dataset is not always available, as real-world datasets are originally corrupted.dataset that is available for training and must be corrected in order to maximize future in- We note $\tilde{X} \in \mathbb{R}^{n \times d}$ the corrupted dataset, either naturally corrupted for a real-world dataset or artificially corrupted from X by adding erroneous and missing values. This is the dataset that is available for training and inference, and must be corrected in order to maximize future prediction results. We note $\mathring{X} \in \mathbb{R}^{n \times d}$ the corrected dataset obtained by applying a correction method to $\tilde{X}$.

1.3.1 Two Types of Noise

It has been estimated that errors in a standard real-world dataset is typically around 5% or more (Zhu and Wu, 2004). The quality of attribute (feature) values and class labels influences drastically inference results. We distinguish two types of noise: attribute noise, which refers to erroneous and missing values in the attributes, and class noise, referring to erroneous values in the labels.

- **Class noise:** Class noise refers to contradictory and mislabeled instances. Contradictory instances are instances that have identical attribute values but a different label value. Mislabeled examples are instances that are not labeled with the right class.
- **Attribute noise:** Attribute noise refers to erroneous and missing values in the attributes. An erroneous value can occur in many situations, such as a human error when typing the value, or a machine issue when recording the value, etc. Missing and unknown values that were reviewed earlier are just a special case of attribute noise.

Figure 1.22 shows an example dataset with both attribute and class noise. We observe attribute noise, with entry errors, such as the glucose value of 890 for the third patient, that is an example of a typing error, the right value should have been 89. Another erroneous value example that could be due to a machine malfunction is the insulin value of 22 for the last patient, which should be higher as this patient has diabetes and a high glucose level. Attribute noise also refers to missing values, identified as NA, which have been extensively reviewed in the previous section. We also observe class noise, where the labels of the first and third patients are wrong.

Age	Sex	Weight	Glucose	Insulin	Pressure	Diabetes	Survival
71	Male	86	187	304	68	True	**True**
58	Female	54	91	86	**NA**	False	**True**
26	Male	72	**890**	37	76	False	**False**
87	Female	65	83	71	78	False	**False**
46	Female	92	171	**22**	110	True	**False**

Figure 1.22: Toy dataset (fake health data) used to illustrate concepts and issues about noise. Toy medical dataset with both attribute and class noise, the target output is the survival column, remaining features are the patients' health data. NA refers to a missing values, red values indicate noise.

It has been repeatedly stated that attribute noise occurs more often in real-world situations than class noise (Van Hulse et al., 2007; Yang et al., 2004; Zhu and Wu, 2004), meaning that it tends to be more harmful and a more important matter than class noise. Despite this fact, extensive research have been conducted to try and limit the impact of class noise (Gupta and Gupta, 2019), while very limited attention has been given to attribute noise. This is often attributed to the fact that handling attribute noise is reputed as harder than handling class noise (Teng, 2000; Zhu and Wu, 2004).

Similarly, methods to handle missing values, which are only a special case of attribute noise, have been, and are still, widely researched, as can be seen from the previous section. While we observe very limited research on erroneous values handling methods. More specifically, to the best of our knowledge, there currently exist no method in the literature to handle attribute noise in its entirety. Often, missing values are addressed, but erroneous values are ignored, or it is assumed that there are no erroneous values in the data. And in the rare cases where both missing values and erroneous ones are handled, they are handled in two distinct steps. In the rest of this section and work, we focus on attribute noise handling methods, as we focus on improving data quality as much as possible before training any future inference model.

1.3.2 Ways of Dealing With Attribute Noise

Dealing with attribute noise in tabular data is usually performed in three different ways: using robust learners, removing noisy instances detected with a filtering method, or correcting erroneous values with a polishing approach (Van Hulse et al., 2007). Using a robust learner to learn on noisy data is probably the most straightforward approach, as no specific noise treatment is needed, the learner is trained on the corrupted data and used directly for inference on test data. Robust learners are expected to learn to ignore noise in training instances during training, but a corrupted training set can still lead to seriously degraded inference performance (Zhu and Wu, 2004). On the other hand, filtering and polishing approaches are used in a preprocessing step, where any learner can then be used to be trained on the cleaned data. Filtering methods aim at detecting noisy instances to remove them from the dataset, while polishing approaches aim to correct erroneous values so that all instances are kept in the dataset. Such preprocessing approaches have the advantage of separating the task that is handling attribute noise from the inference task we ultimately aim to learn. Furthermore, many learners have a higher learning potential but are sensitive to noise (Van Hulse et al., 2007). To use such learners, it is important to get rid of noisy instances prior to the learning phase.

As will be seen in this section, methods used to handle noise in data are only capable of dealing with erroneous values, and unable to deal with missing values. This is an important matter, as no method currently exist to handle attribute noise in its entirety.

Robust Learners. A commonly used tactic against attribute noise are robust algorithms. Robust learners have the advantage of being less likely to be affected by noise in the data (Van Hulse et al., 2007). Such model is trained directly on the corrupted version of the dataset $\tilde{X}$ and relies on its ability to ignore noise. However, relying on the sole ability of the learner to avoid the influence of noise during its learning phase is very limited. Indeed, in addition to its main learning task, the learning model is also given the burden of learning how to deal with noisy attributes (Gupta and Gupta, 2019). In this way, there is no need to try to correct erroneous values beforehand, which makes the whole process easier since it requires less preprocessing. But, the whole weight of dealing with attribute noise relying on the learner makes it harder for the model to focus on its main learning task, and can interfere with the

quality of its results (Teng, 2004). One way for a robust learner to ignore noise is to prevent overfitting. In this way the model cannot learn overly complicated concepts to try to reproduce the noise, but only generalizations about the data. An example of a robust learner is the C4.5 decision tree algorithm, where statistically insignificant portions of the trees are deleted to avoid overfitting the noise present in the data (Salzberg, 1994).

We are less interested in this category of noise handling methods, as learning from noisy instances has the inconvenience of solely relying on the robustness of the learner against noise, which limits Robust Learners in their capacity to give good results. Furthermore, many learners are highly sensitive to attribute noise, and require data preprocessing to improve overall data quality before training (Van Hulse et al., 2007). Therefore, in this work, we are interested in handling data corruptions in a preprocessing step, before focusing on designing an adapted inference model that will be trained on the preprocessed data.

Filtering. Filtering methods aim at detecting and deleting noisy examples from the training set (Brodley and Friedl, 1999; Gamberger et al., 1999, 2000; Van Hulse et al., 2007; Zhu and Wu, 2004). This preprocessing approach have the advantage of separating the task of dealing with noise from the learning task, which makes it easier to achieve good inference results for the final learner. Another advantage of such an approach is that, if the cleaned training dataset still contains some noise after filtering, the learner will be able to deal with it more easily and potentially obtain better results.

The idea behind filtering methods is that removing noisy instances from the training set can facilitate learning and improve generalization on unobserved instances in the test set, and so, inference results. According to research by (Yang et al., 2004), large datasets with excess data can benefit greatly from noise filtering, as it enables the model to learn from a noise-free, yet sufficiently large, training set. However, while most filtering methods address class noise, only a few methods exist to filter attribute noise due to the high complexity of such task (Gupta and Gupta, 2019). This section will concentrate solely on attribute noise filtering techniques.

Authors of (Brodley and Friedl, 1999) proposed the Ensemble Filter (EF) approach, that identifies and removes mislabeled instances in training data using the ensemble paradigm. The EF method trains an ensemble of classifiers on subsets of the training data to filter out noise from the training set, the authors call those classifiers filter algorithms. To identify potentially noisy instances, EF applies k-fold cross-validation on the training data using m classification algorithms, such as the C4.5 tree, 1 Nearest Neighbor, or linear regression, as recommended by the authors. The EF algorithm works as follows: First, the training set is split into k subsets. Each of the m filter algorithms is trained k times, using $k-1$ subsets as training data and being applied on the remaining one, leading to k classifiers for each filter algorithm. The k classifiers are used to label instances in the correspondingly excluded subset. Classification results are used to determine instances that are mislabeled in most cases based on a voting scheme with m votes. Instances that are mostly voted as noisy are removed from the dataset. Authors show that applying this method on a noisy dataset could drastically improve future prediction performance when training an inference model on the filtered training set. The main drawback of this approach is that it relies on classifiers, mean-

ing that it requires the dataset to be labeled in order to be applied. Around the same time, (Gamberger et al., 1999, 2000) proposed the Classification Filter (CF) approach, that identifies and removes mislabeled instances in training data in a similar way to EF. The approach works similarly to Ensemble Filter, but without using the ensemble voting scheme. Instead of training m filter algorithms that are used in a voting scheme, only one is trained and its results are directly used to filter out noisy instances. This method is simpler to implement and faster to execute than EF but probably leads to less robust results. A very simple filtering method was described in (Marcus et al., 2001), where filtering rules are manually defined to detect outlier values in the dataset, instances that include outlier values can be filtered out. An obvious disadvantage of such an approach is the laborious, and eventually impossible, task of manually defining a precise set of rules for each attribute of the dataset. Authors of (Yang et al., 2004) propose Sifting, a filtering approach based on decision trees such as C4.5, the method is equivalent to the Classification Filter approach. Lastly, (Van Hulse et al., 2007) introduced the Pairwise Attribute Noise Detection Algorithm (PANDA), which outputs a ranking of all instances of the dataset from most to least noise, with the advantage of not requiring a labeled dataset unlike other filtering approaches. Most other filtering techniques rely on a prediction performance to identify noisy instances, but such identification is not necessarily reliable, as correctly predicted instances are not necessarily noise-free, and inversely. The PANDA algorithm aims to identify instances with significant deviations from normalcy, given the values of a pair of attributes. When a group of instances shares similar values for one attribute, large deviations for the second attribute can be considered suspicious, with higher deviations indicating stronger evidence of noise. However, caution must be exercised in interpreting this assumption, as the variance and distribution of the two attributes may differ. To address this issue, PANDA standardizes all attributes in the data set. The algorithm produces an ordered list of instances, with each assigned a Noise Factor score to rank its noisiness relative to others in the data set. The Noise Factor computation process is executed for each pair of attributes, and the results are aggregated to produce an overall output score representing the amount of noise present in the instance. This last algorithm has the advantage of being usable whether the dataset is labeled or not, to the best of our knowledge, this is currently the last innovative attribute noise filtering approach that has been proposed in the literature.

There are a few methods that can be used to detect noisy instances in a dataset. The process of filtering aims at removing noisy instances from the dataset in order to improve the overall quality of the dataset. The obvious disadvantage of such approach is that information is lost. An analogous approach to imputation methods that can be used to exploit all clean values in the data to correct erroneous ones is polishing, which aims at correcting noisy values, preserving all other values instead of deleting them.

Polishing. As stated above, removing instances from the training set can become a problem if the dataset contains a limited amount of instances. For example, in a medical real-world case where we are already limited by the number of patient who are part of the study, any deleted data instance means that important information is lost. In that case, using filtering methods would not be the best solution. Another way of dealing with instances detected as noisy is to polish them, that is, correct erroneous values to replace them with more plausible ones, maximizing the amount of retained information. That is very similar to imputation, except in this case we do not know which values are erroneous and which are correct. This constraint makes it harder to develop attribute noise polishing techniques compared to imputation methods.

Teng proposed the Polishing method in (Teng, 2000, 2003, 2004). This approach exploits the correlations between attributes of the data to identify noisy instances and propose plausible corrected values for noisy ones. By utilizing both the label and other features of the instance to polish, the Polishing method predicts the value of the noisy feature. The first step of the algorithm relies on a Filtering algorithm to detect instances that contain noise and separate them from clean ones, this is the identification phase. Any Filtering method can be used at this point, in her papers, Teng uses the Classification Filtering method. Then, one prediction model is trained to predict each feature in the dataset, leading to d trained models for a corrupted dataset $\tilde{X} \in \mathbb{R}^{n \times d}$. Each prediction model f_i is trained on the clean part of the dataset based on all features except the i-th feature that it must predict. Models are applied to the noisy instances to predict new values for all features, $\hat{x}_i = f_i((\tilde{x}_1, \ldots, \tilde{x}_{i-1}, \tilde{x}_{i+1}, \ldots, \tilde{x}_d))$. Attribute values of noisy instances are updated based on empirical thresholds and conditions that must be manually defined. Originally, values of noisy instances are only updated if the absolute difference between the predicted value $\hat{x}_i$ and the original one $\tilde{x}_i$ is above a set threshold. Intuitively, only original values that are drastically different from the predicted values, and so probably noise, are replaced with the prediction. Figure 1.23 shows an illustration of the polishing process such as proposed by Teng.

More recently, (Shahzad et al., 2017), that proposed a Genetic Algorithm based imputation method (cf. section 1.2), have listed in their future perspectives to research and apply their imputation method to noise and erroneous values correction. At the moment, no following paper have been found in the literature, but such a research work could lead to a new interesting polishing method based on a Genetic Algorithm.

In their applicative paper on a real-world medical dataset, (Liebchen et al., 2007) showed that employing the Polishing method in combination with the Classification Filtering approach leads to better inference results than removing filtered instances or not doing any filtering. It has also been shown that the Polishing method improves classification accuracy more than using Filtering or Robust Learners techniques, but that it introduces additional noise in the dataset (Gupta and Gupta, 2019). This is similar to imputation methods that lead to better inference results while leading to worse RMSE results as stated in section 1.2.4.2, which might not indicate poor correction, on the contrary.

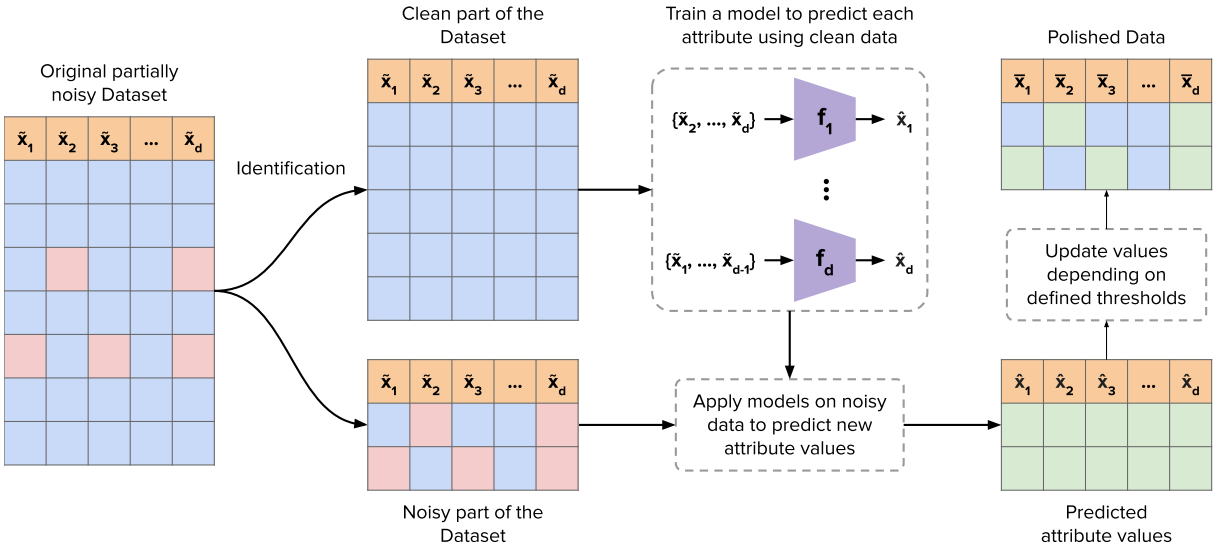


Figure 1.23: Polishing process. Noisy instances are identified using the chosen filtering method. Clean instances are used to train one inference model to predict each data attribute. Models are applied on noisy instances to predict values of corrupted attributes, attribute values are updated if the difference between predicted and original value is above a set threshold, leading to polished instances. The polished dataset is obtained by combining clean instances with polished ones.

1.3.3 How to Evaluate Noise Correction Methods?

Evaluating noise polishing is quite similar to evaluating missing values imputation, as the outcome to evaluate is the same: improving data quality by replacing corrupted data with plausible values. Consequently, the two main ways to evaluate the quality of a polished dataset are the same as when evaluating an imputed dataset: comparison against a known ground-truth or evaluating the predictions of a learner trained on the corrected dataset. Evaluating noise filtering methods can only be based on the evaluation of predictions obtained by training a learner on the filtered dataset. Indeed, filtering removes noisy instances, there is therefore no added information to compare to a known ground-truth.

In a situation where noise is artificially added to corrupt a dataset, as with missing values, we can evaluate how close polishing results are from the ground-truth. Such evaluation is performed using the same distance metrics as with imputation methods detailed in section 1.2.4.2: MAE, RMSE, etc.

Similarly, filtering and polishing results can be evaluated by evaluating a learner trained on the filtered or polished dataset and applied to a test set, such as in (Brodley and Friedl, 1999; Gamberger et al., 1999, 2000; Liebchen et al., 2007; Teng, 2004). In such case, any suitable inference metric can be used to compare the obtained results, such as the Accuracy, AUC or F_1-score for classification. As for evaluating imputation methods, we argue that when we apply a polishing method in preparation for the training of an inference model, then, the most relevant metrics to use are those that evaluate inference quality of a model trained on the corrected data. Indeed, those are the only metrics that give a real indication of the mean-

ingful non-linear correlations that have been captured in corrected values and can then be exploited to maximize learning capacity.

Other specific metrics are sometimes used in the specific literature of attribute noise handling methods. In her paper, (Teng, 2000) proposed the Net Reduction metric, which is computed as:

$$NR = \frac{\sum_{i=1}^{n} d(\tilde{x}_i, \tilde{z}_i) - d(\mathring{x}_i, \mathring{z}_i)}{\sum_{i=1}^{n} d(\tilde{x}_i, \tilde{z}_i)} \tag{1.28}$$

with $\tilde{x}_i$ the original noisy instance i, $\mathring{x}_i$ its polished version, d a distance function, and $\tilde{z}_i$ and $\mathring{z}_i$ the nearest neighbors of $\tilde{x}_i$ and $\mathring{x}_i$ respectively. It indicates how much noise there is in a polished dataset compared to the original version of the data. It relies on the comparison between the original instance and its polished version with their respective nearest neighbors. The NR value is close to 1 if the distance between $\mathring{x}_i$ and $\mathring{z}_i$ is small, that is, if much noise has been corrected in the polished instance. It is close to zero if $d(\mathring{x}_i, \mathring{z}_i) \approx d(\tilde{x}_i, \tilde{z}_i)$, meaning that the polishing process did not manage to properly reduce the noise of $\tilde{x}_i$. It is negative if $d(\mathring{x}_i, \mathring{z}_i) > d(\tilde{x}_i, \tilde{z}_i)$, demonstrating that the polishing process contributed to the addition of more noise than what was originally present in $\tilde{x}_i$. This metric has the advantage of being calculable in real-world scenarios, where the only available dataset is the corrupted one $\tilde{X}$.

Other metrics that compare inference quality between two models, trained on X and $\mathring{X}$ respectively, are sometimes used. A disadvantage of those metrics is that they can only be used in an experimental setting where the ground-truth version of the data is known and noise has been artificially introduced into the data. For example, the Relative Loss of Accuracy (RLA) has been proposed in (Sáez et al., 2011).

$$RLA = \frac{ACC_X - ACC_{\mathring{X}}}{ACC_X} \tag{1.29}$$

Where ACC_X is the average accuracy obtained on the test set by the inference model trained on the ground-truth data X and $ACC_{\mathring{X}}$ is the average test accuracy obtained by the model trained on the corrected data $\mathring{X}$. It measures the relative loss of accuracy between a noise level of 0% in X and a noise level of x% in $\mathring{X}$, the lower the RLA value the better the correction method worked.

1.3.4 Discussion

The attribute noise correction literature is extremely limited, as most researchers focus on designing new imputation methods, while overlooking the issue of erroneous values in tabular data. It has been shown that erroneous values in attributes of tabular data are a serious issue, that drastically decreases the learning performance of ML models, while being an almost unavoidable occurrence in real-world datasets (Zhu and Wu, 2004).

Our exploration of this neglected data science field highlights that no method currently exists to both impute missing values and correct erroneous ones. A few methods exist to deal with erroneous values in tabular data, those are robust learners, filtering approaches, and polishing. Out of those three categories, only the polishing approach aims at correcting erroneous values. Filtering approaches delete instances that are suspected to contain

errors, reducing the size of the dataset, which can be an issue, as important information in non-erroneous attributes is lost. Robust learners rely solely on the capacity of the learner to ignore errors in the dataset, which is very limited and does not lead to the best possible results.

Currently, the only way of implementing a complete attribute noise correction method using existing components of the literature is by combining an imputation method with the polishing approach. Research must be conducted in this field in order to propose innovative, and well-performing, ways of dealing with attribute noise in its entirety as a preprocessing step, paving the way for any kind of subsequent predictive model.

1.4 Domain Adaptation

In a standard Machine Learning setting, an inference model is trained to solve a specific task on a training dataset, and is then applied to perform the same task on test data. That is, the training and test data are assumed to follow an identical latent distribution, and the training and test tasks to solve are identical. This standard learning setting can be used in most cases, but there are situations where the available training data is slightly different from the test data on which the model must be applied, or where the task that is learned on training data is not the same that will be applied on test data. Transfer Learning aims to maximize learning and prediction performance in such contexts.

In Transfer Learning, we aim to exploit knowledge from one or multiple source dataset(s) to enhance learning performance on a different target dataset. For Transfer Learning to be effective, the source dataset(s) should exhibit sufficient similarity to the target dataset. A source dataset that is not similar enough to the target dataset can negatively impact learning performance and should not be used in this context. Examples of Transfer Learning problems are: the transfer of knowledge from source data to distinct target data, the transfer of knowledge from a source task to a distinct target task, or a combination of both (Pan and Yang, 2010; Wilson and Cook, 2020). Specifically, in this work we aim to learn an identical prediction task, while exploiting knowledge from several different source datasets. The setting of Transfer Learning where a model is trained to learn a common task by transferring knowledge from one or more source domain(s) to a target domain, is called Domain Adaptation (DA) (Pan and Yang, 2010; Wilson and Cook, 2020; Zhuang et al., 2020).

Domain Adaptation is the process of adapting a model trained to perform a specific task on one or several source domain(s) to perform the same task on a different target domain, where each domain may have different statistical properties or distributions (Pan and Yang, 2010; Zhuang et al., 2020). DA can greatly enhance prediction performance on the target domain by leveraging more knowledge from the source domain(s) than what is available in the target domain. Obviously, as stated above, for DA to be effective, the source domain(s) should be similar enough to the target domain, so that useful knowledge can be transferred from source(s) to target. An example of a learning setting that can benefit from DA is to use health data from patients suffering from a given pathology in a country A as a source domain, to improve learning performance on data from patients suffering from the same pathology in country B. In such a context, patients from the country A might have a different

lifestyle from patients from country B. Thus, training a model on the first group of patients and applying the model on patients of the second group might lead to poor prediction, as data might be too dissimilar from one country to the other. Therefore, we are in a case with two similar but different datasets (domains). Using Domain Adaptation can help to exploit knowledge from dataset A to maximize prediction performance on dataset B.

Single-Source Domain Adaptation (SSDA) is the specific DA case where a single source domain is used for knowledge transfer to the target domain. SSDA is a widely and actively researched field, most DA applications fall under this setting (Ganin et al., 2017; Long et al., 2015; Saito et al., 2018; Zhu et al., 2021). On the other hand, Multi-Source Domain Adaptation (MSDA) is the DA case where multiple source domains are used for knowledge transfer to the target domain. The MSDA field is a more complex and less researched area compared to SSDA, but exploiting knowledge from several source domains can lead to better inference results on the target domain than when using a single source domain (Peng et al., 2019; Zhao et al., 2018a; Zhu et al., 2019b; Zuo et al., 2021). In the following work, we are more interested in MSDA as we aim to exploit knowledge from several available source domains to maximize inference performance on each target domain.

DA is often employed to enable prediction on a completely unlabeled target domain while exploiting one or several labeled source domain(s), this is known as Unsupervised Domain Adaptation (UDA) (Ganin et al., 2017; Long et al., 2015; Wilson and Cook, 2020). However, in the context of this thesis, the focus is on Supervised Domain Adaptation (SDA), where the target domain is also labeled and the goal is to exploit source domains to improve predictions on the target. Despite being a common occurrence in the real world, Supervised Domain Adaptation (SDA) remains an area with relatively less research attention than Unsupervised Domain Adaptation (UDA).

We can also distinguish homogeneous Domain Adaptation from heterogeneous DA (Day and Khoshgoftaar, 2017; Wilson and Cook, 2020). Homogeneous DA is when the feature spaces of all source and target domains are identical, that is, data is represented by the same features across all domains. On the other hand, heterogeneous DA is when the feature spaces of all source and target domains are not identical. In such case, useful knowledge might be available in the source domain(s), but it is represented differently than in the target domain. Therefore, in such context, it is primordial to employ a tactic to bridge the gap between different feature representations for knowledge transfer.

1.4.1 Theory Behind Domain Adaptation

Here, we introduce definitions and concepts needed for the following, our notations are inspired by the work of Cortes and Mohri (Cortes et al., 2019), we took liberties to adapt them to better suit our specific setting.

Let $\mathcal{X} \in \mathbb{R}^d$ denote an input feature space, with d the number of features, and $\mathcal{Y} = \{1, \ldots, c\}$ a multi-class output label space, with c the number of classes. We define a domain as a pair formed by a marginal distribution over $\mathcal{X}$ and a specific labeling function mapping from $\mathcal{X}$ to $\mathcal{Y}$. We note $\mathbb{D} = (P(X_\mathbb{D}), f_\mathbb{D})$ the domain $\mathbb{D}$, with $P(X_\mathbb{D})$ the marginal distribution of $\mathbb{D}$ over $\mathcal{X}$, and $f_\mathbb{D} : \mathcal{X} \to \mathcal{Y}$ the labeling function mapping from feature to label space. $X_\mathbb{D}$ is the data

sample defined as $X_{\mathbb{D}} = \{x_i \in \mathcal{X}\}_{i=1}^{n_{\mathbb{D}}}$, with $n_{\mathbb{D}}$ the number of instances in the data sample of the domain $\mathbb{D}$.

In a scenario of Single-Source Domain Adaptation, we are given a unique source domain $\mathbb{S}$, that is exploited to improve classification over one target domain, noted $\mathbb{T}$. In a scenario of Multi-Source Domain Adaptation, we are given s source domains, noted $\mathbb{S}_i$ for $i \in [1, s]$, that we want to exploit to improve classification over the target domain $\mathbb{T}$. A unique label space $\mathcal{Y}$ is shared across all domains, we note $\mathcal{X}_{\mathbb{D}}$ the feature space of domain $\mathbb{D}$. In an homogeneous DA setting, the feature space of each domain must be identical, that is, $\mathcal{X}_{\mathbb{S}_1} = \cdots = \mathcal{X}_{\mathbb{S}_s} = \mathcal{X}_{\mathbb{D}}$. In an heterogeneous setting, they can be different across all domains, that is, $\mathcal{X}_{\mathbb{S}_1} \neq \ldots \neq \mathcal{X}_{\mathbb{S}_s} \neq \mathcal{X}_{\mathbb{D}}$.

In a scenario of Multi-Source Domain Adaptation, we have access to s labeled source domains where $\mathbb{S}_i = \{(x_j^{\mathbb{S}_i}, y_j^{\mathbb{S}_i})\}_{j=1}^{n_i}$, with $\{x_1^{\mathbb{S}_i}, \ldots, x_{n_i}^{\mathbb{S}_i}\} \sim P(X_{\mathbb{S}_i})$ and $y_j^{\mathbb{S}_i} = f_{\mathbb{S}_i}(x_j^{\mathbb{S}_i})$. In an Unsupervised Domain Adaptation setting, the target domain is unlabeled, it is defined as $\mathbb{T} = \{x_j^{\mathbb{T}}\}_{j=1}^{n_{\mathbb{T}}}$, where $\{x_1^{\mathbb{T}}, \ldots, x_{n_{\mathbb{T}}}^{\mathbb{T}}\} \sim P(X_{\mathbb{T}})$. While in a Supervised Domain Adaptation setting, the target domain is also labeled, which we note $\mathbb{T} = \{(x_j^{\mathbb{T}}, y_j^{\mathbb{T}})\}_{j=1}^{n_{\mathbb{T}}}$, where $\{x_1^{\mathbb{T}}, \ldots, x_{n_{\mathbb{T}}}^{\mathbb{T}}\} \sim P(X_{\mathbb{T}})$ and $y_j^{\mathbb{T}} = f_{\mathbb{T}}(x_j^{\mathbb{T}})$.

In a UDA setting, we want to exploit knowledge from the labeled source domains to make learning possible on an unlabeled target domain $\mathbb{T}$. While in a SDA setting, we want to exploit knowledge from the labeled source domains and the labeled target domain to improve classification on an unknown and unusable part of $\mathbb{T}$. In most DA scenarios, it is presumed that the covariate shift assumption holds, that is, source domains and target domain share the same labeling function, $f = f_{\mathbb{S}_1} = \cdots = f_{\mathbb{S}_s} = f_{\mathbb{T}}$. We want our Domain Adaptation model to learn to estimate the labeling function f shared upon all domains, that is, the common learning task.

1.4.1.1 Domain Shifts

Domain Adaptation is useful to use when there is a distribution difference between source(s) and target domains. Here, we formally define the three possible distribution differences that can exist between a pair of domains using our mathematical notations.

In their 2012 paper, to unify the terms and definitions used for the various domain shifts that appear in Domain Adaptation literature and provide consistent terminology, (Moreno-Torres et al., 2012) proposed the formalization of three types of shifts: covariate shift, prior shift, and concept shift. More recently, (Kouw and Loog, 2019) reviewed those defined shifts and provided more precise and up-to-date definitions. Figure 1.24 illustrates each of the presented domain shifts.

Definition 6 (Covariate Shift between domains $\mathbb{D}_1$ and $\mathbb{D}_2$, Figure 1.24.a)

The data marginal distributions of the two domains are different, while their conditional distributions are equal, $P(X_{\mathbb{D}_1}) \neq P(X_{\mathbb{D}_2})$ and $P(Y_{\mathbb{D}_1}|X_{\mathbb{D}_1}) = P(Y_{\mathbb{D}_2}|X_{\mathbb{D}_2})$.

A simple cause of covariate shift between two domains would be a different feature representation from a domain to the other, that is, their feature spaces are different, $\mathcal{X}_{\mathbb{D}_1} \neq \mathcal{X}_{\mathbb{D}_2}$,

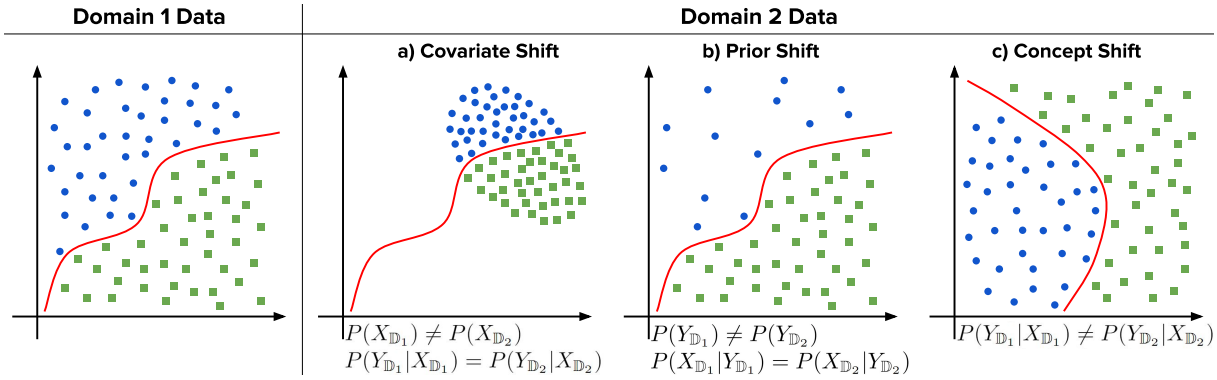


Figure 1.24: Illustration of the three kinds of domain shifts.

naturally leading to different data marginal distributions, $P(X_{\mathbb{D}_1}) \neq P(X_{\mathbb{D}_2})$. In a medical context, such situation can occur if features collected on patients from one hospital ($\mathbb{D}_1$) are collected differently in another hospital ($\mathbb{D}_2$), or if different features are collected between hospitals. Another possible cause of covariate shift between domains is when there exists a domain specific form of sample selection bias (Kouw and Loog, 2019). Sample selection bias between two domains can be seen as an altered probability for an instance to be sampled in one domain compared to the other domain. For example, in a medical context, covariate shift can occur between domains when representative patients of one domain are selected on different health features than the ones of another domain. In this case both domains share the same feature space ($\mathcal{X}_{\mathbb{D}_1} = \mathcal{X}_{\mathbb{D}_2}$) but there is a different sample selection bias between the domains, leading to different data marginal distributions ($P(X_{\mathbb{D}_1}) \neq P(X_{\mathbb{D}_2})$). Under covariate shift, a classifier trained on one domain might struggle when applied on another domain. This is a domain shift that is present in almost all DA applications and is widely studied in the literature (Kouw and Loog, 2019).

Definition 7 (Prior Shift between domains $\mathbb{D}_1$ and $\mathbb{D}_2$, Figure 1.24.b)

The label marginal distributions of the two domains are different, while their conditional distributions are equal, $P(Y_{\mathbb{D}_1}) \neq P(Y_{\mathbb{D}_2})$ and $P(X_{\mathbb{D}_1}|Y_{\mathbb{D}_1}) = P(X_{\mathbb{D}_2}|Y_{\mathbb{D}_2})$.

There exists a prior shift between two domains when the class balance is not the same in each domain. This is a common occurrence that can happen when data from similar domains are gathered differently (Redko et al., 2019), leading to different label marginal distributions between domains. For example, if we use medical data from a poor country as source domain, to transfer knowledge to a target domain with data from a rich country, it is very probable that the disease distribution will be widely different between the two domains ($P(Y_{\mathbb{D}_1}) \neq P(Y_{\mathbb{D}_2})$). This type of domain shift appears less often in the DA literature.

Definition 8 (Concept Shift between domains $\mathbb{D}_1$ and $\mathbb{D}_2$, Figure 1.24.c)

The conditional distributions of the two domains are different, $P(Y_{\mathbb{D}_1}|X_{\mathbb{D}_1}) \neq P(Y_{\mathbb{D}_2}|X_{\mathbb{D}_2})$.

There exists a concept shift between two domains when the decision boundary between

classes is not the same from one domain to another, meaning that the causal relation between features and labels is semantically different from one domain to another. Concept shift occurs between two domains when their joint distributions are defined on totally, or partially, different label spaces, $\mathcal{Y}_{\mathbb{D}_1} \neq \mathcal{Y}_{\mathbb{D}_2}$. Meaning that some classes are not shared or have a different semantic meaning between domains. Concept shift might occur between two domains if classes are semantically inaccurate, which might lead to slightly differently labeled data between them. In such a context, UDA is impossible, as it is not possible to know the target decision boundary if there is a concept shift and no labeled instances in the target domain (Kouw and Loog, 2019). Only SDA is possible in such situation, as source knowledge can be exploited and improve target prediction performance by using labeled instances in the target domain.

1.4.1.2 Negative Transfer

The goal of Domain Adaptation is to exploit knowledge from one or several source domain(s) to improve prediction quality on a target domain. But a common issue with DA is Negative Transfer (Day and Khoshgoftaar, 2017; Pan and Yang, 2010; Zhang et al., 2022). Negative transfer occurs in DA when transferring knowledge from a source domain to a target domain harms the learning performance on the target. Consequently, instead of improving the inference model performance, Negative Transfer leads to a decrease in prediction performance on the target domain.

Negative Transfer can arise due to several reasons (Zhang et al., 2022). A common reason a large dissimilarity between source and target domains. If a source domain is too different from the target domain in terms of data distribution, feature space, or underlying characteristics, the transferred knowledge may not be applicable or may even be misleading in the target domain. Another factor contributing to Negative Transfer is the presence of conflicting or irrelevant information in the source domain(s) that does not align with the target domain. In such cases, the model might mistakenly rely on irrelevant information and fail to generalize well on the target domain. Insufficient, biased, or bad quality source and/or target data can all lead to Negative Transfer. If the source data is limited or unrepresentative of the target domain, the transferred knowledge may not capture the necessary patterns required for effective and useful knowledge transfer.

Negative Transfer is an important issue in the transfer learning field, and more specifically in the Multi-Source Domain Adaptation field, where the use of several sources multiplies the risk of Negative Transfer. In SSDA, Negative Transfer can be limited by choosing a source domain as similar as possible to the target domain, which is harder in MSDA where we aim to use several source domains which are not all equivalently similar to the target domain. Limiting Negative Transfer is an important matter in the MSDA context, and more generally in the DA context, it should be addressed when designing new DA approaches.

1.4.2 Domain Adaptation Approaches

A naive way of performing Supervised Domain Adaptation is to use a Fine-Tuning approach, that is, to pre-train a model on source data and then fit the pre-trained model to target data. Which should improve inference quality on the target domain if source data is similar enough to the target domain. This is only possible when the target domain is labeled. In the case where the target domain is unlabeled, Domain Adaptation is employed to make it possible to train a model on the target domain, which would be impossible without labeled source data. Therefore, the goal of Unsupervised Domain Adaptation is usually different from the goal of Supervised Domain Adaptation, in the first case we want to make the learning task possible, while in the second we want to exploit knowledge from source domain(s) to maximize the learning performance and prediction quality on the target domain. Most, if not all, modern Domain Adaptation approaches aim at training a ML/DL model with a constraint to match the source domain(s) distribution(s) with the target domain distribution. That is, trying to find a domain invariant latent representation between source and target domains. Several methods are employed in the literature to match source and target distributions, the two main solutions are using a discrepancy measure to minimize the discrepancy between source and target domains, or using adversarial training to lose all domain discriminative information in the latent representation. Figure 1.25 shows our classification of modern DA approaches, this list of papers is far from exhaustive and is specifically oriented towards methods that are applicable to tabular data.

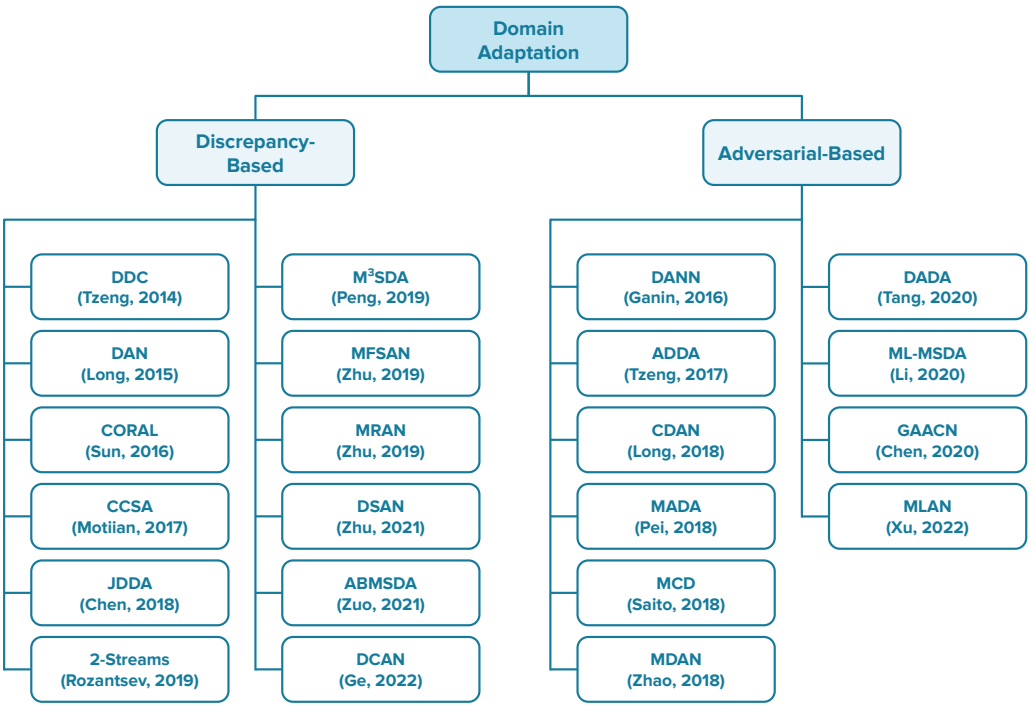


Figure 1.25: Machine/Deep Learning Domain Adaptation Approaches Classification

1.4.2.1 Adaptation Based on Domain Discrepancy Measures

A way of matching source and target distributions to perform Domain Adaptation is to minimize a discrepancy measure computed between source and target distributions. The discrepancy between two distributions refers to the difference or distance between them. There are several ways to measure the discrepancy between two distributions. A well-known discrepancy measure in Machine Learning is the Kullback-Leibler (KL) divergence, which is used in Variational Auto-Encoders to match the latent representation of the VAE with a normal distribution of parameters $\{\mu = 0, \sigma = 1\}$.

In DA, the most widely used discrepancy measure is the Maximum Mean Discrepancy (MMD), which has been introduced in (Borgwardt et al., 2006). The MMD measures the distance between two probability distributions by comparing their means in a high-dimensional feature space. In DA it is common practice to embed source and target data into a common high-dimensional feature space using a Neural Network f_θ. The MMD between source and target latent distributions is then computed as:

$$\text{MMD}(f_\theta, X_\mathbb{S}, X_\mathbb{T}) = \left|\left| \frac{1}{|X_\mathbb{S}|} \sum_{x_\mathbb{S} \in X_\mathbb{S}} f_\theta(x_\mathbb{S}) - \frac{1}{|X_\mathbb{T}|} \sum_{x_\mathbb{T} \in X_\mathbb{T}} f_\theta(x_\mathbb{T}) \right|\right|_\mathcal{F} \tag{1.30}$$

Where $||\cdot||_\mathcal{F}$ is the Frobenius norm. The MMD can be considered as a distance between the means of the two compared distributions in this feature space. A MMD value of 0 means that the two distributions are identical, and the larger the discrepancy value is, the more different the two compared distributions are. Therefore, a common way of performing DA in ML is to minimize the value of the MMD between source and target data as a regularization during training. This leads to a domain invariant latent space, that is, a latent representation that do not contain domain specific information. Using such regularization allows to create a domain invariant latent space, which provides a way to represent data from both the source and target domains in a common and shared representation space that is independent of the domains. By doing so, the domain specific information is separated from the domain invariant information, making it possible to transfer knowledge from the source domain(s) to the target domain. With this regularization, the inference model can focus on learning a latent space that contains only features that are relevant for the prediction task and independent of the original domains, and so, generalize better across different domains. The intuition is that source domain specific information might be irrelevant to the prediction task as it is not present in the target domain. Once the domain invariant latent space is learned, the trained model can be used to perform predictions on the target domain, which should lead to better inference results than without adaptation.

Among the first DA methods to use a discrepancy measure is Deep Domain Confusion (DDC), introduced in (Tzeng et al., 2014). The goal of DDC is to learn a representation that minimizes the distance between source and target distributions, a classifier can then be trained on source data and applied on target data with minimal loss in accuracy. Their method can be used in both unsupervised and supervised Domain Adaptation settings and is designed for Single-Source Domain Adaptation. They are the first to propose the use of the MMD measure as a regularization, which they call a domain confusion loss. They train

a Neural Network to find a common and shared latent representation of source and target data, minimizing the domain confusion term leads to finding model parameters that align both source and target latent distributions. The loss term used to fit DDC is expressed as:

$$\mathcal{L} = \mathcal{L}_{CE}(f_\theta(X_{lab}), y) + \lambda \text{MMD}^2(f_\theta, X_\mathbb{S}, X_\mathbb{T}) \tag{1.31}$$

With $\mathcal{L}_{CE}$ the Cross-Entropy loss for the classification task and X_{lab} the labeled data. In an unsupervised context $X_{lab} = X_\mathbb{S}$ and in a supervised context $X_{lab} = \{X_\mathbb{S}, X_\mathbb{T}\}$. The DDC loss combines the task-specific loss, which is a cross-entropy in the case of classification, and the domain confusion loss, that encourages the network to learn domain invariant features. The authors apply their method on images in the original paper, but DDC can be used to perform adaptation on tabular data as well. DDC is the simplest and most representative DL architecture for Domain Adaptation relying on a discrepancy measure in the literature. Figure 1.26 shows a representation of the DDC architecture.

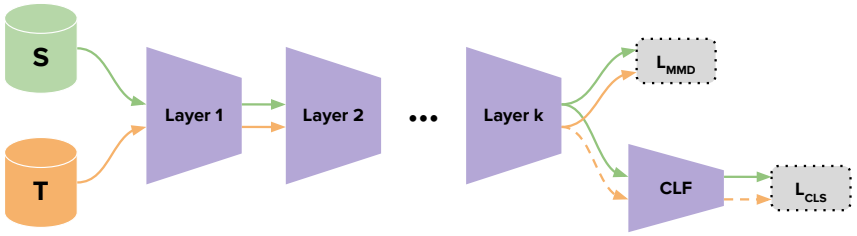


Figure 1.26: Representation of the DDC architecture, with the source domain in green and the target domain in orange. In a supervised context source and target data are used to compute the classification loss with known labels. In an unsupervised context, the classification loss is only computed on source data, with the only alignment being the MMD regularization.

The concept of using MMD as a regularization to align source and target latent representation is a basis of DA approaches. Most other approaches in this category are inspired by DDC and operate in a similar manner.

Authors of (Long et al., 2015) introduced Deep Adaptation Networks (DAN), a reference approach in discrepancy based DA. They propose an SSDA approach to perform adaptation on both unsupervised and supervised settings. They identify one of the main limitations of DDC as the fact that it only adapts a single layer of the NN. This limitation might prevent potentially valuable knowledge captured in other hidden layers from being transferred from the source to the target domain. To solve this limitation, instead of minimizing the discrepancy value on the last NN layer as in DDC, they minimize the multi kernel MMD (MK-MMD) value on the last k layers of the NN, which improves features transfer. Leading to a loss term expressed as:

$$\mathcal{L} = \mathcal{L}_{CLS} + \frac{1}{|k|} \sum_{i \in k} \lambda_i \text{MMD}^2(f_{\theta i}, X_\mathbb{S}, X_\mathbb{T}) \tag{1.32}$$

With k the NN layers for which latent representations are aligned between source and target domains, and $f_{\theta i}$ the latent representation at the i-th layer of the model.

In (Motiian et al., 2017), the authors propose a classification and contrastive semantic

alignment (CCSA) loss for single-source SDA. The model architecture is equivalent to the DDC architecture presented in figure 1.26. They define a three term loss that includes the task-specific loss computed on both source and target instances, a distribution matching MMD regularization computed on the last hidden layer, and the proposed class separation term.

$$\mathcal{L} = \mathcal{L}_{CLS} + \mathcal{L}_{MMD} + \lambda \mathcal{L}_S(f_\theta, X_\mathbb{S}, X_\mathbb{T}) \tag{1.33}$$

This class separation term $\mathcal{L}_S$ relies on labels to encourage class separation, that is, instances with different classes should be mapped as far apart as possible in the latent representation. A suitable metric that yields large values for two instances of different classes while yielding small values for two instances of the same class must be used. The authors do not precise the kind of metric they used for this term.

Instead of defining a NN architecture with shared layers between source and target domains, as it is common practice, (Rozantsev et al., 2019) proposed a two-stream architecture in which the model parameters between the two branches are regularized with an euclidean distance. They propose their method for both unsupervised and supervised SSDA. The loss term can be formalized as:

$$\mathcal{L} = \mathcal{L}_{CLS} + \mathcal{L}_{MMD} + \frac{1}{|k|} \sum_{i \in k} \lambda_i \left|\left| \alpha_i \theta_i^\mathbb{S} + \beta_i - \theta_i^\mathbb{T} \right|\right|_2^2 \tag{1.34}$$

With k the two-stream NN layers, $\theta_i^\mathbb{S}$ the parameters of the i-th layer in the source domain branch and $\theta_i^\mathbb{T}$ the parameters of the i-th layer in the target domain branch. Parameters of both branches are regularized in a way that ensures that latent representations between domains are linear transformations of each others, α and β are parameters which are learned during training.

In their paper, (Zhu et al., 2019b) proposed MFSAN, a discrepancy based adaptation method for unsupervised Multi-Source Domain Adaptation. The goal of their approach is to allow learning on a totally unlabeled target domain while relying on s similar source domains. They design a NN architecture with a shared feature extractor, the architecture then splits with one last hidden layer and one output layer for each source domain. They align the target domain with each source domain using an MMD measure on the last hidden layer and improve the alignment by regularizing the output layers with an L_1 term applied on target data output representations, ensuring the same output probabilities for a same target input instance. MFSAN is an important contribution in the DA literature as there exists few Multi-Source Domain Adaptation methods and it shows an innovative way of exploiting multiple source domains to allow learning on a totally unsupervised target domain. Figure 1.27 shows the MFSAN architecture with MMD, L_1 and task-specific loss terms.

The authors of (Zhu et al., 2019b) propose to use a Conditional MMD to improve supervised SSDA performance with an architecture similar to DDC. They identify that an important limitation of MMD based approaches is that they do not explicitly take account of classes when aligning domain distributions. They propose CMMD, to measure the discrepancy between the class-conditional distributions between source and target domains,

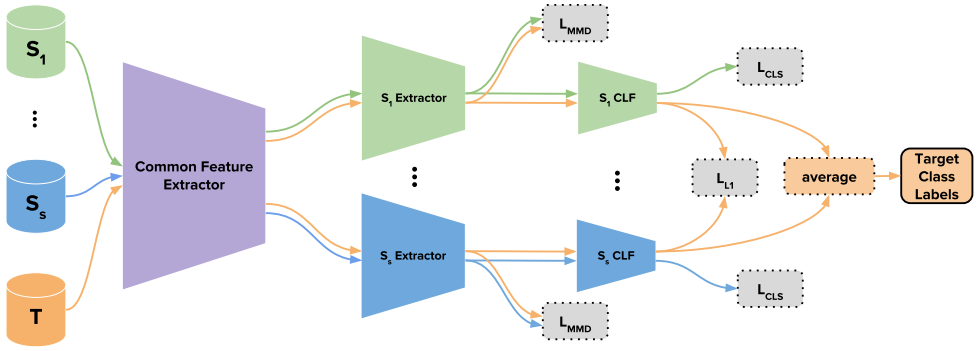


Figure 1.27: Representation of the MFSAN architecture, with s source domains in green to blue and the target domain in orange. The common feature extractor creates a shared latent representation from each domain, the architecture is then split with one feature extractor for each source domain, shared between the specific source domain and the target domain. The MMD regularization is applied between source and target data at this level. Then, the latent representation is fed through the specific classifier layer, the supervised task-specific loss is computed on source data, target data outputs are regularized with an L_1 loss to ensure transfer of coherent features on the target domain. Inference on target data are obtained as the average across all source domain specific classifiers.

defined as:

$$\text{CMMD}(f_\theta, X_\mathbb{S}, X_\mathbb{T}) = \frac{1}{c}\sum_{j=1}^{c}\left|\left|\frac{1}{|X_\mathbb{S}^j|}\sum_{x_\mathbb{S}^j \in X_\mathbb{S}^j} f_\theta(x_\mathbb{S}^j) - \frac{1}{|X_\mathbb{T}^j|}\sum_{x_\mathbb{T}^j \in X_\mathbb{T}^j} f_\theta(x_\mathbb{T}^j)\right|\right|_\mathcal{F} \tag{1.35}$$

With c the amount of classes and x^j an instance of class j. The method can also be applied in an unsupervised adaptation context by computing pseudo-labels of target instances before learning. CMMD has also been used in Deep Conditional Adaptation Network (DCAN) introduced by (Ge et al., 2022).

After introducing CMMD, the authors of (Zhu et al., 2021) propose LMMD. They introduce Deep Subdomain Adaptation Network (DSAN) with a similar architecture to DAN while using the newly proposed Local MMD as a discrepancy measure. LMMD is a direct extension of CMMD with the addition of domain and class specific weights w that are learned with the model parameters:

$$\text{LMMD}(f_\theta, X_\mathbb{S}, X_\mathbb{T}) = \frac{1}{c}\sum_{j=1}^{c}\left|\left|\frac{1}{|X_\mathbb{S}^j|}\sum_{x_\mathbb{S}^j \in X_\mathbb{S}^j} w_\mathbb{S}^j f_\theta(x_\mathbb{S}^j) - \frac{1}{|X_\mathbb{T}^j|}\sum_{x_\mathbb{T}^j \in X_\mathbb{T}^j} w_\mathbb{S}^j f_\theta(x_\mathbb{T}^j)\right|\right|_\mathcal{F} \tag{1.36}$$

This LMMD regularization improves domain alignment by locally aligning classes within aligned domains. Similarly to their previously proposed approach, DSAN can used for both supervised and unsupervised DA, in an unsupervised context pseudo-labels must be computed to be used in the computation of LMMD.

Another kind of domain discrepancy measure has been proposed in (Sun and Saenko, 2016; Sun et al., 2016), that is, the CORAL loss. They introduced the simple concept of min-

imizing the co-variance matrices between source and target latent representations to align their distributions. They define the CORAL loss as:

$$\mathcal{L}_{CORAL} = \frac{1}{4d^2} ||cov(X_{\mathbb{S}}) - cov(X_{\mathbb{T}})||_{\mathcal{F}}^2 \tag{1.37}$$

Where d is the number of features and $cov(X_{\mathbb{S}})$ and $cov(X_{\mathbb{T}})$ are the covariance matrices of source and target domains respectively. The CORAL loss is used as a regularization in parallel to the task-specific loss and must be applied to a high level feature representation, that is, to the output of a hidden layer in the NN. The CORAL loss integrated to Deep Neural Networks led to state-of-the-art results when it was proposed in 2016, with superior results compared to DDC and DAN relying on MMD based adaptation.

Two years later, (Chen et al., 2018) proposed Joint domain alignment and Discriminative feature learning for unsupervised deep Domain Adaptation (JDDA). JDDA uses the CORAL regularization to align source and target domains and proposes a new regularization to further improve the intra-class compactness and inter-class separability of instances in the shared latent representation of source and target instances. Their proposed loss term can be formalized as:

$$\mathcal{L} = \mathcal{L}_{CLS} + \mathcal{L}_{CORAL} + \mathcal{L}_{DIS} \tag{1.38}$$

Where $\mathcal{L}_{DIS}$ is a proposed regularization term that aims at providing a latent representation of source samples where instances of the same class are as close from each others as possible while samples from different classes are well separated. They propose to perform discriminative feature learning, with the instance based discriminative. The instance based discriminative loss between the latent representation of two source instances is defined as:

$$\mathcal{J}(h_1, h_2) = \begin{cases} \max(0, ||h_1 - h_2||_2)^2 & \text{if } y_1 = y_2 \\ \max(0, 100 - ||h_1 - h_2||_2)^2 & \text{otherwise} \end{cases} \tag{1.39}$$

Where $h_i \in \mathbb{R}^l$ is the latent representation of size l of the i-th source instance and y_i is the corresponding label. They report better experimental results compared to DDC, DAN, and CORAL, showing the interest of conditionally regularizing the latent space to cluster same-class instances and separate different-class ones in the latent representation.

Other methods have been proposed in the literature to further improve domain alignment using more advanced discrepancy measures. In Multi-Source Domain Adaptation a discrepancy measure that seems to be more adapted and pertinent than MMD is the Moment Distance (MD) such as defined in (Peng et al., 2019). The authors propose M^{3}SDA, a MSDA approach that matches multiple source domains to the target domain using the MD. They identify that a limitation of MMD is that it only compares the distance between the first moment of two distributions, that is, their mean. They propose MD to further increase the adaptation process by being able to match the k first moments of source and target latent representations. The MD between one source domain and the target domain is defined as:

$$\text{MD}(f_\theta, X_{\mathbb{S}}, X_{\mathbb{T}}) = \sum_{i=1}^{k} \left|\left| \frac{1}{|X_{\mathbb{S}}|} \sum_{x_{\mathbb{S}} \in X_{\mathbb{S}}} f_\theta(x_{\mathbb{S}})^i - \frac{1}{|X_{\mathbb{T}}|} \sum_{x_{\mathbb{T}} \in X_{\mathbb{T}}} f_\theta(x_{\mathbb{T}})^i \right|\right|_{\mathcal{F}} \tag{1.40}$$

Which gives a discrepancy measure between two distributions based on their k first moments. Original authors set $k = 2$, which ensures to match both the mean and variance of the source and target distributions. MD can be considered as an extension of MMD over the k first moments of the distributions to compare. It has been demonstrated in (Peng et al., 2019) that MD is more pertinent and leads to better learning performance than MMD in a multi-source adaptation context.

More recently, (Zuo et al., 2021) proposed ABMSDA, a MSDA approach that introduces a weighted version of MD (WMD) to avoid Negative Transfer by weighting each source domain depending on its contribution to the adaptation process. They simply define WMD as:

$$\text{WMD}(f_\theta, X_\mathbb{S}, X_\mathbb{T}) = \sum_{i=1}^{k} w_\mathbb{S} \left\| \frac{1}{|X_\mathbb{S}|} \sum_{x_\mathbb{S} \in X_\mathbb{S}} f_\theta(x_\mathbb{S})^i - \frac{1}{|X_\mathbb{T}|} \sum_{x_\mathbb{T} \in X_\mathbb{T}} f_\theta(x_\mathbb{T})^i \right\|_\mathcal{F} \tag{1.41}$$

With $w \in \mathbb{R}^s$, s the number of source domains, $w_\mathbb{S}$ is the weight that is associated to the source domain $\mathbb{S}$. Those weights are computed during training depending on the output of a domain classifier that is trained to discriminate the domain of each source and target instance. The output of the domain classifier is used as a domain similarity measure, with the intuition that source domains that are most similar to the target domain should be attributed a higher weight $w_\mathbb{S}$ in the WMD computation. This is a way of avoiding Negative Transfer during Multi-Source Domain Adaptation.

1.4.2.2 Adaptation Based on Adversarial Training

Another way of creating a domain invariant latent space in which relevant information from diverse domains is encoded in the same way is through adversarial learning. Indeed, a common method is to train a classifier to discriminate the original domain of an instance, based on its latent representation, while training the encoder to fool the discriminator. In that way, the discriminator improves in detecting from which domain the data originates from, while the encoder improves in generating a domain invariant latent space.

The first Domain Adaptation approach to rely on adversarial training to ensure a domain invariant latent representation is Domain-Adversarial Neural Network (DANN), proposed in (Ganin and Lempitsky, 2016; Ganin et al., 2017). DANN can be used in both unsupervised and supervised DA settings and is designed for Single-Source Domain Adaptation. It was first specifically designed for image classification, but has been applied to many different tasks and to tabular data with great success in multiple studies. The DANN architecture is composed of an encoder, a task-specific classifier and a domain classifier. The encoder is composed of several NN layers, either convolutional or fully-connected depending on the data to be processed, and is used to transform source and target data to a common and shared latent representation. The task-specific classifier is composed of one or more fully-connected layers and is used to train the model on the specific task to learn. In a supervised setting, the task-specific loss is computed on both source and target data, in an unsupervised setting the task-specific loss is computed on source data only. The domain classifier is trained to discriminate the domain from which the data originates from based on the latent representation. A gradient reversal layer is placed between the encoder and the domain

classifier to invert the domain classifier gradient during back-propagation. This inversion of the gradient makes it possible to simultaneously train the encoder to maximize the domain classifier loss while the domain classifier is trained to minimize its own loss. This adversarial process ensures that the learned latent representation is domain-invariant, while preserving as much task-specific relevant features that can be exploited by the task-specific classifier. This proposed gradient reversal has been widely used in the DA literature based on adversarial training. Formally, we note the encoder part of the architecture $E(\cdot)$, the task-specific classifier $C(\cdot)$, the domain classifier $D(\cdot)$, while the gradient reversal layer is GRL$(\cdot)$. The loss can be expressed as:

$$\mathcal{L} = \mathcal{L}_{CLS} + \text{GRL}(\mathcal{L}_{DOM}) \tag{1.42}$$

where

$$\mathcal{L}_{CLS} = \mathcal{L}_{CE}(C(E(X_{lab})), y) \tag{1.43}$$

and

$$\mathcal{L}_{DOM} = \mathcal{L}_{CE}(D(E(X_{\mathbb{S}})), 0) + \mathcal{L}_{CE}(D(E(X_{\mathbb{T}})), 1) \tag{1.44}$$

Figure 1.28 shows a schematic representation of the DANN architecture.

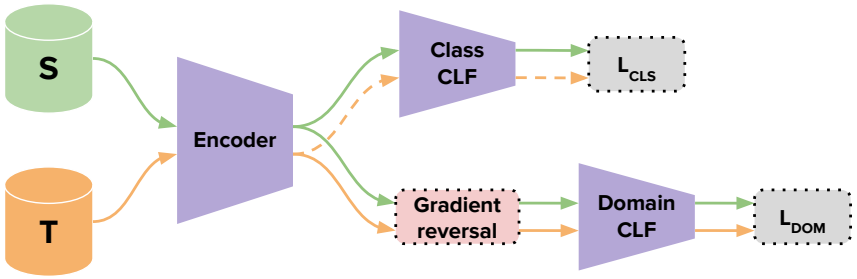


Figure 1.28: Representation of the DANN architecture. Source and target data are fed through a common encoder that transform the data to a shared latent space. The latent space is adversarially regularized through the domain classifier that aims to discriminate the original domain of the data. While the encoder is trained to fool the domain classifier, leading to a domain-invariant latent representation. Simultaneously, a task-specific classifier is trained to properly classify the data based on its latent representation, ensuring that relevant task-specific features are properly captured in the latent space and transferred from the source to the target domain.

In their paper, (Tzeng et al., 2017) proposed Adversarial Discriminative Domain Adaptation (ADDA), a framework to perform UDA in a single-source context. The first step of the approach is to train a model to perform the wanted task on source data, the model is composed of an encoder $E_{\mathbb{S}}(\cdot)$ and a task-specific classifier $C(\cdot)$. The model $C(E_{\mathbb{S}}(\cdot))$ is trained to minimize the task-specific loss on labeled source data. The second step is to train a target encoder $E_{\mathbb{T}}(\cdot)$ to transform target data to the same latent representation as the one built by $E_{\mathbb{S}}(\cdot)$ on source data. This is done by adversarially training the target encoder $E_{\mathbb{T}}(\cdot)$ against a domain classifier $D(\cdot)$, the domain classifier must identify if a latent representation originates from the source or the target domain, while the target encoder must fool the domain classifier. This ensures that the target encoder $E_{\mathbb{T}}(\cdot)$ learns a mapping from target data to the latent space created by the source encoder $E_{\mathbb{S}}(\cdot)$. The trained target encoder $E_{\mathbb{T}}(\cdot)$ can then

be used with the previously trained task-specific classifier $C(\cdot)$ to obtain target inference. This approach can only be used in an unsupervised and single-source setting, as adaptation is performed by mapping target latent representation towards the previously obtained source latent space using adversarial training.

In (Long et al., 2018), the authors introduced Conditional Domain Adversarial Network (CDAN), which relies on multi-linear conditioning to improve the transfer of domain-invariant and task-specific relevant features from source to target domain. CDAN is designed to perform both supervised and unsupervised DA, in a single-source context. In addition to a common adversarially trained domain discriminator as presented with the two previous methods, CDAN uses known labels from both source and target data as an input for the domain classifier, improving its capacity to dicsciminate between domains, and so, improving the domain-invariance of the constructed latent space. In an unsupervised context, pseudo-labels must be computed on target data in order to use CDAN. They also improve the regularization of the latent representation by not only regularizing the final output of the encoder as other methods, but the concatenation of multiple layers in the encoder, ensuring a deeper domain-invariance within the encoder hidden layers. They demonstrate that those simple improvements over DANN lead to consistently better adaptation results.

To improve adaptation results over DANN, (Pei et al., 2018) introduce Multi-Adversarial Domain Adaptation (MADA), an adversarial approach for single-source UDA. Their modification is similar to the one proposed in CDAN, as MADA aims to use the predicted labels obtained with the task-specific classifier to improve the domain discrimination, and so, to further align source and target latent representations. They base their method on the same architecture as DANN. Their proposal is to use as many domain discriminators $D^i(\cdot) \forall i \in [1, \dots, c]$ as there are classes. The i-th domain discriminator receives as input the latent representation of the data, weighted by the probability obtained by the task-specific classifier for the i-th class. Using previously defined formulations when presenting DANN, their modification leads to a loss of the form:

$$\mathcal{L} = \mathcal{L}_{CLS} + \text{GRL}(\mathcal{L}_{DOM}) \tag{1.45}$$

where

$$\mathcal{L}_{CLS} = \mathcal{L}_{CE}(C(E(X_{lab})), y) \tag{1.46}$$

and

$$\mathcal{L}_{DOM} = \sum_{i=1}^{c} \mathcal{L}_{CE}(D^i(\hat{y}^i E(X_{\mathbb{S}})), 0) + \mathcal{L}_{CE}(D^i(\hat{y}^i E(X_{\mathbb{T}})), 1) \tag{1.47}$$

where $\hat{y} = C(E(X_{\mathbb{S}})))$, and so, $\hat{y}^i$ is the predicted probability for the input data to be from the i-th class. Using c discriminators helps aligning classes within the latent space in addition to aligning source with target domains.

To further improve inference results, (Saito et al., 2018) introduced the Maximum Classifier Discrepancy (MCD) method, in which two task-specific classifiers are trained to learn different high-level features. The architecture of MCD is identical to DANN, with the difference that there is not only one task-specific classifier, but two. The model is trained following three steps:

1. In the first step, the model is trained as DANN, where the outputs of both task-specific classifiers are aggregated as the mean.

2. Then, the encoder is fixed and the two task-specific classifiers are trained to maximize their discrepancy. This is simply done by maximizing the L_1 distance between the two classifiers probability outputs.

3. Finally, the two task-specific classifiers are fixed and the encoder is trained to minimize the discrepancy between the two classifiers, that is, minimizing the L_1 distance between the two classifiers probability outputs.

Repeating those steps until convergence leads to a pair of task-specific classifiers that are as discriminative as possible while capturing information as different as possible from one another. This paradigm has been used in the DA literature to improve adaptation results, such as in (Peng et al., 2019) where they propose a second version of their M^{3}SDA approach that implements this paradigm and obtains better inference results on target data.

In (Tang and Jia, 2020), the authors propose Discriminative Adversarial Domain Adaptation (DADA), a SSDA for unsupervised adaptation. They propose a different adversarial approach compared to most other methods from this section. The DADA architecture is composed of an encoder that projects both source and target data to a shared latent space, and a classifier that outputs both the task-specific classification and the domain discrimination as a unique output vector. They propose an adversarial objective that encourages mutual inhibition between task-specific and domain prediction for all input instances.

Recently, (Chen and Hu, 2020) introduced Generative Attention Adversarial Classification Network (GAACN), an approach for single-source UDA specialized for image data. The GAACN architecture is composed of five main components, an encoder $E(\cdot)$, a task-specific classifier $C(\cdot)$, a generator $G(\cdot)$, an attention module $A(\cdot)$ and a discriminator $D(\cdot)$. The encoder and task-specific classifier are trained together to learn on both source and target data by minimizing a standard classification loss, such as $\mathcal{L}_{CLS} = \mathcal{L}_{CE}(C(E(X_{lab})), y)$. Domain alignment of the latent representation obtained after $E(\cdot)$ is performed adversarially through the remaining modules. The generator $G(\cdot)$ is trained to reconstruct images from the latent representation that are able to fool the discriminator $D(\cdot)$ in thinking that they originate from the source domain. The attention module $A(\cdot)$ is introduced in between $G(\cdot)$ and $D(\cdot)$ to focus attention of the discriminator on transferable regions that are considered as discriminative between source and target domains. Therefore, the latent representation obtained through $E(\cdot)$ is indirectly regularized to become domain-invariant through the adversarial training of $G(\cdot)$, $D(\cdot)$ and $A(\cdot)$.

Most Domain Adaptation approaches based on adversarial training are focused on Single-Source Domain Adaptation. In their paper, (Zhao et al., 2018a) proposed Multi-source Domain Adversarial Networks (MDAN), which is an adaptation of the DANN method to handle multiple source domains. The approach is quite simple, instead of training one domain classifier to discriminate between a unique source domain and the target domain as in DANN, MDAN trains s domain classifiers, one for each of the s source domains. The i-th domain classifier is trained to discriminate between data from the i-th source domain and data from

the target domain. Using source domain specific domain classifiers seems to improve results over using a single domain classifier that would be trained to discriminate the original domain of all data points.

Despite dealing with a multiple source domains context, MDAN learns a shared latent space over the s sources and the target domain, relying on adversarial training to regularize the latent space to be domain invariant. In the unsupervised Multi-Source Domain Adaptation context, some authors have tried to learn more than one domain invariant latent space. It is the case of (Li et al., 2020), that proposed Mutual Learning Network for Multiple-Source Domain Adaptation (ML-MSDA). Their architecture is composed of two branches:

1. The first branch learns a shared domain invariant latent space across all source domains and the target domain, which is similar to MDAN.

2. The second branch learns one domain invariant latent space between the pairs of each source domain with the target domain, similarly to MFSAN in discrepancy based approaches.

By jointly learning those multiple latent representations they obtain better experimental results than all previously presented MSDA approaches. Each domain discriminator in the model is adversarially trained to discriminate between data that originates from source domains and data from the target domain. One classifier is trained above each learned latent space, leading to $s+1$ classifiers, which are trained on labeled source data with a standard cross-entropy loss. Unlabeled target data is also used to train classifiers with an unsupervised entropy loss, that is, a loss that maximizes the confidence of the model on the classification of target instances. Finally, in order to align classifiers predictions on target data, they use a symmetric version of the KL divergence to minimize the discrepancy between the classifier of the first branch and each classifier of the second branch.

More recently, (Xu et al., 2022) extended the work of (Li et al., 2020) by proposing a Mutual Learning based Alignment Network (MLAN). They propose a simplified architecture compared to ML-MSDA, they removed the KL based regularization and unsupervised cross-entropy over target predictions. The architecture is composed of the joint alignment branch, which is where the shared latent space across all source and target domains is learned, and the separate alignment branch, where s latent spaces are learned, one for each pair between the target domain and each source domain. Their contribution that differentiates the most their approach MLAN from ML-MSDA is their proposed mutual learning module, that is used to train the entire model. In the training of MLAN, the process alternates between adversarially training each pair of classifier and domain discriminator of the model, and training the model with the proposed mutual learning process. The adversarial training of the model ensures that each latent space is domain invariant and as discriminative as possible on the source and target domains. The mutual learning module is used so that the two branches complement each other for better results on target data. This module minimizes the combination of two loss terms:

$$\mathcal{L}_{MUT} = \mathcal{L}_{CAT} + \lambda \mathcal{L}_{LOG} \tag{1.48}$$

Where $\mathcal{L}_{CAT}$ is the loss for categorical mutual learning, $\mathcal{L}_{LOG}$ is for logits mutual learning and λ is an hyper-parameter used to balance the two terms. The categorical mutual learning assigns pseudo-labels to target instances using both the classifiers outputs and a K-means clustering algorithm applied on latent representations, pseudo-labels are noted $\hat{y}$.

$$\mathcal{L}_{CAT} = \mathcal{L}_{CE}(p_{jnt}, \hat{y}) + \sum_{i=1}^{s} \mathcal{L}_{CE}(p_i, \hat{y}) \tag{1.49}$$

Where p_{jnt} is the prediction output of the classifier of the joint alignment branch and p_i is the prediction output of the i-th classifier of the separate alignment branch. The logits mutual learning loss is used to counter the negative effects that improperly assigned pseudo-labels could have on the training, its loss term $\mathcal{L}_{LOG}$ is defined as a symmetric KL divergence between p_{jnt} and p_{sep}. With p_{jnt} the prediction output of the joint alignment branch and p_{sep} the mean prediction output of the separate alignment branch. This logits mutual learning loss seems to be closely related to the KL based regularization used in ML-MSDA. With MLAN, (Xu et al., 2022) currently obtains state-of-the-art results compared to other multi-source and single-source models of the Domain Adaptation literature.

1.4.3 Discussion

In all previously presented approaches, transfer from source domain(s) to the target domain is realized through learning one or several invariant latent spaces, in which domain specific information is lost, and discriminative information is maximized, maximizing target domain learning performance. This is done in two ways, either by minimizing a discrepancy measure to align source and target latent representations, or using an adversarial approach with a domain discriminator that must be fooled, reducing domain specific information in the latent space in return.

Our exploration of the Domain Adaptation field highlights that most innovations in this field are focused on two categories: proposing a new DA loss, and/or, proposing a new neural architecture. Most innovations in the proposal of new DA losses is either to use a discrepancy measure that had not been used for this goal beforehand, or proposing a slightly modified version of a well-known discrepancy measure to better match an application specific needs. While most innovations in the proposal of a new neural architecture aim at creating an architecture that learns interesting latent spaces, with the goal of transferring information that could not be transferred with simpler architectures.

As can be seen from our exposition of a part of the DA literature, most papers focus on Unsupervised Domain Adaptation, where the goal of the adaptation is to make the learning task on an unlabeled target domain possible, with almost no papers interested in Supervised Domain Adaptation. We believe that many real-world application scenarios fall under the category of SDA, where the target domain is labeled and there exists other labeled datasets related to the target that should be exploited in order to maximize performance on the target domain. In all those learning scenarios, we think that employing more advanced SDA approaches, designed to carefully exploit source knowledge while avoiding Negative Transfer, should lead to much superior prediction results compared to the naive application of

a simple fine-tuning approach. Sadly, there is currently a lack of pertinent methods in the literature for Supervised Domain Adaptation, despite the existing research issue.

Chapter 2

Dealing with Attribute Noise

Contents

In this chapter, we are interested in finding better ways to preprocess data with missing and erroneous values, and improving the learning of Neural Networks (NNs) on completed data. The first section is an introduction about attribute noise, we present the problems we aim to solve and introduce our main contributions. We then briefly present the related works that are most relevant to this chapter work. We describe our proposals in the attribute noise field in section 2.3. Section 2.4 shows the performed experiments to evaluate our proposed

approaches. Finally, we give a global conclusion of our work on attribute noise and discuss our main contributions in section 2.5.

2.1 Context and Introduction

Medical studies are particularly subject to outliers, erroneous, meaningless, or missing values. In most real-life studies, not solely limited to the medical field, the problem of incomplete and erroneous data is unavoidable. Those corruptions can occur at any data collection step. They can be a natural part of the data (patient noncompliance, irrelevant measurement, etc.) or appear from corruption during a later data manipulation phase (Yang et al., 2004). Regardless of their origin, those corruptions are referred to as "noise" in the following work. Noise negatively impacts the interpretation of the data, whether for a manual data analysis or training an inference model on the data. The goal of a Machine Learning (ML) model is to learn patterns and generalizations from training data and use the acquired knowledge to perform predictions on unseen test data later on. Thus, the quality of training data on which a model is based is of critical importance, the less noisy the data is, the better results we can expect from the model.

Noise can be divided into two categories, namely, class noise and attribute noise (Zhu and Wu, 2004). Class noise corresponds to noise in the labels, for example, when data points are labeled with the wrong class, etc. Attribute noise, on the other hand, corresponds to erroneous and missing values in the attribute data, that is, the features of the instances. Attribute noise tends to occur more often than class noise in real-world data (Van Hulse et al., 2007; Yang et al., 2004; Zhu and Wu, 2004). Despite this fact, compared to class noise, very limited attention has been given to attribute noise (Zhu and Wu, 2004). We focused our work on attribute noise to maximize prediction performance while trying to compensate for a lack of appropriate methods within the literature.

Two types of variances occur when multiply imputing an incomplete dataset: the within-variance and the between-variance. The within-variance corresponds to the variance within each imputed dataset. The between-variance corresponds to the variance between all imputed datasets. In real scenarios, it is usually not possible to know the within-variance, as most imputation methods estimate fixed values in place of missing ones, without outputting any probability measure. However, the between-variance can easily be computed between a set of completed datasets. In the following, we refer to the between-variance as imputation uncertainty. As in our background chapter (Chapter 1), we differentiate imputation methods (Definition 5), which aim to impute missing values in a dataset, and imputation frameworks (Definition 4) such as Single-Imputation (SI) or Multiple-Imputation (MI), which rely on any imputation method to train a ML model when dealing with missing data.

When dealing with incomplete data, a naive and overly used framework is SI (Rubin, 2004). That is, to arbitrarily choose an imputation method, use it to impute the dataset, and treat the completed dataset as the new real dataset to perform any kind of future analysis. When training a Neural Network or similarly strong learners on completed data, they are usually able to generalize enough to limit the bias occurring due to imputation uncertainty. This leads to good enough results so that one does not look for better ways to deal with missing

values. It has even been shown that, when using strong inference models, almost any imputation asymptotically leads to optimal prediction (Le Morvan et al., 2021). This is probably one of the main reasons why accounting for imputation uncertainty has not been widely researched. However, strong models might reach good results in such situations, but they are biased by imputation uncertainty, in this chapter we show that accounting for this uncertainty during their training phase helps to reach even better prediction results.

2.1.1 Problem Formulation

In a supervised learning context, the goal of learning a classification task is to learn a mapping function that maps input features to the corresponding set of output labels. In most real-world situations, the input data might contain missing and/or erroneous values, which could negatively impact the performance of the learning algorithm. In the following we describe in a formal way the problem of classification in a supervised learning context on a dataset with corrupted instances, that is, a dataset with missing and/or erroneous attribute values.

We note $\mathcal{X} \in \mathbb{R}^d$ the input attribute space and $\mathcal{Y} \in \{1, \ldots, c\}$ the multi-class output label space, with d the number of features and c of classes. Let $X = \{x_i \in \mathcal{X}\}_{i=1}^n$ be the ground-truth instances, with n the number of instances in the dataset. We assume the associated labels are constructed given the labeling function $f : \mathcal{X} \to \mathcal{Y}$ mapping from feature space to label space. Thus, we note $Y = \{f(x_i)\}_{i=1}^n$ the label sample associated to X. Ideally, we would be given the pair $\{X, Y\}$ to learn the mapping function f by training an inference model to extract and exploit non linear correlations in the clean data X, to predict the right labels Y.

However, in a real-life scenario, it is very common for the input data to be corrupted, with missing and/or erroneous attribute values. In such context we note $\tilde{X} = \{\tilde{x}_i \in \mathcal{X}\}_{i=1}^n$ the corrupted version of X. In this context, the clean data X is not available, only its corrected version $\tilde{X}$ can be used to perform learning and prediction. We note $M \in \{0, 1\}^{n \times d}$ the boolean missing values indicator matrix, where each element is either 0 or 1, and indicates whether the corresponding element in $\tilde{X}$ is missing or not. Unlike for missing values, positions of erroneous values are not known. Therefore, we are given the set $\{\tilde{X}, M, Y\}$ with the goal of learning the labeling function f which associates Y to the ground-truth instances X. The learned mapping function should be able to make accurate predictions for instances with corrupted attribute values as for clean ones.

From this problem formalization there are two possible ways of learning the mapping function f. The first solution is to use a robust learner f_θ, that is, minimizing an adequate loss function between the predictions of the model over the corrupted dataset and the true known labels $\mathcal{L}(f_\theta(\tilde{X}), Y)$. This solution relies on the sole capacity of the model to learn to handle and ignore missing and erroneous values during its training, which is limited and even impossible with most ML models. The second solution is to perform data correction before training an inference model, that is, to estimate plausible values in place of missing or erroneous ones. A preprocessing correction method is used to find an approximation of X from $\tilde{X}$, we note the corrected data instances $\mathring{X}$. Once data are corrected, the set $\{\mathring{X}, M, Y\}$ is used to train an inference model to learn the labeling function f. This facilitates the training

of the learning model f_θ which is relieved of the burden of dealing with corrupted attribute values in addition to learning the classification task.

In this chapter, we focus on the second solution, we aim at designing a preprocessing method to correct the attribute noise in the data, and finding new ways to maximize NN learning performance in such a context.

2.1.2 Our Contributions in the Attribute Noise Field

Our first proposal in this chapter is a new attribute noise correction method based on Auto-Encoders (AEs): data Denoising and Imputation in One Step (DIOS).

- We propose, to the best of our knowledge, the first method in the ML literature for handling attribute noise in tabular data in its entirety (i.e. erroneous values and missing values), in one preprocessing step.
- Our proposed method is able to impute missing values and correct erroneous ones by learning on incomplete and noisy data, requiring no complete or clean instance in the dataset.
- We conduct extensive comparative experiments and show that our proposed method is able to compete with, and exceed, other state-of-the-art methods in both erroneous values correction and data imputation tasks, on both benchmark and real-world medical mixed-type tabular data.

Our second main contribution in this chapter is the proposal of two new imputation frameworks that can be employed to train Neural Networks on imputed datasets while accounting for imputation uncertainty to reduce the natural bias occurring when training on completed datasets: Single-Hotpatching (S-HOT) and Multiple-Hotpatching (M-HOT).

- We propose Single-Hotpatching (S-HOT), a framework that aims at training a unique Neural Network on multiply imputed data by taking account of the variance between imputations to enhance the model generalization capacity.
- We propose Multiple-Hotpatching (M-HOT), an extension of the previous framework that trains an ensemble of Neural Networks while taking account of imputation uncertainty to reach extremely good prediction results, at the expense of a higher computational cost.
- We conduct extensive experiments to compare our proposed S-HOT and M-HOT with the existing SI and MI frameworks, and perform a statistical analysis to assess the obtained results, that shows that S-HOT and M-HOT compete against, and even outperform, existing imputation frameworks.

Our work lead to several scientific papers published in national and international conferences, and an ongoing submission in a journal:

- **T. Ranvier**, H. Elghazel, E. Coquery, K. Benabdeslem. *DIOS: data Denoising and Imputation in One Step*. **IJDSA**: International Journal of Data Science and Analytics, **(Under Review)**.

- **T. Ranvier**, H. Elghazel, E. Coquery, K. Benabdeslem. *Accounting for Imputation Uncertainty During Neural Network Training*. DOI: 10.1007/978-3-031-39831-5_24. **DaWaK 2023**, 28-30 august 2023, Penang, Malaysia.

  The complete results, supplementary material and source code used to conduct the experiments of this paper are available at the following GitHub repository[1].

- **T. Ranvier**, E. Coquery, H. Elghazel, K. Benabdeslem. *Considération de l'Incertitude d'Imputation pour l'Apprentissage des Réseaux de Neurones*. **SFC 2023**: Société Francophone de Classification, 6-7 juillet 2023, Strasbourg, France.

  This paper is a French version of the paper published at DaWaK 2023.

- **T. Ranvier**, H. Elghazel, E. Coquery, K. Benabdeslem. *Autoencoder-based Attribute Noise Handling Method for Medical Data*. DOI: 10.1007/978-981-99-1645-0_18. **ICONIP 2022**, 23-26 november 2022, New Delhi, India.

  The complete source code used to conduct the experiments is available at the following github repository[2].

2.2 Related Works

In this section, we review approaches and concepts of the imputation and correction literature that are related to our work and important for the comprehension of this chapter.

2.2.1 Imputation Frameworks

Such as previously defined with definitions 4 and 5, we differentiate imputation frameworks from imputation methods. We consider that an imputation framework is a precise procedure that makes use of an imputation method to impute missing values and exploit imputations in a formalized way. Whereas an imputation method is an algorithm that outputs a unique imputation value for each missing value in an incomplete dataset.

To the best of our knowledge, there are currently only two imputation frameworks that have ever been proposed in the literature: Single-Imputation (SI) and Multiple-Imputation (MI), which have been previously reviewed in section 1.2.3.1.

SI is probably the most commonly used framework for handling missing values in practice. It simply consists in choosing an imputation method, applying it to the incomplete dataset $\tilde{X}$, which yields a completed dataset $\mathring{X}$ where missing values have been assigned with new

[1] https://github.com/ThomasRanvier/Accounting_for_Imputation_Uncertainty_During_Neural_Network_Training

[2] https://github.com/ThomasRanvier/Autoencoder-based_Attribute_Noise_Handling_Method_for_Medical_Data

plausible values, and using $\mathring{X}$ as one would with any complete dataset (Buuren, 2021; Rubin, 1987). Performing imputation only once implies that the imputed values become the new truth for any future analysis, which is problematic as it can lead to biased inferences, as noted in (Buuren, 2021; Rubin, 1987). SI does not account for the uncertainty of the obtained imputations, leading to biased inference models that lack the ability to generalize. However, it produces a single imputed dataset that can be treated like any complete dataset, making it very simple and convenient to use and integrate into any existing pipeline.

In order to overcome the limitations of SI, (Rubin, 1987) proposed Multiple-Imputation, where missing values are imputed multiple times, leading to a set of plausible values representing a distribution of possibilities, representing the uncertainty about the right value to impute. A learner is trained on each version of the imputed dataset and final inference results are pooled in an ensemble manner. Compared to SI, MI offers the advantage of representing each missing value by a sample of plausible imputation values. This results in pooled inference results, that reflect the uncertainty level associated with each missing value, reducing bias in the pooled results from the ensemble of learners compared to those of each learner taken independently. Despite being an old and well-known imputation framework, MI is still not widely used in practice, with many scientists and users of imputation methods still relying on SI because of its simplicity of usage.

In the following work, we try to address main issues of both SI and MI frameworks in order to propose a new alternative imputation framework that can be used to train a unique learner, such as in SI, while conserving the advantage of taking account of the imputation uncertainty of MI. We also propose a second better performing imputation framework that trains multiple inference models in an ensemble manner to reach the best possible inference results at the cost of a higher computational effort.

2.2.2 Missing Values Imputation Methods

There are two main ways of dealing with missing values to train an inference model. Using a ML model that is able to handle missing values in its input, or artificially imputing missing values with plausible ones in a preprocessing step before training any standard ML model on the completed data. ML models that can handle missing values without explicitly imputing them can be used to naturally handle missing values without requiring imputation, such an approach has the advantage of handling missing data while learning the prediction task in one step. Few ML models have the capacity of naturally handling missing values without requiring prior imputation. In practice, imputation is the most common and by far the best performing way of dealing with missing values.

Deep Learning (DL) based imputation methods rely on the abstraction capacity of NNs to replace missing values with plausible ones. Most DL based imputation methods are based on Auto-Encoders, as this kind of NN is well suited to reconstruct data and can be used to construct a complete version of an incomplete dataset. As presented in section 1.2.3.4, several ways have been explored in the literature to impute missing values using the reconstruction power of AEs. Using standard AEs for imputing missing values is quite limited, most well performing AE based imputation methods rely on Denoising Auto-Encoders or Varia-

tional Auto-Encoders. Denoising Auto-Encoders (DAEs) have the advantage of improving the generalization capacity over standard AEs, while Variational Auto-Encoders (VAEs) have a stronger generative potential. In this work, we propose a different way of imputing missing values using DAEs, but instead of imputing missing values by reconstructing the input data, we train the model to reconstruct a complete version of the data from pure noise. This is a way of using DAEs as generative models. In this way, our method has the advantage of not being limited to imputing missing values, unlike all other DL based imputation methods. It can also correct erroneous values, making it the first attribute noise handling method in the ML literature.

The most similar DL based imputation method to ours in the literature is probably MIDA, Multiple-Imputation using DAEs, proposed in (Gondara and Wang, 2018). They use the advantage of DAEs over standard AEs to increase dimensionality of the latent representation above the size of the input data, leading to enhanced generalization capacity. The architecture of MIDA is a Denoising Auto-Encoder composed of 5 hidden layers, with respective dimensions: $d+\alpha$, $d+2\alpha$, $d+3\alpha$, $d+2\alpha$ and $d+\alpha$, with α a hyper-parameter that determines the number of neurons to add at each layer, set at 7 by default. Training MIDA requires a complete dataset, therefore, the first step of the algorithm is to naively impute missing values using a Substitution approach. Once missing values have been naively replaced, MIDA is trained to reconstruct the input while applying a random dropout operation with a dropout rate of 50%. This means that, at each training epoch, half the input values are masked to the model, and MIDA must learn to reconstruct the whole data from the partial input. The process is iteratively repeated until convergence of the model. The authors compare their experimental results with those obtained using the MICE algorithm on benchmark datasets, evaluating the performance using the Root-Mean-Square Error (RMSE) metric. The findings indicate that MIDA generally achieves better results in terms of Root-Mean-Square Error (RMSE) when compared to the MICE algorithm. However, it is worth noting that the comparison solely focuses on the performance of MIDA against the MICE imputation method. The study does not assess whether the imputations made by MIDA lead to improved inference results, which is a crucial aspect in the context of ML.

In our work, we propose an attribute noise handling approach based on DAEs, similarly to MIDA. A notable difference of our approach with MIDA and other AE based imputation methods is that we train our model to reconstruct a complete version of the data from pure noise. By setting a loss term that computes the error on observed values only, we train our method to reconstruct the corrupted data without requiring previous naive imputation, making it straight forward to apply our method for imputation. In our experiments, we compare our attribute noise correction method on imputation tasks against well-known and high-performing imputation approaches of most categories presented in section 1.2.3.2, on both benchmark and real-world experimental settings. We assess the imputations generated by the different methods using distance metrics, such as RMSE, to measure their proximity to the ground-truth. Additionally, and arguably more importantly, we evaluate the inference results obtained from our imputations using standard classification metrics to ensure that our approach leads to improved predictions.

2.2.3 Image Restoration

Image restoration is an active area of research that encompasses various applications, such as image completion, denoising, super-resolution, enhancement, etc. State-of-the-art approaches to image reconstruction often rely on deep Convolutional Neural Networks (CNNs) (Qin et al., 2021), as this kind of NN is especially adapted to deal with images. In image restoration, CNNs are usually trained on large image datasets to solve a specific image correction task. The training phase of a deep CNNs to learn a specific task is time-consuming and requires a significant amount of computing power and training images.

In response to those issues, an innovative and interesting approach has been proposed by (Ulyanov et al., 2020), called "Deep Image Prior". The method uses untrained CNNs to reconstruct images by fitting the untrained generator model to a single corrupted image. The CNN weights are randomly initialized and fitted to the image, and the network is given random noise as input. During training, the CNN learns to approximate the image from the noise and proceeds through a series of steps. The model output ultimately converges to the corrupted image, passing through a phase during which the generated image is of better quality than the original corrupted one, which is the goal.

This method is based on the simple observation that, in images, it takes more training iterations for a Deep Neural Network (DNN) to learn to reconstruct noise than to learn to reconstruct an image. It is therefore possible to obtain a reconstructed image cleaner than the original when training a model to reconstruct a corrupted image, as the model will be able to reconstruct the image content more easily than the parasitic noise during training. The key idea behind "Deep Image Prior" is to use a randomly initialized DNN f_θ, feed it pure noise while training it to reconstruct the corrupted image, without requiring any other training data than the original corrupted image. The algorithm starts by initializing the weights of the DNN randomly, and then iteratively updates the weights to minimize a loss function that simply measures the difference between the restored image and the corrupted image. The loss function to minimize in the original paper is an MSE defined as: $\mathcal{L} = ||f_\theta(Z) - \tilde{X}||^2$, where $\tilde{X} \in \mathbb{R}^{3\times h\times w}$ is the corrupted image to restore, and $Z \in \mathbb{R}^{3\times h\times w}$ is the input noise, usually defined with identical dimensions to $\tilde{X}$. At each iteration, the algorithm uses the current weights of the network to produce an estimate of the restored image, and then updates the weights to improve this estimate. During training, the similarity of the reconstructed image gets closer to the corrupted one, until total convergence, in order to obtain a corrected image the training process must be interrupted at the right iteration, this stopping criterion is manually defined. As this method does not require any pre-training, it is possible to apply it to any image restoration task. For example, to perform an inpainting task the loss function is redefined as $\mathcal{L} = ||(f_\theta(Z) - \tilde{X}) \odot M||^2$, where $\odot$ is the Hadamard product, and $M \in \{0,1\}^{3\times h\times w}$ is the binary missingness indicator matrix corresponding to missing pixels in the corrupted image.

This approach has shown promising results and competes with other state-of-the-art methods for standard image processing problems, such as denoising, inpainting, super-resolution, and detail enhancement. In the following work, we are inspired by this approach to develop and propose a similar method that can be applied to tabular data for perform-

ing both erroneous values correction and missing values imputation simultaneously, that is, correcting attribute noise in tabular data.

2.2.4 Erroneous Values Correction Methods

As stated in Chapter 1, dealing with erroneous values in tabular data is usually performed in three different ways: using robust learners, removing noisy instances detected with a filtering method, or correcting erroneous values with a polishing approach. In this work we focus on designing a method that is able to correct erroneous values before learning the target prediction task.

The two possible approaches to correct erroneous values in a preprocessing step are filtering and polishing. While filtering approaches aim to detect and delete instances that contain too much erroneous values, our proposed correction method aims to correct those erroneous values, preserving all instances in the dataset and limiting information loss. In that way, our proposed correction method can be considered as a new polishing method, with the advantage of also being able to impute missing values.

The most similar approach from our correction method in the erroneous values correction literature is the Polishing method proposed in (Teng, 2000, 2003, 2004). As described more precisely in 1.3.2, the Polishing method consists in exploiting the correlations between the attributes and labels of the data to identify noisy instances and predict plausible corrected values for noisy ones. The first step of the algorithm relies on a Filtering algorithm to detect instances that contain noise and separate them from clean ones, this is the identification phase. Then, predictive models are trained on non-noisy instances to predict plausible values for noisy instances. Attribute values of noisy instances are then updated based on empirical thresholds and conditions that must be manually defined. The Polishing method relies on a filtering method to detect instances needing correction, while our proposed correction method naturally polishes outlier values more drastically than non-noisy values, making it simpler to apply. The main limitation of the Polishing method is that rules must be manually defined empirically to apply modifications to noisy instances. This makes it hard to apply the Polishing method in many scenarios. With our proposed method, we found that we can apply the method to the entire dataset and train inference models on the raw output data to obtain better predictions.

In our experiments we compare our method to two versions of the Polishing method described in (Teng, 2000, 2003, 2004). The first one relies on standard Classification Filtering, proposed in (Gamberger et al., 1999, 2000), while the second relies on the PANDA algorithm, described in (Van Hulse et al., 2007).

2.2.5 Simulating Corruptions in a Tabular Dataset

To run imputation and error correction experiments, it is common practice to artificially corrupt a clean benchmark dataset and correct it using various methods to compare obtained correction results with the clean dataset. In this section, we review how to simulate the three theoretical mechanisms that cause missing values, Missing Completely At Random (MCAR), Missing At Random (MAR) and Missing Not At Random (MNAR), in order to artificially insert missing values in a dataset. We also define the process we follow to insert erroneous values in a dataset. Artificially inserting both missing and erroneous values into a dataset leads to simulating attribute noise in its entirety.

2.2.5.1 Artificially Inserting Missing Values to a Dataset

Introducing missing values to a dataset can be based on three mechanisms, such as described in (Buuren, 2021). We use the term "masked" to refer to the artificially added missing values.

To generate MCAR missing values, a straightforward method can be used: each value is randomly masked or not, based on a uniform distribution and a predefined threshold, corresponding to the missing rate. A way to introduce MAR missingness, is to use a randomly initialized logistic model, train it on a random subset of features that do not contain missing values, and use its output to either mask or not mask values in the remaining features. In our work we introduce MAR missing values following the implementation of (Muzellec et al., 2020). To generate MNAR missingness, a similar process than for MAR can be used. Where a logistic model is trained on a subset of features to mask random values of other features, and random values of the training subset are then masked with an MCAR mechanism, leading to missing values that were masked based on values that are also potentially masked.

2.2.5.2 Artificially Inserting Erroneous Values to a Dataset

Artificially introducing erroneous values in a tabular dataset to simulate natural erroneous values mechanisms and patterns has been largely less researched than introducing missing values. In their work, (Zhu and Wu, 2004) propose to introduce erroneous values in a tabular dataset by replacing a certain amount of values with a random value drawn from a uniform distribution. The amount of values to replace is defined as a noise level, for example, a noise level of 10% will assign random values to 10% of the dataset values. In the following, we artificially introduce erroneous values to tabular datasets in this way.

2.2.6 On the Potential Bias of Rounding Categorical Missing Values Imputations

When filling-in missing values in tabular data, the resulting predicted values are typically continuous. For quantitative features, the predicted continuous imputation value can be used as is, but in the case of categorical features such continuous imputation might lead to unrealistic values. To address this issue, various methods have been developed, the most commonly used is rounding the continuous imputation value to the closest observed value in the dataset (Horton et al., 2003). In their paper, (Horton et al., 2003) showed that such rounding approach for categorical features can introduce bias to the data, whereas leaving the imputed continuous value as is avoids this bias.

When drawing imputation values from a continuous distribution for a categorical feature, a standard practice is to round the continuous output to the closest observed value in the corresponding feature. As most ML models are trained to learn a continuous distribution from a discrete data sample, such practice is very common when applying ML and DL imputation methods. In (Horton et al., 2003), authors show that for categorical features, rounding the imputation continuous value to the closest known one leads to introducing bias to the data. They show that this bias phenomenon happens more when rounding imputations for binary features and categorical features with few categories. Indeed, for discrete features with many different possible values, the rounding impact is lower as rounded values are closer from original predicted values.

In order to evaluate the pertinence of rounding or not rounding imputation values in categorical features we performed a very simple empirical experiment. We choose a simple dataset containing 690 instances and 14 features, with 8 categorical features and 6 quantitative ones. We call this dataset STATLOG, we detail it more precisely in section 2.4.1. We artificially insert 50% missing values following a simulated MNAR mechanism such as described in the previous section. We perform missing values imputation using the MISSFOREST method, we keep the raw results as the not rounded completed dataset (No Rounding) and apply rounding to the closest known values for all categorical features to obtain the corresponding rounded completed dataset (Rounding). As we are interested in maximizing inference results on imputed data we evaluate the pertinence of rounding or not rounding imputation values on an inference task.

We evaluate the quality of both imputations with and without rounding based on various ML inference models: a Multi-Layer Perceptron (MLP), a KNN classifier and a Decision Tree. Inference results are evaluated with a 5-fold cross validation and stochastic models are trained 25 times to mitigate the randomness impact on final results. All models are instantiated with identical hyper-parameters for each run. Table 2.1 shows the obtained results.

As can be seen in those results, both the MLP and Decision Tree models obtain better results for all metrics when imputations are not rounded. The MLP model obtains largely better Area Under the Curve (AUC) results when no rounding is applied, results with other metrics are very close in both cases. The Decision Tree model obtains better results of about 1% for the three used inference metrics when no rounding is applied, those results are not significantly better but still indicative of an improvement. KNN results on both rounding

Model	Metric	Rounding	No Rounding
MLP	bACC	81.06% ± 2.71%	**81.31% ± 2.30%**
	AUC	84.69% ± 1.24%	**86.75% ± 1.12%**
	F_1	79.31% ± 2.88%	**79.60% ± 2.39%**
KNN	bACC	**80.15% ± 3.78%**	79.76% ± 3.25%
	AUC	84.25% ± 3.70%	**84.75% ± 2.17%**
	F_1	**78.11% ± 4.11%**	77.70% ± 3.56%
TREE	bACC	72.98% ± 2.75%	**73.85% ± 2.37%**
	AUC	72.98% ± 2.75%	**73.85% ± 2.37%**
	F_1	69.56% ± 3.16%	**70.52% ± 2.95%**

Table 2.1: Simple experiment to demonstrate the bias occurring when rounding imputation continuous values to the closest known value for categorical features. Column "Rounding" shows inference results of ML models applied on rounded imputations. Column "No Rounding" shows inference results of ML models applied on not rounded imputations.

and not rounding are very close, with rounding being slightly better when evaluated with balanced Accuracy and with the F_1-score, while not rounding appears to lead to slightly better results when evaluated with AUC. Confidence intervals in those results are quite high and overlapping between results, no significant improvement can be determined from those results only, more extensive experiments would be required to do so. however, those results seem to show that not rounding imputations of categorical features do not degrade inference results, and might even slightly improve them in most cases, which seems to be in accordance to (Horton et al., 2003) findings and comfort us in our choice not to round obtained imputations. Thus, in the rest of this work we follow the advice of (Horton et al., 2003) and do not round imputed values for categorical features to the closest known values for any imputation method.

2.2.7 Noise Regularization

Numerous studies and papers have shown that adding noise to a Neural Network input during training, known as noise regularization, can significantly enhance its generalization performance and act as an effective overfitting regularization (Bishop, 1995; Sietsma and Dow, 1991). The approach consists in introducing random noise into the training data during the learning process to prevent the model from excessively fitting to individual data points, ultimately improving the model ability to generalize (Bishop, 1995). To achieve this, a random vector is added to each data input vector before being fed into the model, thereby introducing new random noise to data points that have already been observed by the NN.

In the following, our proposed imputation frameworks can be compared to a kind of noise regularization that is applied solely to missing values, while scaling the amount of noise to the uncertainty level associated with each missing value. By acting as a focused noise regularization our imputation frameworks lead to less biased Neural Networks capable of more generalization, and thus, better inference results than using standard frameworks.

2.3 Proposed Approaches for Handling Attribute Noise

In this section, we propose and present two contributions for better handling attribute noise in a Machine Learning context:

1. We propose data Denoising and Imputation in One Step (DIOS), a method for correcting attribute noise as a preprocessing step. DIOS is able to correct observed erroneous values in the data while performing imputation of unobserved values. Our method does not require to learn from clean data to correct the corrupted dataset, it learns directly from the whole corrupted version of the dataset. It is also able to impute missing values without requiring any complete instance in the dataset. While it is easier to reach optimal results in a supervised or semi-supervised context, the method can also yield satisfactory results in a fully unsupervised setting. To the best of our knowledge, this is the first method that can truly handle attribute noise in its entirety by performing both missing values imputation and correction of erroneous values on mixed-type tabular data at the same time.

2. We propose Single-Hotpatching (S-HOT) and Multiple-Hotpatching (M-HOT), two frameworks that can be used to account for imputation uncertainty during Neural Network training, leading to better predictions. Those two frameworks take the between-imputation uncertainty into account to improve the training process of NNs, leading to an improvement in their generalization capacity. Those frameworks are based on the computation of the between-imputation uncertainty, which corresponds to the standard deviation between imputed values of all completed datasets. This uncertainty is then used as a scale to add stochasticity to the imputation of missing values directly on batch extraction during training. It leads to a kind of noise regularization that takes into account the imputation uncertainty, improving generalization capacity, and thus, prediction results on unseen data. Those frameworks are to be used in different situations, S-HOT is adapted to train a unique and large NN, while M-HOT can be used to train multiple learners in an ensemble way and reach extremely good prediction results at the expense of a higher computational cost.

2.3.1 DIOS: data Denoising and Imputation in One Step

We propose data Denoising and Imputation in One Step (DIOS), a method based on AEs and inspired from the recent image restoration technique called "Deep Image Prior" (Ulyanov et al., 2020). In the context of image restoration, (Ulyanov et al., 2020) showed that a randomly initialized DNN is able to capture relevant and useful information about an image distribution without requiring any pre-training of the model on similar images. They show that a DNN can be trained on a unique image to generate a corrected version $\mathring{X}$ of the corrupted image $\tilde{X}$. In a tabular data correction context, it is mostly not possible to learn generalizations from other tabular datasets that could be useful to correct the corrupted dataset. That is because, unlike images, tabular datasets do not share a uniform information representation, or similar concepts. Instead, a tabular dataset contains data coming from

a particular source, taken in a specific context, with its own data distribution. To solve the attribute noise correction task we are interested in training a DNN on the corrupted dataset only, with its erroneous and/or missing values, in order to correct them with the most adequate values.

The intuition behind DIOS is to train a model to reconstruct the corrupted data from randomly initialized noise. The model's training process involves learning abstract features and patterns from the data while disregarding any attribute noise in the original data. However, as the training continues, the model may become excessively focused on fitting the data and its attribute noise, leading to overfitting. Therefore, stopping the training before overfitting occurs ensures that the model maintains a generalized understanding of the data. Consequently, the model can generate credible missing values and eliminate any erroneous values present in the observed values.

DIOS is based on a DNN that is trained to reconstruct the original corrupted dataset $\tilde{X} \in \mathbb{R}^{n \times d}$ from a random noise input $Z \in \mathbb{R}^{n \times d}$. The model we use has an AE architecture, but since we feed it pure noise while training it to reconstruct data we can consider the model as a deep generative network. A deep generative model typically generates samples by mapping an input x through a function $f_\theta(\cdot)$ with θ representing the model parameters. Its output, denoted as $\hat{y}$, is a generated sample obtained as $\hat{y} = f_\theta(x)$. In the case of correcting corrupted data $\tilde{X}$, we have $\hat{X} = f_\theta(Z)$, where $\hat{X} \in \mathbb{R}^{n \times d}$ is the reconstructed data and $Z \in \mathbb{R}^{n \times d}$ is the noise input to the model. Unlike AEs, where the model would learn to reconstruct $\tilde{X}$ from $\tilde{X}$ itself, we use a random noise input Z to avoid overfitting on the noise in $\tilde{X}$. By reconstructing the corrupted data $\tilde{X}$ from random noise, the model learns to reconstruct $\tilde{X}$ without being biased by erroneous values during the encoding phase. The model's parameters θ are updated based on the error loss between $\tilde{X}$ and the output of the model $\hat{X}$, as shown in Figure 2.1. We note $\mathring{X}$ the best obtained correction by DIOS, that is, $\mathring{X} = f_{\theta^*}(Z)$ with θ^* the best found model parameters.

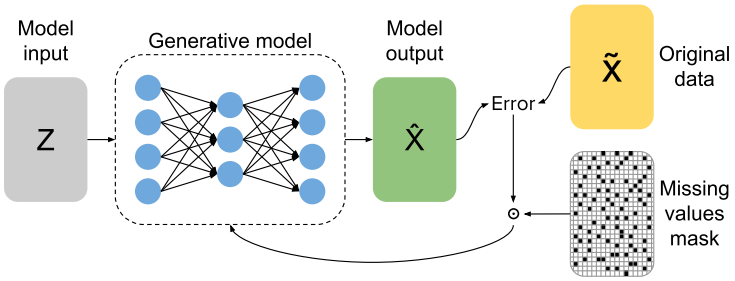


Figure 2.1: The model parameters are trained so that the model learns to reconstruct the original corrupted data $\tilde{X}$. By stopping the training at the right moment, the reconstruction $\hat{X}$ will be cleaner than the original data $\tilde{X}$.

DIOS model parameters are trained minimizing a task dependent loss term, that is, a loss term specific to the task that must be learned. This is done by adjusting the model parameters θ, which are randomly initialized, using a gradient descent optimization method. Any known optimization method can be used to minimize the objective function. The goal is to find the best possible values of θ, noted θ^*, that result in the loss term reaching a local minimum. Once the desired loss value is achieved or another termination condition specified

by the user is met, the training is complete, and the resulting best found reconstruction $\mathring{X}$ is saved.

Fitting a parameter set θ of sufficient size (i.e. containing enough weights and biases organized as a logical architecture) can lead to overfitting and, consequently, exact reconstruction of the corrupted data $\tilde{X}$. To avoid such overfitting, it is crucial to feed random noise as input to the generative model and not the the corrupted data $\tilde{X}$ as we would with an AE. This step prevents the model from being influenced by the noise present in $\tilde{X}$, forcing the model to learn generalizations about the data content before overfitting, and so, reconstruction of the noise in $\tilde{X}$ occurs. It is thus essential to stop the training phase before overfitting happens. We describe more precisely how to detect the right step at which to interrupt training in the next section.

The task specific loss term to use when correcting erroneous values in a corrupted dataset that do not contain missing values is the following:

$$\mathcal{L} = ||\hat{X} - X||^2 \tag{2.1}$$

Since all values are observed in this scenario, we can calculate the Mean Squared Error loss between the reconstructed data $\hat{X}$ and the original corrupted data $\tilde{X}$. By reconstructing the data $\tilde{X}$ without overfitting it, the model is forced to learn to extract and exploit abstractions and generalizations about $\tilde{X}$ without overfitting erroneous values, and so, replacing them with more plausible ones in the output reconstruction. Consequently, reconstruction eliminates outliers and noise and improves data quality. As a result, the best found reconstruction $\mathring{X}$ enhances future inference models capacity to learn from the cleaned data, leading to better performance in tasks such as prediction and classification.

In a data completion setting, the binary missingness indicator matrix $M \in \{0,1\}^{n \times d}$ is known, and so the model can be trained according to the following loss term:

$$\mathcal{L} = ||(\hat{X} - X) \odot M||^2 \tag{2.2}$$

Where $\odot$ is the Hadamard product that is applied between the MSE loss and the missingness indicator matrix M, where $M_{ij} = 1$ if $\tilde{X}_{ij}$ is observed and $M_{ij} = 0$ if $\tilde{X}_{ij}$ is missing. By applying the mask to the difference between the reconstructed data $\hat{X}$ and the original corrupted data $\tilde{X}$, only the observed values are used to compute the MSE loss. Consequently, the reconstruction is optimized to fit the known values while naturally inferring plausible values for the missing ones based on the data distribution learned by the generative model. The generative power of the model allows it to learn abstract knowledge and generalize from all known values in the corrupted dataset, enabling it to identify specific correlations and links between features and use them to complete the missing values. The result is a complete reconstruction of the data with coherent imputations in place of the missing values.

Our approach has the capability to handle both erroneous values correction and missing values imputation simultaneously. To achieve this, we can directly use Equation 2.2 as a loss function, as it can impute missing values while naturally replacing erroneous values with more plausible values. If the objective is solely to fill in missing values without performing erroneous values correction, it is possible to use the missingness indicator matrix as a mask

to replace missing values in $\tilde{X}$ by the ones obtained in $\mathring{X}$ while conserving all known values in $\tilde{X}$ unchanged. As mentioned in section 2.2.6, it has been stated in the literature that not rounding imputed values of categorical features to the nearest known value leads to better inference results, which we experimentally verified. Therefore, our method DIOS is naturally able to handle mixed-type tabular data.

2.3.1.1 Generative Model Architecture

To perform data reconstruction, our method DIOS depends on the generative model architecture. Hence, it is crucial to choose an architecture that suits the dataset we intend to correct. Although any kind of DNN can be used, our findings suggest that for datasets with numerous features, 1D fully Convolutional Neural Networks with skip connections lead to superior outcomes. In contrast, for datasets with fewer features, small fully-connected NN offer better and faster results.

In the case of datasets with large amounts of features, a generic fully convolutional architecture with skip connections is used, which must be empirically dimensioned for each dataset. The architecture that is used is an encoder-decoder model with skip connections, similar to a U-Net (Ronneberger et al., 2015b). When working with images, it is common to use 2D convolutions, as relevant local information is shared in neighboring pixels in both the width and height axis that 2D convolutions are able to capture and learn. As we are interested in a tabular data context, we use 1D convolutions. Indeed, in tabular data, each element is defined by its features in the width dimension and is not associated with other elements in the height dimension. We use 1D convolutions to capture general knowledge on data elements features, while avoiding the capture of knowledge between arbitrary ordered elements through the dataset which would be irrelevant and would encourage overfitting.

An important matter to consider is that using 1D convolutions could introduce bias during the learning phase, as not all features may be related to their neighboring features, since the order of features in the dataset is arbitrary, unlike pixels in an image. To address those interrogations, we experimented by shuffling the order of the features in a dataset, we obtained similar results when features are shuffled and when they are ordered as originally. Our experiments show that it is easier and faster for our method to converge to satisfactory results when features are ordered in a "logical" manner (i.e. when features are clustered depending on their nature) than when they are arbitrarily ordered, that is, shuffled. Nonetheless, if the architecture is properly adapted to the dataset, satisfactory results can be achieved even when the feature order is shuffled. It's worth noting that feature order does not matter in fully-connected architectures.

When working with datasets that have few features, we found that using a fully-connected NNs produced similar results with both small (i.e. small enough so as to not be able to overfit the data) and large architecture designs. One notable difference between fully-connected and convolutional models is that for fully-connected NN, we found better results when using the original corrupted data as input, which reduces the model's generative task and allows it to concentrate solely on learning patterns and generalizations from the data. In such case, it is important to design the fully-connected model to be small enough so that it cannot

learn an identity mapping from input to output, which would yield excellent loss results but would prevent any abstraction and generalization learning for the model. We also noted that it is easier to halt the training process at the appropriate training iteration with a smaller architecture. Consequently, for datasets with few features, we suggest to use small fully-connected architectures and using the original corrupted data as input, that is, $Z = \tilde{X}$.

More details about defining an adequate model architecture will be discussed in the experimental section, but it is worth noting that the best architecture for a given dataset can only be determined empirically. Despite this, the generic convolutional and fully-connected architectures that we designed and used in our experiments can be easily adjusted to suit any mixed-type tabular dataset to correct.

2.3.1.2 DIOS Training

The goal is to correct and/or complete each instance in a corrupted dataset. To do so, DIOS must learn generalizations from known values in the corrupted dataset in order to properly correct each instance. The DNN parameters are learned using a gradient descent optimization algorithm that applies the computed gradient based on the defined loss term. To ensure that the model doesn't overfit to the original corrupted data, we have integrated an early stopping mechanism during the training phase. Moreover, between each epoch, we fully shuffle the dataset instances and split them into mini-batches, which helps preventing the model from overfitting the corrupted erroneous values.

Algorithm 1 is the pseudocode of our generic DIOS correction algorithm: which can either perform erroneous values correction, missing values imputation, or both, based on the chosen loss term. The optimal correction is determined by the minimum loss value or maximum accuracy value, depending on user preference. In scenarios with full or partial supervision, the best correction can be chosen by training a prediction model using the current correction and evaluating its accuracy, or another chosen inference metric. However, in unsupervised settings, the best correction can only be selected based on the loss value. The algorithm takes multiple inputs:

- $\tilde{X} \in \mathbb{R}^{n \times d}$: the original corrupted data.
- $f_\theta(\cdot)$: the generative model.
- $Z \in \mathbb{R}^{n \times d}$: the model input.
- $M \in \{0,1\}^{n \times d}$: the missingness indicator binary mask, which is only used when there are missing values in $\tilde{X}$.
- $y \in \mathbb{R}^n$: the known labels, which are only used in a fully- or semi-supervised setting, where the best correction is selected based on the accuracy obtained on the validation set.

The algorithm outputs the best obtained correction $\mathring{X} \in \mathbb{R}^{n \times d}$, which can be used raw if one wishes a polished correction, or masked with known values in $\tilde{X}$ if one wishes to only impute missing values while conserving known values unchanged.

Algorithm 1: DIOS Correction Algorithm

input : $\tilde{X}$, $f_\theta(\cdot)$, Z, M, y
output: $\mathring{X}$
for $i = 1$ *to max_iter* **do**
  Add small random Gaussian noise to Z as regularization;
  $Z' \longleftarrow split(shuffle(Z, i), nb\_batches)$;
  $X' \longleftarrow split(shuffle(X, i), nb\_batches)$;
  $M' \longleftarrow split(shuffle(M, i), nb\_batches)$;
  for $j = 1$ *to nb_batches* **do**
    $\hat{X}'_j = f_\theta(Z'_j)$;
    Compute loss between $\hat{X}'_j$ and X'_j using Equation 2.1 or 2.2;
    Perform backward propagation on f using $Adam$ optimizer;
  end
  $\hat{X} = unshuffle(\hat{X}', i)$;
  if *best correction selected on accuracy* **then**
    Compute accuracy on validation set using y labels;
    $\mathring{X} = \hat{X}$ if accuracy is the highest ever obtained;
    Break if accuracy did not improve since *max_deter* iterations;
  else if *best correction selected on loss* **then**
    $\mathring{X} = \hat{X}$ if loss is the lowest ever obtained;
    Break if loss did not decrease since *max_deter* iterations;
end

The user needs to define two variables, namely *max_iter* and *max_deter*. These variables respectively determine the maximum number of iterations the algorithm will perform, assuming no breaking condition is met, and the maximum number of deteriorating iterations that will lead to an early stop. Once the algorithm is stopped, the best obtained reconstruction $\mathring{X}$ is returned.

We use the *shuffle* function, which randomizes the order of the elements in a given array along its first dimension, based on a random seed. By using this function, we can apply the same shuffling pattern to multiple arrays given a defined seed, ensuring that each line in a shuffled matrix can be linked to the corresponding line in another matrix shuffled with the same seed. We shuffle the input Z before splitting it into mini-batches, and do the same accordingly to the model target $\tilde{X}$ in order to reduce overfitting and produce better results. The *unshuffle* function is applied to restore the original ordering of the elements in the arrays, given the same seed as used to shuffle them. Lastly, the *split* function generates an array that contains a specified number of mini-batches from the given matrix.

At each iteration, a small random noise drawn on a Gaussian distribution is added to the model input Z, this ensures that the training process of the model does not get stuck before its convergence and avoids overfitting.

2.3.1.3 When and How can DIOS be Used?

Our DIOS correction method can be used and applied in any supervision scenario, from unsupervised to fully supervised. Unlike the polishing method presented in (Teng, 2004), which incorporates labels in the data while polishing, leading to potential label leakage and bias, DIOS does not use labels in its learning process. Instead, validation labels are only employed to regularly evaluate the reconstructed data quality by training a learner on the current data reconstruction and evaluating its inference results on the validation set during training. This ensures that the training of our approach is unsupervised, based solely on attribute values, thereby preventing overfitting the labels. Having at least some labeled elements enhances the selection of the best data correction from DIOS, therefore, a semi- or fully supervised setting helps DIOS yield good results. But labels are not required to train DIOS, it is possible to get satisfactory imputation in a fully unsupervised setting if the architecture is properly designed.

DIOS is a preprocessing technique that should be applied to the entire corrupted dataset, that is, the concatenation of all train and test data. As our method works very well in a semi-supervised setting, our DIOS correction method is viable in most real-world contexts.

DIOS learning phase operates similarly to that of robust learners, in that it is trained to learn abstractions and generalizations from the corrupted data, and the training is stopped before overfitting occurs, which would mean overfitting the parasitic noise in the corrupted data. However, unlike robust learners, DIOS only learns to reconstruct the data while ignoring noise, it does not have the additional burden of learning to perform a prediction task on the corrupted data. This makes it simpler to achieve good generalization, and so, good attribute noise correction. Abstractions and concepts learned by DIOS can then be used for erroneous values correction, similarly to the polishing method, with the advantage that it can also impute missing values.

2.3.2 Accounting for Imputation Uncertainty During Neural Network Training

Multiple-Imputation is a first step towards taking account of between-imputation uncertainty. It naturally takes into account the uncertainty of imputed values through its ensemble nature, but each inference model is still biased by being trained on an arbitrarily fixed completed dataset (Rubin, 2004). We propose two frameworks that take this between-imputation uncertainty into account and show that Neural Networks trained using those frameworks have better generalization capacity. Those frameworks are based on the computation of the between-imputation uncertainty, which corresponds to the standard deviation between imputed values of all completed datasets. This uncertainty is then used as a scale to add stochasticity to the imputation of missing values directly on batch extraction during training. It leads to a kind of noise regularization that takes into account the imputation uncertainty, which improves generalization capacity, and thus, prediction results on unseen data. Those frameworks can be used to train a Neural Network on any inference task.

We introduce the Single-Hotpatching (S-HOT) framework, that addresses the challenge of

handling missing values in tabular datasets when training a Neural Network. Traditional approaches such as Single-Imputation and Multiple-Imputation have their limitations. Single-Imputation is straightforward and fast, but it produces a biased learner by training only one model on a single imputed dataset. On the other hand, Multiple-Imputation trains multiple models on multiple imputations, resulting in better inference results but at the cost of increased computational complexity. S-HOT addresses those issues by training a single strong model while relying on multiple imputations, resulting in a less biased model and requiring less computational time than MI. By using the uncertainty level of each imputed value, S-HOT imputes missing values in a "hotpatching" manner during batch extraction, leading to better generalization capabilities and more robust inference results. Our experimental results show that S-HOT significantly outperforms SI, and leads to close results to MI while necessitating much less running-time, making it an interesting approach to consider for situations where a single large model needs to be trained on incomplete tabular data.

The Multiple-Hotpatching (M-HOT) framework builds on the S-HOT approach to train NNs on imputed tabular data by introducing the ensemble paradigm. Rather than relying on a single strong model trained with multiple imputations, M-HOT trains as many learners as imputations are performed. Each learner takes into account the between-imputation uncertainty to reduce bias. Each learner in the resulting ensemble of Neural Networks is less biased and capable of enhanced generalization than individual learners trained using MI. Empirical results show that M-HOT consistently outperforms MI, while not being computationally more expensive, making it an attractive option for any situation in which several Neural Networks can be trained using an ensemble approach.

2.3.2.1 S-HOT: Single-Hotpatching

Our Single-Hotpatching (S-HOT) framework is similar to MI, but has the advantage of training only one strong model. This framework is named Single-Hotpatching because, when using this approach, missing values are dynamically imputed directly during batch extraction in a "hotpatching" manner.

Training multiple large Neural Networks in an ensemble manner is not always a viable option, due to the substantial amount of time and resources required to train each model. To address this issue, S-HOT aims at training a single model while relying on multiple imputations, such as MI. By doing so, S-HOT can train a less biased model than if it were trained using SI, while also requiring less computational time than MI. Our experimental results demonstrate that S-HOT achieves significantly better performance than SI for identical training times. Thus, S-HOT is a interesting approach to use in situations where a single large model needs to be trained on imputed tabular data.

When we train a NN on a dataset with fixed imputed values that are most certainly non-optimal, the model is repeatedly trained on wrong and imprecise data, leading to a biased NN that cannot generalize well. To overcome this, we propose the S-HOT imputation framework, which performs multiple imputations and calculates the between-imputation standard deviation associated with each imputed value, which we call the uncertainty level. We draw random values from a normal distribution parameterized using the mean and stan-

dard deviation computed between the imputations to train the model on a range of plausible imputations. The NN is trained on a span of plausible values in place of each missing value during its training, leading to a less biased model with better generalization capabilities. Once the NN is trained, prediction results are obtained by randomly patching missing values in the test set using the same process for p iterations, leading to p predictions. The p model output probabilities are pooled in an ensemble manner to obtain the final prediction, making the inference results more robust than a single prediction. Figure 2.2 illustrates the training and testing phases of a NN using the S-HOT framework.

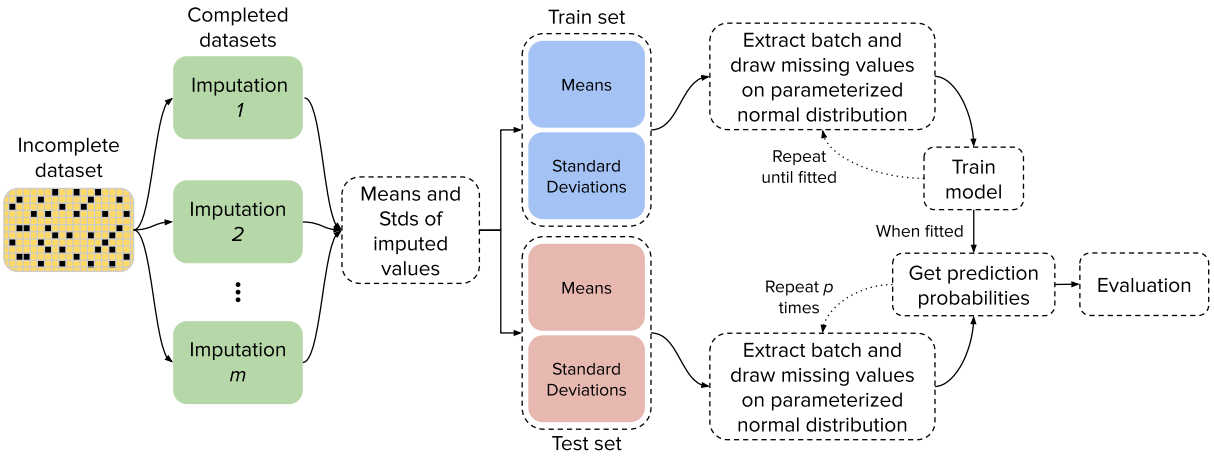


Figure 2.2: Single-Hotpatching. We perform m imputations and compute the means and standard deviations of imputed values between the m sets, leading to two matrices of identical dimension than the corrupted dataset. Those two matrices are split between the train and test sets. During training, every time a batch is extracted, missing values are drawn from a normal distribution parameterized using previously computed means and stds. Once the model is trained, the prediction probabilities are extracted by applying the same process p times for each test instance, resulting in p predictions. The final prediction is computed as the mean of the p output probabilities. The p value is set as a few dozen to obtain robust prediction results.

Mathematically, we note $\tilde{X} \in \mathbb{R}^{n \times d}$ the original incomplete dataset, where each value $\tilde{X}_{ij}$ is either observed or missing, with n and d the number of elements and features in $\tilde{X}$. We perform m imputations, leading to m different completed datasets $\mathring{X}^{1\ldots m}$, with $\mathring{X}^{k} \in \mathbb{R}^{n \times d}$ the k-th completed dataset. Therefore, a missing value $\tilde{X}_{ij}$ is imputed with m different values $\mathring{X}_{ij}^{1\ldots m}$. Then, we compute the means μ and standard deviations σ of each value of the m completed datasets, with

$$\mu_{ij} = \frac{1}{m} \sum_{k=1}^{m} \mathring{X}_{ij}^{k} \tag{2.3}$$

and

$$\sigma_{ij} = \sqrt{\frac{1}{m} \sum_{k=1}^{m} (\mathring{X}_{ij}^{k} - \mu_{ij})^2} \tag{2.4}$$

We note that values in $\mathring{X}^{1\ldots m}$ that are observed in $\tilde{X}$ have a mean of $\mu_{ij} = \tilde{X}_{ij}$ and a standard deviation of $\sigma_{ij} = 0$/ Only missing values in $\tilde{X}$ have a value $\sigma_{ij} > 0$. Then, we train a

Neural Network by feeding it batches that are computed from μ and σ. To extract a batch $B \in \mathbb{R}^{b \times d}$, with b the number of elements in the batch, we draw each value of the batch from a normal distribution parameterized with mean μ_{ij} and standard deviation $\alpha \cdot \sigma_{ij}$, such as $B_{ij} \sim \mathcal{N}(\mu_{ij}, \alpha \cdot \sigma_{ij})$. Where α is a scale hyper-parameter that can be set to 1 in most cases and might need to be set lower depending on the average uncertainty level. If the used imputation method outputs imputed values with a wide uncertainty the α scale should be empirically set lower than 1 to limit the stochastic impact induced by the approach. We never found a situation that benefited from increasing the α value above 1. When a data point is presented to the NN, observed values are set to $\tilde{X}_{ij}$, and missing values are set to a random value that follows the normal distribution of the m imputations. Thus, the NN is not repeatedly trained on arbitrarily fixed (and very probably non-optimal) imputations, as it would be using SI or MI frameworks. Instead, it is trained on values that are randomly drawn from the imputations distribution. This process operates as a noise regularization that takes into account the between-imputation uncertainty, resulting in a less biased and more generalized NN.

Algorithm 2 shows the training phase of the S-HOT framework. $\tilde{X}$ is the original incomplete dataset, m is the number of imputations to perform, $impute(\cdot)$ is the chosen imputation method, $normal(\mu, \sigma)$ is the method that draws each value x_{ij} on the normal distribution parameterized with μ_{ij} and σ_{ij}, α is the hyper-parameter used to scale the standard deviation parameter, and *nb_batches* is the number of batches required to span over the whole dataset.

Algorithm 2: Training phase of the S-HOT framework

input : $\tilde{X}$, m, $impute(\cdot)$, $normal(\cdot,\cdot)$, α, *nb_batches*
output: A trained Neural Network f_{θ^*}
for $i = 1$ *to* m **do**
  $\mathring{X}^i = impute(\tilde{X})$;
end
Compute μ and σ from $\mathring{X}^{1\dots m}$ using equations 2.3 and 2.4;
Randomly initialize the Neural Network f_θ ;
while *convergence is not reached* **do**
  for $b = 1$ *to nb_batches* **do**
    $B = normal(\mu[batch\_slice], \alpha \cdot \sigma[batch\_slice])$;
    Fit the Neural Network f_θ to extracted B;
  end
end

2.3.2.2 M-HOT: Multiple-Hotpatching

The Multiple-Hotpatching (M-HOT) framework extends S-HOT by using the ensemble paradigm. This framework trains as many learners as imputations are performed, those learners are trained while considering the between-imputation uncertainty, leading to less biased individual learners. Our empirical findings demonstrate that using M-HOT to train a NN leads to consistently better inference results than MI while not being computationally more expensive. It is beneficial to apply the M-HOT framework in any situation in which one can afford to train several Neural Networks in an ensemble manner.

The framework is similar to S-HOT with the main difference that we use the ensemble paradigm such as in MI. We use the previously defined mathematical notations, $\tilde{X} \in \mathbb{R}^{n \times d}$ is the original incomplete dataset, where each value $\tilde{X}_{ij}$ is either observed or missing. We perform m imputations which leads to m different completed datasets $\mathring{X}^{1 \dots m}$, with $\mathring{X}^{k} \in \mathbb{R}^{n \times d}$ the k-th completed dataset. One learner is defined for each completed dataset, leading to m models. We compute the standard deviations σ in the same way as in equation 2.4 and do not need to compute the means. Then, the Neural Networks are trained. To extract a batch $B^k \in \mathbb{R}^{b \times d}$ that will be fed to the k-th model, we draw each value of the batch from a normal distribution parameterized with mean $\mathring{X}^k_{ij}$ and standard deviation $\alpha \cdot \sigma_{ij}$, such as $B^k_{ij} \sim \mathcal{N}(\mathring{X}^k_{ij}, \alpha \cdot \sigma_{ij})$. Thus, all Neural Networks are trained in an ensemble manner and are seeing imputed values drawn from a normal distribution centered on the corresponding computed imputation with added diversity during their training. This leads to an ensemble of Neural Networks that are less biased and capable of more generalization than individual learners trained with MI, as they take into account the between-imputation uncertainty with M-HOT. M-HOT can be used as a substitute to MI in any situation in which MI is viable. Figure 2.3 shows, in a more practical way, how Multiple-Hotpatching takes place during the training phase of the Neural Networks.

2.4 Experiments

In this section, we present our experiments to evaluate our proposed approaches and compare their performance against other well-known and state-of-the-art imputation and erroneous values correction methods from the literature. We evaluate our approaches on both widely used benchmark datasets and real-world medical datasets that share similarities with our application context.

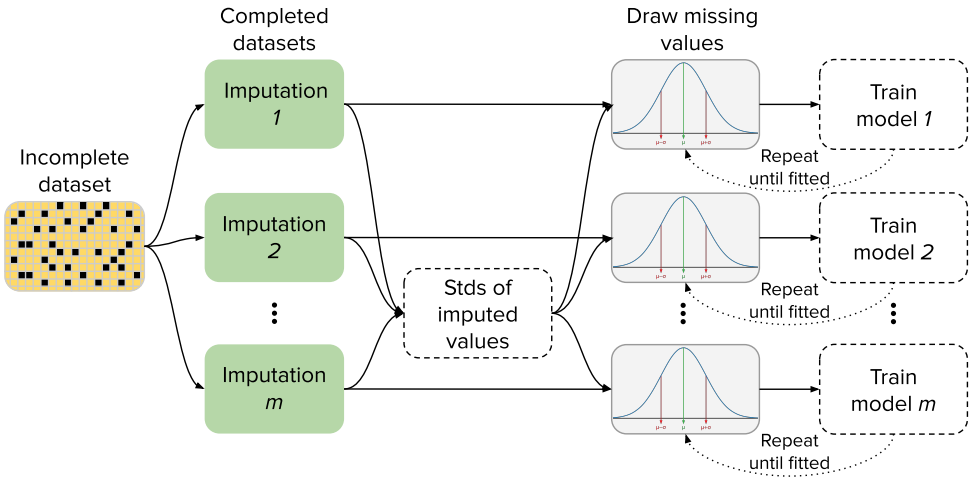


Figure 2.3: Multiple-Hotpatching training phase. We perform m imputations and compute the standard deviations of imputed values. We train m Neural Networks, when a batch is extracted from the k-th completed dataset, missing values are drawn from a normal distribution parameterized using previously computed standard deviations and values from the k-th completed dataset as means. The process is repeated until all Neural Networks are trained. Those can then be used to obtain predictions in the same way as with S-HOT but in an ensemble manner.

2.4.1 Used Datasets

We are interested in using our approaches in a mixed-type tabular data medical context with a limited amount of instances. In order to best evaluate our proposed approaches for our applicative context, we choose several popular benchmark datasets and several real-world medical datasets.

We performed our experiment on various benchmark tabular and mixed-type datasets of various dimensions and various fields:

- STATLOG[3]: This dataset is composed of 690 elements with 14 features, 6 numerical and 8 categorical. The goal is to learn a binary classification task. The dataset concerns credit card applications, all features names and values have been changed to meaningless symbols to protect confidentiality of the data.

- PIMA[4]: This dataset is from the National Institute of Diabetes and Digestive and Kidney Diseases. It is composed of health data of female patients from Pima Indian heritage, of at least 21 years old. The goal is to predict whether a patient suffers from diabetes or not, based on several medical predictor variables, such as number of pregnancies, blood pressure, insulin, etc. It contains 768 instances and 8 mixed-type features.

- ARRHYTHMIA[5]: This medical dataset contains 452 elements and 260 attributes, 206 of which are numerical features and the rest are categorical features: it has been used for

[3] https://archive.ics.uci.edu/ml/datasets/statlog+(australian+credit+approval)
[4] https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database
[5] https://archive.ics.uci.edu/ml/datasets/arrhythmia

arrhythmia analysis. We aim to learn a binary classification task, to predict whether a patient suffers from arrhythmia or not.

- MFEAT[6]: This dataset consists of features that have been extracted from handwritten digits: with 76 Fourier coefficients of the character shapes, 216 profile correlations, 64 Karhunen-Love coefficients, 240 pixel averages in 2×3 windows, 47 Zernike moments and 6 morphological features. Once concatenated, this dataset contains 200 samples per class for a total of 2000 elements with 649 features.

- ORL[7]: We used the multi-view ORL dataset generated by (Zhu et al., 2019a). The original ORL dataset contains 10 different images of each of the 40 distinct subjects. For each subject, the images were taken under varying lighting conditions with different facial expressions (open/closed eyes, smiling/not smiling) and facial details (glasses/no glasses). Zhu et al. adjusted the image size to 48×48 pixels and extracted three types of features: intensity (4096 dimensions), LBP (3304 dimensions), and Gabor (6750 dimensions). There are therefore 400 elements and 14150 features in this tabular ORL dataset.

- IRIS[8]: This very simple dataset is composed of 150 elements with 4 features and 3 classes.

- WINE[9]: This dataset is composed of 178 instances with 13 features and 3 classes.

- ABALONE[10]: This dataset is composed of 4177 elements with 8 features. The goal is initially to learn a regression task to predict the age of abalones, we transformed it into a binary classification task in order to remain in the same experimental setting as with other datasets.

In order to experiment with imputation methods and frameworks on benchmark datasets it is needed to artificially introduce missing values at a chosen rate as described in section 2.2.5.1.

We also evaluate our approaches on three incomplete real-world medical datasets to demonstrate the efficiency and relevance of our approaches in a context highly similar to our application setting.

- COVID[11]: This dataset was publicly released with the paper (Yan et al., 2020), it contains medical information collected in early 2020 on pregnant and breastfeeding women. We based our data preprocessing on the one realized in the original paper, the final preprocessed dataset is composed of 361 patients with 76 features, with about 20% missing data. The goal is to predict the survival outcome of patients.

[6] https://archive.ics.uci.edu/ml/datasets/Multiple+Features
[7] https://github.com/qinghai-zheng/MSCNLG
[8] https://archive.ics.uci.edu/ml/datasets/iris
[9] https://archive.ics.uci.edu/ml/datasets/wine
[10] https://archive.ics.uci.edu/ml/datasets/abalone
[11] https://github.com/HAIRLAB/Pre_Surv_COVID_19/tree/master/data

- NHANES[12]: In the NHANES dataset, US National Health and Nutrition Examination Surveys, we used data from studies spanning from years 2000 to 2008, with 95 features and about 33% missing values. We aim to predict if a patient suffers from diabetes or not.

- MYOCARDIAL[13]: This medical dataset is composed of 1700 patients with 107 features, with about 5% missing values. We aim to predict the survival outcome of patients.

2.4.2 Evaluating our Attribute Noise Correction Method DIOS

First, we evaluate our proposed attribute noise correction method DIOS, technical details about the method and those experiments can be found in Appendix A, section A.3.

2.4.2.1 Experimental Protocol

We demonstrate the application of DIOS in both missing values imputation and erroneous values correction settings, compared to state-of-the-art methods from both those fields.

The first experiment we perform is to evaluate DIOS performance compared to other methods on an imputation task, both on benchmark and real-world datasets. For benchmark datasets, we introduce missing values for each combination between MCAR/MAR/MNAR mechanisms and missing rates of 25/50/75%, and evaluate performance on balanced Accuracy and RMSE. For real-world datasets, missing values are naturally present at various rates, we evaluate all methods based on their inference results using the balanced Accuracy metric. Inference results are obtained through 5-fold cross validation using a simple KNN classifier, with $k = 5$ in all cases. We repeated each experiment 10 times with 10 different stochastic seeds to obtain significant and consistent results with a meaningful average and standard deviation for each method in each experimental setting. In that way, each method is evaluated on the same 10 incomplete datasets at each erroneous values rate, making the comparison fair and allowing evaluation of the robustness of each method.

Our second experiment aims at evaluating DIOS performance compared to other methods on an erroneous values correction task. We only perform this experiment on benchmark data, as real-world datasets contains missing values. To evaluate our method on an erroneous values correction task, we artificially introduce erroneous values into the data at a given rate. The erroneous values rate determines how many attribute values will be replaced by a random value, such as described by (Zhu and Wu, 2004). For example, an erroneous values rate of 10% will assign random values to 10% of the dataset values. We compared the methods on erroneous values levels of 0/5/10/15/20/40/60% for all benchmark datasets. Once again, each experiment was repeated 10 times, with 10 different random seeds, and at each erroneous values rate, such as described above.

The third experiment aims at evaluating DIOS in a complete attribute noise correction task on real-world medical data. We compare DIOS results to those obtained by the combination of the best performing imputation method from previous experiments, followed

[12] https://www.cdc.gov/nchs/nhanes/index.htm
[13] https://archive.ics.uci.edu/ml/datasets/Myocardial+infarction+complications

by each tested state-of-the-art erroneous values correction method. Missing values are already present in used real-world datasets, we artificially add erroneous values at levels 0/5/10/15/20/40/60% in order to illustrate DIOS performance on a complete attribute noise correction task.

Finally, we compare running-times between tested methods on both imputation and erroneous values correction tasks on all datasets. For this running-time experiment, we select an arbitrary noise level or missing rate, as it does not influence the running-times of the methods.

We have chosen eight methods from various categories with which to compare our results. For simple algorithms such as Substitution (SUB) imputation, we used the "SimpleImputer" class from the python library "scikit-learn"[14]. We compared DIOS to the k-nearest neighbors imputation (KNN) and Multivariate Imputation by Chained Equations (MICE) methods. We used implementations from the scikit-learn library for both of them, the "KNNImputer" class for KNN, and the experimental "IterativeImputer" class for MICE. We implemented MIDA from the author description made in the original paper, and from the code template supplied by the author in the following public gist [15]. We used the public implementations given by the respective original authors[16][17] for the SINKHORN and GAIN methods. For SOFTIMPUTE we used the public re-implementation by Travis Brady of the Mazumder and Hastie's R softImpute package[18]. Finally, we compared our results to the MISSFOREST algorithm, and used the "MissForest" class from the python library "missingpy"[19].

There are few attribute erroneous values correction methods in the literature, we implemented four methods with which to compare our results. We compared our method to the standard Filtering (SFIL) and Polishing (SPOL) methods described by Teng in (Teng, 2004). We based our implementation of both methods on the original description from the paper. We used the erroneous values detection method PANDA (Van Hulse et al., 2007) to create improved versions of both Filtering (PFIL) and Polishing (PPOL) methods. Once again, we based our implementation of the PANDA algorithm on the description made in the paper.

In order to statistically assess experimental results, we compute t-tests with a significance level (p-value) of 0.05 between DIOS and every other compared method. The t-tests are computed on the results obtained on the 10 runs for which we conducted our experiments, with each method and for each setting. A global t-test is also computed globally on the complete results with each evaluation metric. The results of these t-tests are symbolized in result tables as either a bullet $\bullet$, a circle $\circ$, or an equivalent symbol $\equiv$. The bullet is used to signify that our method is significantly better than the method we compared it to, the circle signifies the opposite, and the equivalent means that there is no significant difference between DIOS and the compared method. In the following sections, we will refer as "significantly better" all results that have been evaluated using a t-test and that were classified as significantly better in regard to a p-value of 0.05, and "significantly worse" the opposite.

[14] https://scikit-learn.org
[15] https://gist.github.com/lgondara/18387c5f4d745673e9ca8e23f3d7ebd3
[16] https://github.com/BorisMuzellec/MissingDataOT
[17] https://github.com/jsyoon0823/GAIN
[18] https://github.com/travisbrady/py-soft-impute
[19] https://pypi.org/project/missingpy/

Technical details about DIOS and the experiments we performed can be found in Appendix A, section A.3.

2.4.2.2 Comparative Study on an Imputation Task

This experiment evaluates DIOS performance compared to other methods on an imputation task, both on benchmark and real-world datasets. Complete results for this experiment can be found in the Appendix A: Tables A.1, A.2, A.3, A.4, A.5, A.6.

We experiment on two benchmark datasets with large numbers of features (MFEAT with 649 and ORL with 14150), to demonstrate the capacity of our method to scale with dataset dimensions. Complete ORL results can be found in Appendix A, in Table A.4. Results on the ORL dataset show that DIOS obtains significantly better results than any other method and in any case for both Accuracy and RMSE metrics. Several state-of-the-art methods were not usable with this dataset because of scaling issues. Indeed, GAIN could not be executed on our hardware because of a too high RAM usage, and MICE and MISSFOREST algorithms complexities makes them unusable with a too large number of features as they require to train one learner for each feature and iteratively repeat the process until convergence.

Model	Metric	MCAR 25	MCAR 50	MCAR 75	MAR 25	MAR 50	MAR 75	MNAR 25	MNAR 50	MNAR 75
DIOS	RMSE	.136906 ±.000789	.159547 ±.002225	.206618 ±.004068	.134042 ±.002204	.149251 ±.001893	.173816 ±.002857	.137195 ±.001611	.159608 ±.002029	.205436 ±.002275
	ACC	**98.29% ±0.09%**	**98.12% ±0.07%**	96.91% ±0.19%	**98.31% ±0.09%**	**98.28% ±0.09%**	**98.14% ±0.13%**	**98.31% ±0.05%**	**98.23% ±0.08%**	97.02% ±0.20%
SUB	RMSE •	.319362 ±.000359 •	.319532 ±.000171 •	.319656 ±.000100 •	.323797 ±.003016 •	.324849 ±.003200 •	.326267 ±.003431 •	.322167 ±.000659 •	.323362 ±.000454 •	.324760 ±.000277 •
	ACC •	97.78% ±0.12% •	96.24% ±0.24% •	67.83% ±1.87% •	97.91% ±0.13% •	97.79% ±0.21% •	97.31% ±0.32% •	97.83% ±0.13% •	97.00% ±0.22% •	83.38% ±1.44% •
KNN	RMSE •	.171465 ±.000433 •	.182051 ±.000194 •	.213410 ±.000298 •	.173290 ±.002162 •	.181652 ±.002409 •	.196197 ±.002762 •	.172750 ±.000411 •	.184768 ±.000307 •	.220940 ±.000349 •
	ACC •	97.71% ±0.11% •	97.43% ±0.10% •	95.84% ±0.23% •	97.82% ±0.16% •	97.69% ±0.19% •	97.30% ±0.28% •	97.74% ±0.14% •	97.50% ±0.17% •	95.59% ±0.30% •
GAIN	RMSE •	.269801 ±.001455 •	.238402 ±.001020 •	.536074 ±.005791 •	.307802 ±.001784 •	.248680 ±.002288 •	.391710 ±.009960 •	.268669 ±.001205 •	.241067 ±.002235 •	.554727 ±.003461 •
	ACC •	97.57% ±0.13% •	97.40% ±0.20% •	79.17% ±1.60% •	97.73% ±0.11% •	97.58% ±0.13% •	96.81% ±0.27% •	97.68% ±0.15% •	97.47% ±0.16% •	86.66% ±0.91% •
MIDA	RMSE •	.289513 ±.004842 •	.318562 ±.000224 •	.319529 ±.000099 •	.282678 ±.005380 •	.307984 ±.006300 •	.324491 ±.004009 •	.291557 ±.003724 •	.322264 ±.000521 •	.324621 ±.000282 •
	ACC •	97.71% ±0.11% •	96.31% ±0.23% •	68.13% ±1.92% •	97.79% ±0.11% •	97.74% ±0.18% •	97.28% ±0.36% •	97.77% ±0.18% •	97.00% ±0.25% •	83.53% ±1.49% •
SOFT	RMSE •	.297167 ±.000474 •	.310272 ±.000676 •	.385783 ±.001054 •	.302148 ±.003227 •	.315895 ±.003425 •	.367079 ±.004706 •	.300616 ±.001117 •	.315058 ±.001206 •	.389065 ±.001178 •
	ACC •	97.78% ±0.13% •	96.04% ±0.32% •	69.45% ±2.16% •	97.97% ±0.11% •	97.73% ±0.19% •	97.27% ±0.32% •	97.80% ±0.14% •	96.74% ±0.30% •	84.80% ±1.18% •
MICE	RMSE ◦	.124647 ±.000272 ◦	.145737 ±.000250 ◦	.206923 ±.000577 ≡	.127042 ±.001405 ◦	.131368 ±.001965 ◦	.157530 ±.003018 ◦	.125361 ±.000201 ◦	.146554 ±.000298 ◦	.209269 ±.000740 •
	ACC •	98.00% ±0.17% •	97.84% ±0.11% •	**97.09% ±0.29%** ≡	98.03% ±0.09% •	98.02% ±0.13% •	97.84% ±0.17% •	98.08% ±0.13% •	97.86% ±0.10% •	**97.05% ±0.25%** ≡
SINK	RMSE •	.182632 ±.000293 •	.195835 ±.000346 •	.220215 ±.000355 •	.185622 ±.002000 •	.199430 ±.002217 •	.223433 ±.002465 •	.184920 ±.000370 •	.200091 ±.000473 •	.226424 ±.000519 •
	ACC •	97.77% ±0.19% •	97.03% ±0.24% •	92.90% ±0.45% •	97.90% ±0.19% •	97.59% ±0.18% •	97.24% ±0.26% •	97.74% ±0.09% •	97.18% ±0.22% •	93.49% ±0.54% •
MISS	RMSE ◦	**.105803 ±.000459** ◦	**.131395 ±.000367** ◦	**.177283 ±.000749** ◦	**.101257 ±.001610** ◦	**.117162 ±.002386** ◦	**.139133 ±.002813** ◦	**.106212 ±.000529** ◦	**.132246 ±.000835** ◦	**.175603 ±.001144** ◦
	ACC •	97.97% ±0.13% •	97.70% ±0.13% •	96.22% ±0.33% •	98.05% ±0.06% •	97.88% ±0.15% •	97.65% ±0.15% •	98.03% ±0.08% •	97.74% ±0.14% •	96.40% ±0.24% •

DIOS is: • significantly better, ≡ equivalent, ◦ significantly worse, p-value: 0.05

Table 2.2: Experimental results on an imputation task on the benchmark dataset MFEAT with mechanisms MCAR/MAR/MNAR, at 25/50/75% missing rates.

Results for the MFEAT dataset can be seen in Table 2.2. DIOS leads to significantly better inference results in almost all cases compared to other imputation methods. RMSE results are significantly better in many cases, except compared to MICE and MISSFOREST methods that lead to better RMSE results. Better RMSE results mean that the imputation method was

able to provide imputation values closer from the reality. We believe that the better inference results, but worse RMSE results, of DIOS, compared to MICE and MISSFOREST, is mainly due to the erroneous values correction that DIOS performs in addition to imputation. Indeed, if DIOS slightly modifies original values to improve the quality of the dataset, the RMSE value naturally increases. To better assess RMSE results of all methods on the MFEAT dataset, we display their results on a bar plot, in Figure 2.4. As can be seen from this graph, while MICE and MISSFOREST lead to lower RMSE values, DIOS stands quite close behind, and more importantly, other state-of-the-art imputation methods lead to very high RMSE values. This seems to illustrate and confirm what was stated by Stef van Buuren and indicated in section 1.2.4.1: that the RMSE metric is not an informative metric for evaluating imputation methods. For the sake of conformity, we still use the metric as an indication in those experiments, but we consider that inference metrics are a much more important indicator of good performance.

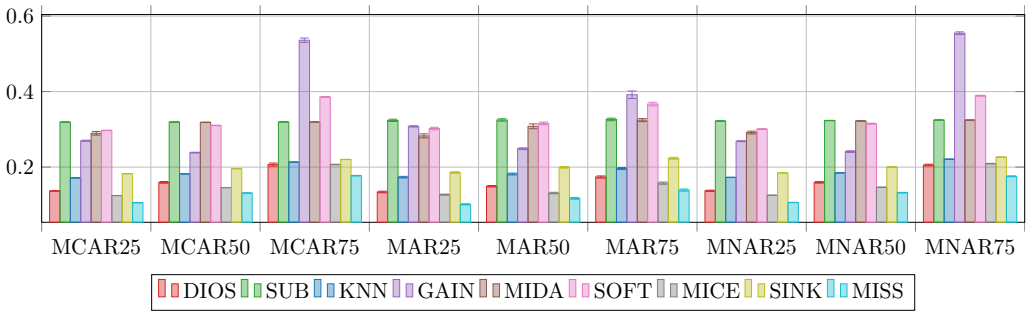


Figure 2.4: RMSE results on an imputation task on the benchmark dataset MFEAT with mechanisms MCAR/MAR/MNAR, at 25/50/75% missing rates.

We also experiment on benchmark datasets with low numbers of features (STATLOG with 14 and PIMA with 8) to show that DIOS is also able to perform well on small datasets. Results on the STATLOG dataset can be found in Appendix A, in Table A.2. As can be seen, results are mitigated, DIOS leads to better predictions in most cases and worse RMSE results in most cases. It is interesting to note that the simple SUB imputation method leads to lower RMSE values than many state-of-the-art methods while leading to poorer predictions, which seems to confirm that the RMSE metric is less pertinent for evaluating data imputation quality. Figure 2.5 shows the accuracies of each imputation method in all missingness settings on the PIMA dataset. As can be seen from this graph, DIOS leads to excellent predictive results on PIMA, significantly better than all other imputation methods in any setting. Those results on the PIMA dataset demonstrate the capacity of DIOS to lead to high quality imputations on datasets with few features.

The ARRHYTHMIA dataset contains 260 features, which is a large number of features for most fields, but quite a regular amount for medical datasets. Figure 2.6 shows the accuracies obtained on a data imputation task on this dataset. DIOS obtains better accuracies than most methods in almost all cases. In cases with a lot of missing values (75%), SINKHORN and MISSFOREST algorithms obtain similar results to DIOS in terms of accuracies.

Table 2.3 shows our imputation experimental results on the three real-world medical datasets. DIOS leads to very competitive prediction results on all datasets, and leads to sig-

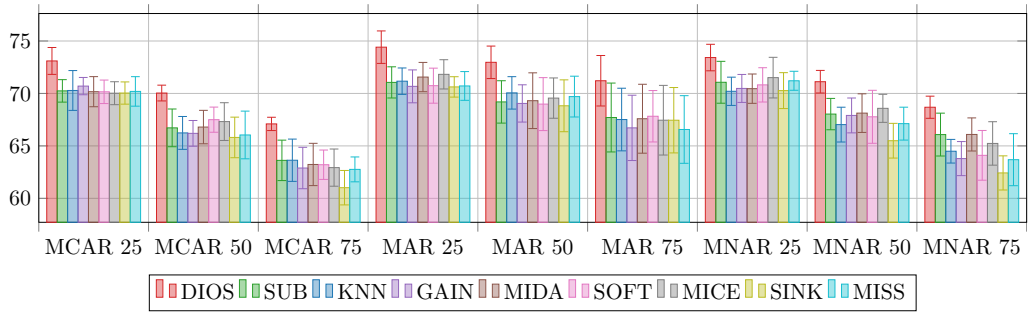


Figure 2.5: Accuracy results on an imputation task on the benchmark dataset PIMA with mechanisms MCAR/MAR/MNAR, at 25/50/75% missing rates.

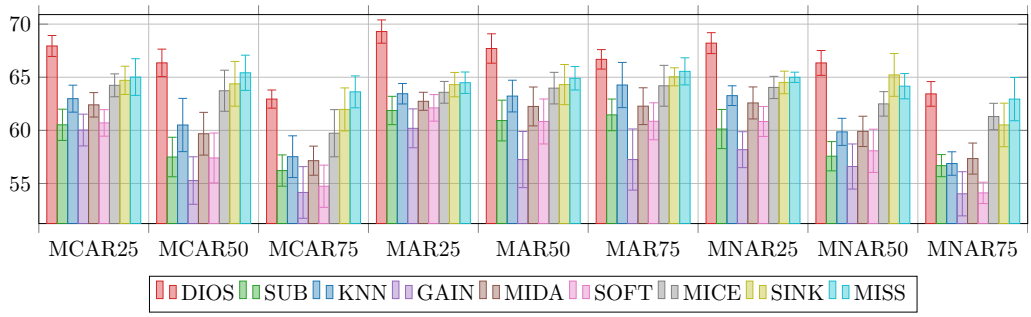


Figure 2.6: Accuracy results on an imputation task on the benchmark dataset Arrhythmia with mechanisms MCAR/MAR/MNAR, at 25/50/75% missing rates.

nificantly better inference results than all other compared methods in the vast majority of cases. The only cases in which DIOS performs significantly worse are compared to KNN and MICE on COVID data on the balanced Accuracy metric. This experiment demonstrates the capacity of DIOS to compete and even outperform other state-of-the-art imputation methods on real-world medical mixed-type tabular data.

To conclude, when it comes to missing values imputation tasks, DIOS is able to provide imputations that give drastically better predictive results in practice than any other method, while RMSE results were sometimes worse than other methods. As discussed, we consider that inference metrics are a much more important indicator of good performance, as an imputation can be evaluated as excellent in regard to RMSE while leading to very bad inference results, as can be seen in the case of the SUB method on the STATLOG dataset.

2.4.2.3 Comparative Study on an Erroneous Values Correction Task

With this next experiment, we aim to evaluate our DIOS method on an erroneous values correction task against other correction methods from the literature. We evaluate the methods on benchmark datasets only in this experiment as the three real-world medical datasets contain missing values, which are not handled by correction methods from the literature. Complete results for this experiment can be found in the Appendix A: Tables A.7, A.8, A.9, A.10, A.11.

Table 2.4 shows the obtained Accuracy and RMSE values for DIOS and each correction

Model	Metric	MYOCARDIAL		NHANES		COVID	
DIOS	bACC	**77.91%** **±1.12%**		**64.17%** **±0.36%**		86.84% ±1.23%	
	AUC	**86.28%** **±0.42%**		**69.92%** **±0.56%**		**92.95%** **±0.93%**	
SUB	bACC	77.30% ±0.00%	≡	60.35% ±0.00%	•	85.91% ±0.00%	•
	AUC	85.09% ±0.00%	•	66.10% ±0.00%	•	91.20% ±0.00%	•
KNN	bACC	68.83% ±0.00%	•	63.00% ±0.00%	•	**88.08%** **±0.00%**	○
	AUC	78.94% ±0.00%	•	67.78% ±0.00%	•	91.53% ±0.00%	•
GAIN	bACC	63.89% ±2.21%	•	61.36% ±0.53%	•	85.14% ±0.91%	•
	AUC	74.22% ±1.11%	•	66.85% ±0.40%	•	91.36% ±0.73%	•
MICE	bACC	76.55% ±0.00%	•	61.70% ±0.00%	•	87.98% ±0.00%	○
	AUC	81.39% ±0.00%	•	67.30% ±0.00%	•	92.43% ±0.00%	≡
MISS	bACC	73.00% ±0.87%	•	61.40% ±1.03%	•	85.15% ±1.67%	•
	AUC	80.82% ±1.60%	•	66.48% ±0.90%	•	91.30% ±1.20%	•
SOFT	bACC	77.24% ±0.99%	≡	61.70% ±0.93%	•	84.48% ±0.78%	•
	AUC	84.88% ±0.77%	•	66.93% ±1.08%	•	91.12% ±0.85%	•
SINK	bACC	75.66% ±1.22%	•	60.77% ±0.98%	•	86.82% ±1.49%	≡
	AUC	83.26% ±1.01%	•	65.42% ±1.18%	•	91.48% ±1.13%	•
MIDA	bACC	75.09% ±0.70%	•	62.15% ±1.26%	•	85.55% ±1.12%	•
	AUC	82.87% ±0.78%	•	66.91% ±1.30%	•	91.67% ±0.62%	•

DIOS is: • significantly better, ≡ equivalent, ○ significantly worse, p-value: 0.05

Table 2.3: Experimental results on an imputation task on three real-world medical mixed-type tabular datasets.

method we tested, on an erroneous values correction task on the STATLOG dataset. The model NONE corresponds to baseline results without applying any correction to the corrupted data. In bold are the best obtained results for each metric at each rate. RMSE values cannot be computed with filtering methods: as they just remove instances from the training set, they do not change any values in the corrupted dataset. We can see that Filtering seems to lead to slightly better inference results than Polishing in most cases. For this dataset, PANDA does not seem to identify noisy instances consistently. Indeed, prediction results on the corrupted data are better before applying PANDA-Filtering (PFIL) or PANDA-Polishing (PPOL) than afterwards. DIOS leads to significantly better Accuracy results than any other method in almost all cases, with the exception of Standard-Filtering (SFIL) and Standard-Polishing (SPOL) at a rate of 60%, that obtained results close to those of DIOS, while remaining slightly worse. This shows that our method is able to drastically improve the prediction potential on this dataset, even when we add no erroneous values to the data. This last point seems to show that this dataset naturally contains erroneous values that DIOS is able to reduce. When erroneous values rates are low, DIOS does not improve the RMSE compared to when no correction is applied. However as the rate increases, DIOS RMSE results improve. This can be accounted for by the fact that the dataset contains some natural erroneous val-

ues. Thus, when we don't add erroneous values, or add a few of them, DIOS is able to correct most of the artificial and natural erroneous values, leading to cleaner data than the original. Since the RMSE metric only accounts for differences between the original data and the correction from DIOS, at low rates the RMSE is higher with DIOS correction.

Model	Metric		NOISE 0		NOISE 0.05		NOISE 0.1		NOISE 0.15		NOISE 0.2		NOISE 0.4		NOISE 0.6	
DIOS	RMSE		.114708 ±.002194		.137553 ±.001855		.162571 ±.005927		**.186799 ±.008194**		**.211346 ±.007440**		**.277133 ±.002640**		**.332875 ±.004374**	
	ACC		**87.48% ±0.25%**		**86.29% ±0.54%**		**85.25% ±0.70%**		**83.29% ±0.97%**		**81.90% ±1.09%**		**75.78% ±1.75%**		**67.55% ±1.12%**	
NONE	RMSE	≡	**.000000 ±.000000**	∘	**.111065 ±.003219**	∘	**.157520 ±.002874**	∘	.193922 ±.001858	•	.223263 ±.001812	•	.311895 ±.003253	•	.383534 ±.003808	•
	ACC	•	84.06% ±0.00%	•	83.80% ±0.73%	•	82.22% ±1.02%	•	80.88% ±1.38%	•	78.78% ±1.73%	•	71.13% ±1.63%	•	63.46% ±2.27%	•
SFIL	RMSE		- -		- -		- -		- -		- -		- -		- -	
	ACC	•	85.33% ±0.67%	•	84.10% ±0.58%	•	82.78% ±1.25%	•	81.03% ±1.96%	•	79.33% ±1.51%	•	72.93% ±0.94%	•	66.19% ±1.94%	≡
SPOL	RMSE	•	.130674 ±.000920	•	.168412 ±.002665	•	.199776 ±.002339	•	.226227 ±.002091	•	.249161 ±.002513	•	.320054 ±.003182	•	.378386 ±.004059	•
	ACC	•	85.12% ±0.60%	•	83.86% ±0.71%	•	82.75% ±1.06%	•	81.23% ±1.39%	•	79.58% ±1.39%	•	73.84% ±1.16%	•	66.96% ±1.56%	≡
PFIL	RMSE		- -		- -		- -		- -		- -		- -		- -	
	ACC	•	84.75% ±0.61%	•	83.46% ±0.76%	•	82.07% ±0.84%	•	80.29% ±1.05%	•	78.43% ±1.64%	•	71.25% ±1.55%	•	62.78% ±1.27%	•
PPOL	RMSE	≡	.103064 ±.000793	∘	.156244 ±.002162	•	.187602 ±.003251	•	.215673 ±.001799	•	.238741 ±.001937	•	.313380 ±.003438	•	.376681 ±.003125	•
	ACC	•	84.87% ±0.66%	•	83.77% ±0.63%	•	82.03% ±0.55%	•	80.41% ±1.49%	•	78.49% ±1.43%	•	70.90% ±1.12%	•	62.28% ±1.24%	•

DIOS is: • significantly better, ≡ equivalent, ∘ significantly worse, p-value: 0.05

Table 2.4: Experimental results on a correction task on the benchmark dataset STATLOG, with erroneous values rates 0/5/10/15/20/40/60%.

Figure 2.7 shows the Accuracy and RMSE values obtained at various erroneous values rates for the ARRHYTHMIA dataset. On the left are the Accuracy values, the higher the curve the better it is, while on the right are the RMSE values, the lower the curve the closer from clean data the data correction is. The black curve corresponds to the results obtained on the original corrupted data: most methods struggle to procure satisfying results as they are mostly under the black curve on the left. On this dataset, DIOS obtains significantly better results than all other methods in both Accuracy and RMSE values. As can be seen on the left plot, DIOS leads to inference results that are drastically higher than any other method, even when no artificial erroneous values has been added. This seems to show that the ARRHYTHMIA dataset naturally contains a lot of noise and erroneous values. DIOS is able to correct those values and maximize the data quality of the corrected dataset.

We were not able to execute the PANDA algorithm for both MFEAT and ORL datasets, as they contain too many features. The complexity of PANDA being $O(d^2 n)$ (Van Hulse et al., 2007), where d is the number of features and n of instances, the execution time of the algorithm is too long with those datasets. For the same reason we were unable to perform Polishing on the ORL dataset. For the MFEAT dataset, DIOS obtained significantly better results than SFIL and SPOL for both accuracies and RMSE values. For the ORL dataset, we could only compare DIOS to SFIL: our method obtained significantly better results than SFIL at any rate.

Our experiments showed that the Polishing method mostly obtained better results than

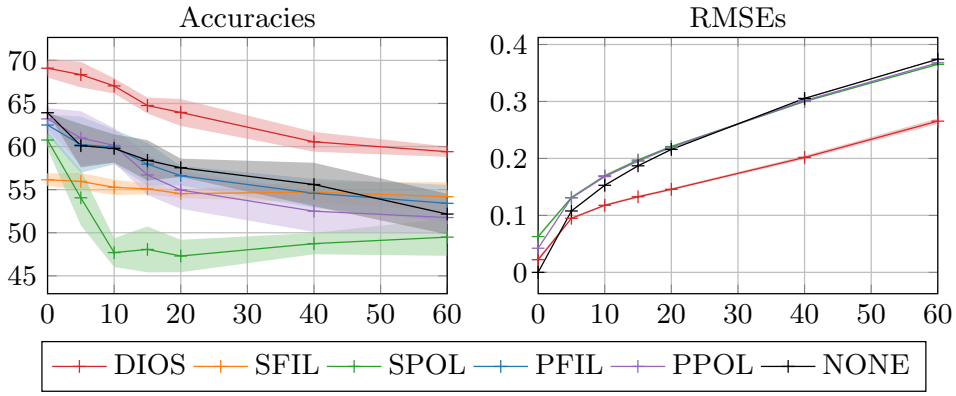


Figure 2.7: RMSE and Accuracy evolution on a correction task on the benchmark dataset ARRHYTHMIA, with erroneous values rates 0/5/10/15/20/40/60%.

Filtering methods at low erroneous values rates, though their results deteriorate considerably as erroneous values rate increases, unlike Filtering methods. Globally, Filtering methods obtained good results, especially on "smaller" datasets like STATLOG, PIMA and ARRHYTHMIA. We can see from our results that DIOS obtains better results than other correcting methods in almost any case, and is able to improve data quality with low to high rates alike.

2.4.2.4 Comparative Study on a Complete Attribute Noise Correction Task

With this third experiment, we aim to evaluate our DIOS method on a complete attribute noise correction task, first, compared with imputation methods only, and then, compared with the combination of imputation and erroneous values correction methods from the literature. We perform this experiment on the three real-world medical datasets to best reflect our specific application context.

We artificially add erroneous values to the already incomplete real-world data at various rates: 0/5/10/15/20/40/60%, and evaluate each imputation method at each rate. Figure 2.8 shows AUC results obtained on NHANES and MYOCARDIAL datasets at each rate against several imputation methods. In both cases, we note that our method globally obtains significantly better results than other methods. The performance of all methods drops when the erroneous values rate increases, which is expected. On NHANES data, our method performs largely better than others, until a noise rate of 60%, where the SUBSTITUTION imputation gets similar results to ours. This can probably be explained by the fact that, with a rate that high, it is nearly impossible to impute coherent values other than the median or mean value for each feature. We can observe the same pattern on MYOCARDIAL data, with the difference that GAIN seems to have learned how to adapt to such an amount of erroneous values in this case. Those results show that on low to high rates, our method can impute missing values while correcting erroneous values. It provides better data correction than most other methods. At extreme rates naive methods might provide better results.

The second step is to compare our method results to those obtained from the combination of an imputation method, followed by a correction method. We chose MICE as the state-of-the-art imputation method, since it obtains competitive results against ours in a not noisy

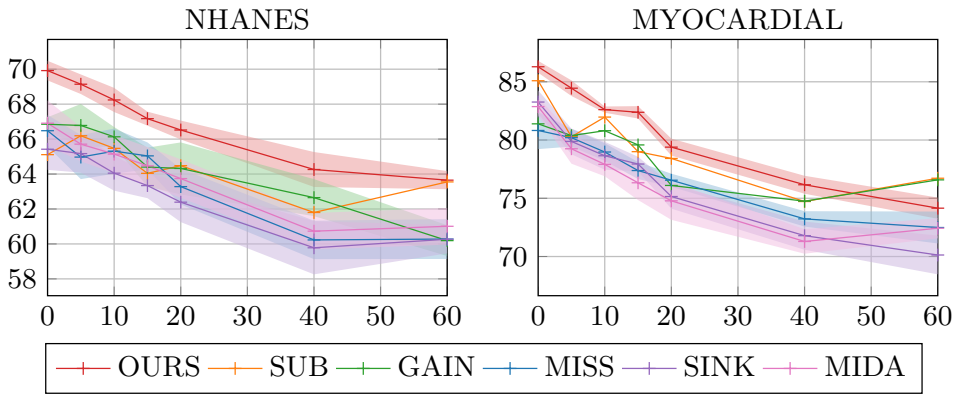


Figure 2.8: AUC results on NHANES and MYOCARDIAL datasets for several imputation methods, at various erroneous values rates compared to DIOS.

context. We then apply the four noise correction methods SFIL, PFIL, SPOL, and PPOL. Figure 2.9 shows AUC results obtained on COVID and MYOCARDIAL datasets at each rate. We note that SFIL and SPOL perform worse than the PANDA alternative of both those methods at all rates. We also note that for both datasets the other state-of-the-art correction methods give very poor results when the rate reaches more than 5%, at higher rates the data quality is better before correction than after. For COVID data, all methods yield similar results at low erroneous values rates, with DIOS on top with a very slight advantage. At high rates, however, DIOS leads to significantly better results than all other methods. For MYOCARDIAL data, the opposite pattern can be observed, our method gives significantly better results up until a noise rate of 40%, after which MICE imputation is slightly better. This experiment completes the conclusions drawn from the previous one, DIOS provides very good data correction, up until the erroneous values rate becomes too extreme, at that point, simpler methods can achieve better results. The fact that the opposite is observed on COVID data is probably due to a remarkable original data quality, which would explain why DIOS becomes significantly better only at higher noise levels.

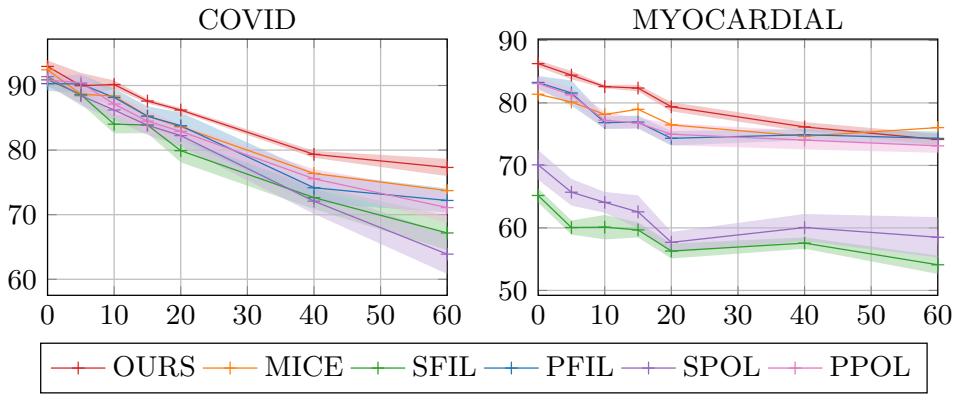


Figure 2.9: AUC results on COVID and MYOCARDIAL datasets for a pipeline of MICE and each correction method at various erroneous values rates compared to DIOS.

2.4.2.5 Running-Time Comparison

We performed an empirical comparison of the running-times of all tested methods on all used datasets. Table 2.5 recapitulates our entire results.

	Erroneous Values Correction					Missing Values Imputation							
Method	DIOS	SFIL	PFIL	SPOL	PPOL	DIOS	KNN	GAIN	MIDA	SOFT	MICE	SINK	MISS
MFEAT	02:03	00:01	-	13:01	-	02:03	00:13	02:08	00:47	00:01	20:10	01:48	58:20
ARRHYTHMIA	00:58	00:00	11:26	00:13	12:54	00:58	00:01	01:07	00:38	00:01	00:56	01:45	10:20
PIMA	00:06	00:00	00:01	00:00	00:01	00:06	00:00	00:24	00:35	00:01	00:00	01:22	00:26
STATLOG	00:14	00:00	00:03	00:00	00:03	00:14	00:00	00:23	00:36	00:00	00:00	01:23	00:47
ORL	16:08	00:02	-	-	-	16:08	00:17	-	05:26	00:01	-	02:10	-
NHANES	00:13	00:01	07:52	00:06	07:50	00:13	00:02	00:59	00:25	00:00	00:16	01:39	02:28

Table 2.5: Average running-time comparison between all tested methods on each dataset, on an erroneous values correction task on the left and an imputation task on the right. The format is “minutes:seconds”, the symbol “-” means that the method could not be executed in this setting.

DIOS, GAIN, SINKHORN and MIDA methods are based on gradient descent optimization and have benefited from GPU acceleration, while all other methods were executed on CPUs. Running-time of the DIOS method remains identical between erroneous values correction and imputation task as the training process is the same in both cases. Therefore, for a same dataset and with identical hyper-parameters, DIOS execution time is the same for both tasks.

On an erroneous values correction task, we note that standard Filtering is very fast, with a running-time of a few seconds at most on the largest dataset. As this method is based on a KNN method to filter out outliers, we easily understand that it can scale well in terms of number of features, but will struggle as the amount of samples in the dataset increases. The standard Polishing method is also based on a KNN method and additionally trains as many regression models as there are features in the dataset, which explains why its running-time gets so high on the Mfeat dataset. For this reason, this method could not be executed on the ORL dataset. Both variants using the PANDA algorithm in place of the KNN could not be executed on both Mfeat and ORL. That is because the PANDA algorithm needs to compute a noise score for each instance in the dataset by comparing each pair of features, which does not scale well when the amount of features increases.

On a missing values imputation task, we see that the KNN imputation algorithm obtains fast results in all cases. GAIN and MIDA methods are the most comparable to DIOS in terms of running-times as they are all based on Neural Networks. GAIN obtains extremely similar running-times to our method, it could not be executed on our machine for the ORL dataset because of an out of memory error, but it would certainly lead to a similar running-time as DIOS in this case too if it was executable on our available hardware. Our method takes longer to run than MIDA in some cases as we perform regular supervised validation to chose an early stopping step that maximizes the quality of the obtained imputation. This allows our method to reach better results than MIDA at the cost of a more demanding computational process. SOFTIMPUTE obtains extremely fast results, but as can be noted from our experimental results, it can lead to bad results when the missing rate is too high. Some methods do not scale well when the amount of features gets too large, it is the case of methods MICE and MISSFOREST. That is because both those models rely on iteratively training one logis-

tic model for each feature in the dataset, which does not scale well for datasets with lots of features. For this reason, both those methods could not be executed in the case of the ORL dataset. Finally, SINKHORN running-times do not vary much with the amount of features in the dataset, it is slower than other methods on small datasets but quite fast on larger datasets.

As can be seen from our results, DIOS scales well when the amount of features in the dataset drastically increases. As our method is based on a Neural Network it also naturally scales well as the amount of samples in the dataset increases.

2.4.3 Evaluating our Imputation Frameworks S-HOT and M-HOT

Secondly, we evaluate the pertinence and usefulness of our proposed imputation frameworks S-HOT and M-HOT.

2.4.3.1 Experimental Protocol

We apply the frameworks using five different state-of-the-art imputation methods to demonstrate that our frameworks yield good results and are pertinent to use to improve generalization of Neural Networks in a context with missing values imputed with any stochastic imputation method. We experiment with each imputation framework by training NNs using each stochastic state-of-the-art imputation method previously used in our experiments with DIOS. Those are: MISSFOREST, SOFTIMPUTE, GAIN, MIDA, and SINKHORN. In our experiments we treat MICE as an imputation framework, to which our frameworks results are compared. Indeed, as described earlier, MICE perform Multiple-Imputation within its own algorithm, it can be considered both an imputation method and an imputation framework. In our experiments, we are not interested in comparing the performances of the imputation methods, instead, we are interested in the results we can observe for the same imputation method when applying each of the compared imputation frameworks.

As previously, missing values are artificially introduced in benchmark datasets with each combination between MCAR/MAR/MNAR mechanisms and missing rates of 10/15/25%. Our experiments aim to evaluate NN performances in a supervised learning setting to check the bias and generalization capacity of the model. We use a simple Neural Network with the scikit-learn library[20], parameterized using well-performing and identical hyper-parameters for each dataset to ensure a fair and unbiased comparison. We used the Adam optimizer with a default learning rate of 0.001. The used NNs are composed of two fully-connected layers in all our experiments, for datasets IRIS, STATLOG, PIMA and ABALONE both layers contains 32 units, for WINE, MYOCARDIAL and NHANES layers contain 64 and 32 units respectively, for the COVID dataset layers are composed of 128 and 32 units. We evaluate NN inference performance with the following classification metrics: balanced Accuracy, AUC and F_1-score.

We first perform a comparative study on benchmark datasets between the four frameworks SI, MI, S-HOT, and M-HOT, using different imputation methods. We show that, no

[20] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

matter the used imputation method and missingness setting, our hotpatching frameworks lead to consistent and good results. In each missingness setting, we ran 20 imputations using each of the tested imputation methods and then ran the four frameworks 200 times, each run using a different random seed and train-test set to ensure the results are not biased by the stochastic nature of NN training. Secondly, we compare our results to those of MICE, an imputation method that takes into account imputation uncertainty within its own algorithm, which we consider as a framework. We then experiment on three real-world medical datasets containing missing values to evaluate our frameworks results in a real situation. Finally, we compare the required running time for each framework.

To better assess the obtained results we base our comparisons on the Friedman and Nemenyi statistical tests such as described in (Demsar, 2006). The Friedman test is first used to check if the null hypothesis that all compared frameworks are statistically equivalent for a given p-value is rejected or not. It ranks the compared frameworks for each dataset from best to worst, ties are assigned an average rank. The average rank for each compared framework is computed over all datasets and the Friedman test checks whether the measured average ranks are significantly different from the mean rank by computing the Friedman statistic:

$$\chi_F^2 = \frac{12N}{k(k+1)}\left(\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4}\right) \tag{2.5}$$

derived in:

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2} \tag{2.6}$$

The Friedman statistic is distributed according to the F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom, with k the number of frameworks compared and N the number of datasets. If the Friedman statistic is above the critical value of $F(k-1,(k-1)(N-1))$ given the p-value, then the test shows a significant difference between the compared frameworks. If that is the case, a post-hoc test is performed. We use the post-hoc Nemenyi test to compare the frameworks to each other in a pairwise manner, two frameworks are considered significantly different if their average ranks differ by more than the computed critical distance:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{2.7}$$

where α is the p-value, q_α comes from (Demsar, 2006). The Nemenyi test can be easily visualized through a simple diagram, which makes its results easy to analyze.

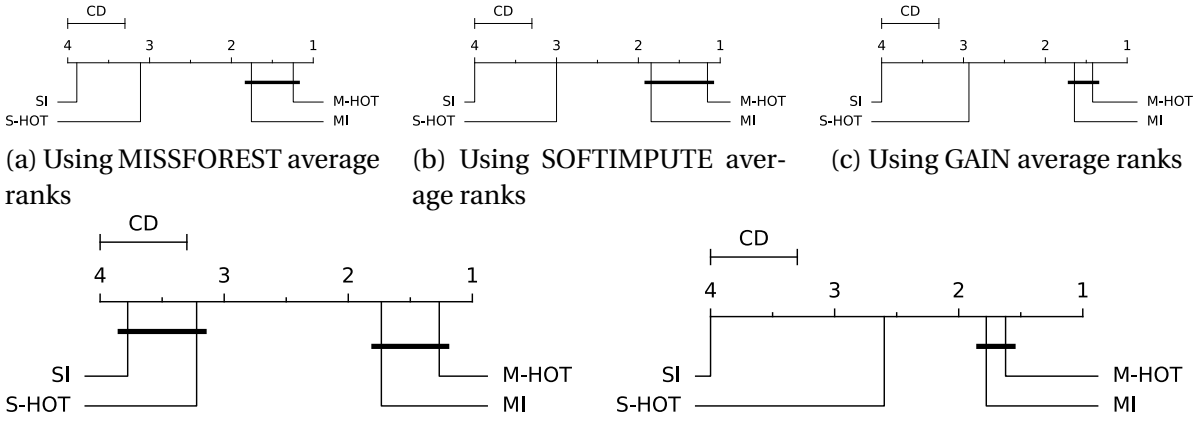
We choose to evaluate our experimental results with the Friedman and Nemenyi tests as they are relevant to statistically assess the superiority of an approach based on the consistency of the results. That is, those tests assess how many times an approach leads to better results than another, even if the obtained results are close from one another. In the case of results too close, a t-test would always conclude that the two approaches are equivalent, even if an approach consistently leads to better results than the other. This is why it makes more sense to use the Friedman and Nemenyi tests to assess the results in this context.

2.4.3.2 Results on Benchmark Datasets

Complete results for this experiment can be found in the Appendix B: Tables B.1, B.2, B.3, B.4, B.5.

The first step of this experiment is to compute $m = 20$ imputations with each tested imputation method on each dataset at each previously described missingness setting. Once all imputations have been computed, we run the four compared frameworks SI, MI, S-HOT and M-HOT, following the previously described experimental protocol. We base our analysis for this experiment on the AUC experimental results. Table 2.6 shows AUC results when applying the four compared frameworks with imputations obtained with the MISSFOREST method. We do not observe any influence from the missingness mechanism or missing rate on the obtained results, which seems to show that our frameworks are not sensitive to those parameters, and so, can be used in any circumstances. We note that the M-HOT framework obtains the best results in the vast majority of cases, while S-HOT consistently achieve better results than SI.

We use the Friedman and Nemenyi tests such as described in our protocol to statistically assess our empirical results. Using equations 2.5 and 2.6, we compute the Friedman statistic on the average ranks for the results obtained using the MISSFOREST imputation method, we obtain $F_F \approx 344.07$. The number of compared frameworks is $k = 4$, and number of experimental settings is $N = 5 \cdot 9 = 45$ for 5 datasets with 9 missingness settings for each. Under a significance level of 0.05 the critical value of $F(3,132)$ is 2.673, the null-hypothesis is rejected since in our case $F_F > F(3,132)$. We reiterate the same process to compute the Friedman statistic for each of the other imputation methods used, and find that the null hypothesis is rejected in all cases. Thus, we continue with the post-hoc Nemenyi test as described in our protocol. We check the significant difference under a p-value of 0.05, given those values we have $q_{0.05} = 2.569$ from table 5 in (Demsar, 2006). We find $CD \approx 0.6992$, meaning that two frameworks can be considered as significantly different if their average ranks differ by more than 0.6992. We repeat the exact same process with the four other used imputation methods to obtain five Nemenyi graphs.



(a) Using MISSFOREST average ranks

(b) Using SOFTIMPUTE average ranks

(c) Using GAIN average ranks

(d) Using MIDA average ranks

(e) Using SINKHORN average ranks

Figure 2.10: Nemenyi tests comparing SI, MI, S-HOT and M-HOT frameworks using each tested imputation method.

Dataset	Pattern		*SI*		*S-HOT*		*MI*		*M-HOT*	
IRIS	MCAR	10%	0.9972681	(4)	**0.9973865**	(3)	0.9976699	(2)	**0.9976913**	(1)
		15%	0.9942281	(4)	**0.9944422**	(3)	0.9946880	(2)	**0.9947552**	(1)
		25%	0.9835323	(4)	**0.9839692**	(3)	0.9841479	(2)	**0.9843596**	(1)
	MAR	10%	**0.9975236**	(3)	0.9974886	(4)	0.9978336	(2)	**0.9978845**	(1)
		15%	**0.9970188**	(3)	0.9970006	(4)	**0.9973150**	(1)	0.9973097	(2)
		25%	0.9941960	(4)	**0.9943554**	(3)	0.9943643	(2)	**0.9943890**	(1)
	MNAR	10%	0.9972209	(4)	**0.9972343**	(3)	**0.9973548**	(1)	0.9973011	(2)
		15%	0.9937917	(4)	**0.9938451**	(3)	0.9941570	(2)	**0.9941622**	(1)
		25%	0.9891964	(4)	**0.9902221**	(3)	0.9905233	(2)	**0.9907251**	(1)
STAT	MCAR	10%	0.9105582	(4)	**0.9106386**	(3)	**0.9132492**	(1)	0.9132475	(2)
		15%	0.9060337	(4)	**0.9066862**	(3)	0.9091019	(2)	**0.9091584**	(1)
		25%	0.9047888	(4)	**0.9059597**	(3)	0.9077214	(2)	**0.9078041**	(1)
	MAR	10%	**0.9127799**	(3)	0.9127328	(4)	**0.9144097**	(1)	0.9142985	(2)
		15%	**0.9079620**	(3)	0.9078667	(4)	**0.9093156**	(1)	0.9092031	(2)
		25%	0.8959471	(4)	**0.8968849**	(3)	0.8990894	(2)	**0.8992881**	(1)
	MNAR	10%	0.9070192	(4)	**0.9071004**	(3)	0.9095510	(2)	**0.9095576**	(1)
		15%	0.9040681	(4)	**0.9042151**	(3)	**0.9061872**	(1)	0.9060699	(2)
		25%	0.8951658	(4)	**0.8967363**	(3)	0.8992252	(2)	**0.8992736**	(1)
WINE	MCAR	10%	0.9987736	(4)	**0.9988008**	(3)	0.9989672	(2)	**0.9989811**	(1)
		15%	0.9955454	(4)	**0.9956407**	(3)	0.9960062	(2)	**0.9960307**	(1)
		25%	0.9910498	(4)	**0.9914148**	(3)	0.9919927	(2)	**0.9923485**	(1)
	MAR	10%	0.9961058	(4)	**0.9961142**	(3)	0.9965056	(2)	**0.9965060**	(1)
		15%	0.9977720	(4)	**0.9978677**	(3)	**0.9982010**	(1)	0.9981809	(2)
		25%	0.9952116	(4)	**0.9959576**	(3)	0.9965157	(2)	**0.9968702**	(1)
	MNAR	10%	**0.9987205**	(3)	0.9987058	(4)	**0.9988244**	(1)	0.9988131	(2)
		15%	0.9974746	(4)	**0.9974850**	(3)	0.9976465	(2)	**0.9976683**	(1)
		25%	0.9808498	(4)	**0.9822719**	(3)	0.9841291	(2)	**0.9844587**	(1)
PIMA	MCAR	10%	0.8193054	(4)	**0.8196586**	(3)	**0.8211340**	(1)	0.8211193	(2)
		15%	0.8073739	(4)	**0.8078378**	(3)	0.8095505	(2)	**0.8095824**	(1)
		25%	0.8029002	(4)	**0.8043589**	(3)	0.8060367	(2)	**0.8065611**	(1)
	MAR	10%	0.8238900	(4)	**0.8242472**	(3)	**0.8257089**	(1)	0.8256414	(2)
		15%	0.8045918	(4)	**0.8061503**	(3)	0.8080830	(2)	**0.8083194**	(1)
		25%	0.8017568	(4)	**0.8025144**	(3)	**0.8041203**	(1)	0.8040062	(2)
	MNAR	10%	0.8280685	(4)	**0.8284115**	(3)	0.8302729	(2)	**0.8303076**	(1)
		15%	0.8279577	(4)	**0.8283792**	(3)	0.8298929	(2)	**0.8300464**	(1)
		25%	0.8005008	(4)	**0.8021749**	(3)	0.8043448	(2)	**0.8047250**	(1)
ABAL	MCAR	10%	0.8737739	(4)	**0.8740180**	(3)	0.8748059	(2)	**0.8749393**	(1)
		15%	0.8714861	(4)	**0.8717831**	(3)	0.8725539	(2)	**0.8726250**	(1)
		25%	0.8663833	(4)	**0.8666186**	(3)	0.8674332	(2)	**0.8675645**	(1)
	MAR	10%	0.8742551	(4)	**0.8743399**	(3)	0.8751972	(2)	**0.8753671**	(1)
		15%	0.8720722	(4)	**0.8721856**	(3)	0.8731502	(2)	**0.8731739**	(1)
		25%	0.8697060	(4)	**0.8699619**	(3)	0.8708046	(2)	**0.8709102**	(1)
	MNAR	10%	0.8760444	(4)	**0.8760447**	(3)	0.8768033	(2)	**0.8769350**	(1)
		15%	0.8753217	(4)	**0.8754605**	(3)	0.8763271	(2)	**0.8764456**	(1)
		25%	0.8683507	(4)	**0.8690281**	(3)	0.8696780	(2)	**0.8697714**	(1)
Average rank			3.8889		3.1111		1.7556		1.2444	

Table 2.6: AUC results when applying imputation frameworks SI, MI, S-HOT and M-HOT using the MISSFOREST imputation method on benchmark datasets. Bold values are the best results between pairs of frameworks SI vs S-HOT, and MI vs M-HOT.

Figure 2.10 shows the obtained Nemenyi results for the comparison between SI, MI, S-HOT and M-HOT frameworks, using each of the five tested imputation methods. The critical distance for each imputation method is visualized on the top left corner of each diagram, two frameworks can be considered significantly different if they are not linked by the same black bar. We can see that in all cases the ordering of the frameworks from worst to best is the same, SI performs the worst, then S-HOT, followed by MI, and finally M-HOT performs the best. S-HOT obtains significantly better results than SI in four cases out of the five, showing that it is a good alternative to SI when one wants to train a unique large model. We note that M-HOT obtains consistently better results than MI despite not being significantly better, which seems to show that it is always a good and viable alternative to MI.

This experiment shows that our frameworks are not sensible to the missingness setting, neither in the missingness mechanism nor in the rate of missing values. It shows that our frameworks give consistent results no matter the imputation method used for the imputations. We can conclude that S-HOT and M-HOT are both viable, and almost always better, alternatives to SI and MI respectively.

2.4.3.3 Comparative Study Between MICE and Imputation Frameworks.

In this experiment, we consider the MICE method as an imputation framework, as multiple imputations are performed within the method algorithm. We perform the same experiment as above on the benchmark datasets and with each missingness setting, the best performing imputation method from the previous experiment is chosen and used for each dataset. Table 2.6 shows AUC results when applying MICE and the four compared frameworks with the experimental protocol previously described. At the bottom of the table are the average ranks used to compute the Friedman statistic, and in bold are the highest results on each line. As previously, we observe that the M-HOT framework leads to the best inference results in most cases, while S-HOT leads to better results than SI and MICE in most cases.

We reiterate the same calculations as in the first experiment to check if the Friedman test rejects the null-hypothesis that all compared frameworks and methods are equivalent. We find that in this case $F_F \approx 81.74$, we have $k = 5$ for four frameworks plus MICE and $N = 5 \cdot 9 = 45$. Under a p-value of 0.05 the critical value of $F(4,176)$ is 2.423, since $F_F > F(4,176)$ we find that the compared frameworks and methods are not equivalent.

As before, we use the post-hoc Nemenyi test, we compute $CD \approx 0.9093$, figure 2.11 shows the obtained Nemenyi results for comparison between the four frameworks and MICE. We note that MICE performs better than the SI framework, despite not being statistically different. Our S-HOT framework leads to significantly better inference results than SI and slightly better results than MICE. Both MI and M-HOT frameworks are significantly better than the remaining tested frameworks and methods, once again M-HOT performs better than MI while both frameworks are statistically equivalent.

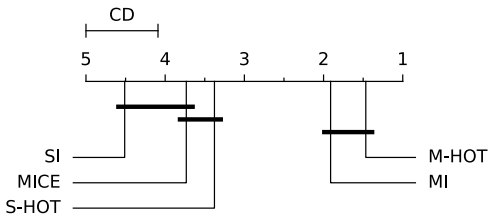


Figure 2.11: Nemenyi test comparing the MICE imputation method with the best imputation method for each dataset, $CD \approx 0.9093$.

This second experiment shows that our frameworks compete and even outperform MICE, a very widely used, and one of the very few imputation methods that takes imputation uncertainty into account.

Dataset	Pattern		MICE		Best Imputation Method							
					SI		*S-HOT*		*MI*		*M-HOT*	
IRIS	MCAR	10%	0.9964543	(4)	0.9958323	(5)	0.9968544	(3)	**0.9973218**	(1)	0.9972525	(2)
		15%	0.9916368	(5)	0.9947010	(4)	0.9961452	(3)	0.9967011	(2)	**0.9967655**	(1)
		25%	0.9878213	(5)	0.9879101	(4)	0.9920251	(3)	0.9931490	(2)	**0.9933714**	(1)
	MAR	10%	0.9963453	(5)	0.9977831	(4)	0.9980598	(3)	**0.9983718**	(1)	0.9983512	(2)
		15%	0.9968731	(3)	0.9964883	(5)	0.9966728	(4)	0.9969672	(2)	**0.9970707**	(1)
		25%	**0.9972507**	(1)	0.9934700	(5)	0.9937325	(4)	0.9940230	(3)	0.9940345	(2)
	MNAR	10%	0.9973377	(3)	0.9968828	(5)	0.9969866	(4)	0.9975337	(2)	**0.9975402**	(1)
		15%	0.9923837	(4)	0.9922183	(5)	0.9939836	(3)	0.9953051	(2)	**0.9953053**	(1)
		25%	0.9877847	(3)	0.9800315	(5)	0.9851027	(4)	**0.9893560**	(1)	0.9882223	(2)
STAT	MCAR	10%	0.9087270	(5)	0.9105582	(4)	0.9106386	(3)	**0.9132492**	(1)	0.9132475	(2)
		15%	0.9038454	(5)	0.9060337	(4)	0.9066862	(3)	0.9091019	(2)	**0.9091584**	(1)
		25%	0.9013979	(5)	0.9047888	(4)	0.9059597	(3)	0.9077214	(2)	**0.9078041**	(1)
	MAR	10%	0.9058957	(5)	0.9127799	(3)	0.9127328	(4)	**0.9144097**	(1)	0.9142985	(2)
		15%	**0.9142847**	(1)	0.9079620	(4)	0.9078667	(5)	0.9093156	(2)	0.9092031	(3)
		25%	0.8976313	(3)	0.8959471	(5)	0.8968849	(4)	0.8990894	(2)	**0.8992881**	(1)
	MNAR	10%	0.9085572	(3)	0.9070192	(5)	0.9071004	(4)	0.9095510	(2)	**0.9095576**	(1)
		15%	0.8998035	(5)	0.9040681	(4)	0.9042151	(3)	**0.9061872**	(1)	0.9060699	(2)
		25%	**0.8998697**	(1)	0.8951658	(5)	0.8967363	(4)	0.8992252	(3)	0.8992736	(2)
WINE	MCAR	10%	0.9987802	(3)	0.9986694	(5)	0.9987253	(4)	0.9988537	(2)	**0.9988668**	(1)
		15%	0.9959565	(5)	0.9975506	(4)	0.9976749	(3)	**0.9979885**	(1)	0.9979830	(2)
		25%	0.9922771	(5)	0.9947814	(4)	0.9956436	(3)	0.9958397	(2)	**0.9958428**	(1)
	MAR	10%	**0.9979961**	(1)	0.9963918	(5)	0.9966461	(4)	0.9968344	(3)	0.9969354	(2)
		15%	0.9980099	(4)	0.9978826	(5)	0.9984737	(3)	0.9986410	(2)	**0.9986593**	(1)
		25%	0.9950532	(5)	0.9966551	(4)	0.9973227	(3)	0.9976426	(2)	**0.9977587**	(1)
	MNAR	10%	0.9972888	(5)	0.9990950	(4)	0.9991137	(3)	**0.9991486**	(1)	0.9991412	(2)
		15%	**0.9991038**	(1)	0.9981937	(5)	0.9982292	(4)	0.9984342	(3)	0.9984523	(2)
		25%	**0.9953236**	(1)	0.9901818	(5)	0.9940507	(4)	0.9948135	(3)	0.9948292	(2)
PIMA	MCAR	10%	0.8195130	(4)	0.8183348	(5)	0.8207717	(3)	0.8225089	(2)	**0.8225461**	(1)
		15%	0.8097860	(4)	0.8060807	(5)	0.8103984	(3)	**0.8123862**	(1)	0.8123352	(2)
		25%	0.7993334	(4)	0.7932394	(5)	0.8020177	(3)	0.8040140	(2)	**0.8055621**	(1)
	MAR	10%	**0.8252737**	(1)	0.8201186	(5)	0.8218285	(4)	0.8235582	(2)	0.8235392	(3)
		15%	**0.8147294**	(1)	0.8078931	(5)	0.8127261	(4)	0.8143046	(2)	0.8142421	(3)
		25%	0.8007801	(4)	0.7989430	(5)	0.8058399	(3)	**0.8072073**	(1)	0.8071192	(2)
	MNAR	10%	0.8211049	(5)	0.8250540	(4)	0.8278898	(3)	0.8299199	(2)	**0.8303720**	(1)
		15%	0.8157382	(4)	0.8137489	(5)	0.8189948	(3)	0.8211264	(2)	**0.8213390**	(1)
		25%	0.7972827	(4)	0.7916129	(5)	0.8011662	(3)	0.8035721	(2)	**0.8036587**	(1)
ABAL	MCAR	10%	0.8732051	(5)	0.8737739	(4)	0.8740180	(3)	0.8748059	(2)	**0.8749393**	(1)
		15%	0.8717183	(4)	0.8714861	(5)	0.8717831	(3)	0.8725539	(2)	**0.8726250**	(1)
		25%	0.8649143	(5)	0.8663833	(4)	0.8666186	(3)	0.8674332	(2)	**0.8675645**	(1)
	MAR	10%	0.8752625	(2)	0.8742551	(5)	0.8743399	(4)	0.8751972	(3)	**0.8753671**	(1)
		15%	0.8716980	(5)	0.8720722	(4)	0.8721856	(3)	0.8731502	(2)	**0.8731739**	(1)
		25%	0.8680810	(5)	0.8697060	(4)	0.8699619	(3)	0.8708046	(2)	**0.8709102**	(1)
	MNAR	10%	0.8748891	(5)	0.8760444	(4)	0.8760447	(3)	0.8768033	(2)	**0.8769350**	(1)
		15%	0.8734301	(5)	0.8753217	(4)	0.8754605	(3)	0.8763271	(2)	**0.8764456**	(1)
		25%	0.8678228	(5)	0.8683507	(4)	0.8690281	(3)	0.8696780	(2)	**0.8697714**	(1)
Average rank			3.7333		4.5111		3.3778		1.9111		1.4667	

Table 2.7: Comparison between MICE results and the results of the four SI, MI, S-HOT and M-HOT frameworks, using the imputation method yielding the best results for each dataset. Bold values are the single best inference results for each dataset and missingness setting.

2.4.3.4 Results on Real-World Medical Datasets

In this experiment, we compared inference results obtained after applying each of the four frameworks on three real-world medical mixed-type tabular datasets that contains missing values. Complete results for this experiment can be found in the Appendix B: Tables B.6, B.7, B.8, B.9, B.10.

Table 2.8 shows the complete prediction results obtained by applying each of the four compared framework using the SINKHORN imputation method. We confirm the results observed on benchmark datasets, S-HOT leads to consistently better inference results compared to SI, while M-HOT leads to better prediction results in most cases compared to MI

The complete results show that our frameworks perform well on real-world medical conditions, leading to better prediction results than their counterpart in the vast majority of cases no matter the used imputation method. S-HOT leads to consistently better prediction results than SI, which seems to show that S-HOT is a good alternative to SI in most

Dataset	Metric	*SI*	*S-HOT*	*MI*	*M-HOT*
COVI	bACC	88.0521 ±2.7099	**88.7577 ±2.4153**	88.1914 ±2.444	**88.5625 ±2.4207**
	AUC	0.954824 ±0.0147369	**0.9625726 ±0.0124882**	0.9624413 ±0.0123389	**0.9630812 ±0.012202**
	F1	88.0857 ±2.7305	**88.8315 ±2.3945**	88.2002 ±2.4686	**88.5704 ±2.4137**
MYOC	bACC	69.8555 ±2.7178	**70.0475 ±2.6415**	70.3021 ±2.5083	**70.4734 ±2.5403**
	AUC	0.8158175 ±0.0219278	**0.8188904 ±0.0214235**	0.8447799 ±0.0186636	**0.8456001 ±0.0184988**
	F1	85.1389 ±1.3661	**85.1642 ±1.412**	85.9508 ±1.3636	**85.9812 ±1.3924**
NHAN	bACC	63.1886 ±1.5701	**63.3785 ±1.5872**	**64.0025 ±1.5978**	63.9914 ±1.6632
	AUC	0.6926347 ±0.0176282	**0.6965183 ±0.0174091**	0.709166 ±0.0152069	**0.7095322 ±0.0156614**
	F1	62.9453 ±1.6887	**63.0783 ±1.6842**	**63.7281 ±1.7522**	63.7012 ±1.8955

Table 2.8: Experimental results when applying imputation frameworks SI, MI, S-HOT and M-HOT using the SINKHORN imputation method on three real-world medical datasets. Bold values are the best results between pairs of frameworks SI vs S-HOT, and MI vs M-HOT.

real-life scenarios. M-HOT also performs better than MI in most those real-world scenarios. Overall, while no statistically significant difference can be noted between frameworks, this experiment demonstrates the good capacity of our imputation frameworks in real-world situations.

2.4.3.5 Running-Time Comparison

Finally, we compare the computational running-time required to execute each framework in each of the tested scenarios. Table 2.9 shows the average running-times in seconds required to execute each framework. In each column, the left is the imputation time and the right is the training+test running time, the training and test phases are independent of the used imputation method.

Most of the computational time comes from computing the m imputations. SI benefits largely from that point, the three remaining frameworks all require the same amount of time to compute the multiple imputations. The difference in training+test running time between SI and S-HOT is negligible, the main difference is on the time required to compute the multiple imputations in the case of S-HOT, that is m times higher than for SI. In the case of MI and M-HOT, the required time to compute the imputations is the same, and there is a small difference in the training+test running times with M-HOT being very slightly slower than MI. The overall difference in total running time between MI and M-HOT is negligible. When using a fast imputation method, such as SOFTIMPUTE, the imputation computational time is almost negligible in all cases, using S-HOT over SI in this context leads to better results for the same computational cost.

Dataset	Method	*SI*		*S-HOT*		*MI*		*M-HOT*	
IRIS	MISS	10.61	0.62	212.30	0.60	212.30	12.13	212.30	12.60
	SOFT	0.02		0.34		0.34		0.34	
	GAIN	17.76		355.12		355.12		355.12	
	MIDA	21.02		420.40		420.40		420.40	
	SINK	43.27		865.39		865.39		865.39	
STAT	MISS	41.09	0.29	821.84	0.31	821.84	4.82	821.84	5.56
	SOFT	0.01		0.14		0.14		0.14	
	GAIN	25.58		511.67		511.67		511.67	
	MIDA	16.71		334.11		334.11		334.11	
	SINK	46.62		932.40		932.40		932.40	
WINE	MISS	35.74	0.58	714.80	0.60	714.80	12.30	714.80	12.51
	SOFT	0.01		0.13		0.13		0.13	
	GAIN	21.48		429.58		429.58		429.58	
	MIDA	16.89		337.80		337.80		337.80	
	SINK	45.74		914.75		914.75		914.75	
ABAL	MISS	21.99	5.98	439.77	6.45	439.77	123.53	439.77	127.83
	SOFT	0.01		0.18		0.18		0.18	
	GAIN	19.48		389.58		389.58		389.58	
	MIDA	17.94		358.85		358.85		358.85	
	SINK	43.89		877.82		877.82		877.82	
PIMA	MISS	24.84	1.31	496.75	1.33	496.75	24.38	496.75	25.44
	SOFT	0.01		0.12		0.12		0.12	
	GAIN	20.87		417.40		417.40		417.40	
	MIDA	19.26		385.26		385.26		385.26	
	SINK	46.61		932.21		932.21		932.21	

Table 2.9: Average imputation (left) and training+test (right) computational times in seconds for each framework on each benchmark dataset using each tested imputation method. The missingness setting was arbitrarily chosen to be MNAR at 15%, as it does not impact running times.

2.5 Discussion and Conclusion

In this chapter, we proposed an attribute noise correction method, data Denoising and Imputation in One Step (DIOS), that can be used to both impute missing values and correct erroneous ones simultaneously. To the best of our knowledge, DIOS is the first method able to handle attribute noise in its entirety as a preprocessing step in the Machine Learning literature. We experimentally showed that DIOS competes and even outperforms other state-of-the-art imputation and correction methods, and that it is pertinent to use in a real-world medical application context.

We also proposed two new imputation frameworks, Single-Hotpatching (S-HOT) and Multiple-Hotpatching (M-HOT), that aim to improve the training of one or several Neural Networks when dealing with missing values. Those two frameworks aim to exploit the variance between multiple imputations in order to improve the training process of NNs by taking account of the imputation uncertainty. Our work is a first step towards finding better ways to deal with missing values imputation in Machine Learning. We hope that it spikes the interest of other researchers throughout the world on this important, and often overlooked matter, in the Machine Learning literature.

2.5.1 Discussion and Conclusion on DIOS

There is currently a need, in Machine Learning, for an approach able to properly handle attribute noise in its entirety. Such a method needs to be able to correct erroneous values and impute missing values from a corrupted mixed-type tabular dataset. No method able to fully handle attribute noise currently exists in the ML literature. Most existing methods are able to impute missing values only. A few methods are able to correct erroneous values, but most of them are focused on class noise. In this chapter, we proposed a novel approach, data Denoising and Imputation in One Step (DIOS), the first attempt to perform both erroneous values correction and missing values imputation in one preprocessing step in the literature.

Our method produces a corrected version of the original dataset by imputing missing values while correcting erroneous values without requiring any complete or clean instance in the original data. Our experiments show that our method competes against and even outperforms other imputation methods on benchmark and real-world medical mixed-type tabular data. In our applicative medical context, we will be able to employ DIOS to correct attribute noise in our data as a preprocessing step, before training a model to predict survival outcome or Immune-Related Adverse Events (irAEs).

DIOS relies on Denoising Auto-Encoders, it is therefore fundamental to define an architecture that is adapted to the dataset to correct, and to find the best possible hyper-parameters. This tuning phase is fundamental to obtain the best possible results. We were able to define generic architectures that can easily be scaled to the specific dataset of interest, but finding a well-suited architecture and hyper-parameters can be laborious and time-consuming. An obvious resulting limitation is the computational running-time of the method, which can get extensively long with larger datasets.

Since DIOS requires an empirical tuning phase, an interesting future work could be to define an algorithm able to automatically find an adapted architecture, depending on dataset dimensions. Indeed, during our experiments, we found that the main difference between our architectures depends simply on scaling the number of channels at each layer to the amount of features in the corrupted dataset, which we think could be automatized with a bit more research. This would highly improve the usability of DIOS, so that anyone could use and apply the method on any tabular dataset. Finally, if we knew which values are erroneous, or if we could consistently detect the most likely erroneous values, we could mask them as missing. This would certainly highly reduce the overall noise in the corrupted dataset before the learning phase of DIOS, which would yield better results.

2.5.2 Discussion and Conclusion on S-HOT and M-HOT

Taking account of imputation uncertainty while training a Neural Network is not typically researched. That is because strong learners, such as NNs, are naturally able of enough generalization to neglect the consequences of the bias induced by imputation uncertainty. In this chapter, we introduced and proposed two new frameworks: Single-Hotpatching (S-HOT) and Multiple-Hotpatching (M-HOT). Those can be used to improve Neural Network training when dealing with an incomplete dataset by taking account of imputation uncertainty, leading to models able of more generalization and better inference results on unseen data.

An evident extension of S-HOT and M-HOT might be to use them in the context of handling attribute noise. Where the between-variance can be computed between corrected values, and not only between imputed ones. Intuitively, in the exact same way as with imputed missing values, this would lead to NNs that are less prone to overfitting and not biased by the modifications performed by the correction method. In our applicative chapter (Chapter 4), we use the S-HOT framework as an attribute noise framework to train a unique Neural Network with better generalization capacity. Successfully extending the imputation framework concept to handling attribute noise. The only other existing correction method in the literature (Polishing (Teng, 2004)) has only ever been applied using a straight forward framework, analogous to the Single-Imputation framework from the imputation field. Therefore, extending our frameworks to the correction field makes them the first attribute noise correction frameworks, that can be used to better train Neural Networks when dealing with missing and erroneous values, that is, data with attribute noise.

In our frameworks, we assume a Gaussian distribution between imputed values. It leads to good empirical results but a normal distribution might not always be pertinent, other distributions might make more sense depending on the missing feature nature or used imputation method, future works should focus on this matter. It might be possible to automatically adapt the distribution for each missing value, given the proposed imputations. We have empirically shown that our new frameworks improve generalization capacity of Neural Networks on imputed tabular data, future works could focus on experimenting with similar frameworks on image or sequential data.

This work is a first step towards better ways of training Neural Networks on incomplete data, we hope that it sparks the interest of other ML researchers throughout the world to work on researching more advanced ways of taking account of imputation uncertainty during training.

Chapter 3

Learning with Multiple Labeled and Imbalanced Domains

Contents

In this chapter, we are interested in proposing a Domain Adaptation (DA) approach that we can use to maximize prediction quality in the multi-source and imbalanced medical data context of the QUALITOP project. We first set the context and gives an introduction of our specific learning scenario. Section 3.2 briefly presents the related works that are linked to our work. Next, we describe our proposed approach to solve our Multi-Source Domain Adaptation (MSDA) problem on limited and imbalanced data. In section 3.4, we show the performed experiments to evaluate our proposed approach. Finally, section 3.5 gives a global conclusion of our work and discuss our main contributions in Domain Adaptation.

3.1 Context and Introduction

Learning from imbalanced data requires a specific learning approach in order to pay specific attention to the class distribution during the training phase. Deep Learning (DL) is notoriously hard when dealing with limited data. Indeed, if the data is too limited, it is impossible to properly train a deep model, and simpler Machine Learning (ML) approaches should be preferred. When independent limited but similar datasets are available, it becomes adequate and beneficial to use deep MSDA to improve predictions on the target domain (Day and Khoshgoftaar, 2017; Kouw and Loog, 2019; Pan and Yang, 2010; Wilson and Cook, 2020). Exploiting knowledge from several source domains can help to minimize the negative impact of both limited data and class imbalance. In this chapter, we propose a new interesting Multi-Source Supervised Domain Adaptation (MSSDA) approach, which we ultimately aim to apply on real-world limited and imbalanced medical tabular data.

In transfer learning, we aim to exploit knowledge from one or several source dataset(s) to improve learning performance on another target dataset. For transfer learning to be beneficial, the dataset(s) used as source(s) should be similar enough to the target dataset. A source that is not similar enough to the target will negatively impact learning performance, and should not be used in this context. We talk about DA when we aim to learn a single and common task by transferring knowledge from one or several source domain(s) to a target domain. A very well-researched area of DA is Single-Source Domain Adaptation (SSDA) (Ganin et al., 2017; Long et al., 2015; Saito et al., 2018; Zhu et al., 2021), that is, when we use only one source domain to transfer towards the target domain. A more complex and less researched area is Multi-Source Domain Adaptation (Peng et al., 2019; Zhao et al., 2018a; Zhu et al., 2019b; Zuo et al., 2021), where we use several source domains to transfer as much knowledge as possible to the target. Domain Adaptation can help largely improve prediction performance on the target domain by exploiting more knowledge from source domain(s) than available on the sole target domain. Domain Adaptation is often used to make prediction possible on an entirely unlabeled target domain, that is, Unsupervised Domain Adaptation (UDA). In our work, we are interested in a case where the target domain is labeled as any standard dataset, in this case we talk about Supervised Domain Adaptation (SDA), which is a less researched domain adaptation area, despite being a common real-world occurrence.

In this chapter, we propose a new original approach for MSDA in a supervised context, and demonstrate its performance on limited and imbalanced data. Namely, Weighted Multi-Source Supervised Domain Adaptation (WMSSDA). WMSSDA transfers knowledge from s source domains to a similar target domain. It learns a domain invariant latent space, regularized using both statistical and adversarial approaches, where shared knowledge across source domains is transferred to the target domain, and s source domain specific latent spaces, in which source specific knowledge is transferred. With such an architecture, our proposed approach WMSSDA is able to exploit both common knowledge across all domains and source specific knowledge that is useful for inference on the target domain. We compute source domain specific transfer contribution weights during training, those are applied during training to weight the importance of each source domain on learning, reducing as much as possible Negative Learning, that unavoidably appears in MSDA. We conduct ex-

tensive experiments to compare our approach with other baseline and state-of-the-art DA approaches in a data-limited and class imbalance context. We show that WMSSDA outperforms other state-of-the-art DA approaches using statistical analysis on both image benchmark datasets and real-world medical tabular data. We perform an ablation study to validate the pertinence and positive impact of each component in our method.

3.1.1 Domains Shifts in Our Adaptation Scenario

In our specific application context on QUALITOP data, we face different kinds of shifts between data from different hospitals, which we consider as different domains. It is important to be aware of the various shifts between the available domains to ensure the proposal of an adequate approach. We base our analysis on our previously provided shift definitions: 6, 7, and 8.

Data from the QUALITOP project is collected in various hospitals from different countries. Those hospitals have different ways of collecting data, leading to heterogeneous feature spaces between domains, that is, $\mathcal{X}_{\mathbb{S}_1} \neq \ldots \neq \mathcal{X}_{\mathbb{S}_s} \neq \mathcal{X}_{\mathbb{T}}$. As data are collected on patients from different hospitals, there is a high probability of the presence of a sample selection bias (Kouw and Loog, 2019). Indeed, it is quite common to observe sample selection bias in a medical context, that is, an altered probability for an instance to be sampled in one domain compared to another. In this case, the data marginal distributions are different between domains, $P(X_{\mathbb{S}_1}) \neq \ldots \neq P(X_{\mathbb{S}_s}) \neq P(X_{\mathbb{T}})$. The heterogeneity between domains feature spaces, and probable sample selection bias, leads to a covariate shift between QUALITOP domains. In such context, naively training a classifier on one domain to apply it to another might lead to poor inference results. Such shift necessitates to rely on a DA method.

As defined in Definition 7, there exists a prior shift between two domains when the class balance is not the same in each domain. In QUALITOP data, we observe different survival outcome and Immune-Related Adverse Events (irAEs) distributions between domains, which is a perfect example of prior shift, $P(Y_{\mathbb{S}_1}) \neq \ldots \neq P(Y_{\mathbb{S}_s}) \neq P(Y_{\mathbb{T}})$. In this prior shift context, the DA model must be carefully trained while taking account of the class imbalance between domains.

In QUALITOP data, patients do not suffer from the same cancer types and are not treated with the same immunotherapy treatments across hospitals. This is an issue as it leads to a concept shift between domains, which means that the causal relation between features and labels is semantically different from one domain to another, that is, $P(Y_{\mathbb{S}_1}|X_{\mathbb{S}_1}) \neq \ldots \neq P(Y_{\mathbb{S}_s}|X_{\mathbb{S}_s}) \neq P(Y_{\mathbb{T}}|X_{\mathbb{T}})$. Obviously, in our adaptation context, we hope that this concept shift is limited between the domains. Indeed, a concept shift too large between two domains leads to the impossibility to transfer knowledge between domains. In a case with a concept shift, but similar enough data, it is still possible to perform Domain Adaptation from source to target domain at the condition that the target domain is labeled, that is, in a context of Supervised Domain Adaptation.

3.1.2 Problem Formulation

We introduce definitions and concepts needed for the following chapter, our notations are inspired by the work of (Cortes et al., 2019), we took liberties to adapt them to our MSSDA context.

Let $\mathcal{X} \in \mathbb{R}^d$ denote an input feature space, with d the number of features, and $\mathcal{Y} = \{1, \dots, c\}$ a multi-class output label space, with c the total number of classes. We define a domain as a pair formed by a distribution over $\mathcal{X}$ and a labeling function mapping from $\mathcal{X}$ to $\mathcal{Y}$. We note $\mathbb{D} = (P(X_\mathbb{D}), f_\mathbb{D})$ the domain $\mathbb{D}$, with $P(X_\mathbb{D})$ the marginal distribution of $\mathbb{D}$ over $\mathcal{X}$, $f_\mathbb{D} : \mathcal{X} \to \mathcal{Y}$ is the labeling function mapping from feature to label space, $X_\mathbb{D}$ is the data sample defined as $X_\mathbb{D} = \{x_i \in \mathcal{X}\}_{i=1}^{n_\mathbb{D}}$, with $n_\mathbb{D}$ the number of instances in the data sample of the domain $\mathbb{D}$.

In the scenario of Multi-Source Supervised Domain Adaptation we consider, we are given s source domains, noted $\mathbb{S}_i$ for $i \in [1, s]$, that we want to exploit to improve classification over one target domain, noted $\mathbb{T}$. A unique label space $\mathcal{Y}$ is shared across all domains, feature space of each domain can be different from other domains, we note $\mathcal{X}_\mathbb{D}$ the feature space of domain $\mathbb{D}$. In our scenario, we have access to s labeled source domains where $\mathbb{S}_i = \{(x_j^{\mathbb{S}_i}, y_j^{\mathbb{S}_i})\}_{j=1}^{n_i}$, with $\{x_1^{\mathbb{S}_i}, \dots, x_{n_i}^{\mathbb{S}_i}\} \sim P(X_{\mathbb{S}_i})$, and $y_j^{\mathbb{S}_i} = f_{\mathbb{S}_i}(x_j^{\mathbb{S}_i})$. We have access to a labeled target domain, similarly, $\mathbb{T} = \{(x_j^\mathbb{T}, y_j^\mathbb{T})\}_{j=1}^{n_\mathbb{T}}$, where $\{x_1^\mathbb{T}, \dots, x_{n_\mathbb{T}}^\mathbb{T}\} \sim P(X_\mathbb{T})$ and $y_j^\mathbb{T} = f_\mathbb{T}(x_j^\mathbb{T})$.

We want to exploit knowledge from labeled source domains and the labeled target domain, to improve classification on an unknown and unusable part of $\mathbb{T}$. As we consider a scenario in which the three types of shifts are present between domains, we consider that the covariate shift assumption does not hold, $f_{\mathbb{S}_1} \neq \dots \neq f_{\mathbb{S}_s} \neq f_\mathbb{T}$. Solving such a problem is only possible if the target domain is labeled, as it is necessary to rely on supervision to properly align domains with concept shifts. Therefore, we want our Supervised Domain Adaptation model to learn to estimate the labeling function $f_\mathbb{T}$, while exploiting knowledge from the source domains through the learning of the different source labeling functions $\{f_{\mathbb{S}_1}, \dots, f_{\mathbb{S}_s}\}$.

3.1.3 Our Contribution in the Supervised Domain Adaptation Field

The main contributions in this chapter are as follows:

- We propose Weighted Multi-Source Supervised Domain Adaptation (WMSSDA), an new innovative approach for Multi-Source Supervised Domain Adaptation that learns both common and source specific latent spaces in two branches, by combining statistical and adversarial learning to maximize domain invariance in the shared space.
- We propose to dynamically compute transfer contribution weights during training, and apply them as a scaling of the source domains on learning. Increasing the transfer contribution of relevant source domains while decreasing the contribution of less related, or too dissimilar, source domains to maximize useful knowledge transfer towards the target domain.
- We experimentally demonstrate the relevance and superiority of our approach on data limited and imbalanced domains, an experimental setting close to the application setting of QUALITOP.

- We perform an ablation study to validate the relevance of each element in our method.

Our work has lead to the redaction and submission of a scientific paper at the following international journal:

- **T. Ranvier**, H. Elghazel, E. Coquery, K. Benabdeslem. *Deep Multi-Source Supervised Domain Adaptation with Class Imbalance*. DOI: 10.21203/rs.3.rs-3160713/v1. **KAIS**: Knowledge and Information Systems, **(Under Review)**.

The research work performed in the Domain Adaptation context during this thesis also helped contributing to a research work in the field of multi-view Unsupervised Domain Adaptation in collaboration with PhD. student Mehdi Hennequin, leading to the international publication of the following research paper:

- M. Hennequin, K. Benabdeslem, H. Elghazel, **T. Ranvier**, E. Michoux. *Multi-view Self-attention for Regression Domain Adaptation with Feature Selection*. DOI: 10.1007/978-3-031-30105-6_15. **ICONIP 2022**, 23-26 november 2022, New Delhi, India, (Online).

3.2 Related Works

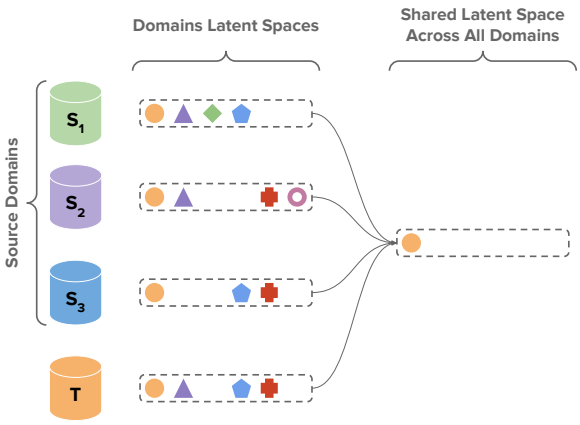
In this section, we review approaches and concepts of the Domain Adaptation literature that are related to our work, and important for the comprehension of this chapter.

3.2.1 Domain Adaptation Approaches

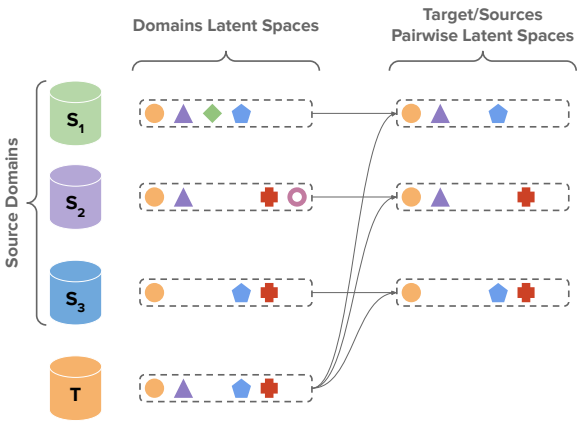
Most DA approaches in the literature focus on learning a shared domain invariant latent space between domains to capture shared information between source(s) and target (Wilson and Cook, 2020). This leads to a latent space where instances of a domain are indistinguishable from instances of other domains, while classification relevant information is conserved, leading to better inference results on the target domain. There are two main ways of reaching this goal, relying on statistic distribution matching, or relying on an adversarial loss that encourages samples from different domains to lose all domain specific information. Adversarial approaches are notorious for reaching better results than statistic distribution matching in Domain Adaptation (Zhu et al., 2021).

We believe that learning a shared domain invariant latent space for Multi-Source Domain Adaptation is limited, and that learning pairwise invariant latent spaces between the target domain and each source domain allows for the capture and transfer of more useful information between sources and target. Figure 3.1 shows an intuitive representation of why we think learning pairwise latent spaces between the target domain and each source domain is important, as it captures more relevant information than a shared invariant latent space across all domains.

For this reason, in our proposed MSDA approach, we used an architecture where a shared domain invariant latent space is learned across all domains in one branch, while s source



(a) Shared domain invariant latent space across all domains leads to a limited knowledge transfer between domains.



(b) Pairwise latent spaces built between the target and each source domain leads to optimal knowledge transfer from sources to target.

Figure 3.1: When learning a domain invariant latent space between two domains, intuitively, the only information that is captured within the shared representation is the common information. Therefore, when building a shared representation across multiple domains, only the common information across all domains is captured, which becomes a limiting factor as the number of domains and the dissimilarities between them increases. This is why we believe that learning several latent spaces in a pairwise manner between target and sources is pertinent in order to capture and transfer as much relevant information as possible.

specific latent spaces are learned between the target domain and each of the s source domains in another branch.

There exist two MSDA methods in the literature that also rely on learning both a shared domain invariant latent space while also learning pairwise latent spaces between sources and target domain: ML-MSDA (Li et al., 2020), and MLAN (Xu et al., 2022). Unlike us, their methods have been proposed and applied in a UDA context, which differs from SDA. As presented in Chapter 1, Mutual Learning Network for Multiple-Source Domain Adaptation (ML-MSDA) (Li et al., 2020) is composed of two branches. The first branch learns a shared invariant latent space across all domains, while the second learns pairwise latent spaces between the target and each source domain. By jointly learning those multiple latent representations, they obtain better experimental results than all previously presented MSDA approaches. They rely on adversarial learning to ensure the domain invariance of the learned latent spaces. Similarly, (Xu et al., 2022) extended the work of (Li et al., 2020) by proposing a Mutual Learning based Alignment Network (MLAN). The model architecture is identical to ML-MSDA, but is trained slightly differently, through the proposed mutual learning module. The module relies on pseudo-labeling of target instances to maximize target prediction performance. With MLAN, (Xu et al., 2022) currently obtains state-of-the-art results compared to other multi-source and single-source models of the Domain Adaptation literature.

In our work, we propose a MSSDA method with a two-branch architecture, similarly to ML-MSDA and MLAN. We exploit the fact that we are working in a SDA context to train the model on labeled instances from all sources and target domains.

3.2.2 Negative Transfer

One of the most common reasons for Negative Transfer is a too large dissimilarity between source and target domains (Zhang et al., 2022). This risk is multiplied in the Multi-Source Domain Adaptation field, as multiple sources can contribute to Negative Transfer.

In their paper, (Zuo et al., 2021) proposed the ABMSDA method, which avoids Negative Transfer by weighting each source domain depending on their similarity with the target domain. Their model architecture is composed of a domain classifier, a common feature extractor regularized using WMD (a modified version of Moment Distance), and a shared task-specific classifier. They train the domain classifier, separately and prior from the rest of the model, to predict the probability that target images belong to each source domain. They use the probability output of the domain classifier as a metric that indicates the statistical similarity between the target domain and each source domains, with the intuition that source domains that are most similar to the target domain should be attributed a higher weight $w_{\mathbb{S}}$. They apply those weights w when computing WMD:

$$\text{WMD}(f_\theta, X_{\mathbb{S}}, X_{\mathbb{T}}) = \sum_{i=1}^{k} w_{\mathbb{S}} \left\| \frac{1}{|X_{\mathbb{S}}|} \sum_{x_{\mathbb{S}} \in X_{\mathbb{S}}} f_\theta(x_{\mathbb{S}})^i - \frac{1}{|X_{\mathbb{T}}|} \sum_{x_{\mathbb{T}} \in X_{\mathbb{T}}} f_\theta(x_{\mathbb{T}})^i \right\|_{\mathcal{F}} \tag{3.1}$$

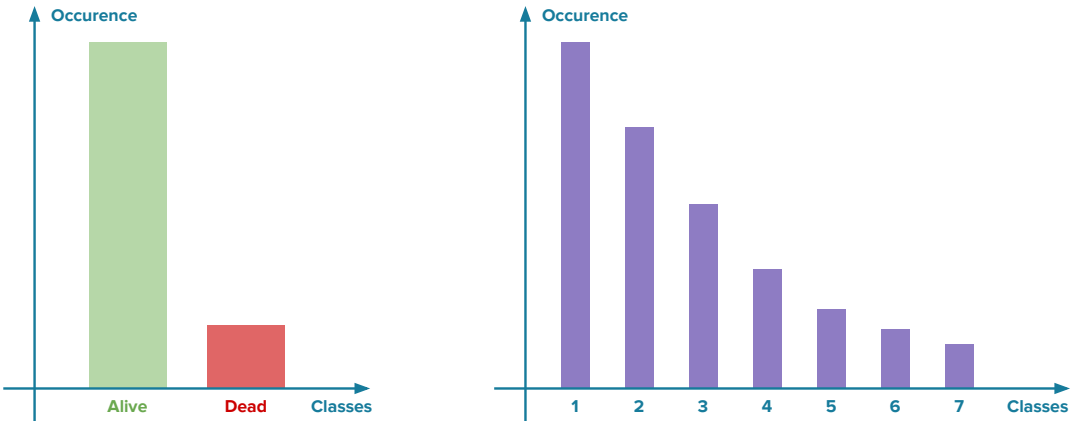
They also apply those weights when combining the probability outputs of the classifier during training, leading to a classifier less prone to Negative Transfer. This is a way of avoiding Negative Transfer during Multi-Source Domain Adaptation.

In our proposed approach, we compute transfer contribution weights with a discrepancy measure, directly during training. Those weights are associated to each source domain, based on supervised target domain results, and applied to scale the importance of source instances during training. The goal being to increase the importance of instances from relevant source domains, while decreasing the importance of instances from less related source domains.

3.2.3 Learning With Imbalanced Data

Class imbalance occurs when the labels we aim to predict are not uniformly distributed over the dataset, resulting in certain classes having a much higher, or lower, number of instances compared to others. Figure 3.2 show examples of class imbalance in both binary and multi-class settings. In the multi-class example, properly classifying instances of the tailing classes (the classes with less occurrence on the right) would be harder, because of a high bias towards more commonly represented classes. Similarly, in the binary imbalance setting, a standardly trained inference model would be highly biased towards the positive class “Alive”, and would miss-classify many patients that should be classified as in danger of death. Such an example shows how important it is to train inference models in the less biased way possible when in presence of class imbalance.

When trained on an imbalanced dataset, standard Machine Learning approaches will lead to poorer results than when trained with a similar balanced dataset (Haibo He and Garcia, 2009). When dealing with class imbalance it is primordial to use approaches that help im-



(a) Binary class imbalance.

(b) Multi-class imbalance.

Figure 3.2: Examples of class imbalance in both binary (left) and multi-class (right) settings

prove learning performance. Nowadays, two main ways are used to deal with imbalanced data in the literature, sampling approaches, and cost-sensitive approaches (Wang et al., 2016).

Sampling approaches aim to artificially adjust the class distribution, by either removing instances the majority class(es), and/or adding more instances from the minority class(es). Under-sampling aims to reduce the number of majority class instances to achieve a more balanced class distribution. The simplest under-sampling approach that can be used to artificially re-balance a dataset is random under-sampling (Fernández et al., 2018; Krawczyk, 2016; Wang et al., 2016). With random under-sampling instances from the majority class(es) are randomly removed until the desired class distribution is achieved. This simple approach can lead to improved inference results but can also result in the loss of important information from the majority class(es), and so, reduce the generalization ability of the model as it misses crucial information. To assess this drawback more researched and advanced under-sampling approaches have been proposed, such as: Condensed Nearest Neighbor (Hart, 1968), or Tomek Links (Tomek, 1976). Overall, under-sampling leads to improved inference results but might also lead to losing important information for inference if the amount of instances is too low to afford removing instance from the majority class(es). On the other hand, over-sampling involves increasing the representation of minority classes by duplicating or generating synthetic examples. The simplest over-sampling approach that can be used to artificially re-balance a dataset is random over-sampling (Fernández et al., 2018; Krawczyk, 2016; Wang et al., 2016). Random over-sampling duplicates randomly selected instances from the minority class(es) until the desired class distribution is achieved. As with random under-sampling, over-sampling leads to improved inference results but highly increases the risk of overfitting, leading to biased inference models that lack in generalization capacity. The Synthetic Minority Oversampling Technique (SMOTE) method is the most popular and most widely used advanced over-sampling method, it has been proposed in (Chawla et al., 2002) and it is known to largely improve inference results on imbalanced data (Fernández et al., 2018; Wang et al., 2016). SMOTE works by creating synthetic examples of the minority class(es) by interpolating between existing instances of the minority class(es). SMOTE

is known to largely improve prediction results on imbalanced data. A known drawback of SMOTE, and over-sampling in general, is the risk of leading to overfitting, and the risk of generating synthetic examples that are unrealistic or less informative, leading to a limited improvement in prediction quality.

Removing and generating synthetic data leads to improved learning results, but with important drawbacks. Removing data is often non viable in real-world scenarios with limited data, and generating synthetic data comes with the disadvantage of potentially generating implausible instances. Another approach that can be used in ML to deal with class imbalance is cost-sensitive learning, where modifications are made to the algorithm, and/or to the training process, to take account of imbalance and improve prediction results. It has been shown in several empirical studies that cost-sensitive learning leads to superior inference results than sampling approaches on imbalanced data (Haibo He and Garcia, 2009; Kukar and Kononenko, 1998). Therefore, cost-sensitive techniques are usually a better solution than sampling methods. In Neural Network training, the most popular way of implementing cost-sensitive learning is to adapt the error function to take into account the class cost of each training instance during the learning phase, as defined in (Kukar and Kononenko, 1998). The error function is corrected by introducing the cost factor of the class as a weight that is applied during training. Class weights applied to the loss function are commonly computed as the inverse of the class distribution of training data, though other weighting techniques can be used. This approach obtained by far the best results in (Kukar and Kononenko, 1998), it is still a very commonly used approach as it is very easy to implement and use, and it reaches better results than most other existing approaches to handle imbalanced data. This is why, in this chapter, we use a cost-sensitive approach to account for the class imbalance between domains to maximize prediction quality.

3.3 Proposed Approach for Learning with Multiple Supervised Domains

In this section, we describe in details our proposed approach: Weighted Multi-Source Supervised Domain Adaptation (WMSSDA).

The idea behind WMSSDA is to create a common domain invariant latent space and s source domain specific latent spaces, perform classification on each latent space, and draw final weighted prediction results in an ensemble manner. The common domain invariant latent space is trained on a label classification task on both source and target batches. Two regularization techniques are employed to minimize domain-specific information into the common latent space. First, by using a Moment Distance (MD) regularization to match the distributions of source and target batches, and secondly, through the adversarial training of a domain discriminator. For the s source domain specific latent spaces the ideology is the opposite, we want each latent space to retain as much source specific information as possible. This is possible as in a Supervised Domain Adaptation context, the target domain is labeled, it is therefore possible to train each classifier on a supervised classification task on both source and target instances, leading to a pairwise fine-tuning between each source do-

main and the target domain. Those specific latent representations are regularized using a collaboratively trained specific domain classifier, while performing label classification, leading to latent spaces retaining as much domain specific information as possible. Using MSDA naturally helps dealing with the class imbalance problem, as it increases the total amount of available data for training. We further deal with the class imbalance in each domain using a cost-sensitive learning approach, by scaling the loss with a class weight, computed as the inverse of the class distribution of training data. We use the output of the MD measure between the target domain and each source domain to determine transfer contribution weights, higher weights are attributed to source domains with closer latent distributions compared to the target domain latent representation, and inversely. Those transfer contribution weights are then applied to weight the classification loss on source instances, giving less importance to less relevant source domains in training. Ultimately, the final predictions for the target domain are obtained by passing a target batch through all trained modules. The average of the outputs from all specific classifiers is computed and combined with the outputs of the common classifier to obtain the final probabilities for the target classes. Figure 3.3 shows a simple representation of the architecture of the approach.

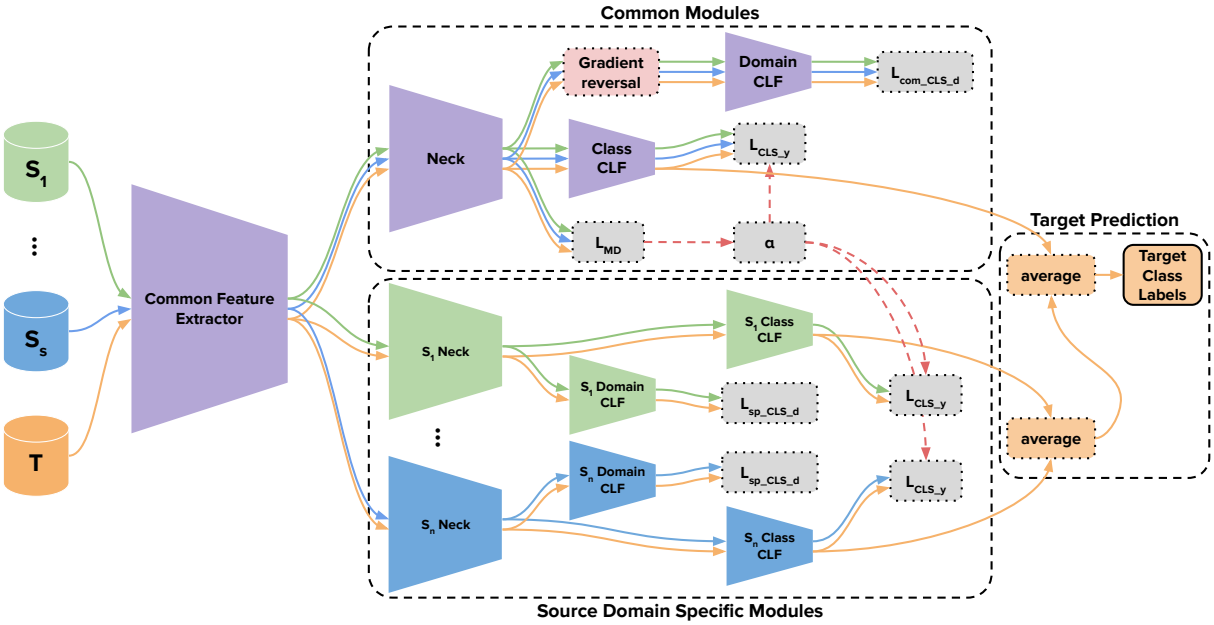


Figure 3.3: Architecture of our approach WMSSDA, common modules appear in purple, i-th source domain specific modules appear in the same color as the i-th associated source domain. Lines of color symbolize the data flow, each color corresponding to a batch of the corresponding domain. Dashed red lines symbolize the computation and application of transfer contribution weights, computed from the MD measure and applied as scaling in the classification loss terms.

When training the approach, we iterate through one batch from each source domain for each target domain batch. Batches are processed as a pair between the b-th target batch and the b-th batch of the i-th source domain, which we will refer to as a pair of batches in the following. Each pair of batches is fed through a common feature extractor used to extract

low-level features on all domains alike. The approach architecture is then divided in two main parts: common modules and source domain specific modules.

- Common modules. They are fed pairs of batches between the target domain and any source domain. The first component of this part is a common neck, comparable to the previous feature extractor, which extracts higher-level features on all domains. We call the output of the previous component the common domain invariant latent space, which we note $Z_{\mathbb{T}com}$ and $Z_{\mathbb{S}com}$, for the target and source batch latent representations respectively. To ensure that this latent space is domain invariant, we use both statistical distribution matching and adversarial domain discrimination.

  We regularize the latent representations by minimizing the standard Moment Distance, such as defined in (Peng et al., 2019; Zuo et al., 2021), between target and source representation, $\mathcal{L}_{MD} = \sum_{i=1}^{k} \left|\left|\mathbb{E}(Z^i_{\mathbb{S}com}) - \mathbb{E}(Z^i_{\mathbb{T}com})\right|\right|_{\mathcal{F}}$.

  We use the output of the MD measure between the target domain batch and batches of all source domains to compute transfer contribution weights. We note those weights $\alpha \in \mathbb{R}^s$, with α_i the weight associated to the i-th source domain. Those transfer contribution weights are used to scale the classification loss of each source domains, giving more weight to close and related source domains and less weight to less useful domains. They are computed as:

$$\alpha_i = \frac{s+1}{s} - \frac{e^{D_i - \max(D)}}{\sum e^{D_i - \max(D)}}$$

  with $D = \{\text{MD}(Z_{\mathbb{S}_i com}, Z_{\mathbb{T}com})\}_{i=1}^{s}$ the set of MD measures between the target domain batch and each source domain batch. The weights are computed such as $\sum_{i=1}^{s} \alpha_i = s$, which ensures that source instances are given as much importance as target instances overall, while a different weight is given to each source domain.

  We associate the MD regularization with the training of an adversarial common domain classifier applied to the latent space. This common domain classifier learns to discriminate the domain from which originates each sample in the pair of batches, while the common feature extractor and neck try to fool the discriminator, leading to a domain invariant representation in this latent space. Parameters of the common feature extractor and neck are tuned to maximize the loss of the domain discriminator, while the discriminator parameters are tuned to minimize their own loss. As it is hard to optimize minimax problems using gradient descent algorithms, a common practice in DA research is to use a gradient reversal operation, and apply it between the latent representation and the domain classifier, such as defined in (Ganin et al., 2017). Which solves the adversarial problem by minimizing a single loss. We note $\mathcal{L}_{adv\_d}$ the loss of the common domain discriminator applied on the gradient reversed common latent representation of the pair of batches.

  We found that using both statistical and adversarial strategies simultaneously led to better empirical results, which suggests that using both approaches cooperatively leads to a better domain invariant representation.

Finally, the latent representation is fed through a task-specific classifier that discriminates samples on their class label.

- Source domain specific modules. They are only fed pairs of batches between the target domain and their associated source domain, that is, a pair of batches between the i-th source domain and the target domain is fed to the i-th specific module. Each specific module is composed of a specific neck, which extracts higher-level features, a specific task-specific classifier that discriminates samples on their class label, and a specific domain classifier that discriminates samples on the domain they originate from. In the opposite way to the above, this specific domain classifier is trained collaboratively, which pushes each specific latent space to retain as much domain specific information as possible. The source and target batches are treated differently:

  - The source batch from domain $\mathbb{S}_i$ is fed through the i-th specific neck, the resulting latent representation is noted $Z_{\mathbb{S}i}$. The i-th domain classifier is fed $Z_{\mathbb{S}i}$ and is trained to recognize that those samples originate from domain $\mathbb{S}_i$. The i-th task-specific classifier is fed $Z_{\mathbb{S}i}$ and is trained to discriminate the class of each sample.
  - The target batch is fed through all specific necks, and the resulting latent representations are noted $\{Z_{\mathbb{T}1}, \ldots, Z_{\mathbb{T}s}\}$. The i-th domain classifier is fed $Z_{\mathbb{T}i}$ and is trained to recognize that those samples originate from domain $\mathbb{T}$. Each task-specific classifier is fed $Z_{\mathbb{T}i}$, their outputs are noted $\{\hat{Y}_{\mathbb{T}1}, \ldots, \hat{Y}_{\mathbb{T}s}\}$. Only the i-th classifier is trained to discriminate the class of each target sample to avoid overfitting.

  We note $\mathcal{L}_{sp\_d}$ the loss of the i-th specific domain discriminator applied on the i-th latent representations of the pair of batches.

We note $\mathcal{L}_y$ the global task-specific classification loss, which is the average between common and specific classification probabilities on the pair of batches. To take account of class imbalance during training we compute class weights, noted W, that are applied in a cost-sensitive learning manner during task-specific classification loss computation, they are computed as the inverse of the class distribution observed in the training data of each domain: $W_{\mathbb{D}} = 1/P(Y_{\mathbb{D}})$. Finally, the loss we minimize using gradient descent is computed for each pair of batches in the following way:

$$\mathcal{L} = \mathcal{L}_y + \lambda_1 \mathcal{L}_{adv\_d} + \lambda_2 \mathcal{L}_{sp\_d} + \lambda_3 \mathcal{L}_{MD}$$

Where $\lambda_1, \lambda_2, \lambda_3$ are hyper-parameters that are defined to balance each component of the final loss.

Final prediction results are obtained by feeding target instances through all task-specific classifiers, the average of source specific classifiers outputs is computed and averaged with the output of the common classifier, leading to final class probabilities.

Pseudo-code 3 describes, in a more formal way, the training steps of the entire approach. In this pseudo-code, $model$ includes a common feature extractor model, common and spe-

cific necks, and common and specific classifiers, com_clf_d is the common domain classifier and sp_clf_d is the set of i specific domain classifiers. We note $adv(\cdot)$ the gradient reversal operation, classification outputs are noted $\{\hat{Y}_{\mathbb{D}com}, \hat{Y}_{\mathbb{D}1}, \ldots, \hat{Y}_{\mathbb{D}s}\}$ and correspond to the label probabilities obtained on the common latent representation and all specific representations of the domain $\mathbb{D}$ respectively. Latent spaces are referred to with the same notations as above, $Z_{\mathbb{D}com}$ for the common latent representation and $Z_{\mathbb{D}i}$ for the i-th specific latent representation of domain $\mathbb{D}$. Parameter E is the number of epochs to perform, s is the number of source domains and λ is the set of hyper-parameters used to balance all loss components together. Finally, W are class weights specific to each domain computed as $1/P(Y_{\mathbb{D}})$, W_i for the i-th source domain and $W_{\mathbb{T}}$ for the target domain.

Our proposal combines the advantages of a domain invariant latent space with a set of domain specific representations, with transfer contribution weights applied to minimize Negative Transfer during training, leading to a Supervised Domain Adaptation approach able to transfer knowledge from multiple source domains to a target domain. Similarly to DANN (Ganin et al., 2017), our proposed WMSSDA learns a domain invariant latent space on which is performed classification and similarly to MFSAN (Zhu et al., 2019b), WMSSDA builds s domain specific latent spaces. The main difference between the second part of WMSSDA and MFSAN is that MFSAN aims to match source and target distributions between all specific latent spaces, which we find counterproductive as it means that all specific latent spaces are pushed toward an identical latent space. We choose to combine both the adversarial and statistical approaches to regularize the common latent space, since we found better results in this way, and could use the MD results to compute contribution weights. As in the two recent Unsupervised MSDA approaches, ML-MSDA (Li et al., 2020) and MLAN (Xu et al., 2022), we chose to organize our architecture in two branches, one to build a common latent space, and the other to build s domain specific latent spaces. We do so as we believe that only learning a shared latent space across all domains in a MSDA context prevents part of useful knowledge to be transferred from source to target domain. We also noted superior experimental results by using two branches instead of one. We choose to use the Moment Distance to match both target and source common distributions, since (Peng et al., 2019) demonstrated that MD is better suited as a statistical distribution matching approach for MSDA than the most common Maximum Mean Discrepancy (MMD). We compute a scale variable γ as described in (Zhu et al., 2021), to scale the impact of the MD regularization throughout the training. Its value starts at a 0 and increases logarithmically towards a value of 1 over the total number of epochs, giving more importance to this regularization at the middle and end of the training phase. We apply the computed transfer contribution weights from the MD output on the classification loss of source instances, minimizing as much as possible Negative Learning during training. We apply class weights to scale the computed classification loss to take account of class imbalance during training in a cost-sensitive learning manner. Using a Domain Adaptation approach also naturally helps in dealing with limited data since source domain knowledge helps improving the overall learning performance. Our proposed Weighted Multi-Source Supervised Domain Adaptation approach allows the exploitation of both common knowledge and source domain specific information that is useful for target domain classification.

Algorithm 3: WMSSDA Training Pseudo-Code.

input: E, s, $model$, com_clf_d, sp_clf_d, W, λ
for $epoch \leftarrow 1$ *to* E **do**
  $X_{\mathbb{T}}, Y_{\mathbb{T}} \leftarrow$ extract_batch($\mathbb{T}$);
  for $i \leftarrow 1$ *to* s **do**
    // Feed target batch to model and compute target loss components
    $\{\hat{Y}_{\mathbb{T}com}, \hat{Y}_{\mathbb{T}1}, \ldots, \hat{Y}_{\mathbb{T}s}\} \leftarrow$ forward($model$, $X_{\mathbb{T}}$);
    $\hat{Y}_{\mathbb{T}} \leftarrow (\hat{Y}_{\mathbb{T}com} + \hat{Y}_{\mathbb{T}i})/2$;
    $\mathcal{L}_{\mathbb{T}y}$ = cross_entropy($Y_{\mathbb{T}}$, $\hat{Y}_{\mathbb{T}}$, $W_{\mathbb{T}}$);
    // Feed source batch to model and compute source loss components
    $X_{\mathbb{S}}, Y_{\mathbb{S}} \leftarrow$extract_batch($\mathbb{S}_i$);
    $\{\hat{Y}_{\mathbb{S}com}, \hat{Y}_{\mathbb{S}1}, \ldots, \hat{Y}_{\mathbb{S}s}\} \leftarrow$ forward($model$, $X_{\mathbb{S}}$);
    $\hat{Y}_{\mathbb{S}} \leftarrow (\hat{Y}_{\mathbb{S}com} + \hat{Y}_{\mathbb{S}i})/2$;
    $\mathcal{L}_{\mathbb{S}y}$ = cross_entropy($Y_{\mathbb{S}}$, $\hat{Y}_{\mathbb{S}}$, W_i);
    // Feed adversarially trained common domain classifier
    $\{\hat{Y}_{\mathbb{S}adv_d}, \hat{Y}_{\mathbb{T}adv_d}\} \leftarrow$forward($com\_clf\_d$, adv($\{Z_{\mathbb{S}com}, Z_{\mathbb{T}com}\}$));
    $\mathcal{L}_{\mathbb{S}adv\_d} \leftarrow$ cross_entropy($\hat{Y}_{\mathbb{S}adv\_d}$, $\{i, \ldots, i\}$);
    $\mathcal{L}_{\mathbb{T}adv\_d} \leftarrow$ cross_entropy($\hat{Y}_{\mathbb{T}adv\_d}$, $\{0, \ldots, 0\}$);
    $\mathcal{L}_{adv\_d} \leftarrow (\mathcal{L}_{\mathbb{S}adv\_d} + \mathcal{L}_{\mathbb{T}adv\_d})/2$;
    // Feed i-th collaboratively trained specific domain classifier
    $\{\hat{Y}_{\mathbb{S}sp\_d}, \hat{Y}_{\mathbb{T}sp\_d}\} \leftarrow$ forward($sp\_clf\_d_i$, $\{Z_{\mathbb{S}sp_i}, Z_{\mathbb{T}sp_i}\}$);
    $\mathcal{L}_{\mathbb{S}sp\_d} \leftarrow$ cross_entropy($\hat{Y}_{\mathbb{S}sp\_d}$, $\{1, \ldots, 1\}$);
    $\mathcal{L}_{\mathbb{T}sp\_d} \leftarrow$ cross_entropy($\hat{Y}_{\mathbb{T}sp\_d}$, $\{0, \ldots, 0\}$);
    $\mathcal{L}_{sp\_d} \leftarrow (\mathcal{L}_{\mathbb{S}sp\_d} + \mathcal{L}_{\mathbb{T}sp\_d})/2$;
    // Compute MD regularization
    $\gamma \leftarrow 2/(1 + exp(-10e/E))) - 1$;
    $D_i \leftarrow MD(Z_{\mathbb{S}com}, Z_{\mathbb{T}com})$;
    $\mathcal{L}_{MD} \leftarrow \gamma \times D_i$;
    $\alpha_i \leftarrow ((s+1)/s) - (e^{D_i - \max(D)} / \sum e^{D_i - \max(D)})$;
    // Compute global loss and back-propagate
    $\mathcal{L}_y \leftarrow (\mathcal{L}_{\mathbb{T}y} + \alpha_i \times \mathcal{L}_{\mathbb{S}y})/(1 + \alpha_i)$;
    $\mathcal{L} = \mathcal{L}_y + \lambda_1 \mathcal{L}_{adv\_d} + \lambda_2 \mathcal{L}_{sp\_d} + \lambda_3 \mathcal{L}_{MD}$;
    Train $model$, com_clf_d and sp_clf_d by back-propagating $\mathcal{L}$;
  end
end

As we consider, in our learning scenario, that the covariate shift assumption might not hold, meaning that there might be concept shift between domains, we implement a second version of our method. In their paper, (Saito et al., 2018) proposed an interesting and easy-to-implement way to allow Domain Adaptation approaches to handle concept shift by successfully aligning conditional distribution between two domains $P(Y_{\mathbb{D}_1}|X_{\mathbb{D}_1}) = P(Y_{\mathbb{D}_2}|X_{\mathbb{D}_2})$. Similarly to (Zhu et al., 2019b), we implement a second version of our method in which we include this modification, we name this variation WMSSDA-β. We replace each task-specific classifier in WMSSDA-β with a pair of classifiers and follow the following three steps in our

training:

- We train our entire model as previously defined, where label probabilities are defined as the mean of each pair of classifier outputs.
- We then fix the feature extractor and necks and train the classifier pairs to maximize their discrepancy. The discrepancy between two classifiers C and C' for an instance x is defined as $|C(x) - C'(x)|$.
- Finally, we train the feature extractor and necks to minimize this same discrepancy with fixed classifiers.

We repeat those steps until global convergence to simultaneously align both marginal and conditional distributions, leading to successful domain adaptation given our scenario. This β version of WMSSDA should be able to better handle concept shift than the standard version, leading to better inference results when in presence of concept shift.

3.4 Experiments

This section presents the experiments we led to evaluate and compare our proposed approach WMSSDA to other state-of-the-art Domain Adaptation approaches in a data-limited and class imbalance context.

3.4.1 Used Datasets

With our experiments we want to show that our method performs well compared to other state-of-the-art approaches in an experimental setting close to the real application setting of QUALITOP, on both popular benchmark DA datasets and a real-world medical DA dataset composed of mixed-type tabular data, the Covid dataset. We are interested in comparing state-of-the-art DA approaches in the context of limited data and class imbalance.

The 5-Digits multi-domain dataset is widely used in Domain Adaptation studies (Ganin et al., 2017; Peng et al., 2019; Zhao et al., 2018a; Zhu et al., 2021; Zuo et al., 2021), it is composed of five digits recognition datasets, with grayscale or color images of various sizes:

1. MNIST[1], the widely known handwritten digit recognition dataset.
2. MNIST-M[2], a more complex version of MNIST, created by combining MNIST images with randomly extracted patches of photos of the BSDS500 dataset as their background.
3. Street View House Numbers (SVHN)[3], a Real-World image dataset of house numbers extracted from Google Street View images.

[1] http://yann.lecun.com/exdb/mnist
[2] https://www.kaggle.com/datasets/aquibiqbal/mnistm
[3] http://ufldl.stanford.edu/housenumbers

4. Synthetic Digits (SYN)[4], synthetically generated images of digits with random backgrounds.

5. USPS[5], a handritten digit recognition dataset similar to MNIST.

The DomainNet dataset is a multi-domain dataset recently released with paper (Peng et al., 2019), and accessible at the following webpage[6]. It aims to provide a new difficult and more advanced benchmark dataset for DA approaches. It is composed of six domains of color images of various sizes, with a total of 345 common categories across all domains:

1. Clipart, a collection of clipart images.

2. Infograph, infographic images of specific objects.

3. Painting, artistic depictions of objects in the form of paintings.

4. Quickdraw, drawings of the worldwide players of the game "Quick Draw!".

5. Real, photos and real-world images.

6. Sketch, sketches of specific objects.

The 5-Digits and DomainNet datasets are DA benchmark image datasets that are known for their covariate shift between domains. We perform experiments on those two datasets to demonstrate the capacity of our approach to perform well on known experimental DA datasets. As we are interested in an experimental scenario with limited data in each domain and class imbalance, we preprocessed the datasets to follow our setting of interest. We randomly selected subsets of each dataset domain to create both a limited amount of data and a class imbalance. After this selection step, the class representation in the 5-Digits dataset spans from 4.22% (10 samples) for the least represented class to 21.09% (500 samples) for the most represented class in each domain. For the DomainNet dataset, the class representation spans from 1.1% (10 samples) for the least represented class to 13.33% (120 samples) for the most represented class in each domain, out of 17 classes, leading to a total of 900 instances in each domain.

The Covid dataset was provided by the Mexican government and is composed of health data about patients that suffer from, or have symptoms that could be related to, Covid19. It is downloadable from the following Kaggle repository[7], it originally contains 1,048,576 samples from patients suspected of suffering from Covid19, with 20 tabular mixed-type (continuous and categorical) features. The predictive task for this dataset is to predict the survival outcome of patients. The dataset is naturally imbalanced, odds of survival to Covid19 being, fortunately, higher than the odds of death. We used the categorical feature "Medical Unit" to split the original data into 5 domains depending on the type of medical institution that provided the care to the patient. We could not find more information about those kinds of

[4] https://www.kaggle.com/datasets/prasunroy/synthetic-digits
[5] https://www.kaggle.com/datasets/bistaumanga/usps-dataset
[6] http://ai.bu.edu/M3SDA/#dataset
[7] https://www.kaggle.com/datasets/meirnizri/covid19-dataset

medical institutions, apart from their ID in the dataset, we consider in the following that they correspond to different hospitals as we observe a covariate shift between the institutions. During our preprocessing, we selected 800 unique patients per domain to simulate a limited amount of training data, and we dropped two features containing almost only missing values. With the subtraction of the feature used to split the data into separate domains, there remain 17 features after preprocessing. More information about our preprocessing, along with an exploratory analysis of the dataset, can be found in chapter 4, where we base the application of the entire work of this thesis on this real-world dataset. The exploratory analysis of the Covid dataset shows that there is a covariate shift between the five domains. We conclude that the covariate shift between Covid domains is certainly quite similar to our QUALITOP context, as patients are also split between distinct medical institutions in QUALITOP data.

Table 3.1 shows the class representation in each domain of the Covid dataset, revealing a prior shift between the domains of the dataset. Indeed, we can observe that the label distribution is imbalanced in each domain and the representation is different from one domain to the other, thus, $P(Y_{\mathbb{D}_1}) \neq P(Y_{\mathbb{D}_2}) \neq \ldots \neq P(Y_{\mathbb{D}_5})$.

Class	Representation	Domain 1	Domain 2	Domain 3	Domain 4	Domain 5	Overall
Negative	Percentage	7.37%	12.37%	13.75%	10.5%	4.87%	9.78%
	Samples Count	59	99	110	84	39	391
Positive	Percentage	92.63%	87.63%	86.25%	89.5%	95.13%	90.22%
	Samples Count	741	701	690	716	761	3,609

Table 3.1: Covid label distribution per domain, showing that there is a prior shift across domains, $P(Y_{\mathbb{D}_1}) \neq P(Y_{\mathbb{D}_2}) \neq \ldots \neq P(Y_{\mathbb{D}_5})$.

After the preprocessings described above for each dataset, we observe both covariate and prior shifts in all datasets, with limited and imbalanced data in each domain. All datasets might suffer from limited concept shift, in precaution, the β version of our approach is designed to better handle concept shift. A summary of the details about the three DA datasets are shown in Table 3.2.

Dataset	5-Digits	DomainNet	Covid
Data Type	Image	Image	Mixed-Type Tabular
Context	DA Benchmark	DA Benchmark	Real-World Medical
Domains	5	6	5
Classes	10	17	2
Samples per Domain	2,370	900	800

Table 3.2: Multi-Source Domain Adaptation datasets technical details.

3.4.2 Compared Approaches

We compared our proposed WMSSDA to four single-source and four Multi-Source Domain Adaptation state-of-the-art approaches. Most of those approaches are initially Unsupervised Domain Adaptation approaches, that is, approaches that do not use labels from the target domain. We modified those approaches to allow them to use target domain labels in their training phase for a fair comparison. The modification for each method is usually as simple as adding the supervised classification task on the target domain into the loss term of each approach. This phase of adapting UDA approaches for the SDA context is crucial as it would not be fair to compare our approach, that is able to use target domain labels in its supervised learning, with UDA methods that are able to use them. This would lead to far better inference results for our approach compared to unsupervised ones and would not be representative of the real capacity of unsupervised approaches.

In our experiments, single-source approaches are evaluated using two settings, such as in (Peng et al., 2019). In the single best setting, we evaluate the approach on all possible pairs of domains as source and target and select the best-obtained results for each target domain. In the source combine setting, we combine all source domains as one unique domain to obtain only one source domain. We used the following single-source domain adaptation approaches from the literature:

- DAN, Deep Adaptation Network (Long et al., 2015) is among the first deep learning Unsupervised Domain Adaptation approaches to have been proposed. DAN uses multi-kernel MMD to minimize the distribution divergence between features extracted on the source and target data, fed through common layers followed by domain specific classifiers. We based our implementation on the following PyTorch implementation[8].

- DANN, Domain-Adversarial Neural Network (Ganin et al., 2017), a model that builds a domain invariant latent space, using an adversarial domain classifier, on which classification is performed. We based our implementation on this PyTorch implementation[9].

- MCD, Maximum Classifier Discrepancy (Saito et al., 2018), an approach that trains a generator and a pair of classifiers by alternating between training the classifiers to maximize their discrepancy, and training the generator to minimize their discrepancy. We based our implementation on this PyTorch implementation[10].

- DSAN, Deep Subdomain Adaptation Network (Zhu et al., 2021), a similar approach to DAN that replaces the MK-MMD with a Local MMD that aligns the distributions of the relevant subdomains. We based our implementation on this PyTorch implementation[11].

We used the following Multi-Source Domain Adaptation approaches from the literature:

[8] https://github.com/CuthbertCai/pytorch_DAN
[9] https://github.com/fungtion/DANN
[10] https://github.com/mil-tokyo/MCD_DA
[11] https://github.com/easezyc/deep-transfer-learning/tree/master/UDA/pytorch1.0/DSAN

- MDAN, Multi-source Domain Adversarial Network (Zhao et al., 2018a), a similar approach to DANN that uses as many adversarial domain classifiers as source domains. We based our implementation on this PyTorch implementation[12].

- MFSAN, Multiple Feature Spaces Adaptation Network (Zhu et al., 2019b), the architecture of this model is composed of a common feature extractor followed by source domain specific parallel layers blocks. Those layers are regularized using MMD and domain specific classifier outputs are aligned with an L1 operation. Classifiers outputs are combined in an ensemble way to obtain the final target prediction. We based our implementation on this PyTorch implementation[13].

- M^{3}SDA, Moment Matching for Multi-Source Domain Adaptation (Peng et al., 2019), the architecture is composed of a common feature extractor followed by source domain specific parallel classifiers. The output of the common feature extractor is regularized using a moment-matching distribution distance between source and target data. We based our implementation on this PyTorch implementation[14].

- ABMSDA, Attention-Based Multi-Source Domain Adaptation (Zuo et al., 2021), the architecture of the model is comparable to that of M^{3}SDA, the main difference is that ABMSDA uses a common classifier instead of domain specific ones. ABMSDA computes attention weights with a domain classifier that is fed raw data, the domain classifier outputs are used to derive weights that are applied to the moment matching regularization and to the training loss in an attempt to weight each source domain and avoid negative transfer.

We also used two simple baseline approaches to get reference results:

- NN, a simple Neural Network, in the first evaluation setting, the model is trained only on the target domain, in the second setting, the model is trained on all combined domains, sources, and target domains alike.

- FT, a simple fine-tuning approach in which a Neural Network (NN) is pre-trained on source data and is then fine-tuned on target data.

[12] https://github.com/hanzhaoml/MDAN
[13] https://github.com/easezyc/deep-transfer-learning/blob/master/MUDA/MFSAN
[14] https://github.com/VisionLearningGroup/VisionLearningGroup.github.io

3.4.3 Experimental Protocol

We performed three main experiments, one to evaluate our approach compared to other DA methods on popular benchmark datasets, a second one to evaluate WMSSDA on a real-world medical MSDA dataset in a very similar setting to the one of QUALITOP, and finally, an ablation study to evaluate the impact and usefulness of each component of our method. We evaluate all approaches on a Supervised Domain Adaptation classification task with limited data and class imbalance, all UDA methods have been slightly modified to handle a labeled target domain as described in the previous section. All our experimental results are compared using the three following classification metrics: the balanced Accuracy (bACC), the Area Under the Curve (AUC), and the F_1-score.

All datasets are split between a training set and a test set, all approaches are trained on the same training data and evaluated on the same test data. To obtain significant results we conduct each experiment 5 times and report the mean and standard deviation of each evaluation metric. To be as fair as possible in our comparisons, in each experiment, we define the model architecture for each approach as similarly as possible across them, while taking account of their architectural differences. For image datasets, our feature extractor is a convolutional network, and all other modules are fully connected networks, for the Covid tabular dataset all modules are fully-connected. In the same line of thought, identical hyper-parameters, tuned on the NN approach, are used across all compared approaches for each experiment. All approaches are given the same class weights, those are applied in a cost-sensitive learning manner during loss computation to better handle class imbalance. Those class weights are computed, for each domain, as the inverse of the class distribution observed in the training data: $W_{\mathbb{D}} = 1/P(Y_{\mathbb{D}})$.

To better assess the obtained experimental results, we statistically compare WMSSDA results to each of the compared state-of-the-art approaches using t-tests. The results of those statistical tests are used to determine if our approach performs significantly better, even, or worse than each other, based on a p-values set to 0.05. The results of the t-tests are symbolized in result tables as either a bullet $\bullet$, a circle $\circ$, or an equivalent symbol $\equiv$. The bullet is used to signify that our method is significantly better than the method we compared it to, the circle signifies the opposite, and the equivalent means that there is no significant difference between WMSSDA and the compared method. In the following sections, we will refer as "significantly better" all results that have been evaluated using a t-test and that were classified as significantly better in regard to a p-value of 0.05, and "significantly worse" the opposite.

3.4.4 Comparative Study on Benchmark Datasets

This first experiment is a comparative study of several baseline and state-of-the-art DA approaches to evaluate WMSSDA performance on benchmark DA datasets. We aim to show that our approach is able to compete against, and even outperform, other state-of-the-art DA approaches on well-known image benchmark multi-domain datasets in a context of limited data and class imbalance.

Setting	Method	Metric	MNIST	MNIST-M	SVHN	SYN	USPS	Avg
Single Best	NN	bACC	84.74 ± 0.29 •	63.08 ± 1.07 •	46.36 ± 1.07 •	61.72 ± 0.70 •	89.49 ± 0.53 •	69.08 •
		AUC	.9915 ± .0004 •	.9353 ± .0046 •	.8578 ± .0081 •	.9333 ± .0028 •	.9934 ± .0007 •	.9423 •
		F1	83.85 ± 0.35 •	58.98 ± 1.25 •	31.47 ± 0.81 •	56.77 ± 1.36 •	88.91 ± 0.40 •	64.00 •
	DAN	bACC	77.56 ± 1.54 •	53.95 ± 2.10 •	41.12 ± 0.14 •	56.46 ± 2.01 •	78.70 ± 1.13 •	61.56 •
		AUC	.9832 ± .0005 •	.8963 ± .0077 •	.8196 ± .0036 •	.9073 ± .0082 •	.9900 ± .0010 •	.9193 •
		F1	75.10 ± 2.15 •	48.61 ± 2.61 •	24.88 ± 0.39 •	49.47 ± 2.79 •	76.74 ± 1.58 •	54.96 •
	DANN	bACC	90.12 ± 0.51 •	69.62 ± 0.85 •	69.37 ± 0.53 •	78.40 ± 0.52 •	96.02 ± 0.24 •	80.71 •
		AUC	.9967 ± .0002 •	.9622 ± .0030 •	.9479 ± .0013 •	.9734 ± .0008 •	.9989 ± .0002 •	.9758 •
		F1	89.43 ± 0.71 •	68.38 ± 1.14 •	**68.26 ± 0.63** ≡	78.06 ± 0.53 •	96.21 ± 0.28 •	80.07 •
	DSAN	bACC	91.00 ± 0.25 •	70.42 ± 1.78 •	67.57 ± 1.03 •	77.30 ± 0.79 •	96.30 ± 0.31 •	80.52 •
		AUC	.9971 ± .0001 •	.9549 ± .0052 •	.9394 ± .0022 •	.9669 ± .0022 •	.9988 ± .0002 •	.9714 •
		F1	90.73 ± 0.29 •	69.44 ± 2.05 •	67.52 ± 1.04 ≡	77.27 ± 0.80 •	96.35 ± 0.39 •	80.26 •
	FT	bACC	88.95 ± 0.56 •	68.11 ± 0.98 •	59.32 ± 1.97 •	67.50 ± 1.13 •	93.91 ± 0.34 •	75.56 •
		AUC	.9958 ± .0002 •	.9568 ± .0029 •	.9274 ± .0049 •	.9562 ± .0029 •	.9980 ± .0002 •	.9669 •
		F1	88.54 ± 0.67 •	65.58 ± 0.97 •	51.94 ± 2.35 •	65.03 ± 1.57 •	94.10 ± 0.26 •	73.04 •
	MCD	bACC	92.47 ± 1.00 •	74.09 ± 2.43 •	59.94 ± 4.38 •	76.84 ± 3.16 •	94.95 ± 0.28 •	79.66 •
		AUC	.9968 ± .0005 •	.9666 ± .0084 •	.8972 ± .0178 •	.9665 ± .0081 •	.9982 ± .0005 •	.9651 •
		F1	92.25 ± 1.10 •	73.32 ± 2.27 ≡	58.00 ± 4.22 •	76.38 ± 3.41 •	94.81 ± 0.29 •	78.95 •
Source Combine	NN	bACC	91.92 ± 1.00 •	68.66 ± 1.78 •	62.21 ± 1.05 •	78.36 ± 0.76 •	95.77 ± 0.29 •	79.39 •
		AUC	.9971 ± .0005 •	.9572 ± .0029 •	.9354 ± .0030 •	.9737 ± .0013 •	.9983 ± .0002 •	.9724 •
		F1	91.57 ± 1.19 •	67.77 ± 1.52 •	61.04 ± 1.99 •	78.23 ± 0.74 •	95.93 ± 0.34 •	78.91 •
	DANN	bACC	90.12 ± 1.62 •	66.76 ± 1.68 •	54.77 ± 8.47 •	73.98 ± 3.86 •	93.65 ± 1.38 •	75.86 •
		AUC	.9945 ± .0008 •	.9471 ± .0038 •	.9078 ± .0291 •	.9617 ± .0090 •	.9971 ± .0009 •	.9616 •
		F1	89.94 ± 1.62 •	66.22 ± 1.39 •	54.63 ± 7.70 •	73.97 ± 3.77 •	94.06 ± 1.13 •	75.76 •
	DAN	bACC	75.91 ± 1.34 •	53.67 ± 1.02 •	42.48 ± 1.48 •	56.00 ± 0.69 •	81.27 ± 1.65 •	61.87 •
		AUC	.9847 ± .0017 •	.9018 ± .0056 •	.8449 ± .0100 •	.9138 ± .0059 •	.9910 ± .0013 •	.9272 •
		F1	72.56 ± 1.96 •	47.78 ± 1.02 •	24.84 ± 1.13 •	48.14 ± 0.98 •	79.88 ± 2.36 •	54.64 •
	DSAN	bACC	94.11 ± 0.54 •	72.40 ± 0.48 •	65.76 ± 1.01 •	80.41 ± 0.29 •	96.35 ± 0.16 •	81.80 •
		AUC	.9978 ± .0003 •	.9608 ± .0017 •	.9348 ± .0018 •	.9730 ± .0005 •	.9986 ± .0004 •	.9730 •
		F1	94.02 ± 0.57 •	71.88 ± 0.24 •	63.40 ± 1.08 •	80.39 ± 0.31 •	96.38 ± 0.25 •	81.21 •
	FT	bACC	92.76 ± 0.47 •	70.71 ± 0.55 •	59.97 ± 1.08 •	73.49 ± 0.22 •	95.85 ± 0.26 •	78.56 •
		AUC	.9977 ± .0001 •	.9627 ± .0013 •	.9296 ± .0032 •	.9671 ± .0009 •	.9985 ± .0002 •	.9711 •
		F1	92.62 ± 0.50 •	69.38 ± 0.70 •	52.48 ± 1.26 •	72.69 ± 0.30 •	95.93 ± 0.24 •	76.62 •
	MCD	bACC	93.33 ± 0.91 •	64.97 ± 3.06 •	51.95 ± 3.47 •	70.68 ± 0.99 •	93.71 ± 0.91 •	74.93 •
		AUC	.9950 ± .0016 •	.9067 ± .0193 •	.8475 ± .0236 •	.9315 ± .0040 •	.9880 ± .0035 •	.9337 •
		F1	93.33 ± 0.91 •	66.18 ± 2.56 •	50.63 ± 4.02 •	70.52 ± 0.99 •	94.22 ± 0.78 •	74.98 •
Multi Source	MDAN	bACC	87.63 ± 0.98 •	53.44 ± 1.66 •	43.47 ± 2.08 •	60.64 ± 1.27 •	87.58 ± 2.12 •	66.55 •
		AUC	.9946 ± .0005 •	.9194 ± .0036 •	.8925 ± .0113 •	.9469 ± .0042 •	.9946 ± .0016 •	.9496 •
		F1	87.29 ± 1.11 •	49.77 ± 1.59 •	26.62 ± 1.26 •	57.22 ± 1.39 •	87.85 ± 2.02 •	61.75 •
	MFSAN	bACC	90.97 ± 0.46 •	70.75 ± 1.19 •	62.76 ± 2.64 •	75.06 ± 0.97 •	95.87 ± 0.34 •	79.08 •
		AUC	.9970 ± .0003 •	.9607 ± .0055 •	.9378 ± .0050 •	.9734 ± .0011 •	.9986 ± .0002 •	.9735 •
		F1	90.67 ± 0.55 •	69.45 ± 1.30 •	57.28 ± 3.75 •	74.07 ± 1.14 •	96.10 ± 0.30 •	77.51 •
	M3SDA	bACC	92.38 ± 1.41 •	65.45 ± 1.19 •	57.56 ± 2.61 •	75.85 ± 0.43 •	94.69 ± 0.46 •	77.19 •
		AUC	.9963 ± .0009 •	.9421 ± .0039 •	.9178 ± .0078 •	.9660 ± .0012 •	.9976 ± .0004 •	.9639 •
		F1	92.20 ± 1.57 •	64.67 ± 1.46 •	56.41 ± 2.87 •	75.56 ± 0.39 •	94.84 ± 0.52 •	76.74 •
	ABMSDA	bACC	93.53 ± 0.50 •	67.01 ± 1.84 •	53.47 ± 4.63 •	77.45 ± 0.56 •	95.23 ± 0.77 •	77.34 •
		AUC	.9977 ± .0003 •	.9471 ± .0041 •	.9079 ± .0112 •	.9677 ± .0014 •	.9983 ± .0004 •	.9637 •
		F1	93.37 ± 0.58 •	66.78 ± 2.09 •	52.12 ± 5.24 •	77.22 ± 0.62 •	95.57 ± 0.75 •	77.01 •
	WMSSDA	bACC	**95.30 ± 0.21**	75.05 ± 1.06	70.50 ± 1.21	82.85 ± 0.49	96.77 ± 0.30	84.10
		AUC	.9988 ± .0001	.9737 ± .0017	.9519 ± .0014	.9838 ± .0005	.9992 ± .0001	.9815
		F1	**95.25 ± 0.21**	74.62 ± 1.17	68.17 ± 2.01	82.70 ± 0.49	96.90 ± 0.29	83.53
	WMSSDA-β	bACC	95.21 ± 0.53	**77.58 ± 1.75**	**71.83 ± 1.57**	**84.74 ± 0.59**	**96.91 ± 0.24**	**85.26**
		AUC	**.9989 ± .0001**	**.9790 ± .0025**	**.9596 ± .0027**	**.9876 ± .0006**	**.9993 ± .0002**	**.9849**
		F1	95.14 ± 0.56	**76.68 ± 2.05**	67.80 ± 1.95	**84.60 ± 0.62**	**97.00 ± 0.22**	**84.24**

WMSSDA is: • significantly better, ≡ equivalent, ∘ significantly worse, p-value: 0.05

Table 3.3: Comparative Study on the 5-Digits Domain Adaptation Benchmark Dataset, with Limited and Imbalanced Data. The best and second-best results for each metric and each target domain appear in bold and underlined respectively. Results are evaluated on a multi-class classification task using metrics: balanced Accuracy, Area Under the Curve (AUC), and the F_1-Score.

Table 3.3 reports our entire experimental results on the 5-Digits dataset, with visual in-

dications of the results of the t-tests between our best-performing models and each compared approach in all settings. As can be seen in the table, our two models, WMSSDA and WMSSDA-β, perform largely better than other approaches in the vast majority of cases, obtaining the best or second-best result for almost all metrics and target domains. We note that apart from our approach, other state-of-the-art multi-source approaches do not reach particularly better results than single-source approaches, we hypothesize that this is a manifestation of Negative Transfer that multi-source approaches are not yet able to fully avoid. The statistical comparison of the results on 5-Digits shows that our best-performing approach, WMSSDA-β, obtains significantly better results in the vast majority of cases, with only a few settings in which our approach leads to equivalent results to those of other methods. If we consider average performance across all target domains, we can conclude that WMSSDA-β obtains the best prediction results overall, with significantly better results than any other compared approach.

Table 3.4 reports our entire experimental results on the DomainNet dataset. Prediction results are overall quite low, as this dataset is notoriously hard. The fact that we drastically reduced the amount of available training data worsens classification results, but this does not affect the comparison potential of the results. We can see that our proposed WMSSDA-β obtains the best or second-best results in the majority of cases, while WMSSDA obtains overall good results but not better than other approaches. It is important to note that the overall best prediction results for the Quickdraw domain are reached by a simple NN trained on the target domain only. This means that no DA method is currently able to avoid Negative Transfer enough to match those results, nor are they able to surpass them. This shows that Negative Transfer in DA is still an open-problem, and that it is crucial to find better ways of solving this issue. The second best results on the Quickdraw domain are reached by the Fine-Tuning approach, followed by our WMSSDA-β approach, showing that our method is currently the best-performing MSDA approach to avoid Negative Transfer. The simple Fine-Tuning approach leads to significantly better average balanced Accuracy results than ours and almost all other methods on both single best and source combine settings. This probably shows that adding a short phase of pre-training to other state-of-the-art approaches would drastically improve the overall results of all methods. Another well-performing approach in this setting is MFSAN, which leads to significantly better balanced Accuracy than ours on average. Overall, our two versions of WMSSDA lead to competitive experimental results on this dataset, with WMSSDA-β leading to significantly better results than other approaches in the vast majority of cases. When considering the average over all target domains, WMSSDA obtains significantly better results than all other approaches on the three evaluation metrics except for Fine-Tuning and MFSAN on the balanced accuracy metric.

We can conclude from this experiment on benchmark datasets that our proposed approach is able to compete, and even surpass, other baseline and state-of-the-art Domain Adaptation approaches in our supervised multi-domain with limited and imbalanced data context.

Setting	Method	Metric	CLIP		INFO		PAINT		REAL		SKETCH		QUICK		Avg	
Single Best	NN	bACC	27.43 ± 0.90	≡	16.16 ± 0.81	≡	23.14 ± 0.19	•	35.51 ± 1.36	•	17.04 ± 1.75	•	**44.76 ± 0.77**	○	27.34	•
		AUC	.7466 ± .0044	•	.6283 ± .0043	•	.7171 ± .0062	•	.8372 ± .0063	•	.6489 ± .0086	•	**.8707 ± .0052**	○	.7415	•
		F1	22.85 ± 0.94	•	13.21 ± 0.82	•	21.22 ± 0.48	•	32.26 ± 1.47	•	14.76 ± 1.60	•	**39.89 ± 0.97**	○	24.03	•
	DAN	bACC	23.08 ± 1.90	•	13.78 ± 1.02	•	17.78 ± 2.99	•	28.84 ± 1.42	•	15.29 ± 0.63	•	36.08 ± 1.23	•	22.48	•
		AUC	.7072 ± .0101	•	.6070 ± .0152	•	.6504 ± .0100	•	.7722 ± .0163	•	.6367 ± .0060	•	.8129 ± .0029	•	.6977	•
		F1	19.22 ± 2.11	•	11.20 ± 1.25	•	14.01 ± 4.13	•	24.50 ± 0.91	•	13.30 ± 0.84	•	29.57 ± 1.20	•	18.63	•
	DANN	bACC	28.37 ± 0.33	≡	17.24 ± 0.77	≡	23.75 ± 1.17	≡	37.22 ± 1.09	•	19.02 ± 1.42	•	34.12 ± 1.65	•	26.62	•
		AUC	.7593 ± .0040	•	.6514 ± .0058	•	.7354 ± .0068	•	.8499 ± .0054	•	.6849 ± .0082	•	.8188 ± .0077	•	.7499	•
		F1	25.85 ± 0.70	•	**16.39 ± 0.64**	≡	22.42 ± 0.97	•	36.51 ± 1.07	•	17.41 ± 1.26	•	29.53 ± 1.67	•	24.69	•
	DSAN	bACC	5.90 ± 0.40	•	5.88 ± 0.00	•	14.82 ± 1.65	•	25.51 ± 5.35	•	5.88 ± 0.00	•	18.57 ± 2.51	•	12.76	•
		AUC	.5741 ± .0105	•	.5465 ± .0061	•	.6466 ± .0113	•	.7632 ± .0443	•	.5444 ± .0027	•	.7574 ± .0109	•	.6387	•
		F1	0.95 ± 0.53	•	0.65 ± 0.00	•	11.14 ± 2.14	•	21.23 ± 7.06	•	0.65 ± 0.00	•	13.32 ± 2.35	•	7.99	•
	FT	bACC	29.41 ± 1.16	≡	16.29 ± 1.16	≡	23.51 ± 1.07	•	37.37 ± 0.92	•	19.61 ± 0.74	≡	44.51 ± 1.10	○	28.45	○
		AUC	.7664 ± .0077	•	.6446 ± .0038	•	.7386 ± .0069	•	.8513 ± .0027	•	.6858 ± .0055	•	.8663 ± .0023	≡	.7588	•
		F1	25.61 ± 1.21	•	13.97 ± 1.20	≡	21.95 ± 0.83	•	34.63 ± 1.21	•	17.87 ± 0.44	•	39.81 ± 1.18	○	25.64	•
	MCD	bACC	26.47 ± 1.43	≡	16.90 ± 0.74	≡	21.90 ± 0.66	•	34.94 ± 0.83	•	19.96 ± 1.03	≡	37.98 ± 0.86	•	26.36	•
		AUC	.7328 ± .0142	•	.6389 ± .0056	•	.6952 ± .0055	•	.8216 ± .0039	•	.6667 ± .0046	•	.8253 ± .0055	•	.7301	•
		F1	25.48 ± 1.07	•	15.86 ± 0.20	≡	20.85 ± 0.67	•	34.27 ± 0.57	•	18.84 ± 1.15	≡	33.68 ± 1.00	≡	24.83	•
Source Combine	NN	bACC	26.51 ± 1.60	≡	14.65 ± 1.22	≡	22.88 ± 1.48	•	34.67 ± 1.45	•	19.73 ± 1.25	≡	26.92 ± 0.98	•	24.23	•
		AUC	.7451 ± .0122	•	.6357 ± .0075	•	.7103 ± .0072	•	.8223 ± .0098	•	.6903 ± .0085	•	.7518 ± .0083	•	.7259	•
		F1	25.78 ± 1.35	≡	14.52 ± 1.09	≡	22.08 ± 1.32	•	34.78 ± 1.74	•	18.25 ± 0.72	•	21.65 ± 1.54	•	22.84	•
	DANN	bACC	25.41 ± 0.99	•	14.82 ± 1.18	≡	22.10 ± 1.44	•	32.53 ± 2.46	•	20.51 ± 1.63	≡	27.98 ± 1.07	•	23.89	•
		AUC	.7344 ± .0031	•	.6324 ± .0069	•	.7104 ± .0126	•	.8073 ± .0179	•	.6920 ± .0061	•	.7465 ± .0161	•	.7205	•
		F1	24.79 ± 1.16	•	14.85 ± 1.06	≡	21.15 ± 1.35	•	32.86 ± 2.76	•	19.42 ± 1.77	≡	22.21 ± 1.84	•	22.55	•
	DAN	bACC	21.67 ± 1.02	•	11.41 ± 1.41	•	16.78 ± 1.20	•	28.75 ± 0.77	•	15.71 ± 0.75	•	33.31 ± 0.94	•	21.27	•
		AUC	.6988 ± .0065	•	.5923 ± .0160	•	.6602 ± .0167	•	.7782 ± .0103	•	.6264 ± .0049	•	.7946 ± .0079	•	.6918	•
		F1	17.33 ± 1.23	•	7.76 ± 2.36	•	14.49 ± 0.79	•	25.02 ± 1.06	•	12.56 ± 0.97	•	26.22 ± 0.65	•	17.23	•
	DSAN	bACC	5.88 ± 0.06	•	5.76 ± 0.11	•	5.84 ± 0.05	•	5.92 ± 0.08	•	5.84 ± 0.08	•	5.88 ± 0.00	•	5.86	•
		AUC	.5056 ± .0137	•	.5107 ± .0115	•	.5153 ± .0232	•	.4794 ± .0377	•	.5054 ± .0096	•	.5037 ± .0234	•	.5034	•
		F1	0.82 ± 0.11	•	0.68 ± 0.06	•	0.85 ± 0.31	•	0.73 ± 0.15	•	0.68 ± 0.06	•	0.65 ± 0.00	•	0.74	•
	FT	bACC	29.76 ± 1.00	○	17.00 ± 0.58	≡	24.22 ± 0.55	•	38.69 ± 1.04	≡	20.71 ± 0.72	≡	41.37 ± 0.67	≡	**28.62**	○
		AUC	.7762 ± .0061	•	.6463 ± .0050	•	.7362 ± .0053	•	.8567 ± .0044	•	.6978 ± .0092	•	.8431 ± .0028	•	.7594	•
		F1	26.83 ± 0.68	≡	15.49 ± 0.28	≡	22.80 ± 0.31	•	36.67 ± 0.93	•	19.08 ± 0.72	≡	36.52 ± 0.75	≡	26.23	•
	MCD	bACC	25.63 ± 0.64	•	13.04 ± 0.78	•	21.88 ± 1.36	•	33.47 ± 1.74	•	18.18 ± 1.50	•	27.80 ± 2.22	•	23.33	•
		AUC	.7315 ± .0076	•	.6166 ± .0066	•	.6899 ± .0116	•	.8050 ± .0161	•	.6542 ± .0111	•	.7579 ± .0111	•	.7092	•
		F1	25.14 ± 0.77	•	12.52 ± 0.67	•	20.32 ± 1.24	•	32.12 ± 2.07	•	16.82 ± 1.18	•	23.50 ± 2.48	•	21.74	•
Multi Source	MDAN	bACC	27.88 ± 0.89	≡	15.82 ± 0.90	≡	22.43 ± 0.32	•	35.69 ± 1.41	•	18.73 ± 0.85	•	30.00 ± 2.42	•	25.09	•
		AUC	.7789 ± .0074	•	.6515 ± .0035	•	.7310 ± .0111	•	.8560 ± .0018	•	.6887 ± .0083	•	.8269 ± .0094	•	.7555	•
		F1	24.60 ± 1.31	•	13.45 ± 1.06	•	20.40 ± 0.72	•	32.99 ± 1.46	•	16.36 ± 1.08	•	23.14 ± 2.85	•	21.82	•
	MFSAN	bACC	**30.29 ± 0.72**	○	**17.39 ± 0.50**	○	**26.16 ± 0.41**	≡	38.51 ± 0.83	≡	21.96 ± 0.75	≡	36.41 ± 1.01	•	28.45	○
		AUC	.7708 ± .0035	•	.6548 ± .0058	•	.7437 ± .0057	•	.8379 ± .0042	•	.7066 ± .0072	≡	.8104 ± .0125	•	.7540	•
		F1	26.43 ± 0.45	≡	15.69 ± 0.78	≡	24.23 ± 0.51	≡	35.77 ± 1.15	•	19.98 ± 0.93	≡	29.52 ± 1.13	•	25.27	•
	M3SDA	bACC	26.14 ± 1.19	•	15.55 ± 0.90	≡	22.12 ± 0.42	•	34.24 ± 1.82	•	19.88 ± 0.89	≡	23.10 ± 1.70	•	23.50	•
		AUC	.7386 ± .0070	•	.6342 ± .0056	•	.7102 ± .0048	•	.8120 ± .0057	•	.6804 ± .0087	•	.7107 ± .0058	•	.7143	•
		F1	25.47 ± 1.49	≡	15.42 ± 1.20	≡	21.22 ± 0.45	•	34.11 ± 1.63	•	18.84 ± 1.08	≡	16.84 ± 2.03	•	21.98	•
	ABMSDA	bACC	25.43 ± 1.50	•	14.84 ± 0.82	≡	21.73 ± 0.71	•	32.20 ± 1.32	•	19.80 ± 0.67	≡	24.94 ± 2.02	•	23.16	•
		AUC	.7269 ± .0063	•	.6201 ± .0024	•	.6991 ± .0046	•	.7966 ± .0122	•	.6755 ± .0065	•	.7245 ± .0129	•	.7071	•
		F1	24.75 ± 1.55	•	14.87 ± 1.11	≡	20.80 ± 0.48	•	32.03 ± 1.34	•	18.59 ± 0.76	≡	19.24 ± 2.14	•	21.71	•
	WMSSDA	bACC	29.84 ± 1.38		16.25 ± 1.19		24.96 ± 0.89		39.04 ± 0.70		**22.02 ± 0.69**		32.88 ± 2.08		27.50	
		AUC	.7737 ± .0103		.6504 ± .0078		.7447 ± .0048		.8500 ± .0059		.7044 ± .0094		.8007 ± .0088		.7540	
		F1	**28.31 ± 1.28**		16.06 ± 1.09		24.08 ± 0.87		**38.87 ± 0.64**		**20.74 ± 0.74**		28.08 ± 3.06		26.02	
	WMSSDA-β	bACC	28.18 ± 0.58		15.96 ± 1.04		25.27 ± 0.70		**39.29 ± 1.26**		21.14 ± 1.15		40.04 ± 0.99		28.31	
		AUC	**.7971 ± .0033**		**.6791 ± .0055**		**.7663 ± .0059**		**.8821 ± .0036**		**.7162 ± .0063**		.8620 ± .0045		**.7838**	
		F1	27.28 ± 0.64		15.50 ± 1.11		**24.45 ± 0.83**		38.72 ± 1.45		20.17 ± 1.25		34.85 ± 1.48		**26.83**	

WMSSDA is: • significantly better, ≡ equivalent, ○ significantly worse, p-value: 0.05

Table 3.4: Comparative Study on the DomainNet Domain Adaptation Benchmark Dataset, with Limited and Imbalanced Data. The best and second-best results for each metric and each target domain appear in bold and underlined respectively. Results are evaluated on a multi-class classification task using metrics: balanced Accuracy, AUC, and the F_1-Score.

3.4.5 Comparative Study on Real-World Tabular Medical Dataset

The second experiment is a comparative study between all tested DA approaches and our proposed WMSSDA, on the real-world mixed-type tabular medical Covid dataset. We aim to show that our approach can reach good results on mixed-type tabular data in addition to image data. We also aim to show that WMSSDA competes and outperforms other DA approaches in an experimental setting very close to the QUALITOP setting, that is, real-world medical context with limited data and class imbalance.

Table 3.5 reports our entire experimental results on the Covid dataset. Results show that both WMSSDA variations lead to the best or second-best results in most cases, and in all cases on average. Those experimental results show that our approach can perform very well on tabular data. In this particular setting, we note that it is our standard version of WMSSDA

Setting	Method	Metric	1		2		3		4		5		Avg	
Single Best	NN	bACC	86.15 ± 1.15	•	85.57 ± 0.46	•	85.41 ± 0.37	•	83.47 ± 0.30	•	85.91 ± 2.92	•	85.30	•
		AUC	.9257 ± .0083	•	.9145 ± .0042	•	.8983 ± .0042	•	.8878 ± .0066	•	.9621 ± .0048	•	.9177	•
		F1	81.52 ± 1.14	•	82.06 ± 0.83	•	80.79 ± 1.00	•	78.80 ± 0.68	•	81.79 ± 3.55	•	80.99	•
	DAN	bACC	87.09 ± 1.01	•	80.96 ± 7.17	≡	79.50 ± 4.82	•	82.36 ± 1.14	•	92.90 ± 1.20	•	84.56	•
		AUC	.9261 ± .0074	•	.9052 ± .0079	•	.8910 ± .0069	•	.8928 ± .0055	•	.9646 ± .0090	•	.9159	•
		F1	85.64 ± 2.25	≡	75.02 ± 13.25	≡	71.73 ± 9.18	•	78.83 ± 2.88	≡	92.00 ± 1.36	•	80.64	•
	DANN	bACC	85.10 ± 0.82	•	85.82 ± 0.58	•	84.44 ± 0.97	•	83.31 ± 1.54	•	92.63 ± 1.05	•	86.26	•
		AUC	.9331 ± .0021	•	.9210 ± .0011	•	.9047 ± .0043	•	.9071 ± .0043	≡	.9701 ± .0025	•	.9272	•
		F1	80.01 ± 1.14	•	81.68 ± 1.31	•	78.74 ± 2.01	•	77.18 ± 2.39	•	90.60 ± 1.35	•	81.64	•
	DSAN	bACC	84.62 ± 3.17	•	84.82 ± 1.52	•	82.99 ± 1.32	•	83.30 ± 0.62	•	91.20 ± 2.03	•	85.39	•
		AUC	.9146 ± .0089	•	.9112 ± .0088	•	.8941 ± .0073	•	.8944 ± .0067	•	.9575 ± .0088	•	.9143	•
		F1	81.45 ± 4.51	•	84.56 ± 1.14	≡	77.88 ± 2.07	•	79.67 ± 1.03	•	90.94 ± 2.63	•	82.90	•
	FT	bACC	84.47 ± 0.97	•	85.12 ± 0.63	•	84.43 ± 0.61	•	83.31 ± 1.42	•	90.17 ± 2.39	•	85.50	•
		AUC	.9311 ± .0040	•	.9179 ± .0022	•	.9061 ± .0018	•	.9049 ± .0018	≡	.9689 ± .0014	•	.9258	•
		F1	79.23 ± 1.03	•	81.33 ± 1.98	•	78.70 ± 1.02	•	77.48 ± 2.00	•	86.71 ± 3.09	•	80.69	•
	MCD	bACC	86.67 ± 1.35	•	85.49 ± 0.34	•	85.23 ± 0.40	•	84.35 ± 0.85	≡	91.82 ± 2.42	•	86.71	•
		AUC	.9350 ± .0016	•	.9207 ± .0029	≡	.9089 ± .0014	•	**.9114 ± .0026**	○	.9718 ± .0010	•	.9296	•
		F1	82.38 ± 1.70	•	82.51 ± 1.18	•	79.79 ± 0.64	•	78.94 ± 1.69	•	89.26 ± 2.97	•	82.58	•
Source Combine	NN	bACC	87.93 ± 0.44	•	86.53 ± 0.22	•	85.26 ± 0.18	•	83.90 ± 0.34	•	94.45 ± 0.23	•	87.61	•
		AUC	.9389 ± .0019	•	.9168 ± .0033	•	.9111 ± .0037	≡	.9036 ± .0031	≡	.9751 ± .0009	•	.9291	•
		F1	83.86 ± 0.77	•	83.95 ± 0.52	•	79.07 ± 0.48	•	78.66 ± 0.44	•	93.94 ± 0.28	•	83.90	•
	DANN	bACC	88.12 ± 0.27	•	86.79 ± 0.40	≡	85.41 ± 0.15	•	83.98 ± 0.32	•	94.78 ± 0.13	≡	87.82	•
		AUC	.9391 ± .0023	≡	.9188 ± .0038	•	.9118 ± .0007	•	.9074 ± .0016	≡	.9747 ± .0002	•	.9304	•
		F1	84.34 ± 0.42	•	84.27 ± 0.51	•	79.15 ± 0.28	•	78.66 ± 0.28	•	94.05 ± 0.16	•	84.09	•
	DAN	bACC	87.40 ± 1.23	•	86.22 ± 0.22	•	84.78 ± 0.75	•	83.99 ± 0.38	•	94.25 ± 0.24	•	87.33	•
		AUC	.9306 ± .0077	•	.9238 ± .0023	≡	.9078 ± .0034	•	.9026 ± .0029	•	.9726 ± .0012	•	.9275	•
		F1	84.57 ± 1.18	•	83.13 ± 1.24	•	80.11 ± 1.31	•	79.18 ± 0.73	•	93.38 ± 0.83	≡	84.07	•
	DSAN	bACC	87.53 ± 1.02	•	85.85 ± 0.23	•	85.67 ± 1.02	•	84.21 ± 0.77	•	92.27 ± 1.70	•	87.11	•
		AUC	.9307 ± .0027	•	.9197 ± .0052	≡	.9069 ± .0046	•	.9053 ± .0051	≡	.9585 ± .0082	•	.9242	•
		F1	84.15 ± 1.52	•	84.15 ± 0.89	•	80.44 ± 1.71	•	78.76 ± 0.83	•	92.11 ± 1.57	•	83.92	•
	FT	bACC	85.88 ± 0.52	•	86.08 ± 0.50	•	85.39 ± 0.42	•	84.43 ± 1.12	≡	92.43 ± 1.53	•	86.84	•
		AUC	.9358 ± .0031	•	.9210 ± .0022	•	.9083 ± .0030	•	.9055 ± .0039	≡	.9706 ± .0019	•	.9282	•
		F1	81.00 ± 0.68	•	82.40 ± 0.72	•	79.76 ± 0.78	•	79.51 ± 1.02	•	89.73 ± 2.06	•	82.48	•
	MCD	bACC	84.74 ± 1.75	•	84.25 ± 1.73	•	83.81 ± 0.87	•	83.59 ± 0.60	•	90.98 ± 1.33	•	85.47	•
		AUC	.9288 ± .0028	•	.9103 ± .0094	•	.9060 ± .0028	•	.9091 ± .0027	≡	.9674 ± .0033	•	.9243	•
		F1	79.64 ± 2.83	•	80.80 ± 1.98	•	77.31 ± 1.40	•	77.65 ± 1.33	•	88.38 ± 1.78	•	80.76	•
Multi Source	MDAN	bACC	66.96 ± 7.40	•	73.42 ± 8.83	•	77.47 ± 2.40	•	72.11 ± 4.53	•	71.34 ± 8.91	•	72.26	•
		AUC	.8415 ± .0451	•	.8474 ± .0244	•	.8460 ± .0285	•	.8215 ± .0259	•	.8669 ± .0363	•	.8447	•
		F1	52.83 ± 17.89	•	65.41 ± 19.93	≡	72.76 ± 3.70	•	63.68 ± 11.61	•	63.02 ± 20.23	•	63.54	•
	MFSAN	bACC	86.42 ± 0.35	•	85.22 ± 0.39	•	83.07 ± 0.43	•	83.75 ± 0.77	•	92.20 ± 0.85	•	86.13	•
		AUC	.9300 ± .0035	•	.9196 ± .0018	•	.9059 ± .0027	•	.9108 ± .0024	≡	.9643 ± .0050	•	.9261	•
		F1	82.82 ± 0.40	•	81.87 ± 1.11	•	76.25 ± 0.84	•	77.36 ± 1.10	•	90.07 ± 1.06	•	81.67	•
	M3SDA	bACC	86.45 ± 0.73	•	85.00 ± 0.47	•	84.51 ± 0.27	•	82.92 ± 0.77	•	93.70 ± 0.86	•	86.52	•
		AUC	.9351 ± .0022	•	.9174 ± .0057	≡	.9067 ± .0015	•	.9010 ± .0024	•	.9695 ± .0022	•	.9259	•
		F1	82.24 ± 1.04	•	81.22 ± 0.76	•	78.10 ± 0.57	•	76.76 ± 1.20	•	92.85 ± 1.60	≡	82.23	•
	ABMSDA	bACC	86.78 ± 1.09	•	85.88 ± 0.97	≡	83.66 ± 1.16	•	81.00 ± 0.93	•	93.45 ± 1.91	≡	86.15	•
		AUC	.9281 ± .0043	•	.9190 ± .0033	•	.9068 ± .0034	•	.9049 ± .0021	≡	.9695 ± .0022	•	.9256	•
		F1	83.52 ± 1.48	•	82.60 ± 2.07	•	77.00 ± 2.01	•	73.44 ± 1.48	•	92.08 ± 2.63	≡	81.73	•
	WMSSDA	bACC	**89.24 ± 0.30**		**86.97 ± 0.28**		**87.03 ± 0.14**		**85.33 ± 0.24**		**94.88 ± 0.20**		**88.69**	
		AUC	**.9413 ± .0006**		.9240 ± .0014		**.9147 ± .0003**		.9074 ± .0016		**.9764 ± .0006**		**.9328**	
		F1	**86.87 ± 0.37**		**85.65 ± 0.21**		**83.05 ± 0.33**		**81.32 ± 0.26**		**94.34 ± 0.13**		**86.25**	
	WMSSDA-β	bACC	88.85 ± 0.61		86.61 ± 0.17		86.29 ± 0.24		84.60 ± 0.62		94.21 ± 0.27		88.11	
		AUC	.9378 ± .0027		**.9242 ± .0017**		.9112 ± .0017		.9065 ± .0064		.9735 ± .0010		.9306	
		F1	85.31 ± 1.39		84.50 ± 0.74		82.05 ± 0.89		80.36 ± 0.53		92.86 ± 0.50		85.02	

WMSSDA is: • significantly better, ≡ equivalent, ○ significantly worse, p-value: 0.05

Table 3.5: Comparative Study on the Real-World Medical Covid Dataset Dataset, with Limited and Imbalanced Data. The best and second-best results for each metric and each target domain appear in bold and underlined respectively. Results are evaluated on a binary classification task using metrics: balanced Accuracy, AUC, and the F_1-Score.

that performs the best, which is probably an indication that there is almost no concept shift in this dataset, unlike with the two benchmark image datasets, rendering the β version of the approach no better than the standard one. The results between the two versions of WMSSDA are close, with slightly better results for our standard version, thus, we performed the statistical tests evaluation based on the standard version of WMSSDA. The statistical comparison of the results shows that our approach WMSSDA leads to significantly better results than most other approaches, in the majority of cases. Our approach leads to significantly better results than any other state-of-the-art approach when considering the average performance over all target domains.

3.4.6 Ablation Study

Our method WMSSDA is composed of several important elements, in this section we perform an ablation study in order to evaluate the pertinence and usefulness of each component of WMSSDA. An ablation study is a type of experiment that is conducted to investigate the impact of removing, or disabling, specific components or features of an approach. To do so, we eliminate parts of our model and evaluate the results to understand their individual contributions to the overall performance and validate the pertinence of each component.

Approach	Branches	Regus	Weights
WMSSDA-A	Common	MD+ADV	No
WMSSDA-B	Specific	-	No
WMSSDA-C	Common+Specific	ADV	No
WMSSDA-D	Common+Specific	MD	No
WMSSDA-E	Common+Specific	MD+ADV	No
WMSSDA	Common+Specific	MD+ADV	Yes

Table 3.6: Ablation study compared approaches.

In this ablation study we compare five ablated versions of our WMSSDA approach with our complete method. Table 3.6 shows all WMSSDA versions compared in this study. In the column "Branches" it is indicated if the method contains both the common modules branch and the source domain specific modules branch, or only one of the two. Column "Regus" indicates if both the statistical and adversarial regularizations of the common branch are used, or only one of the two, or none in the case where only the specific branch is used. Finally, column "Weights" indicates if transfer contribution weights are computed and used during training to minimize Negative Transfer or not.

Method	Metric	MNIST	MNIST-M	SVHN	SYN	USPS	Avg
WMSSDA-A	bACC	94.95 ± 0.58	72.02 ± 1.24	67.69 ± 2.28	80.94 ± 0.49	96.57 ± 0.22	82.43
	AUC	.9985 ± .0001	.9620 ± .0024	.9429 ± .0054	.9783 ± .0012	.9989 ± .0002	.9761
	F1	94.89 ± 0.61	71.20 ± 1.35	65.64 ± 2.70	80.63 ± 0.53	96.83 ± 0.19	81.84
WMSSDA-B	bACC	93.77 ± 0.90	74.62 ± 1.50	66.34 ± 2.24	79.72 ± 1.20	95.89 ± 0.35	82.07
	AUC	.9980 ± .0003	.9721 ± .0021	.9414 ± .0048	.9805 ± .0013	.9982 ± .0005	.9781
	F1	93.71 ± 0.92	73.79 ± 1.72	62.83 ± 2.40	79.48 ± 1.18	96.06 ± 0.47	81.17
WMSSDA-C	bACC	93.95 ± 0.61	73.52 ± 1.32	69.61 ± 1.60	81.58 ± 0.95	96.60 ± 0.16	83.05
	AUC	.9982 ± .0002	.9709 ± .0017	.9482 ± .0049	.9813 ± .0014	.9984 ± .0004	.9794
	F1	93.79 ± 0.71	72.98 ± 1.58	67.29 ± 2.87	81.41 ± 0.99	96.73 ± 0.11	82.44
WMSSDA-D	bACC	95.01 ± 0.46	74.60 ± 1.33	69.27 ± 2.05	81.87 ± 1.42	96.28 ± 0.21	83.41
	AUC	.9988 ± .0002	.9736 ± .0025	.9505 ± .0035	.9828 ± .0019	.9987 ± .0003	.9809
	F1	94.93 ± 0.50	74.19 ± 1.40	67.16 ± 2.34	81.70 ± 1.43	96.38 ± 0.14	82.87
WMSSDA-E	bACC	95.16 ± 0.25	74.94 ± 0.76	69.20 ± 1.17	82.60 ± 0.46	96.60 ± 0.28	83.70
	AUC	**.9989 ± .0002**	.9737 ± .0024	.9501 ± .0016	.9833 ± .0008	.9987 ± .0003	.9809
	F1	95.11 ± 0.26	74.50 ± 0.88	67.11 ± 1.56	82.45 ± 0.43	96.65 ± 0.30	83.17
WMSSDA	bACC	**95.30 ± 0.21**	**75.05 ± 1.06**	**70.50 ± 1.21**	**82.85 ± 0.49**	**96.77 ± 0.30**	**84.10**
	AUC	.9988 ± .0001	**.9737 ± .0017**	**.9519 ± .0014**	**.9838 ± .0005**	**.9992 ± .0001**	**.9815**
	F1	**95.25 ± 0.21**	**74.62 ± 1.17**	**68.17 ± 2.01**	**82.70 ± 0.49**	**96.90 ± 0.29**	**83.53**

Table 3.7: Ablation study results.

Table 3.7 shows our experimental results on the Covid dataset for this ablation study. We compare our method with only the common modules branch with method WMSSDA-A and only the source domain specific modules branch with method WMSSDA-B. In both cases, average results are similar, with slightly better results for WMSSDA-A, showing the importance

of a shared latent space. Method WMSSDA-E contains both branches and obtains largely better results than WMSSDA-A and WMSSDA-B, showing the pertinence of an architecture combining both the common and specific branches to obtain the best possible results. We believe that the two branches architecture naturally decreases Negative Transfer as classifiers with higher confidence are given more importance in the ensemble pooling of results, which highly contributes to improving prediction quality. We also evaluate the pertinence of using both statistical and adversarial regularizations to learn a shared domain invariant latent space, with method WMSSDA-C using only the adversarial regularization, and method WMSSDA-D using only the statistical MD regularization. The results of WMSSDA-C and WMSSDA-D show that using a statistical only regularization leads to slightly better results than the adversarial one alone. This is contradictory with the actual consensus in the literature, that states that adversarial regularization is superior to statistical regularization for learning a common domain-invariant latent space. This can probably be explained by the fact that MD has been shown to be more pertinent and lead to better learning performance than MMD in a Multi-Source Domain Adaptation context in (Peng et al., 2019), making it slightly superior to adversarial alignment in this case. Method WMSSDA-E, which uses both MD and adversarial regularization for its shared latent space obtains better results than both WMSSDA-C and WMSSDA-D, showing that using both kind of regularizations allows to further align the latent space and lead to even better adaptation results. Finally, we observe that our complete WMSSDA approach leads to the best results overall, which seems to indicate that our transfer contribution weights are useful to limit Negative Transfer during training and have a positive effect on learning performance.

3.5 Discussion and Conclusion

In this chapter, we proposed an innovative MSSDA approach, Weighted Multi-Source Supervised Domain Adaptation (WMSSDA), which we evaluate and compare to other state-of-the-art approaches on limited and imbalanced data on both benchmark and real-world medical datasets. Our proposed approach is composed of a two branch architecture, learning both a shared domain invariant latent space and source domain specific latent spaces. The shared latent representation is learned and regularized both statistically and adversarially, the statistical regularization relies on a MD measure between source and target domains. The output of the MD regularization is used to compute transfer contribution weights that are applied to weight the impact of each source domain during training, limiting Negative Transfer. We show that our proposed WMSSDA outperforms most state-of-the-art approaches on both image benchmarks datasets and a real-world tabular medical dataset. We further analyze the relevance and importance of each component of our method by performing an ablation study, validating the overall architecture of our approach.

Overall, our experimental results seem to show that most Multi-Source Domain Adaptation approaches do not obtain significantly better results than single-source approaches in our experimental scenario. This seems to show that despite researchers efforts, Negative Transfer in Multi-Source Domain Adaptation is still an open critical problem that seems to limit the overall potential performance of state-of-the-art Multi-Source Domain Adaptation

approaches. Best performing DA approaches are still not able to fully avoid Negative Transfer. In our proposal of a new MSDA approach, we tried to limit Negative Transfer through the computation of transfer contribution weights that are applied as a scaling of the impact of each source domain in the training of the entire model. Our experimental results and ablation study show that this element of our approach is relevant and improves overall results. But even with this component, our proposed approach is yet not able to fully avoid Negative Transfer. Future works in the Domain Adaptation field should focus on finding better ways to handle this important matter.

We have evaluated our proposed approach in a specific data limited and imbalanced setting, as this is our setting of interest for the QUALITOP project. However, our approach WMSSDA is a generic MSDA approach and should perform well under any amount of data and class balance setting. An interesting perspective could be to evaluate the approach in a more standard MSDA evaluation setting, unrelated to the medical field and without imbalanced and limited data, to show how it performs and compares to other state-of-the-art DA approaches in such scenario.

Chapter 4

Application for Survival Outcome Prediction for Covid Patients

Contents

In this chapter, we showcase the complete application of this thesis work as a predictive Machine Learning (ML) pipeline, on a real-world Covid medical dataset. First, we define the application setting of the QUALITOP project. In section 4.2, we present the dataset on which we apply our entire pipeline. We then detail the design of the complete prediction pipeline that unifies the various works previously proposed within this manuscript. Section 4.4 shows our application results, with the analysis and interpretation of various components of the pipeline. We discuss how such a ML pipeline is to be deployed and integrated within a real application in the next section. Finally, we conclude the chapter with a summary and discussion of our applicative work.

4.1 The Application Setting of the QUALITOP Project

This thesis is part of the European research project QUALITOP: Monitoring multidimensional aspects of QUAlity of Life after cancer ImmunoTherapy - an Open smart digital Platform for personalized prevention and patient management[1] [2]. This project aims at improving the quality of life of patients suffering of cancer and undergoing immunotherapy treatment. Cancer immunotherapy has significantly progressed in the treatment of cancer. But immunotherapy is also responsible for lots of Immune-Related Adverse Events (irAEs), spanning from benign symptoms to life threatening reactions. The QUALITOP project aims to design a European immunotherapy-specific open smart digital platform, that will feature predictive Machine Learning models that will offer real-time recommendations to enhance patient care based on patient profiles, and help identify the factors influencing patients health status.

Our main goal within the QUALITOP project is to design ML prediction models that will be useful to provide real-time recommendations for a given patient that medical experts will use to help in their decisions. This thesis work focuses on designing such ML and Deep Learning (DL) tools, to predict the risks that a patient may face when undergoing an immunotherapy treatment. In such a context, there are two main predictive tasks of interest, survival outcome prediction, and Immune-Related Adverse Events prediction. The ability to evaluate the survival outcome of a cancer patient that would undergo immunotherapy is of critical importance. Indeed, it allows medical experts to make well-informed decisions regarding the patient treatment and care. By accurately predicting the potential outcomes and risks associated with immunotherapy for each individual patient, healthcare professionals can choose the best suited treatment depending to each patient specific needs, maximizing the chances of positive outcome and maximizing the overall patient quality of life. This information helps medical experts to optimize treatment strategies, provide appropriate support, and improve as much as possible the patient quality of life throughout their cancer journey. Similarly, the ability to anticipate the severity of adverse reactions that might occur because of an immunotherapy treatment is extremely valuable. Presently, there are no identified bio-markers that can help in predicting irAEs. Designing a ML model capable of predicting irAEs before they occur will not only help medical experts take better and more informed decisions, but will also show that crucial bio-markers actually exist within patients health data and can be exploited to anticipate immunotherapy adverse events. Such a predictive model helps to take proactive measures to mitigate potential irAEs and significantly improve patient outcomes.

A crucial objective of QUALITOP is to collect and aggregate real-world data from diverse European sources, to monitor the health status and quality of life of cancer patients undergoing immunotherapy. Health data of cancer patients undergoing immunotherapy treatments are collected at five partner hospitals within the European Union, in France, Spain, Portugal, United Kingdom, and Netherlands. There are several practical particularities about QUALITOP data that it is crucial to take account of in order to maximize predictive results of a ML

[1] https://cordis.europa.eu/project/id/875171
[2] https://h2020qualitop.liris.cnrs.fr/wordpress/index.php/project

model:

- Attribute noise. As defined in chapter 1 and 2, attribute noise is the combination of missing and erroneous values. As stated in previous chapters, missing and erroneous values are an issue that is present in almost all real-world data, and especially in a medical context. It is estimated that health features collected within the QUALITOP study contain about 20% to 30% missing values. As health features are manually collected and entered in the system, it is very probable that there are erroneous values in QUALITOP data.

- Multiple domains. QUALITOP data is collected within five distinct hospitals, each medical institution in a different European country. This is a real-world example of a Multi-Source Domain Adaptation (MSDA) setting, where we want to exploit knowledge from all sources in order to maximize inference results in each source, that is, each hospital.

- Class imbalance. QUALITOP data is imbalanced, whether we aim to predict the survival outcome of a patient or Immune-Related Adverse Events. For example, benign irAEs are much more common than severe ones. This must be accounted for when designing a predictive model. Failing to account for this imbalance might result in a predictive model that can only predict benign adverse events, while overlooking the more serious and significant adverse reactions.

Those matters have all been addressed in this manuscript, the complete predictive ML pipeline that is presented and applied in this chapter is the combination of all our previously proposed and discussed approaches.

While working on the QUALITOP project, we encountered data sharing issues that were not initially anticipated during project conception. Indeed, hospitals data sharing policies are very strict, and even within the framework of the European QUALITOP project, it was impossible to get access to other countries data before the end of this thesis. Therefore, we searched for a real-world public medical dataset that would be as close as possible to the real QUALITOP applicative setting. In this chapter, we unite our complete work within a Machine Learning predictive pipeline, and showcase its application on a real-world medical Covid dataset, in an applicative setting that simulates closely the QUALITOP context.

4.2 A Real-World Medical Dataset Close to QUALITOP Data

As we were not able to obtain access to QUALITOP data in time, we searched for a dataset that would be as close as possible to our application context. We found a dataset provided by the Mexican government that contains anonymized health data from an important number of patients that suffer from, or have symptoms that could be related to, Covid19. It is downloadable from the following Kaggle repository[3]. A total of 1,048,576 unique patients

[3] https://www.kaggle.com/datasets/meirnizri/covid19-dataset

suspected of suffering from Covid19 are included in this dataset, with 20 tabular mixed-type (continuous and categorical) features. It is known that the majority of people that are infected with Covid19 experience mild to moderate respiratory symptoms, which often resolve without the need for specialized medical intervention. Those cases are characterized by mild to moderate symptoms, such as fever, cough, fatigue, and shortness of breath. These individuals typically recover through supportive care and self-isolation, contributing to the relatively low overall fatality rate of the disease. On the other hand, certain vulnerable groups face a higher risk of developing severe illness, and so, a higher risk of dying. This is the case of older individuals, and those with pre-existing medical conditions. This vulnerable population includes patients with underlying health issues like cardiovascular disorders, diabetes, chronic respiratory diseases such as asthma, and other chronic illnesses. This Covid dataset includes indicators of such diseases, which are primordial information in order to evaluate the risk an infected patient is facing. Having access to such an amount of Covid19 related data is extremely valuable, as it allows researchers and healthcare professionals to gain significant insights into the patterns, risk factors, and outcomes associated with Covid19. In this application chapter, we will showcase the application of our complete Machine Learning predictive pipeline on this real-world medical dataset, with a scenario that is closely related to the QUALITOP medical setting.

This Covid dataset under consideration presents interesting similarities with QUALITOP data. They are both medical datasets containing anonymised health data, with which we aim to predict the survival outcome of individuals based on their actual data. Just like QUALITOP data, the patients in this Covid dataset also originate from various distinct medical institutions, making it a multi-source context. However, there are also some important differences to account for between the two datasets. The first significant difference is the size of the datasets. The Covid dataset is notably larger, as it includes a substantial number of patients. In contrast, QUALITOP data includes only around 1,000 patients per medical institute. The second major difference is in the presence of missing values. The Covid dataset contains almost no missing values. Whereas QUALITOP data exhibits an important amount of missing values, estimated to be between 20% to 30% in health features. In order to align the Covid dataset more closely with our scenario of interest, it requires additional preprocessing. This preprocessing aims to simulate the limited number of patients, and relatively high amount of missing value, observed in the QUALITOP data. Making the Covid dataset more comparable to QUALITOP data, and suitable for the application of our ML predictive pipeline.

4.2.1 Preprocessing the Dataset

The first step is to split the datasets in domains, based on the medical institution in which patients are treated. Initially, the medical institution of each patient is indicated in a categorical variable, we split patients in separate datasets grouped by domains. There are 13 distinct medical units in the original dataset. We choose to select 5 medical institutions out of those 13, as there are five partner hospitals actively collecting cancer immunotherapy related health data in the QUALITOP project, leading to a total of 5 domains. We based the selection of those medical units on an exploratory analysis of the dataset. We considered

only medical institutions with more than two thousand patients, as we want at least 1,000 training patients and 1,000 test patients in each domain. We then selected the 5 institutions showing the most feature distribution variability from one another, as shown in the next subsection. This ensures that the domain shift is maximized across domains of the dataset. Applying a well-performing MSDA approach to such a dataset would show a good capacity at handling domain shift in a real-world setting. Finally, we randomly select 1,000 training patients in each medical institute to align the size of the Covid dataset with the number of patients included in QUALITOP data. We also randomly select another 1,000 patients for the test set of each medical institution, which will be used for the evaluation of our predictive pipeline.

The second step of this preprocessing is the injection of missing values within the Covid dataset, to better align the application scenario with the QUALITOP one. It is estimated that health features collected within the QUALITOP study contain about 20% to 30% missing values. Therefore, we inject missing values in the Covid dataset at a missing rate set to 25%. We inject those missing values based on the Missing Not At Random (MNAR) mechanism, such as previously described in chapter 2, as it is the most commonly encountered missingness mechanism in real-life data, and the hardest to deal with. We choose not to include missing values within the "Sex" and "Age" features, as missing values in such basic information are rare or nonexistent in practice. Values of all other features have the same 25% probability to be missing. This leads to a total amount of missing values of about 22% on average for each domain.

4.2.2 Exploratory Analysis of the Covid Dataset

In data science, an important step is the exploratory analysis of the dataset of interest. It consists in using various techniques and visualizations to examine the data, which helps to gain insights, identify patterns, trends, and potential relationships between variables. In our multi-source context, we are especially interested in detecting and evaluating the domain shifts in the dataset.

Figure 4.1 shows the univariate marginal distributions of four features of the Covid dataset across the five domains. Visualizing the univariate marginal distribution of a feature in each domain visually highlights differences between domains. As the Covid dataset has been anonymized, it is impossible to know the kind of hospital to which each medical institution corresponds. Visualizing the distribution of features can help better understand the particularities of each domain.

The first visualized distributions, on the top left of the figure, are those of the "Sex" feature across domains. We observe very balanced sex distributions across all domains, except for domain 4, where there are considerably more males than females, with about 61.4% male patients in this institute. This is a drastically different sex distribution compared to the other four domains overall balanced distributions. On the top right of the figure, we visualize the "Obesity" feature marginal distributions across domains. In domains 1, 2, 3 and 5, we observe a vast majority of non-obese patients, with less than 20% obese patients in those four institutes. As previously, domain 4 exhibits a very different feature distribution, as there are

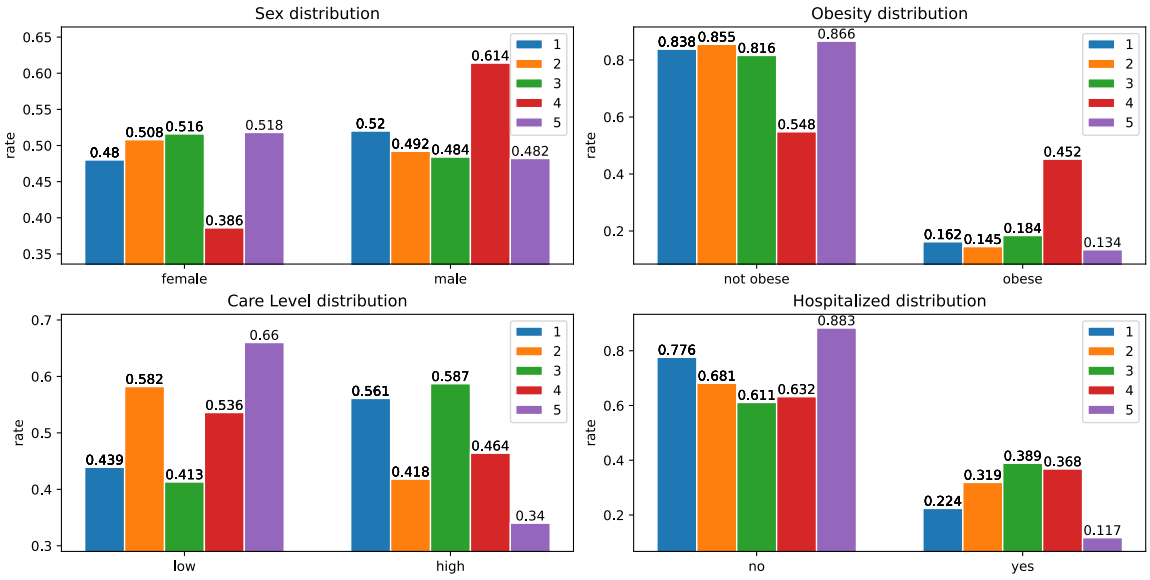


Figure 4.1: Distributions of four binary features across the 5 domains in the Covid dataset. We observe a covariate shift between the five domains, as univariate marginal distributions are different across pairs of features. Therefore, $P(X_{\mathbb{D}_1}) \neq P(X_{\mathbb{D}_2}) \neq \ldots \neq P(X_{\mathbb{D}_5})$.

almost as many obese patients as non-obese patients in this medical unit. This might be correlated to the fact that there nearly two thirds of patients are males in this institution.

On the bottom left of the figure, we visualize the "Care Level" feature distributions across the five institutions. We note a higher proportion of patients in institutions 1 and 3 at a high care level than in other domains, and note that most patients of institution 5 are at a low care level, with about 66% of patients at a low care level. We then visualize the "Hospitalized" feature on the bottom right of the figure, where "yes" means that patients are being hospitalized for their treatment, while "no" signifies that patients are sent home to recover. The marginal distributions are relatively similar for most institutions, except for institution 5 that is largely dissimilar to others. In domain 5, almost 90% of patients are sent home to recover, which seems correlated to the fact that most patients of institution 5 are at a low care level. On the other hand, we also note that many patients in domain 1 are also sent home to recover, while most patients are at a high care level. This might indicate that those patients are treated more intensively despite being hospitalized from their home.

We do not know to which kind of hospital each medical institution corresponds, but we observe different marginal distributions for most features between domains. This is probably due to a sample selection bias, and thus, different kinds of patients across institutions.

This simple visualization of the distributions across the Covid dataset domains is enough to conclude that there is a covariate shift between the five domains. The covariate shift observed between the domains of this dataset is probably very similar to the one existing between the medical institutions in our QUALITOP context.

There is no standard way of measuring covariate shift between multiple domains in the literature. Therefore, we propose a way of measuring the covariate shift between each pair of domains in the Covid dataset, relying on the normalized Fréchet distance (Dowson and

Landau, 1982). It is noted and computed between two domains as:

$$\Delta(\mathbb{D}_1, \mathbb{D}_2) = ||\mu_{\mathbb{D}_1} - \mu_{\mathbb{D}_2}||_2^2 + tr(cov(\mathbb{D}_1) + cov(\mathbb{D}_2) - 2\sqrt{cov(\mathbb{D}_1) \cdot cov(\mathbb{D}_2)})$$

With $cov(\mathbb{D})$ the covariate matrix of the data sample of domain $\mathbb{D}$. The Fréchet distance is a measure of similarity between two multivariate distributions, its normalized version is the mean of the distance between each features in the data. The normalized Fréchet distance value between two domains does not make much sense on its own, as all that can be concluded is that the further it is from 0, the most unrelated the two domains are. In our proposed way of measuring the covariate shift, we first compute the normalized Fréchet distance between the train set and the validation set of each domain, which gives a reference distance value for each domain, noted and computed as $\Delta_0(\mathbb{D}) = \Delta(\mathbb{D}_{train}, \mathbb{D}_{val})$. Then, we compute the normalized Fréchet distances between each pair of domains, and compute the ratio between the mean of the reference values of both domains, and the normalized Fréchet distance value between the domain and the compared one. We note $\Delta_{cov}(\mathbb{D}_1, \mathbb{D}_2)$, the covariate distance between domains 1 and 2, computed as:

$$\Delta_{cov}(\mathbb{D}_1, \mathbb{D}_2) = \frac{\Delta(\mathbb{D}_1, \mathbb{D}_2)}{\frac{1}{2} \cdot (\Delta_0(\mathbb{D}_1) + \Delta_0(\mathbb{D}_2))}$$

We consider that a covariate distance above 1 means that the two domains are different, and the higher the value is, the more covariate shift there is between the two domains. Table 4.1 shows the results of our proposed way of measuring covariate shift between each pair of domains on the Covid dataset.

	Domain 1	Domain 2	Domain 3	Domain 4	Domain 5	Avg
Domain 1	1	4.1353	6.6404	15.8727	9.4139	9.0156
Domain 2	4.1353	1	5.4521	12.9550	7.7180	7.5651
Domain 3	6.6404	5.4521	1	10.9721	19.7236	10.6970
Domain 4	15.8727	12.9550	10.9721	1	24.9527	16.1881
Domain 5	9.4139	7.7180	19.7236	24.9527	1	15.4520

Table 4.1: Covariate shift ratios between each pair of domains in the Covid dataset.

In each cell of this table, the covariate distance value can be understood as the amount of times the covariate shift is superior between a pair of domains compared to the average reference shift that exists between the training and validation set of both domains respectively. For example, we can consider that there is a shift about 15.9 times higher between domains 1 and 4 compared to their mean intrinsic shift between their training and validation sets. In the last column we see the average of the covariate distances for each domain. This emphasizes the fact that domains 4 and 5 are the most different domains compared to the others, with average distances much higher than domains 1, 2 and 3.

Table 4.2 shows the survival outcome distribution in each domain of the Covid dataset, revealing a prior shift between the domains of the dataset. Indeed, we observe that the label distribution is imbalanced in each domain and the representation is different from one domain to the other, thus, $P(Y_{\mathbb{D}_1}) \neq P(Y_{\mathbb{D}_2}) \neq \ldots \neq P(Y_{\mathbb{D}_5})$. For example, we observe a low

mortality in domain 5, with 96% patients that survive, whereas we observe a relatively high mortality in domain 3, with only 86.3% patients that survive. This highlights an important prior shift, with a high class imbalance in all domains. Indeed, there is an average ratio of 9:1 patients that survive compared to those that do not survive. It is important to be aware of such matter before applying a ML predictive model to such dataset, as imbalance needs to be accounted for in classification tasks to ensure good inference results.

Class	Representation	Domain 1	Domain 2	Domain 3	Domain 4	Domain 5	Overall
Negative	Percentage	8%	13.5%	13.7%	11.1%	4%	10.06%
	Samples Count	80	135	137	111	40	503
Positive	Percentage	92%	86.5%	86.3%	88.9%	96%	89.94%
	Samples Count	920	865	863	889	960	4497

Table 4.2: Covid label distribution per domain, showing that there is a prior shift across domains, $P(Y_{\mathbb{D}_1}) \neq P(Y_{\mathbb{D}_2}) \neq \ldots \neq P(Y_{\mathbb{D}_5})$.

We note from this table that patients from domain 5 have much higher chances of survival compared to other domains, with only 4% patients not surviving. This seems correlated to the fact that most patients in this medical unit are sent home for their treatment (feature "Hospitalized" in figure 4.1). This probably means that patients in this medical institution are low risk patients, we also note that domain 5 is the medical institution with the least obese patients which further supports this theory.

4.3 A Complete Prediction Pipeline

In this section, we design a predictive Machine Learning pipeline that is the unification of all the approaches proposed in this manuscript and that will be applied on the Covid dataset. Figure 4.2 shows our entire Machine Learning predictive pipeline, from preprocessing of the raw domains data to the final survival outcome prediction.

In this figure the raw domains of the Covid dataset are on the far left, those are noted $\tilde{\mathbb{S}}_1, \ldots, \tilde{\mathbb{S}}_s$ for the s source domains, and $\tilde{\mathbb{T}}$ for the target domain. The pipeline must be trained and applied once for each target domain. The first step in the pipeline is to apply our attribute noise correction method, data Denoising and Imputation in One Step (DIOS), m times per domain. Which leads to m corrected datasets per domain, noted $\mathring{\mathbb{S}}_{i,1}, \ldots, \mathring{\mathbb{S}}_{i,m}$ for the i-th source domain, and $\mathring{\mathbb{T}}_1, \ldots, \mathring{\mathbb{T}}_m$ for the target domain. In order to take account of attribute noise correction uncertainty we integrate our framework Single-Hotpatching (S-HOT) into the training of the predictive model. Therefore, means and standard deviations of the m corrections are computed for each domain, noted $\mu_{\mathbb{S}_i}$ and $\sigma_{\mathbb{S}_i}$ for the i-th source domain, and $\mu_{\mathbb{T}}$ and $\sigma_{\mathbb{T}}$ for the target domain. Then, during batch extraction for the training of the predictive model, corrected values are drawn on a Gaussian distribution, parameterized with the means and standard deviations of the corresponding domain. Based on this batch extraction process, we train our MSDA predictive model, Weighted Multi-Source Supervised Domain Adaptation (WMSSDA). This model exploits knowledge from source domains in order to improve predictions on the target domain. Once the WMSSDA predictive model is trained, its output probabilities are calibrated through a Dirichlet calibration (Kull

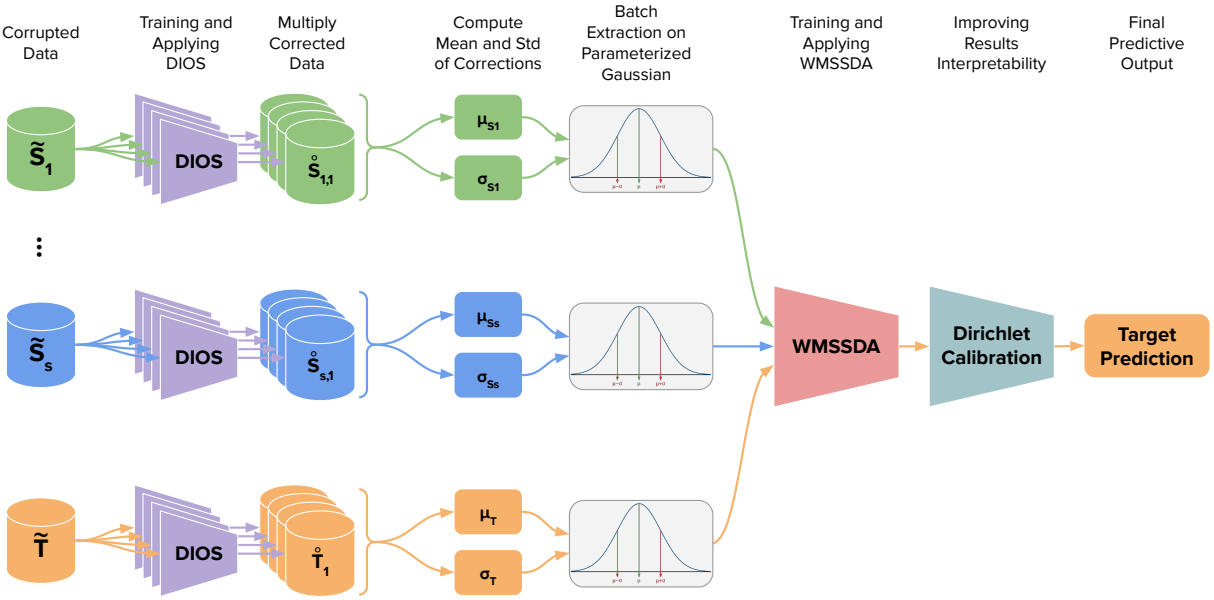


Figure 4.2: Complete pipeline training and application.

et al., 2019), which improves the reliance and interpretability of the results. Finally, once the entire pipeline has been trained for a target domain, high quality predictions can be obtained for new patients.

4.3.1 Attribute Noise Correction

In chapter 2, we proposed and discussed an attribute noise correction approach, data Denoising and Imputation in One Step (DIOS). This Auto-Encoder (AE) based approach is able to impute missing values and correct erroneous ones in a dataset simultaneously. We experimentally showed that applying such an approach as a preprocessing step could greatly help a predictive model reach better inference results. In our pipeline, we apply DIOS m times to each domain, leading to m corrections per domain. Applying DIOS m times is useful in the next steps of the pipeline, where we want to train a predictive model while taking account of correction uncertainty, that is, taking account of the variability in DIOS results from one execution to the other.

In our pipeline, the architecture of DIOS is a fully-connected Neural Network (NN), with dimensions $\{80, 80, 2\}$, with an input of size 17 for each domain, the output is binary as the pipeline is applied to a survival outcome prediction. DIOS is trained with a learning rate set to 10^{-4}, for a maximum number of 750 epochs.

4.3.2 Multi-Source Prediction

In chapter 3, we proposed and discussed a Multi-Source Supervised Domain Adaptation (MSSDA) approach, Weighted Multi-Source Supervised Domain Adaptation (WMSSDA). WMSSDA is a NN based approach that improves predictions on a target domain by exploiting related labeled source domains data. It is composed of a two branch architecture, where both a shared domain invariant latent space and source domain specific latent spaces are learned. We previously evaluated our WMSSDA approach and showed that exploiting related source domains data could drastically improve target domain predictions, which is highly pertinent in our specific applicative setting.

In our pipeline, the architecture of the WMSSDA is composed of a fully-connected encoder layer (size 128), followed by a neck layer (size 64), and the classifier layer (size 2). A batch normalization operation is used in-between each fully-connected layer for better numerical stability, a dropout of 20% is applied after the encoder output to improve the generalization capacity of the predictive model and avoid overfitting. The model is trained for 100 epochs, the learning rate is set to 10^{-5}, and the activation function applied after each layer is a Leaky ReLU, with a slope coefficient set to 0.2. In our pipeline, the training of WMSSDA is based on the S-HOT framework, this point is explained and discussed in the next subsection.

4.3.3 Improving Training by Taking Account of Correction Uncertainty

In chapter 3, we proposed and discussed a framework to take account of imputation uncertainty during the training of a Neural Network to improve its generalization capacity, Single-Hotpatching (S-HOT). This framework relies on computing m imputations of a dataset, and draw imputed values on a normal distribution parameterized with the mean and standard deviation of the m imputations during the training of a Neural Network. In this application, we extend the S-HOT framework to an application in a complete attribute noise correction. Thus, we compute m corrections of each domain of the Covid dataset and apply S-HOT in the same way as above to train our WMSSDA model. Our results in the next section show the successful application of the S-HOT framework to improve a Neural Network prediction results by taking account of attribute noise correction uncertainty during its training.

4.3.4 Improving Prediction Reliance With Dirichlet Calibration

It has been recently showed that modern NNs are poorly calibrated (Guo et al., 2017). In a real-world situation, a NN prediction should be as accurate as possible, but also, the output probabilities of the model should reflect how likely it is that the prediction is correct. Essentially, the output probabilities of a predictive model should serve as an indication of the model confidence. Despite their superior performance compared to older Machine Learning models, modern NNs have been found to be poorly calibrated. In fact, they tend to overestimate their own confidence in the probability output. As a result, it becomes challenging to estimate the likelihood of a correct prediction based on the provided probabilities. This mis-calibration issue is critical across various applications, and it is especially concerning in

a medical context. Making decisions based on model predictions in healthcare can significantly impact patients health and overall quality of life. If a NN is poorly calibrated, it may assign high probabilities to incorrect predictions, leading to misguided decisions in patient care and treatment. To make predictive models usable and reliable in a medical setting, it is essential that the model produces probabilities that better reflect the correctness of the predictions. A well-calibrated predictive model ensures that a medical expert can confidently use the model predictions to make informed decisions.

Most existing solutions for ML models calibration in the literature are post-processing approaches (Guo et al., 2017), which is a huge advantage as it is possible to apply such approach to any trained ML model. Among calibration methods is the isotonic regression (Zadrozny and Elkan, 2002), one of the most common non-parametric calibration method. It consists in fitting a piecewise-constant function to the predicted probabilities. It adjusts the original probabilities to be monotonically increasing, thereby improving calibration. In (Platt, 2000), the authors propose Platt scaling, a parametric approach for calibration, where the output of the model to calibrate is fed as input for a logistic regression model, which is trained to learn probabilities on a validation set. Once the logistic model has been fit with the validation set, it can simply be applied to calibrate the predictions obtained on the test set. More recently, (Guo et al., 2017) proposed a simple extension of Platt scaling, namely, Temperature scaling. Temperature scaling is based on the optimization of a temperature parameter T, that is applied to scale the model probabilities before the softmax operation. The parameter T is found by minimizing the NLL loss over the validation set. As the scale T is applied to all probabilities before the softmax operation, it does not affect the accuracy of the model, instead, it only re-scales the probabilities to better reflect the model confidence. In paper (Kull et al., 2019), the authors propose Dirichlet calibration, another post-processing approach that can be applied both to binary and multi-class classification predictions. It consists in applying a log transformation to the uncalibrated probabilities, followed by a fully-connected NN layer and a softmax operation, leading to calibrated probabilities. Other calibration approaches that are not applied as a post-processing step exist, such as (Lakshminarayanan et al., 2017), where the authors propose a simple approach to model calibration based on uncertainty estimation from the training of an ensemble of models. Such an approach is very effective for calibration, but is unusable in many real-world scenarios, as training multiple versions of a DL model might take too much time.

In order to calibrate the probabilities output of our ML predictive pipeline, we use the Dirichlet calibration implementation from (Kull et al., 2019)[4]. We choose the Dirichlet calibration as it is a state-of-the-art post-processing calibration approach that is very simple to use on any trained Machine Learning model. The method can be applied for binary classification, as it is the case in our application on survival outcome prediction, but is also effective in a multi-class context, as would be the case in an irAEs prediction context. By applying this post-processing calibration technique, we aim to significantly enhance the relevance and reliability of the model predictions, making it a better and more valuable tool for helping medical experts in their decisions.

[4] https://github.com/dirichletcal/dirichlet_python

4.4 Results

In this section, we present our complete applicative results on the real-world Covid dataset.

4.4.1 Application Setting

We apply several pipelines in a comparative and ablation study, to validate that all components of our proposed pipeline play an important role in improving the overall predictions quality. First, we define four baseline pipelines that are exclusively composed of components from the literature.

- KNN-SPOL-NN. This baseline pipeline is composed of a KNN imputer (Troyanskaya et al., 2001), that imputes missing values based on known values of closest instances in the training set, followed by the standard Polishing method (Teng, 2004), that corrects erroneous values such as described in chapter 2. Once raw data have been imputed and corrected, a Neural Network is trained and applied on the target domain to obtain final predictions.
- KNN-PPOL-NN. Similarly to the previous pipeline, this one is composed of a KNN imputer, which is followed by the PANDA-Polishing method (Van Hulse et al., 2007), a more advanced polishing approach described in chapter 2. The last component of the pipeline is a Neural Network trained and applied on the target domain to obtain final predictions.
- MISS-SPOL-NN. This pipeline is composed of the MISSFOREST imputation method (Stekhoven and Bühlmann, 2012), an imputation approach based on random forest described in chapter 1, followed by the standard Polishing method, and a Neural Network trained and applied on the target domain to obtain final predictions.
- MISS-PPOL-NN. This pipeline is composed of the MISSFOREST imputation method, followed by the PANDA-Polishing method, and a Neural Network trained and applied on the target domain to obtain final predictions.

Then, we define a similar baseline pipeline based on DIOS corrections, which shows the advantage of using DIOS compared to a combination of an imputation and a correction approach from the literature. We extend this pipeline with the addition of a calibration component, such as described in section 4.3.4.

- DIOS-NN. The first component of the pipeline is the application of our attribute noise correction approach DIOS, which imputes missing values and correct erroneous ones in one step. it is followed by a Neural Network, trained and applied on the target domain to obtain final predictions.
- DIOS-NN-CAL. This pipeline adds a calibration component to the previous DIOS-NN pipeline, improving the overall prediction performance, the relevance and the reliability of the obtained prediction probabilities.

Finally, we define four Multi-Source Supervised Domain Adaptation (MSSDA) pipelines, based on DIOS corrections.

- DIOS-M^{3}SDA. This pipeline is composed of the application of DIOS, followed by the multi-source approach M^{3}SDA (Peng et al., 2019), which matches multiple source domains to the target domain using a moment matching discrepancy, the Moment Distance (MD), described in chapter 1. This pipeline serves as a Domain Adaptation (DA) baseline to compare the results obtained with our more advanced MSSDA pipelines.
- DIOS-WMSSDA. This pipeline is the equivalent of the previous one, where we replace the M^{3}SDA approach with our own WMSSDA approach. Comparing this pipeline results with those of the previous one will validate the fact that our proposed WMSSDA approach is superior and more pertinent to use in our specific MSSDA context.
- DIOS-WMSSDA-SHOT. This pipeline is an improved version of the previous one, where the WMSSDA model is trained through our S-HOT framework, such as described in section 4.3.3. Results obtained with this pipeline will demonstrate the advantage of taking account of attribute noise correction uncertainty during NN training for improved inference results.
- DIOS-WMSSDA-SHOT-CAL. Finally, our final and complete applicative pipeline builds on the previous one, with the addition of a calibration component, such as described in section 4.3.4. This component acts to calibrate the confidence evaluation of the results, and to improve the relevance of obtained prediction probabilities, increasing the reliability of results for medical experts.

Predictive results are evaluated with three classification metrics: the balanced Accuracy, which is indicative of the pertinence of the predictions per class, the Area Under the Curve (AUC), which is indicative of the discriminatory power of the model, and the F_1-score, which is indicative of the balance between precision and recall. All pipelines are trained and applied for a total of 10 times per target domain, to obtain consistent results and for fair comparison. To better assess the obtained results, we statistically compare the results of the DIOS-WMSSDA-SHOT-CAL pipeline to each of the compared pipelines with t-tests. The results of those statistical tests are used to determine if the DIOS-WMSSDA-SHOT-CAL pipeline performs significantly better, even, or worse than other pipelines, based on a p-values set to 0.05. The results of the t-tests are symbolized in the result table as either a bullet $\bullet$, a circle $\circ$, or an equivalent symbol $\equiv$. The bullet is used to signify that DIOS-WMSSDA-SHOT-CAL is significantly better than the compared pipeline, the circle signifies the opposite, and the equivalent means that there is no significant difference between the two compared pipelines.

4.4.2 Prediction Results

Table 4.3 shows the evaluation of the entire application results of all compared pipelines for each target domain of the Covid dataset.

Setting	Method	Metric	1		2		3		4		5		Avg	
Baselines	KNN-SPOL-NN	bACC	80.81 ± 1.74	•	78.53 ± 1.14	•	78.03 ± 0.92	•	78.40 ± 0.95	•	83.58 ± 2.85	•	79.87	•
		AUC	$.8772 \pm .0116$	•	$.8353 \pm .0206$	•	$.8454 \pm .0058$	•	$.8352 \pm .0072$	•	$.9291 \pm .0126$	•	.8644	•
		F1	80.78 ± 2.38	•	83.08 ± 0.77	•	77.14 ± 1.17	•	78.13 ± 1.27	•	81.91 ± 4.43	•	80.21	•
	KNN-PPOL-NN	bACC	80.18 ± 2.12	•	81.06 ± 0.91	≡	77.42 ± 0.48	•	78.99 ± 0.64	•	80.74 ± 3.34	•	79.68	•
		AUC	$.8868 \pm .0098$	•	$.8838 \pm .0031$	•	$.8413 \pm .0053$	•	$.8493 \pm .0062$	•	$.9391 \pm .0206$	•	.8801	•
		F1	77.19 ± 4.09	•	79.26 ± 1.91	•	74.53 ± 1.09	•	77.84 ± 0.93	•	74.42 ± 4.75	•	76.65	•
	MISS-SPOL-NN	bACC	81.08 ± 1.07	•	80.03 ± 1.08	•	69.80 ± 3.19	•	75.37 ± 1.85	•	78.70 ± 3.79	•	76.99	•
		AUC	$.8883 \pm .0054$	•	$.8841 \pm .0059$	•	$.7628 \pm .0264$	•	$.8155 \pm .0124$	•	$.9103 \pm .0265$	•	.8522	•
		F1	78.93 ± 1.53	•	79.07 ± 1.22	•	73.71 ± 1.61	•	75.07 ± 1.96	•	77.08 ± 4.10	•	76.77	•
	MISS-PPOL-NN	bACC	78.78 ± 2.25	•	81.56 ± 0.88	≡	75.31 ± 0.89	•	78.54 ± 0.72	•	77.40 ± 4.77	•	78.32	•
		AUC	$.8990 \pm .0063$	•	$.8800 \pm .0075$	•	$.8120 \pm .0080$	•	$.8311 \pm .0066$	•	$.8861 \pm .0200$	•	.8616	•
		F1	73.16 ± 3.18	•	80.22 ± 1.35	•	72.91 ± 0.80	•	76.36 ± 1.40	•	76.91 ± 7.19	•	75.91	•
	DIOS-NN	bACC	82.20 ± 1.52	•	79.82 ± 1.23	•	81.12 ± 0.76	•	79.14 ± 0.85	•	81.44 ± 3.10	•	80.74	•
		AUC	$.8923 \pm .0043$	•	$.8762 \pm .0089$	•	$.8647 \pm .0055$	•	$.8450 \pm .0064$	•	$.9261 \pm .0139$	•	.8809	•
		F1	78.85 ± 3.37	•	80.64 ± 1.11	•	76.73 ± 1.10	•	75.47 ± 1.54	•	78.65 ± 4.51	•	78.07	•
	DIOS-NN-CAL	bACC	82.38 ± 1.82	•	79.73 ± 1.36	•	81.12 ± 0.96	•	79.11 ± 0.80	•	84.46 ± 2.00	•	81.36	•
		AUC	$.8923 \pm .0043$	•	$.8762 \pm .0089$	•	$.8647 \pm .0055$	•	$.8456 \pm .0071$	•	$.9261 \pm .0139$	•	.8810	•
		F1	85.26 ± 0.72	•	82.48 ± 0.71	•	79.33 ± 0.54	•	79.11 ± 0.96	•	88.31 ± 1.69	•	82.90	•
Domain Adaptation	DIOS-M3SDA	bACC	82.26 ± 1.34	•	81.80 ± 0.96	≡	79.29 ± 2.06	•	78.40 ± 1.36	•	89.80 ± 1.60	•	82.31	•
		AUC	$.9068 \pm .0045$	•	$.8949 \pm .0044$	•	$.8738 \pm .0048$	•	$.8598 \pm .0044$	•	$.9608 \pm .0096$	•	.8992	•
		F1	78.32 ± 2.83	•	81.41 ± 1.32	•	71.53 ± 3.90	•	71.51 ± 2.71	•	87.67 ± 1.90	•	78.09	•
	DIOS-WMSSDA	bACC	85.57 ± 1.06	≡	81.67 ± 1.59	≡	81.89 ± 0.53	•	80.87 ± 0.72	≡	89.91 ± 1.23	•	83.98	•
		AUC	$.9118 \pm .0032$	≡	$.8985 \pm .0050$	≡	$\mathbf{.8799 \pm .0028}$	≡	$.8679 \pm .0036$	•	$.9668 \pm .0043$	≡	.9050	•
		F1	83.92 ± 1.75	•	$\mathbf{84.09 \pm 0.98}$	≡	79.60 ± 0.77	•	79.23 ± 0.72	•	90.09 ± 2.02	•	83.38	•
	DIOS-WMSSDA-SHOT	bACC	85.26 ± 0.43	≡	81.95 ± 1.44	≡	82.19 ± 0.35	≡	$\mathbf{81.50 \pm 0.71}$	≡	90.78 ± 0.60	≡	84.34	•
		AUC	$\mathbf{.9129 \pm .0026}$	≡	$\mathbf{.8996 \pm .0041}$	≡	$.8798 \pm .0052$	≡	$\mathbf{.8711 \pm .0021}$	≡	$\mathbf{.9700 \pm .0044}$	≡	**.9067**	○
		F1	83.74 ± 1.40	•	83.92 ± 0.56	≡	79.19 ± 1.16	•	79.50 ± 1.12	•	91.31 ± 0.92	≡	83.53	•
	DIOS-WMSSDA-SHOT-CAL	bACC	$\mathbf{85.59 \pm 0.56}$		$\mathbf{81.97 \pm 1.16}$		$\mathbf{82.47 \pm 0.33}$		81.29 ± 0.47		$\mathbf{91.18 \pm 0.82}$		**84.50**	
		AUC	$\mathbf{.9129 \pm .0026}$		$\mathbf{.8996 \pm .0041}$		$.8798 \pm .0052$		$.8711 \pm .0021$		$\mathbf{.9700 \pm .0044}$		.9067	
		F1	$\mathbf{86.91 \pm 0.47}$		83.90 ± 0.23		$\mathbf{80.55 \pm 0.30}$		$\mathbf{81.18 \pm 0.35}$		$\mathbf{91.96 \pm 0.46}$		**84.90**	

DIOS-WMSSDA-SHOT-CAL is: • significantly better, ≡ equivalent, ○ significantly worse

Table 4.3: Evaluation of the entire application results of all compared pipelines for each target domain of the Covid dataset. Bold values are the highest value for each metric and each target domain, while underlined values are the second best value for each metric and each target domain. The first four pipelines are baselines composed of elements from the literature, with an imputation method, a correction method and a fully-connected Neural Network for predictions. The fifth line is our baseline pipeline, composed of our DIOS attribute noise correction approach, followed by a fully-connected Neural Network for predictions. The sixth line is an extension of our previous baseline pipeline, to which a calibration component is added. The seventh line is a Domain Adaptation baseline pipeline, composed of DIOS and a state-of-the-art MSDA approach, M^3SDA. The bottom three lines are our DA pipelines, composed of all the proposed components in this thesis, DIOS, WMSSDA, and S-HOT, with a Dirichlet calibration component added to the last pipeline. Our final and complete applicative pipeline can be found on the last line, obtaining significantly better results than all other compared pipelines.

First, we note that DIOS-NN results are overall better than other baseline pipelines. Indeed, other pipelines rely on a combination of an imputation method and a polishing method from the literature, which has been experimentally shown to be less effective than applying DIOS in chapter 2. By both imputing missing values and correcting erroneous ones, DIOS is able to better improve the data quality compared to a combination of imputation and correction. The same fully-connected Neural Network is used for predictions in all baseline pipelines, with dimensions $\{128, 128, 32, 2\}$ and batch normalization operations in-between each layer, trained for 100 epochs on the target domain data with a learning rate

of 10^{-5}. We also note that the DIOS-NN-CAL pipeline results are largely better than those obtained without calibration. This establishes the fact that the calibration component has a significant positive impact on prediction performance. The experimental results of those two baseline pipelines show the pertinence of using DIOS as an attribute noise preprocessing approach, as well as the positive impact of integrating a post-processing calibration component to the pipeline. Our next pipelines will further improves those results by implementing a Multi-Source Domain Adaptation process, they all rely on DIOS as a pre-processing step, and our complete pipeline also includes a calibration component.

The bottom four lines of table 4.3 are all Multi-Source Supervised Domain Adaptation pipelines, that is, pipelines that include a DA prediction model, which uses source domains data to improve predictions on the target domain, drastically improving predictive performance compared to a standard predictor trained on target data only. Logically, we observe that pipeline DIOS-M^{3}SDA leads to significantly better inference results than all baselines, demonstrating the advantage of exploiting source domains data. When comparing our pipeline DIOS-WMSSDA with DIOS-M^{3}SDA, it is evident that our DIOS-WMSSDA leads to significantly better results than the other, and thus, to significantly better results than all baselines. Indeed, we note an increase of about 1.6% of balanced Accuracy and of more than 5% of F_1-score on average for DIOS-WMSSDA compared to DIOS-M^{3}SDA. This is a clear indication of the pertinence of our proposed WMSSDA approach in our applicative setting, that is able to learn both common and source domain specific information to improve target domain inference, while limiting negative transfer, as showed in chapter 3.

Pipeline DIOS-WMSSDA-SHOT builds on pipeline DIOS-WMSSDA by training the WMSSDA module using our proposed S-HOT framework, which has the advantage of improving NN generalization capacity by taking account of corrections uncertainty during training. Our application results show the pertinence of using the S-HOT framework in such a context, as DIOS-WMSSDA-SHOT results are significantly better than those of pipeline DIOS-WMSSDA on average.

Finally, the last line in table 4.3 shows the results of our complete application pipeline DIOS-WMSSDA-SHOT-CAL. It is clearly apparent that this complete pipeline leads to better results than all other compared pipelines, with pipeline DIOS-WMSSDA-SHOT being a close second. This demonstrates that Dirichlet calibration helps improving prediction results in addition to making probabilities more relevant for medical experts, next section will focus on examining the positive effects of calibration on output probabilities.

In our complete pipeline, output probabilities are calibrated with a Dirichlet calibration after each experimental run. In practice, it is possible to even further improve predictive results by applying calibration on the average predictions over several runs of the pipeline, essentially transforming the pipeline into an ensemble approach. Here, by applying Dirichlet calibration on the average predictions probabilities over the 10 performed runs, we obtain the following overall average metrics values, balanced Accuracy: 84.62%, AUC: 0.9094, and F_1-score: 85.10%. Those results are better than all obtained results in table 4.3, showing that an ensemble approach maximizes results in practice. Obviously, training and applying the entire pipeline several times is not always doable in practice, which is the main limit of the ensemble paradigm. Nevertheless, if the necessary resources and time are available, an

ensemble approach should be used to obtain the best possible predictions quality.

4.4.3 Examining the Impact of Calibration

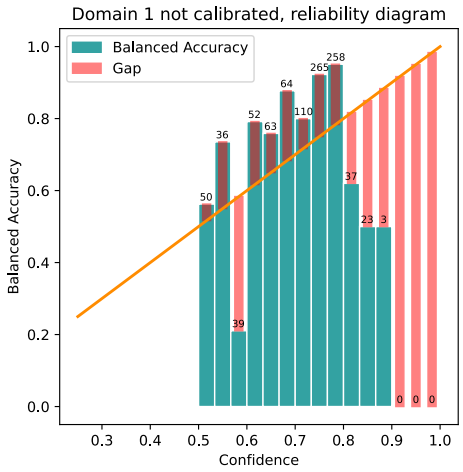
In this section, we are interested in examining the impact of calibration on the output probabilities. Calibration is performed with a Dirichlet calibration, such as described in (Kull et al., 2019), which aims at aligning the confidence of the model with its accuracy. That is, a well calibrated model confidence should be indicative of its accuracy. For a set of patients, if the model predicts that they will survive with a confidence of 75% for all of them, the real outcome should be the survival of about 75% of the patients in the set. A well-calibrated model, with aligned confidence and accuracy, is said to be reliable.

To better visualize the calibration of a model, it is possible to plot a reliability diagram, as it is common practice in the literature (Guo et al., 2017; Kull et al., 2019). A reliability diagram plots the accuracy rate of the model on the test set as a function of the model confidence. On such a diagram, a perfectly calibrated model should plot the identity function, where the accuracy in a sample is always equal to the correspondingly outputted confidence of the model. Any deviation from the identity function is a sign of miscalibration.

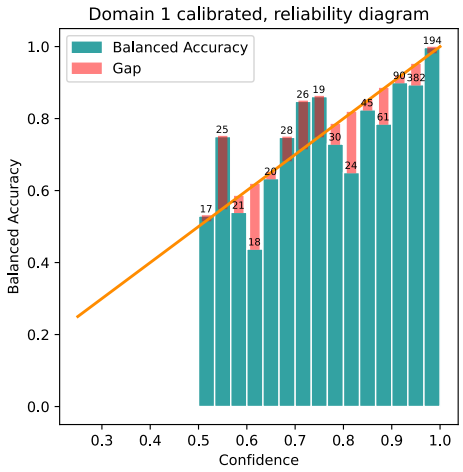
In figure 4.3, we plot the reliability diagrams of the uncalibrated and calibrated outputs of pipeline DIOS-WMSSDA-SHOT, for target domains 1 and 5. In order to obtain such plot, the outputs of the pipeline are grouped into interval bins based on their confidence level. The balanced Accuracy value is computed for samples of each bin and plotted as a bar in the corresponding confidence bin. A perfectly calibrated model should obtain bars that end exactly on the orange diagonal line, which is the identity function. The gap between the diagonal and the actual model reliability bars is signified in red. As we are in a binary classification context, the confidence of the model cannot be under 0.5, as in such a context, a confidence of 50% means that the model output is equivalent to a random output. We display the number of elements in each bin above the corresponding bar, to show that the calibration process improves the confidence of the pipeline while improving reliability of probability outputs.

It appears clearly that the DIOS-WMSSDA-SHOT pipeline is initially not well calibrated, we note that there are no predictions with a confidence above 90%, and that there is an important gap between the confidence of the model and the real balanced Accuracy rate. As can be seen in sub-figures 4.3a and 4.3c, most of the predictions are made with a confidence in the range $70-80\%$, but real balanced Accuracy results are closer to 90%. This shows that the confidence of the model should be higher for those predictions. On sub-figures 4.3b and 4.3d, after Dirichlet calibration, we observe far better calibrated results, with outputs ranging from 50% to 100% of confidence and a better aligned balanced Accuracy rate. We note that most predictions are now made with a high confidence, with most predictions at a confidence of 90% or above, with a correspondingly matching high balanced Accuracy. The gap between predictions and the identity function have been greatly reduced.

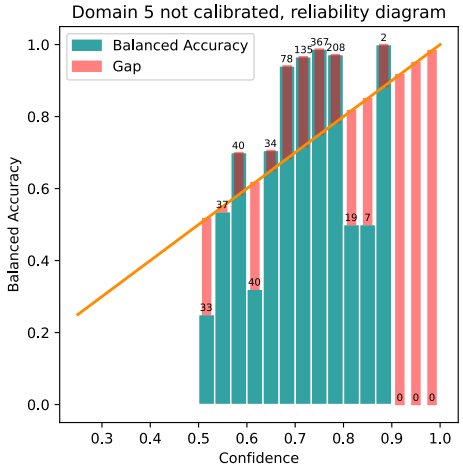
Applying Dirichlet calibration to our pipeline output greatly improved the reliability and pertinence of the results by aligning the model confidence with its expected accuracy. Such an alignment is crucial when deploying a ML model in a medical context, where model predictions hold a significant weight in the decision-making process of medical experts, as it



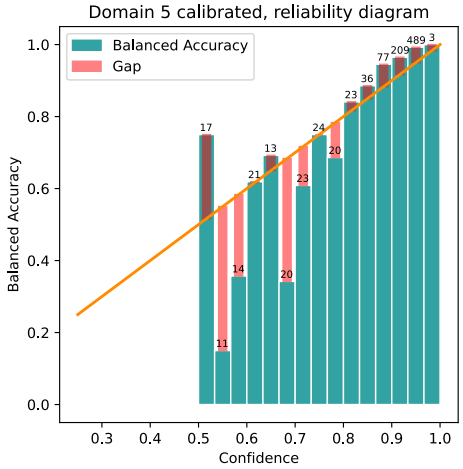
(a) Target domain 1, not calibrated.



(b) Target domain 1, calibrated.



(c) Target domain 5, not calibrated.



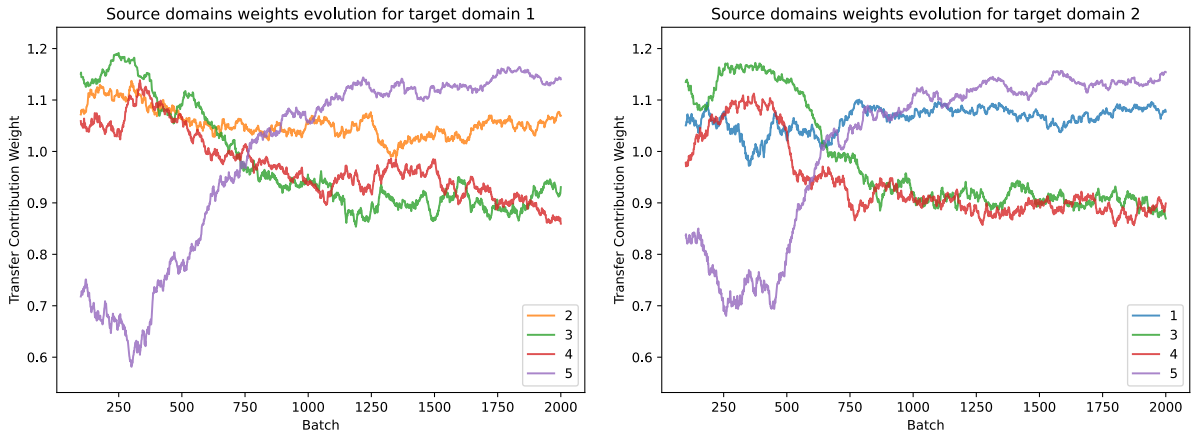
(d) Target domain 5, calibrated.

Figure 4.3: Reliability diagrams of the uncalibrated and calibrated outputs of pipeline DIOS-WMSSDA-SHOT ,for target domains 1 and 5, on the Covid dataset.

directly influences the level of trust and confidence that medical experts can place in predictions.

4.4.4 Source Domains Transfer Contribution Weights

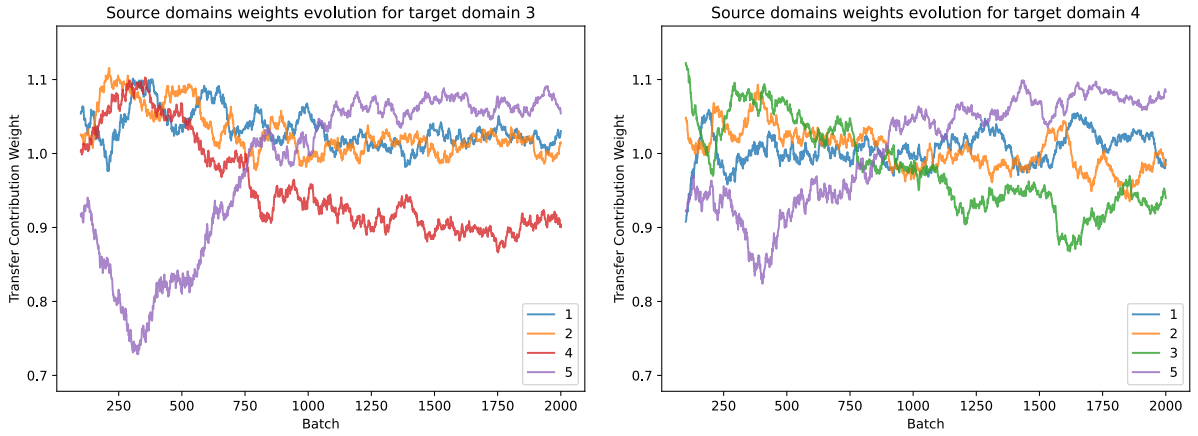
Our proposed WMSSDA approach computes source domains transfer contribution weights during its training phase, those weights are dynamically applied as a scaling of the impact of each source domain on the training of the model. The goal of computing and applying such weights is to reduce Negative Transfer as much as possible, that is, to prevent transferring detrimental information from a source domain to the target domain. In this section, we are interested in visualizing the evolution of those weights during the learning phase of WMSSDA on the Covid dataset.



(a) Contribution weights for target domain 1. (b) Contribution weights for target domain 2.

Figure 4.4: Evolution of source domains contribution weights during training for target domains 1 and 2.

Figure 4.4 shows the evolution of source domains contribution weights during the training phase of the DIOS-WMSSDA-SHOT pipeline for target domains 1 and 2. We observe a very similar pattern in source transfer contribution weights for those two target domains, sub-figures 4.4a and 4.4b respectively. Indeed, when domain 1 is the target, source domain 2 has a stable weight of about 1 for the whole training phase, and inversely. In both cases, weights of other source domains follow an almost identical pattern, with source domains 3 and 4 contributing equally, with a weight decreasing throughout the whole training, while source domain 5 seems to be more important in transfer contribution, with a weight drastically increasing during training in both cases. We can conclude that domain 5 has a high importance in transferring useful information towards domain 1 and 2, whereas domains 3 and 4 contain less interesting knowledge for adaptation towards domains 1 and 2.



(a) Contribution weights for target domain 3. (b) Contribution weights for target domain 4.

Figure 4.5: Evolution of source domains contribution weights during training for target domains 3 and 4.

Figure 4.5 shows the evolution of source domains contribution weights during training for target domains 3 and 4. In both sub-figures, the weights of source domains 1 and 2 remain stable around 1, meaning that both those source domains contribute equally to the training phase for target domains 3 and 4. We note that in both cases, the weight of source domain 5 increases largely during training, reaching the highest transfer contribution across source domains. This is interesting, because despite a huge covariate shift between domains 3 and 4 and domain 5 (covariate distance 19.7 and 24.95 in table 4.1), it seems to be the source domain with the most relevant information for transfer towards target domains 3 and 4. This is an indication that it is not possible to judge the importance of using a source domain simply based on the observed discrepancy between two domains, as two dissimilar domains can contain crucial information for transfer towards the other. We think that this is an issue in the ABMSDA method (Zuo et al., 2021), where source domains weights are computed based on the output of a domain classifier used as a discrepancy measure, with source domains that are most similar to the target domain being attributed higher weights in an attention mechanism. Thus, those weights are computed based on a discrepancy measure effectuated before training, attributing low weights to dissimilar domains, and preventing important information from dissimilar domains of being transferred to the target domain. We argue that our dynamic mechanism that evolves during training allows for more pertinent information transfer between domains while preventing Negative Transfer.

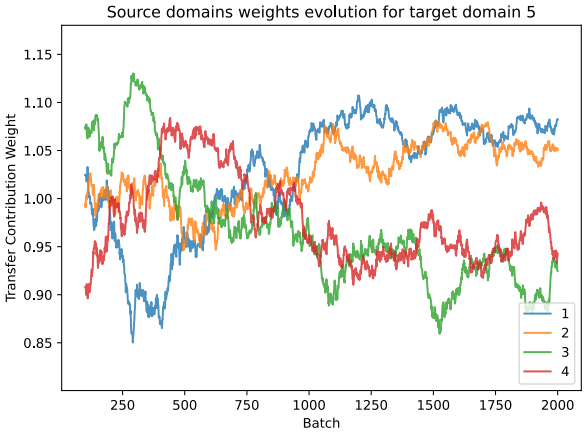


Figure 4.6: Evolution of source domains contribution weights during training for target domain 5.

Finally, figure 4.6 shows the evolution of transfer contribution weights for target domain 5. We observe high transfer contribution weights for source domains 1 and 2 towards target domain 5, whereas weights of source domains 3 and 4 are decreasing during training. This correlates with covariate distances of the four source domains with target domain 5 in table 4.1, where we note a similar and moderate covariate shift between source domains 1 and 2 and target domain 5 (distance 9.4 and 7.7), while domains 3 and 4 are drastically different from domain 5 (distance 19.7 and 24.95). The weights evolution shows that source domains 1 and 2 provide more valuable information to transfer towards domain 5 than the other two source domains. In addition to providing an efficient mechanism to prevent Negative Trans-

fer, our system of dynamic transfer contribution weights allows for a simple visualization of the contribution of each source domain during the training of the model, providing valuable insights on their respective importance.

4.5 Deploying the Pipeline

The goal of the QUALITOP project is to create a European immunotherapy-specific open smart digital platform. Patients and medical experts will consult this platform to obtain various recommendations based on the outputs of several ML models, including the complete predictive pipeline designed in this chapter. In this chapter we have presented the design, training, and experimental application of our complete predictive pipeline. In order to deploy such a pipeline into the QUALITOP open smart digital platform, there are further steps to follow.

A common and effective way of integrating a ML model into an application is to deploy it as an Application Programming Interface (API) endpoint hosted on a server. An API serves as a simple interface that enables smooth communication between two separate applications, through predefined and standardized guidelines. This allows two programs to communicate with one another without the need to understand how the other program works. An API endpoint is a specific address or location within the API that provides a specific function. When an HTTP request is sent to an API endpoint, the associated function is executed, and the response is sent back to the requesting application. In the context of deploying a ML model, the model prediction endpoint receives an HTTP request containing the features of one or several patient(s). The pipeline is fed the patient(s) features on the server, generating output probabilities. These probabilities are then sent back to the application that initiated the request. This approach presents a straightforward and standardized way of interacting with the ML model. It makes easy the integration of the model across one or several front-end application(s), irrespective of their specific functionalities or technologies. By deploying the model as an API endpoint, any developer can seamlessly integrate the model predictions into their application(s), without needing any understanding of its intricate workings.

There are two main approaches of deploying a ML model in a real-world application, namely, online and batch processing. Each approach has its own advantages and considerations, depending on the nature of the data, and the predictive task and usage of the model.

- Online Deployment. In online deployment, the ML model interactively processes data when it arrives, making predictions in real-time. Data is processed as it is received, without waiting for a full batch. The model interactively generates predictions for all incoming data points, providing immediate results. Online deployment is best suited for applications requiring immediate and up-to-date responses. A disadvantage of this approach is that processing each data point individually can be resource intensive, as the model is executed each time a user requests a prediction.

- Batch Deployment. In batch deployment, the ML model processes data in batches. Data is collected and stored until enough has been collected. The newly collected

data batch is fed through the model, and output probabilities are stored on the server. When an end user requests access to a prediction, the stored probabilities are sent back. This makes it very fast to obtain predictions, as the model is regularly executed in background, last predictions of the model can be sent back instantly to the user without requiring an intensive execution. The main disadvantage of batch deployment is that predictions for newly added patients are only available once a full batch has been collected, which can lead to an important delay depending on data arrival.

In the applicative context of the QUALITOP platform, the most pertinent choice is to choose an online deployment for our pipeline. Indeed, this ensures to maximize interaction with the user, as it is then possible to instantly obtain real-time predictions for a new patient. A batch deployment would be less resource intensive, but predictions for new patients could only be obtained once a full batch of new patients is collected, the resulting delay would render the usage of the model predictions impossible.

After deploying a ML model, it is absolutely essential to maintain vigilant performance monitoring throughout its lifecycle. This observation is vital for identifying as soon as possible any potential issue that may arise, and that might necessitate an update of the model. After being deployed to production, it is quite common for the performance of a ML model to degrade over time. This is usually due to an increasing discrepancy between the data that has been used to train the model and the real-world data that the model is continuously being fed in production. This increasing discrepancy through time is known as concept drift, which is exactly the same thing as the concept shift defined in chapter 1, except that it appears and amplifies through time. Meticulously tracking aspects such as model performance, data quality, API traffic, and more, can help determine when the deployed ML model should be updated. There are many possible strategies that can be used to determine when the deployed model should be updated. A performance threshold can be defined, the model should be updated if its monitored performance drops below the defined threshold. An automatic concept drift detection between the data used for training and the data observed in the application can be defined, once concept drift is detected, the model should be updated. A threshold in the accumulation of newly collected data can be defined, if the amount of collected data goes above the threshold, the model must be updated. Even more simply, regular updates can be scheduled. Similarly, diverse policies can be used to determine how the model should be updated. The model can be fully retrained on the concatenation of the old and newly collected data, which is simple but slow and resource intensive. A less expensive approach is to fine-tune the previous version of the model on the newly collected data, which drastically reduces the required training time. This second policy is usually less expensive, faster, and leads to an updated model that should perform well again on real-world data.

In this section we have explored ways of deploying our complete and best-performing DIOS-WMSSDA-SHOT-CAL pipeline in the context of the QUALITOP platform. The most simple and pertinent way of deploying our pipeline is as an API endpoint, hosted on a server. When a user of the QUALITOP platform requests a prediction, the model deployed in an online manner receives the patient data and sends back its probabilities output in real-time. During the life-cycle of the deployed pipeline, automatic monitoring helps detecting perfor-

mance drop and/or concept drift, triggering the retraining of the pipeline. The most pertinent way of retraining our pipeline is through fine-tuning, when the previous version of the model is used as a pre-trained model, and the newly collected data is used to fine-tune the model, optimizing the required time and resources. These policies ensure that the deployed production model consistently delivers optimal and pertinent output probabilities, successfully assisting medical experts in making informed decisions.

4.6 Discussion and Conclusion

In this application chapter, we demonstrated the practical application of the entire work proposed and exposed in this thesis. The central focus is the design and application of a Machine Learning pipeline, unifying all our proposed components. Its efficacy is showcased by its application on a real-world medical Covid dataset. Data sharing issues are preventing partner hospital of the QUALITOP project of freely sharing their data within the project consortium. Consequently, the decision was made to demonstrate the application of our work on a real-world Covid dataset. This dataset shares an extremely similar setting to the QUALITOP applicative setting, allowing us to demonstrate the pertinence of our work for survival outcome prediction in a multi-source supervised medical setting.

Our application to the Covid dataset leads to impressively good prediction results, showing the pertinence of unifying the various proposed approaches of this thesis within a unified pipeline. Our attribute noise correction method, DIOS, is used as the first component of the pipeline, providing a robust foundation for following components by multiply correcting the data. Applying the S-HOT framework for improving NN training while taking account of attribute noise correction uncertainty significantly improves inference results. This successfully extends this proposed imputation framework as a more global and generic correction framework. The application of our pipeline shows the pertinence of using a MSSDA approach in our supervised multi-source context, where important information lies within labeled source domains and should be exploited to improve learning performance on the target domain, leading to drastically better inference results. The last component of our applicative pipeline is the Dirichlet calibration, which is used to significantly improve the reliability and relevance of the pipeline output probabilities, aligning the confidence evaluation with the expected accuracy. This calibration step has shown to drastically improve prediction results, while increasing the reliability of results for medical experts. In the last section, we explained how the designed pipeline is to be integrated within the QUALITOP immunotherapy-specific open smart digital platform, where patients and medical experts will benefit from its predictive output. We define when and how the pipeline should be updated, in order to maximize results quality at all times.

We have shown, in this chapter, that the unification of our work leads to an advanced Machine Learning predictive pipeline, that is applicable and leads to very high quality results in a real-world medical survival outcome prediction context. The application of our pipeline to the Covid dataset showed the pertinence of the proposed approaches in this thesis for our application context. Once access to the data of all partner hospital within the QUALITOP consortium will be granted, the transfer from our pipeline application on the Covid dataset

to the application on QUALITOP data will be straightforward, as the applicative context is almost identical. Integration of the pipeline results has already been demonstrated by deploying our Covid prediction pipeline as an API endpoint, binding the predictive capabilities of our pipeline to the QUALITOP web platform that is currently being developed . With the objective of maintaining maximum predictive efficacy, the guidelines defined in section 4.5 will be followed to monitor and update the pipeline when required. By adhering to these guidelines, we ensure the consistent delivery of output predictions that are not only optimal in quality, but also reliable and pertinent throughout the entire lifespan of the QUALITOP open smart digital platform.

Conclusion and Perspectives

In this thesis, we were interested in several research issues, that were all focused on improving survival outcome and Immune-Related Adverse Events (irAEs) predictions for cancer patients treated with immunotherapy. We tackled different challenges, some we approached from a research point of view, by proposing innovative solutions, and others we tackled from an engineering perspective, by applying the best possible existing solutions from the literature for our specific needs. Our work led to the proposal of innovative approaches, insights, and frameworks, across various fields within the realm of Machine Learning.

Contributions

Several scientific contributions have been made to the field of Machine Learning during this thesis.

Our first main contribution is the design of a new and innovative attribute noise correction method, data Denoising and Imputation in One Step (DIOS), a method based on Auto-Encoders (AEs) and inspired from the recent image restoration technique called "Deep Image Prior" (Ulyanov et al., 2020). This simple, yet effective, method is able to both impute missing values and correct erroneous ones simultaneously, that is, it is able to correct attribute noise in one preprocessing step. The intuition behind DIOS is to train a model to reconstruct the corrupted data from randomly initialized noise. The model's training process involves learning abstract features and patterns from the data while disregarding any attribute noise in the original data. However, as the training continues, the model may become excessively focused on fitting the data and its attribute noise, leading to overfitting. To avoid this, stopping the training before overfitting occurs ensures that the model maintains a generalized understanding of the data. Consequently, the model generates credible missing values and correct erroneous ones in the dataset. DIOS does not require to learn from clean data to correct the corrupted dataset, it learns directly from the whole corrupted version of the dataset. It is also able to impute missing values without requiring any complete instance in the dataset. To the best of our knowledge, DIOS is the first method that can truly handle attribute noise in mixed-type tabular data as a preprocessing step in the Machine Learning literature.

Our second main contribution is the proposal of two frameworks that can be used to account for imputation uncertainty during Neural Network training, leading to better predic-

tions, Single-Hotpatching (S-HOT) and Multiple-Hotpatching (M-HOT). Those two frameworks take the between-imputation uncertainty into account to improve the training process of Neural Networks (NNs), leading to an improvement in their generalization capacity. S-HOT and M-HOT are based on the computation of the between-imputation uncertainty, which corresponds to the standard deviation between imputed values of multiple imputed datasets. This uncertainty is then used as a scale to add stochasticity to the imputation of missing values directly on batch extraction during training. It leads to a kind of noise regularization that takes into account the imputation uncertainty, improving generalization capacity, and thus, prediction results on unseen data. Those frameworks are to be used in different situations, S-HOT is adapted to train a unique and large NN, while M-HOT can be used to train multiple learners in an ensemble way and reach extremely good prediction results at the expense of a higher computational cost. Our work on the matter of improving NN training by taking account of imputation uncertainty is a first step towards finding better ways to deal with missing values imputation for training predictive models in Machine Learning. We hope that it spikes the interest of other researchers throughout the world on this important, and often overlooked matter, in the Machine Learning literature.

Another major contribution of this work is the design of an innovative Multi-Source Supervised Domain Adaptation approach, Weighted Multi-Source Supervised Domain Adaptation (WMSSDA). WMSSDA is a predictor that exploits valuable information from multiple source domains in order to improve learning performance on a related target domain. Our proposed approach is composed of a two branch architecture, learning both a shared domain invariant latent space and source domain specific latent spaces. The shared latent representation is learned and regularized both statistically and adversarially, the statistical regularization relies on a Moment Distance (MD) measure between source and target domains. The output of the MD regularization is used to compute transfer contribution weights that are applied to weight the impact of each source domain during training, limiting Negative Transfer. We propose a new component for limiting Negative Transfer through the computation of transfer contribution weights that are applied as a scaling of each source domain in its impact in the training of the entire model. Our ablation study highlights the importance of this new element for the Domain Adaptation (DA) literature.

Finally, our last contribution is a practical one, it is the design and application of a Machine Learning pipeline, unifying all our previously proposed components. Its efficacy is showcased by its application on a real-world medical Covid dataset, highlighting the pertinence of our work for survival outcome prediction in a multi-source supervised medical setting. Our attribute noise correction method, DIOS, is used as the first component of the pipeline, providing a robust foundation for following components by multiply correcting the data. Applying the S-HOT framework for improving NN training while taking account of attribute noise correction uncertainty significantly improves inference results. The predictive model used in the pipeline is our proposed WMSSDA approach, that exploits important information from labeled source domains to improve learning performance on the target domain, leading to drastically better inference results. The last component of our applicative pipeline is the Dirichlet calibration, which is used to significantly improve the reliability and relevance of the pipeline output probabilities, aligning the confidence evaluation with the

expected accuracy. This calibration step has shown to drastically improve prediction results, while increasing the reliability of results for medical experts. With this last contribution, we show that the unification of our work leads to an advanced Machine Learning predictive pipeline, that is applicable and leads to very high quality results in an application context almost identical to QUALITOP.

Perspectives

There are many possibilities to explore that could improve and build upon the work presented in this thesis.

Our method DIOS relies on Denoising Auto-Encoders, a crucial step of our approach is to carefully design the best possible architecture for the dataset to correct, and to find hyper-parameters leading to the best results. This is a highly limiting factor for anyone that would be interested in applying our approach to their project. An interesting perspective would be to define an algorithm able to automatically find a well-suited architecture, depending on the particularities of the dataset to correct. Indeed, the best performing architectures for each datasets were very similar in our experiments, we think that a kind of generic scaling could be found and applied with good results with a bit more research. This would highly improve the usability of DIOS, and facilitate the application of the method on any tabular dataset.

Another interesting improvement would be to pair DIOS with a method able to detect which values are erroneous and should be polished. Then, we would simply mask them as missing, which would certainly reduce the overall attribute noise in the corrupted dataset before the learning phase of DIOS, which would yield better results.

In our frameworks, S-HOT and M-HOT, we assume that the imputation uncertainty follows a Gaussian distribution. We designed our frameworks with this assumption in mind and found good empirical results. However, assuming a normal distribution might work in most cases, but it might not be pertinent in some others, depending on the missing feature nature or used imputation method. An interesting perspective could be to try to automatically adapt the distribution kind for each missing value, given the proposed multiple imputations. An obvious complication to finding a distribution through regression based on imputed values is the very limited sample size, corresponding to the number of imputations performed. Therefore, we believe that a theoretical work should be performed in order to determine distributions of interest, from which the most adequate distribution could be chosen based on the observed distribution of imputation values.

We designed S-HOT and M-HOT with the goal to improve Neural Network training when dealing with missing values in tabular data. Future works could focus on adapting those frameworks to other data natures, such as sequential data or image data, where important components, such as the time dimension and pixel neighborhood, should be taken under account in the design of the approach.

With our approach WMSSDA, we proposed a new component to reduce Negative Transfer during the training phase of the model. This component aims to dynamically scale the impact of each source domain during the training of the complete method, to maximize the

transfer contribution of important source domains, while minimizing detrimental transfer towards the target domain. Despite this new component, and better performance than concurring approaches, our experiments have shown that our method is not able to fully avoid Negative Transfer. Negative Transfer is still an open problem in the Domain Adaptation field, future works should focus on finding better ways to handle this important matter.

In our last chapter, we have showcased the application of an advanced predictive pipeline that unifies our work on a real-world medical dataset. The application of this pipeline led to very high quality prediction results and showed the pertinence of the proposed approaches in this thesis for our application context. As we were not able to get access to the complete QUALITOP data in the time-frame of this thesis, we choose to apply our work on a real-world application setting very close to the QUALITOP setting. An obvious perspective is the application of this same pipeline on QUALITOP data once access will be granted. As the applicative context is almost identical, the application and integration of our pipeline within the QUALITOP platform will be straightforward.

Bibliography

Baligh Al-Helali, Qi Chen, Bing Xue, and Mengjie Zhang. A new imputation method based on genetic programming and weighted KNN for symbolic regression with incomplete data. *Soft Computing*, 25(8):5993–6012, April 2021. ISSN 1432-7643, 1433-7479. doi: 10.1007/s00500-021-05590-y. URL http://link.springer.com/10.1007/s00500-021-05590-y.

Rebecca R. Andridge and Roderick J. A. Little. A Review of Hot Deck Imputation for Survey Non-response. *International statistical review = Revue internationale de statistique*, 78(1): 40–64, April 2010. ISSN 0306-7734. doi: 10.1111/j.1751-5823.2010.00103.x. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3130338/.

Emily Grace Armitage, Joanna Godzien, Vanesa Alonso-Herranz, Ángeles López-Gonzálvez, and Coral Barbas. Missing value imputation strategies for metabolomics data. *Electrophoresis*, 36(24):3050–3060, December 2015. ISSN 1522-2683. doi: 10.1002/elps.201500352.

Saqib Ejaz Awan, Mohammed Bennamoun, Ferdous Sohel, Frank Sanfilippo, and Girish Dwivedi. Imputation of missing data with class imbalance using conditional generative adversarial networks. *Neurocomputing*, 453:164–171, September 2021. ISSN 0925-2312. doi: 10.1016/j.neucom.2021.04.010. URL https://www.sciencedirect.com/science/article/pii/S0925231221005282.

Dzmitry Bahdanau, Kyunghyun Cho, and Y. Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *ArXiv*, 1409, September 2014.

Kirk Baker. Singular Value Decomposition Tutorial. 2005.

Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, April 2021. URL http://arxiv.org/abs/2003.05991. arXiv:2003.05991 [cs, stat].

Amanda N. Baraldi and Craig K. Enders. An introduction to modern missing data analyses. *Journal of School Psychology*, 48(1):5–37, February 2010. ISSN 00224405. doi: 10.1016/j.jsp.2009.10.001. URL https://linkinghub.elsevier.com/retrieve/pii/S0022440509000661.

Brett K. Beaulieu-Jones, Jason H. Moore, and THE POOLED RESOURCE OPEN-ACCESS ALS CLINICAL TRIALS CONSORTIUM. MISSING DATA IMPUTATION IN THE ELECTRONIC HEALTH RECORD USING DEEPLY LEARNED AUTOENCODERS. In *Biocomputing 2017*, pages 207–218, Kohala Coast, Hawaii, USA, January 2017. WORLD SCIENTIFIC. ISBN 978-981-320-780-6 978-981-320-781-3. doi: 10.1142/9789813207813_0021. URL http://www.worldscientific.com/doi/abs/10.1142/9789813207813_0021.

Rianne van den Berg, Thomas N. Kipf, and Max Welling. Graph Convolutional Matrix Completion. In *KDD18*, 2018.

Patricia Berglund and Steven Heeringa. *Multiple Imputation of Missing Data Using SAS®*. SAS Institute, 1 edition, 2014. ISBN 978-1-62959-203-9.

Chris M. Bishop. Training with Noise is Equivalent to Tikhonov Regularization. *Neural Computation*, 7(1):108–116, January 1995. ISSN 0899-7667. doi: 10.1162/neco.1995.7.1.108. Conference Name: Neural Computation.

Christopher M. Bishop. *Pattern recognition and machine learning*. Information science and statistics. Springer, New York, 2006. ISBN 978-0-387-31073-2.

Karsten M. Borgwardt, Arthur Gretton, Malte J. Rasch, Hans-Peter Kriegel, Bernhard Schölkopf, and Alex J. Smola. Integrating structured biological data by Kernel Maximum Mean Discrepancy. *Bioinformatics*, 22(14):e49–e57, July 2006. ISSN 1367-4803. doi: 10.1093/bioinformatics/btl242. URL https://doi.org/10.1093/bioinformatics/btl242.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 978-0-89791-497-0. doi: 10.1145/130385.130401. URL https://dl.acm.org/doi/10.1145/130385.130401.

Leo Breiman. *Classification and Regression Trees*. Routledge, New York, October 2017. ISBN 978-1-315-13947-0. doi: 10.1201/9781315139470.

Carla E. Brodley and Mark A. Friedl. Identifying Mislabeled Training Data. *Journal of Artificial Intelligence Research*, 11(1):131–167, 1999. ISSN 1076-9757. doi: 10.1613/jair.606.

L. F. Burgette and J. P. Reiter. Multiple Imputation for Missing Data via Sequential Regression Trees. *American Journal of Epidemiology*, 172(9):1070–1076, November 2010. ISSN 0002-9262, 1476-6256. doi: 10.1093/aje/kwq260. URL https://academic.oup.com/aje/article-lookup/doi/10.1093/aje/kwq260.

Stef van Buuren. *Flexible Imputation of Missing Data, Second Edition*. Taylor & Francis Limited, September 2021. ISBN 978-1-03-217863-9. Google-Books-ID: LpOWzgEACAAJ.

Stef van Buuren and Karin Groothuis-Oudshoorn. mice: Multivariate Imputation by Chained Equations in R. *Journal of Statistical Software*, 45(3):1–67, December 2011. ISSN 1548-7660. doi: 10.18637/jss.v045.i03. URL https://doi.org/10.18637/jss.v045.i03.

Trond Hellem Bø, Bjarte Dysvik, and Inge Jonassen. LSimpute: accurate estimation of missing values in microarray data with least squares methods. *Nucleic Acids Research*, 32(3): e34, February 2004. ISSN 1362-4962. doi: 10.1093/nar/gnh026.

N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002. ISSN 1076-9757. doi: 10.1613/jair.953.

Chao Chen, Zhihong Chen, Boyuan Jiang, and Xinyu Jin. Joint Domain Alignment and Discriminative Feature Learning for Unsupervised Deep Domain Adaptation, November 2018. URL http://arxiv.org/abs/1808.09347. arXiv:1808.09347 [cs, stat].

Wendong Chen and Haifeng Hu. Generative attention adversarial classification network for unsupervised domain adaptation. *Pattern Recognition*, 107:107440, November 2020. ISSN 0031-3203. doi: 10.1016/j.patcog.2020.107440. URL https://www.sciencedirect.com/science/article/pii/S0031320320302430.

Linda M. Collins, Joseph L. Schafer, and Chi-Ming Kam. A comparison of inclusive and restrictive strategies in modern missing data procedures. *Psychological Methods*, 6(4): 330–351, 2001. ISSN 1939-1463, 1082-989X. doi: 10.1037/1082-989X.6.4.330. URL http://doi.apa.org/getdoi.cfm?doi=10.1037/1082-989X.6.4.330.

Corinna Cortes, Mehryar Mohri, and Andrés Muñoz Medina. Adaptation Based on Generalized Discrepancy. *Journal of Machine Learning Research*, 20(1):1–30, 2019. doi: 10.5555/3322706.3322707.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, December 1989. ISSN 1435-568X. doi: 10.1007/BF02551274. URL https://doi.org/10.1007/BF02551274.

Paul D.Allison. *Missing Data*. SAGE Publications, Inc., 2002. ISBN 978-1-4129-8507-9. doi: 10.4135/9781412985079. URL https://methods.sagepub.com/book/missing-data.

Oscar Day and Taghi M. Khoshgoftaar. A survey on heterogeneous transfer learning. *Journal of Big Data*, 4(1):29, December 2017. doi: 10.1186/s40537-017-0089-0.

Edith D de Leeuw, Joop Hox, and Mark Huisman. Prevention and Treatment of Item Nonresponse. *Journal of Official Statistics*, 19(2):153–176, January 2003.

Marcilio CP de Souto, Pablo A. Jaskowiak, and Ivan G. Costa. Impact of missing data imputation methods on gene expression clustering and classification. *BMC Bioinformatics*, 16(1):64, February 2015. ISSN 1471-2105. doi: 10.1186/s12859-015-0494-3. URL https://doi.org/10.1186/s12859-015-0494-3.

Olivier Delalleau, Aaron Courville, and Yoshua Bengio. Efficient EM Training of Gaussian Mixtures with Missing Data, January 2018. URL http://arxiv.org/abs/1209.0521. arXiv:1209.0521 [cs, stat].

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1): 1–38, 1977. ISSN 0035-9246. URL https://www.jstor.org/stable/2984875. Publisher: [Royal Statistical Society, Wiley].

Janez Demsar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7:30, 2006. doi: 10.5555/1248547.1248548.

D. C Dowson and B. V Landau. The Fréchet distance between multivariate normal distributions. *Journal of Multivariate Analysis*, 12(3):450–455, September 1982. ISSN 0047-259X. doi: 10.1016/0047-259X(82)90077-X. URL https://www.sciencedirect.com/science/article/pii/0047259X8290077X.

Wenjie Du, David Côté, and Yan Liu. SAITS: Self-attention-based imputation for time series. *Expert Systems with Applications*, 219:119619, June 2023. ISSN 0957-4174. doi: 10.1016/j.eswa.2023.119619. URL https://www.sciencedirect.com/science/article/pii/S0957417423001203.

John Duchi and Elad Hazan. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159, July 2011.

Tlamelo Emmanuel, Thabiso Maupong, Dimane Mpoeleng, Thabo Semong, Banyatsang Mphago, and Oteng Tabona. A survey on missing data in machine learning. *Journal of Big Data*, 8(1):140, October 2021. ISSN 2196-1115. doi: 10.1186/s40537-021-00516-9. URL https://doi.org/10.1186/s40537-021-00516-9.

Alberto Fernández, Salvador García, Mikel Galar, Ronaldo C. Prati, Bartosz Krawczyk, and Francisco Herrera. *Learning from Imbalanced Data Sets*. Springer International Publishing, Cham, 2018. ISBN 978-3-319-98073-7 978-3-319-98074-4. doi: 10.1007/978-3-319-98074-4. URL http://link.springer.com/10.1007/978-3-319-98074-4.

Juan Carlos Figueroa-García, Roman Neruda, and German Hernandez–Pérez. A genetic algorithm for multivariate missing data imputation. *Information Sciences*, 619:947–967, January 2023. ISSN 0020-0255. doi: 10.1016/j.ins.2022.11.037. URL https://www.sciencedirect.com/science/article/pii/S0020025522013263.

Dragan Gamberger, Nada Lavrac, and Ciril Groselj. Experiments with Noise Filtering in a Medical Domain. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 143–151, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-612-8.

Dragan Gamberger, Nada Lavrac, and Sašo Džeroski. Noise Detection and Elimination in data Proprocessing: Experiments in Medical Domains. *Applied Artificial Intelligence*, 14: 205–223, 2000. doi: 10.1080/088395100117124.

Yaroslav Ganin and Victor Lempitsky. Unsupervised Domain Adaptation by Backpropagation. *Journal of Machine Learning Research*, 17(1):2096–2030, 2016. doi: 10.5555/2946645.2946704.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-Adversarial Training of Neural Networks. In *Domain Adaptation in Computer Vision Applications*, pages 189–209. Springer International Publishing, Cham, 2017. ISBN 978-3-319-58346-4 978-3-319-58347-1. doi: 10.1007/978-3-319-58347-1_10. URL http://link.springer.com/10.1007/978-3-319-58347-1_10. Series Title: Advances in Computer Vision and Pattern Recognition.

Pengfei Ge, Chuan-Xian Ren, Dao-Qing Dai, and Hong Yan. Domain Adaptation and Image Classification via Deep Conditional Adaptation Network, May 2022. URL http://arxiv.org/abs/2006.07776. arXiv:2006.07776 [cs].

Lovedeep Gondara and Ke Wang. MIDA: Multiple Imputation Using Denoising Autoencoders. volume 10939, pages 260–272, Cham, 2018. Springer International Publishing. ISBN 978-3-319-93039-8 978-3-319-93040-4. doi: 10.1007/978-3-319-93040-4_21. URL http://link.springer.com/10.1007/978-3-319-93040-4_21.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL http://www.deeplearningbook.org.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On Calibration of Modern Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1321–1330. PMLR, July 2017. URL https://proceedings.mlr.press/v70/guo17a.html. ISSN: 2640-3498.

Shivani Gupta and Atul Gupta. Dealing with Noise Problem in Machine Learning Datasets: A Systematic Review. *Procedia Computer Science*, 161:466–474, January 2019. ISSN 1877-0509. doi: 10.1016/j.procs.2019.11.146. URL https://www.sciencedirect.com/science/article/pii/S1877050919318575.

Haibo He and E.A. Garcia. Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, September 2009. doi: 10.1109/TKDE.2008.239.

P. Hart. The condensed nearest neighbor rule (Corresp.). *IEEE Transactions on Information Theory*, 14(3):515–516, May 1968. ISSN 0018-9448. doi: 10.1109/TIT.1968.1054155. URL http://ieeexplore.ieee.org/document/1054155/.

Trevor Hastie, Robert Tibshirani, Gavin Sherlock, Michael Eisen, Patrick Brown, and David Botstein. Imputing Missing Data for Gene Expression Arrays. *Technical report, Stanford Statistics Department*, 1, December 2001.

R. Hecht-Nielsen. Kolmogorov"s Mapping Neural Network Existence Theorem. 1987. URL https://www.semanticscholar.org/paper/Kolmogorov%27%27s-Mapping-Neural-Network-Existence-Hecht-Nielsen/257843bde91001dc94fffdba287879fcb3b2f89a.

Geoffrey Hinton. RMSProp, 2012. URL https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. *Neural computation*, 9:1735–80, December 1997. doi: 10.1162/neco.1997.9.8.1735.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, January 1989. ISSN 08936080. doi: 10.1016/0893-6080(89)90020-8. URL https://linkinghub.elsevier.com/retrieve/pii/0893608089900208.

Nicholas J. Horton and Ken P. Kleinman. Much ado about nothing: A comparison of missing data methods and software to fit incomplete data regression models. *The American Statistician*, 61(1):79–90, February 2007. ISSN 0003-1305. doi: 10.1198/000313007X172556.

Nicholas J Horton, Stuart R Lipsitz, and Michael Parzen. A Potential for Bias When Rounding in Multiple Imputation. *The American Statistician*, 57(4):229–232, November 2003. ISSN 0003-1305. doi: 10.1198/0003130032314. URL https://doi.org/10.1198/0003130032314. Publisher: Taylor & Francis _eprint: https://doi.org/10.1198/0003130032314.

Vugar E. Ismailov. A three layer neural network can represent any multivariate function. *Journal of Mathematical Analysis and Applications*, 523(1):127096, July 2023. ISSN 0022-247X. doi: 10.1016/j.jmaa.2023.127096. URL https://www.sciencedirect.com/science/article/pii/S0022247X23000999.

Anil K. Jain and Richard C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., USA, 1988. ISBN 978-0-13-022278-7.

Jireh Jam, Connah Kendrick, Kevin Walker, Vincent Drouard, Jison Gee-Sern Hsu, and Moi Hoon Yap. A comprehensive review of past and present image inpainting methods. *Computer Vision and Image Understanding*, 203:103147, February 2021. ISSN 1077-3142. doi: 10.1016/j.cviu.2020.103147. URL https://www.sciencedirect.com/science/article/pii/S1077314220301661.

Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, April 2016. doi: 10.1098/rsta.2015.0202.

URL https://royalsocietypublishing.org/doi/10.1098/rsta.2015.0202. Publisher: Royal Society.

Julie Josse and François Husson. Handling missing values in exploratory multivariate data analysis methods. *Journal de la Société Française de Statistique*, 153(2):79–99, December 2012. ISSN 2102-6238. URL http://journal-sfds.fr/article/view/122. Number: 2.

Julie Josse, Nicolas Prost, Erwan Scornet, and Gaël Varoquaux. *On the consistency of supervised learning with missing values*. February 2019.

Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80(5):8091–8126, February 2021. ISSN 1573-7721. doi: 10.1007/s11042-020-10139-6. URL https://doi.org/10.1007/s11042-020-10139-6.

Shahidul Islam Khan and Abu Sayed Md Latiful Hoque. SICE: an improved missing data imputation technique. *Journal of Big Data*, 7(1):37, 2020. ISSN 2196-1115. doi: 10.1186/s40537-020-00313-w.

Hyunsoo Kim, Gene H. Golub, and Haesun Park. Missing value estimation for DNA microarray gene expression data: local least squares imputation. *Bioinformatics (Oxford, England)*, 21(2):187–198, January 2005. ISSN 1367-4803. doi: 10.1093/bioinformatics/bth499.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *CoRR*, December 2014. URL https://www.semanticscholar.org/paper/Adam%3A-A-Method-for-Stochastic-Optimization-Kingma-Ba/a6cb366736791bcccc5c8639de5a8f9636bf87e8.

Kevin H. Knuth. Optimal Data-Based Binning for Histograms. *arXiv e-prints*, page physics/0605197, May 2006. doi: 10.48550/arXiv.physics/0605197. URL https://ui.adsabs.harvard.edu/abs/2006physics...5197K. ADS Bibcode: 2006physics...5197K.

Marietta Kokla, Jyrki Virtanen, Marjukka Kolehmainen, Jussi Paananen, and Kati Hanhineva. Random forest-based imputation outperforms other methods for imputing LC-MS metabolomics data: a comparative study. *BMC Bioinformatics*, 20(1):492, October 2019. ISSN 1471-2105. doi: 10.1186/s12859-019-3110-0. URL https://doi.org/10.1186/s12859-019-3110-0.

Wouter M. Kouw and Marco Loog. An introduction to domain adaptation and transfer learning, January 2019. URL http://arxiv.org/abs/1812.11806.

Bartosz Krawczyk. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 5(4):221–232, November 2016. ISSN 2192-6352, 2192-6360. doi: 10.1007/s13748-016-0094-0. URL http://link.springer.com/10.1007/s13748-016-0094-0.

M. Kukar and I. Kononenko. Cost-Sensitive Learning with Neural Networks. 1998. URL https://www.semanticscholar.org/paper/Cost-Sensitive-Learning-with-Neural-Networks-Kukar-Kononenko/bdef7eb9b62e2a12b870957879f7a097b41f6012.

Meelis Kull, Miquel Perello Nieto, Markus Kängsepp, Telmo Silva Filho, Hao Song, and Peter Flach. Beyond temperature scaling: Obtaining well-calibrated multi-class probabilities with Dirichlet calibration. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/hash/8ca01ea920679a0fe3728441494041b9-Abstract.html.

Miguel Lacerda, Cally Ardington, and Murray Leibbrandt. Sequential Regression Multiple Imputation for Incomplete Multivariate Data using Markov Chain Monte Carlo. *Southern Africa Labour and Development Research Unit, University of Cape Town, SALDRU Working Papers*, January 2007.

Xiaochen Lai, Xia Wu, Liyong Zhang, Wei Lu, and Chongquan Zhong. Imputations of missing values using a tracking-removed autoencoder trained with incomplete data. *Neurocomputing*, 366:54–65, November 2019. ISSN 0925-2312. doi: 10.1016/j.neucom.2019.07.066. URL https://www.sciencedirect.com/science/article/pii/S0925231219310732.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/hash/9ef2ed4b7fd2c810847ffa5fa85bce38-Abstract.html.

Marine Le Morvan, Julie Josse, Erwan Scornet, and Gael Varoquaux. What's a good imputation to predict with missing values? In *Advances in Neural Information Processing Systems*, volume 34, pages 11530–11540. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/5fe8fdc79ce292c39c5f209d734b7206-Abstract.html.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1 (4):541–551, December 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.4.541. Conference Name: Neural Computation.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. ISSN 1476-4687. doi: 10.1038/nature14539. URL https://www.nature.com/articles/nature14539. Number: 7553 Publisher: Nature Publishing Group.

Jae-woong Lee and Jongwuk Lee. IDAE: Imputation-boosted Denoising Autoencoder for Collaborative Filtering. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pages 2143–2146, New York, NY, USA, November 2017. Association for Computing Machinery. ISBN 978-1-4503-4918-5. doi: 10.1145/3132847.3133158. URL https://doi.org/10.1145/3132847.3133158.

Pengzhi Li, Yan Pei, and Jianqiang Li. A comprehensive survey on design and application of autoencoder in deep learning. *Applied Soft Computing*, 138:110176, May 2023. ISSN 1568-4946. doi: 10.1016/j.asoc.2023.110176. URL https://www.sciencedirect.com/science/article/pii/S1568494623001941.

Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):6999–7019, December 2022. ISSN 2162-2388. doi: 10.1109/TNNLS.2021.3084827. Conference Name: IEEE Transactions on Neural Networks and Learning Systems.

Zhenpeng Li, Zhen Zhao, Yuhong Guo, Haifeng Shen, and Jieping Ye. Mutual Learning Network for Multi-Source Domain Adaptation, March 2020. URL http://arxiv.org/abs/2003.12944. arXiv:2003.12944 [cs].

Gernot Liebchen, Bheki Twala, Martin Shepperd, Michelle Cartwright, and Mark Stephens. Filtering, Robust Filtering, Polishing: Techniques for Addressing Quality in Software Data. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 99–106, Madrid, Spain, September 2007. IEEE. ISBN 978-0-7695-2886-1. doi: 10.1109/ESEM.2007.70. URL http://ieeexplore.ieee.org/document/4343737/.

Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers. *AI Open*, 3:111–132, January 2022. ISSN 2666-6510. doi: 10.1016/j.aiopen.2022.10.001. URL https://www.sciencedirect.com/science/article/pii/S2666651022000146.

Wei-Chao Lin and Chih-Fong Tsai. Missing value imputation: a review and analysis of the literature (2006–2017). *Artificial Intelligence Review*, 53(2):1487–1509, February 2020. ISSN 0269-2821, 1573-7462. doi: 10.1007/s10462-019-09709-4. URL http://link.springer.com/10.1007/s10462-019-09709-4.

Roderick Little and Donald Rubin. *Statistical Analysis with Missing Data, Third Edition*. Wiley Series in Probability and Statistics. Wiley, 1 edition, April 2019. ISBN 978-0-470-52679-8 978-1-119-48226-0. doi: 10.1002/9781119482260. URL https://onlinelibrary.wiley.com/doi/book/10.1002/9781119482260.

Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I. Jordan. Learning Transferable Features with Deep Adaptation Networks, May 2015. URL http://arxiv.org/abs/1502.02791. arXiv:1502.02791 [cs].

Mingsheng Long, ZHANGJIE CAO, Jianmin Wang, and Michael I Jordan. Conditional Adversarial Domain Adaptation. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://papers.nips.cc/paper/2018/hash/ab88b15733f543179858600245108dd8-Abstract.html.

Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. ISSN 1533-7928. URL http://jmlr.org/papers/v9/vandermaaten08a.html.

Andrian Marcus, Jonathan Maletic, and King-Ip Lin. Ordinal Association Rules for Error Identification in Data Sets. *International Conference on Information and Knowledge Management, Proceedings*, September 2001. doi: 10.1145/502585.502700.

Pierre-Alexandre Mattei and Jes Frellsen. MIWAE: Deep Generative Modelling and Imputation of Incomplete Data Sets. In *Proceedings of the 36th International Conference on Machine Learning*, pages 4413–4423. PMLR, May 2019. URL https://proceedings.mlr.press/v97/mattei19a.html. ISSN: 2640-3498.

Rahul Mazumder, Trevor Hastie, and Robert Tibshirani. Spectral Regularization Algorithms for Learning Large Incomplete Matrices. *Journal of Machine Learning Research*, 11(80):2287–2322, 2010. URL http://jmlr.org/papers/v11/mazumder10a.html.

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, December 1943. ISSN 1522-9602. doi: 10.1007/BF02478259. URL https://doi.org/10.1007/BF02478259.

Xiaoye Miao, Yangyang Wu, Jun Wang, Yunjun Gao, Xudong Mao, and Jianwei Yin. Generative Semi-supervised Learning for Multivariate Time Series Imputation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10):8983–8991, May 2021. ISSN 2374-3468. doi: 10.1609/aaai.v35i10.17086. URL https://ojs.aaai.org/index.php/AAAI/article/view/17086. Number: 10.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill series in computer science. McGraw-Hill, New York, 1997. ISBN 978-0-07-042807-2.

Frank J. Molnar, Brian Hutton, and Dean Fergusson. Does analysis using "last observation carried forward" introduce bias in dementia research? *CMAJ : Canadian Medical Association Journal*, 179(8):751–753, October 2008. ISSN 0820-3946. doi: 10.1503/cmaj.080820. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2553855/.

Jose G. Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V. Chawla, and Francisco Herrera. A unifying view on dataset shift in classification. *Pattern Recognition*, 45(1):521–530, January 2012. ISSN 00313203. doi: 10.1016/j.patcog.2011.06.019. URL https://linkinghub.elsevier.com/retrieve/pii/S0031320311002901.

Saeid Motiian, Marco Piccirilli, Donald A. Adjeroh, and Gianfranco Doretto. Unified Deep Supervised Domain Adaptation and Generalization, September 2017. URL http://arxiv.org/abs/1709.10190. arXiv:1709.10190 [cs].

Carol M. Musil, Camille B. Warner, Piyanee Klainin Yobas, and Susan L. Jones. A Comparison of Imputation Techniques for Handling Missing Data. *Western Journal of Nursing Research*, 24(7):815–829, November 2002. ISSN 0193-9459, 1552-8456. doi:

10.1177/019394502762477004. URL http://journals.sagepub.com/doi/10.1177/019394502762477004.

Boris Muzellec, Julie Josse, Claire Boyer, and Marco Cuturi. Missing Data Imputation using Optimal Transport. In *Proceedings of the 37th International Conference on Machine Learning*, pages 7130–7140. PMLR, November 2020. URL https://proceedings.mlr.press/v119/muzellec20a.html. ISSN: 2640-3498.

Alfredo Nazabal, Pablo M. Olmos, Zoubin Ghahramani, and Isabel Valera. Handling Incomplete Heterogeneous Data using VAEs, May 2020. URL http://arxiv.org/abs/1807.03653. arXiv:1807.03653 [cs, stat].

Sanaz Nikfalazar, Chung-Hsing Yeh, Susan Bedingfield, and Hadi A. Khorshidi. Missing data imputation using decision trees and fuzzy clustering with iterative learning. *Knowledge and Information Systems*, 62(6):2419–2437, June 2020. ISSN 0219-3116. doi: 10.1007/s10115-019-01427-1. URL https://doi.org/10.1007/s10115-019-01427-1.

Xiuli Ning, Yingcheng Xu, Xiaohong Gao, and Ying Li. Missing data of quality inspection imputation algorithm base on stacked denoising auto-encoder. In *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*, pages 84–88, March 2017. doi: 10.1109/ICBDA.2017.8078781.

Shigeyuki Oba, Masa-aki Sato, Ichiro Takemasa, Morito Monden, Ken-ichi Matsubara, and Shin Ishii. A Bayesian missing value estimation method for gene expression profile data. *Bioinformatics*, 19(16):2088–2096, November 2003. ISSN 1367-4803. doi: 10.1093/bioinformatics/btg287. URL https://doi.org/10.1093/bioinformatics/btg287.

Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010. doi: 10.1109/TKDE.2009.191.

Zhongyi Pei, Zhangjie Cao, Mingsheng Long, and Jianmin Wang. Multi-adversarial domain adaptation. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18, pages 3934–3941, New Orleans, Louisiana, USA, 2018. AAAI Press. ISBN 978-1-57735-800-8.

Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. Moment Matching for Multi-Source Domain Adaptation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1406–1415, Seoul, Korea (South), October 2019. IEEE. ISBN 978-1-72814-803-8. doi: 10.1109/ICCV.2019.00149. URL https://ieeexplore.ieee.org/document/9010750/.

Ricardo Cardoso Pereira, Pedro Henriques Abreu, and Pedro Pereira Rodrigues. VAE-BRIDGE: Variational Autoencoder Filter for Bayesian Ridge Imputation of Missing Data. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, July 2020a. doi: 10.1109/IJCNN48605.2020.9206615. ISSN: 2161-4407.

Ricardo Cardoso Pereira, Miriam Seoane Santos, Pedro Pereira Rodrigues, and Pedro Henriques Abreu. Reviewing Autoencoders for Missing Data Imputation: Technical Trends, Applications and Outcomes. *Journal of Artificial Intelligence Research*, 69:1255–1285, December 2020b. ISSN 1076-9757. doi: 10.1613/jair.1.12312. URL https://www.jair.org/index.php/jair/article/view/12312.

Jonathan Pirnay and Keng Chai. Inpainting Transformer for Anomaly Detection. In Stan Sclaroff, Cosimo Distante, Marco Leo, Giovanni M. Farinella, and Federico Tombari, editors, *Image Analysis and Processing – ICIAP 2022*, Lecture Notes in Computer Science, pages 394–406, Cham, 2022. Springer International Publishing. ISBN 978-3-031-06430-2. doi: 10.1007/978-3-031-06430-2_33.

John Platt. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. *Adv. Large Margin Classif.*, 10, June 2000.

B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, January 1964. ISSN 0041-5553. doi: 10.1016/0041-5553(64)90137-5. URL https://www.sciencedirect.com/science/article/pii/0041555364901375.

Zhen Qin, Qingliang Zeng, Yixin Zong, and Fan Xu. Image inpainting based on deep learning: A review. *Displays*, 69:102028, September 2021. ISSN 0141-9382. doi: 10.1016/j.displa.2021.102028. URL https://www.sciencedirect.com/science/article/pii/S0141938221000391.

Yeping Lina Qiu, Hong Zheng, and Olivier Gevaert. Genomic data imputation with variational auto-encoders. *GigaScience*, 9(8):giaa082, August 2020. ISSN 2047-217X. doi: 10.1093/gigascience/giaa082. URL https://academic.oup.com/gigascience/article/doi/10.1093/gigascience/giaa082/5881619.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986. ISSN 0885-6125, 1573-0565. doi: 10.1007/BF00116251. URL http://link.springer.com/10.1007/BF00116251.

Md. Geaur Rahman and Md Zahidul Islam. Missing value imputation using a fuzzy clustering-based EM approach. *Knowledge and Information Systems*, 46(2):389–422, February 2016. ISSN 0219-3116. doi: 10.1007/s10115-015-0822-y. URL https://doi.org/10.1007/s10115-015-0822-y.

Ievgen Redko, Emilie Morvant, Amaury Habrard, Marc Sebban, and Younès Bennani. *Advances in domain adaptation theory*. Computer engineering. ISTE Press Ltd ; Elsevier Ltd, London, UK : Kidlington, Oxford, UK, 2019. ISBN 978-1-78548-236-6. URL https://www.elsevier.com/books/advances-in-domain-adaptation-theory/redko/978-1-78548-236-6. OCLC: ocn988168970.

Joe Rodgers and Alan Nicewander. Thirteen Ways to Look at the Correlation Coefficient. *American Statistician - AMER STATIST*, 42:59–66, February 1988. doi: 10.1080/00031305.1988.10475524.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, volume 9351, pages 234–241. Springer International Publishing, Cham, 2015a. ISBN 978-3-319-24573-7 978-3-319-24574-4. doi: 10.1007/978-3-319-24574-4_28. URL http://link.springer.com/10.1007/978-3-319-24574-4_28. Series Title: Lecture Notes in Computer Science.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, Lecture Notes in Computer Science, pages 234–241, Cham, May 2015b. Springer International Publishing. ISBN 978-3-319-24574-4. doi: 10.1007/978-3-319-24574-4_28.

Artem Rozantsev, Mathieu Salzmann, and Pascal Fua. Beyond Sharing Weights for Deep Domain Adaptation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41 (4):801–814, April 2019. ISSN 1939-3539. doi: 10.1109/TPAMI.2018.2814042. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

Donald B. Rubin. Inference and Missing Data. *Biometrika*, 63(3):581–592, 1976. ISSN 0006-3444. doi: 10.2307/2335739. URL https://www.jstor.org/stable/2335739. Publisher: [Oxford University Press, Biometrika Trust].

Donald B. Rubin, editor. *Multiple Imputation for Nonresponse in Surveys*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., Hoboken, NJ, USA, June 1987. ISBN 978-0-470-31669-6 978-0-471-08705-2. doi: 10.1002/9780470316696. URL http://doi.wiley.com/10.1002/9780470316696.

Donald B. Rubin. Multiple Imputation After 18+ Years. *Journal of the American Statistical Association*, 91(434):473–489, 1996. ISSN 0162-1459. doi: 10.2307/2291635. URL https://www.jstor.org/stable/2291635. Publisher: [American Statistical Association, Taylor & Francis, Ltd.].

Donald B. Rubin. *Multiple Imputation for Nonresponse in Surveys*. John Wiley & Sons, June 2004. ISBN 978-0-471-65574-9. Google-Books-ID: bQBtw6rx_mUC.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986. ISSN 1476-4687. doi: 10.1038/323533a0. URL https://www.nature.com/articles/323533a0. Number: 6088 Publisher: Nature Publishing Group.

Kuniaki Saito, Kohei Watanabe, Yoshitaka Ushiku, and Tatsuya Harada. Maximum Classifier Discrepancy for Unsupervised Domain Adaptation, April 2018. URL http://arxiv.org/abs/1712.02560. arXiv:1712.02560 [cs].

Steven L. Salzberg. C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993. *Machine Learning*, 16(3):235–240, September 1994. ISSN 1573-0565. doi: 10.1007/BF00993309. URL https://doi.org/10.1007/BF00993309.

J. L. Schafer. *Analysis of Incomplete Multivariate Data*. Chapman and Hall/CRC, New York, August 1997. ISBN 978-0-367-80302-5. doi: 10.1201/9780367803025.

Shaun Seaman, John Galati, Dan Jackson, and John Carlin. What Is Meant by "Missing at Random"? *Statistical Science*, 28(2), May 2013. ISSN 0883-4237. doi: 10.1214/13-STS415. URL https://projecteuclid.org/journals/statistical-science/volume-28/issue-2/What-Is-Meant-by-Missing-at-Random/10.1214/13-STS415.full.

Waseem Shahzad, Qamar Rehman, and Ejaz Ahmed. Missing Data Imputation using Genetic Algorithm for Supervised Learning. *International Journal of Advanced Computer Science and Applications*, 8, March 2017. doi: 10.14569/IJACSA.2017.080360.

Jocelyn Sietsma and Robert J. F. Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4(1):67–79, January 1991. ISSN 0893-6080. doi: 10.1016/0893-6080(91)90033-2. URL https://www.sciencedirect.com/science/article/pii/0893608091900332.

Sho Sonoda and Noboru Murata. Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*, 43(2):233–268, September 2017. ISSN 1063-5203. doi: 10.1016/j.acha.2015.12.005. URL https://www.sciencedirect.com/science/article/pii/S1063520315001748.

Indro Spinelli, Simone Scardapane, and Aurelio Uncini. Missing data imputation with adversarially-trained graph convolutional networks. *Neural Networks*, 129:249–260, September 2020. ISSN 0893-6080. doi: 10.1016/j.neunet.2020.06.005. URL https://www.sciencedirect.com/science/article/pii/S0893608020302185.

Daniel J. Stekhoven and Peter Bühlmann. MissForest—Non-Parametric Missing Value Imputation for Mixed-Type Data. *Bioinformatics*, 28(1):112–118, January 2012. ISSN 1367-4803, 1460-2059. doi: 10.1093/bioinformatics/btr597. URL https://doi.org/10.1093/bioinformatics/btr597.

Baochen Sun and Kate Saenko. Deep CORAL: Correlation Alignment for Deep Domain Adaptation, July 2016. URL http://arxiv.org/abs/1607.01719. arXiv:1607.01719 [cs].

Baochen Sun, Jiashi Feng, and Kate Saenko. Return of Frustratingly Easy Domain Adaptation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), March 2016. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v30i1.10306. URL https://ojs.aaai.org/index.php/AAAI/article/view/10306.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1139–1147. PMLR, May 2013. URL https://proceedings.mlr.press/v28/sutskever13.html. ISSN: 1938-7228.

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 978-0-262-03924-6.

José A. Sáez, Julián Luengo, and Francisco Herrera. Fuzzy Rule Based Classification Systems versus crisp robust learners trained in presence of class noise's effects: A case of study. In *2011 11th International Conference on Intelligent Systems Design and Applications*, pages 1229–1234, November 2011. doi: 10.1109/ISDA.2011.6121827. ISSN: 2164-7151.

Adrián Sánchez-Morales, José Luis Sancho-Gómez, Juan Martínez García, and Aníbal Figueiras-Vidal. Improving deep learning performance with missing values via deletion and compensation. *Neural Computing and Applications*, 32, September 2020. doi: 10.1007/s00521-019-04013-2.

Adrián Sánchez-Morales, José Luis Sancho-Gómez, and Aníbal Figueiras-Vidal. Complete autoencoders for classification with missing values. *Neural Computing and Applications*, 33, March 2021. doi: 10.1007/s00521-020-05066-4.

Fei Tang and Hemant Ishwaran. Random forest missing data algorithms. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 10(6):363–377, 2017. ISSN 1932-1872. doi: 10.1002/sam.11348. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sam.11348. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sam.11348.

Hui Tang and Kui Jia. Discriminative Adversarial Domain Adaptation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):5940–5947, April 2020. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v34i04.6054. URL https://ojs.aaai.org/index.php/AAAI/article/view/6054.

Choh Man Teng. Evaluating Noise Correction. In Riichiro Mizoguchi and John Slaney, editors, *PRICAI 2000 Topics in Artificial Intelligence*, Lecture Notes in Computer Science, pages 188–198, Berlin, Heidelberg, 2000. Springer. ISBN 978-3-540-44533-3. doi: 10.1007/3-540-44533-1_22.

Choh Man Teng. Combining Noise Correction with Feature Selection. In Yahiko Kambayashi, Mukesh Mohania, and Wolfram Wöß, editors, *Data Warehousing and Knowledge Discovery*, Lecture Notes in Computer Science, pages 340–349, Berlin, Heidelberg, 2003. Springer. ISBN 978-3-540-45228-7. doi: 10.1007/978-3-540-45228-7_34.

Choh Man Teng. Polishing Blemishes: Issues in Data Correction. *IEEE Intelligent Systems*, 19(2):34–39, March 2004. ISSN 1941-1294. doi: 10.1109/MIS.2004.1274909. Conference Name: IEEE Intelligent Systems.

Le Van Thinh, Nguyen Xuan Hau, and Truong Dinh Tu. User-based AutoenCoder for QoS prediction. In *2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 308–311, August 2016. doi: 10.1109/ICSESS.2016.7883073. ISSN: 2327-0594.

Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. Learning Deep Representations for Graph Clustering. *Proceedings of the AAAI Conference on Artificial Intelligence*, 28(1), June 2014. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v28i1.8916. URL https://ojs.aaai.org/index.php/AAAI/article/view/8916.

Ivan Tomek. Two Modifications of CNN. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(11):769–772, November 1976. ISSN 0018-9472, 2168-2909. doi: 10.1109/TSMC.1976.4309452. URL http://ieeexplore.ieee.org/document/4309452/.

O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. B. Altman. Missing value estimation methods for DNA microarrays. *Bioinformatics (Oxford, England)*, 17(6):520–525, June 2001. ISSN 1367-4803. doi: 10.1093/bioinformatics/17.6.520.

Bhekisipho Twala. An Empirical Comparison of Techniques for Handling Incomplete Data Using Decision Trees. *Applied Artificial Intelligence*, 23(5):373–405, May 2009. ISSN 0883-9514. doi: 10.1080/08839510902872223. URL https://doi.org/10.1080/08839510902872223. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/08839510902872223.

Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep Domain Confusion: Maximizing for Domain Invariance, December 2014. URL http://arxiv.org/abs/1412.3474. arXiv:1412.3474 [cs].

Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial Discriminative Domain Adaptation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2962–2971, Honolulu, HI, July 2017. IEEE. ISBN 978-1-5386-0457-1. doi: 10.1109/CVPR.2017.316. URL http://ieeexplore.ieee.org/document/8099799/.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep Image Prior. *International Journal of Computer Vision*, 128(7):1867–1888, July 2020. ISSN 0920-5691, 1573-1405. doi: 10.1007/s11263-020-01303-4. URL https://doi.org/10.1007/s11263-020-01303-4.

S. van Buuren, H. C. Boshuizen, and D. L. Knook. Multiple imputation of missing blood pressure covariates in survival analysis. *Statistics in Medicine*, 18(6):681–694, March 1999. ISSN 0277-6715. doi: 10.1002/(sici)1097-0258(19990330)18:6<681::aid-sim71>3.0.co;2-r.

Jason D. Van Hulse, Taghi M. Khoshgoftaar, and Haiying Huang. The Pairwise Attribute Noise Detection Algorithm. *Knowledge and Information Systems*, 11(2):171–190, February 2007. ISSN 0219-1377, 0219-3116. doi: 10.1007/s10115-006-0022-x. URL http://link.springer.com/10.1007/s10115-006-0022-x.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Cédric Villani. *Optimal Transport*, volume 338 of *Grundlehren der mathematischen Wissenschaften*. Springer, Berlin, Heidelberg, 2009. ISBN 978-3-540-71049-3 978-3-540-71050-9. doi: 10.1007/978-3-540-71050-9. URL http://link.springer.com/10.1007/978-3-540-71050-9.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 1096–1103, Helsinki, Finland, 2008. ACM Press. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390294. URL http://portal.acm.org/citation.cfm?doid=1390156.1390294.

Akbar K Waljee, Ashin Mukherjee, Amit G Singal, Yiwei Zhang, Jeffrey Warren, Ulysses Balis, Jorge Marrero, Ji Zhu, and Peter DR Higgins. Comparison of imputation methods for missing laboratory data in medicine. *BMJ Open*, 3(8):e002847, August 2013. ISSN 2044-6055. doi: 10.1136/bmjopen-2013-002847. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3733317/.

Shoujin Wang, Wei Liu, Jia Wu, Longbing Cao, Qinxue Meng, and Paul J. Kennedy. Training deep neural networks on imbalanced data sets. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 4368–4374, Vancouver, BC, Canada, July 2016. IEEE. doi: 10.1109/IJCNN.2016.7727770.

Xian Wang, Ao Li, Zhaohui Jiang, and Huanqing Feng. Missing value estimation for DNA microarray gene expression data by Support Vector Regression imputation and orthogonal coding scheme. *BMC Bioinformatics*, 7(1):32, December 2006. ISSN 1471-2105. doi: 10.1186/1471-2105-7-32. URL https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-7-32.

Yufeng Wang, Dan Li, Xiang Li, and Min Yang. PC-GAIN: Pseudo-label conditional generative adversarial imputation networks for incomplete data. *Neural Networks*, 141:395–403, September 2021. ISSN 0893-6080. doi: 10.1016/j.neunet.2021.05.033. URL https://www.sciencedirect.com/science/article/pii/S089360802100229X.

Zhengwei Wang, Qi She, and Tomás E. Ward. Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy. *ACM Computing Surveys*, 54(2):1–38, March 2022. ISSN 0360-0300, 1557-7341. doi: 10.1145/3439723. URL https://dl.acm.org/doi/10.1145/3439723.

Garrett Wilson and Diane J. Cook. A Survey of Unsupervised Deep Domain Adaptation. *ACM Transactions on Intelligent Systems and Technology*, 11(5):51:1–51:46, 2020. doi: 10.1145/3400066.

Richard Wu, Aoqian Zhang, Ihab F Ilyas, and Theodoros Rekatsinas. Attention-based Learning for Missing Data Imputation in HoloClean. In *Proceedings of the 3 rd MLSys Conference*, Austin, TX, USA, 2020.

Yuanyuan Xu, Meina Kan, Shiguang Shan, and Xilin Chen. Mutual Learning of Joint and Separate Domain Alignments for Multi-Source Domain Adaptation. In *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1658–1667, Waikoloa, HI, USA, January 2022. IEEE. ISBN 978-1-66540-915-5. doi: 10.1109/WACV51458.2022.00172. URL https://ieeexplore.ieee.org/document/9707089/.

Li Yan, Hai-Tao Zhang, Jorge Goncalves, Yang Xiao, Maolin Wang, Yuqi Guo, Chuan Sun, Xiuchuan Tang, Liang Jing, Mingyang Zhang, Xiang Huang, Ying Xiao, Haosen Cao, Yanyan Chen, Tongxin Ren, Fang Wang, Yaru Xiao, Sufang Huang, Xi Tan, Niannian Huang, Bo Jiao, Cheng Cheng, Yong Zhang, Ailin Luo, Laurent Mombaerts, Junyang Jin, Zhiguo Cao, Shusheng Li, Hui Xu, and Ye Yuan. An interpretable mortality prediction model for COVID-19 patients. *Nature Machine Intelligence*, 2(5):283–288, May 2020. ISSN 2522-5839. doi: 10.1038/s42256-020-0180-7. URL http://www.nature.com/articles/s42256-020-0180-7.

Banghua Yang, Davy Janssens, Da Ruan, Mario Cools, Tom Bellemans, and Geert Wets. A Data Imputation Method with Support Vector Machines for Activity-Based Transportation Models. In Yinglin Wang and Tianrui Li, editors, *Foundations of Intelligent Systems*, Advances in Intelligent and Soft Computing, pages 249–257, Berlin, Heidelberg, 2012. Springer. ISBN 978-3-642-25664-6. doi: 10.1007/978-3-642-25664-6_29.

Ying Yang, Xindong Wu, and Xingquan Zhu. Dealing with Predictive-but-Unpredictable Attributes in Noisy Data Sources. In Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *Knowledge Discovery in Databases: PKDD 2004*, Lecture Notes in Computer Science, pages 471–483, Berlin, Heidelberg, 2004. Springer. ISBN 978-3-540-30116-5. doi: 10.1007/978-3-540-30116-5_43.

Jinsung Yoon, James Jordon, and Mihaela Schaar. GAIN: Missing Data Imputation using Generative Adversarial Nets. In *Proceedings of the 35th International Conference on Machine Learning*, pages 5689–5698. PMLR, July 2018. URL https://proceedings.mlr.press/v80/yoon18a.html. ISSN: 2640-3498.

Jiaxuan You, Xiaobai Ma, Yi Ding, Mykel J Kochenderfer, and Jure Leskovec. Handling Missing Data with Graph Representation Learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 19075–19087. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/dc36f18a9a0a776671d4879cae69b551-Abstract.html.

Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation*, 31(7):1235–1270, July 2019. ISSN 0899-7667. doi: 10.1162/neco_a_01199. URL https://doi.org/10.1162/neco_a_01199.

Bianca Zadrozny and Charles Elkan. Transforming Classifier Scores into Accurate Multiclass Probability Estimates. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2002. doi: 10.1145/775047.775151.

Chengqi Zhang, Yongsong Qin, Xiaofeng Zhu, Jilian Zhang, and Shichao Zhang. Clustering-based Missing Value Imputation for Data Preprocessing. In *2006 4th IEEE International Conference on Industrial Informatics*, pages 1081–1086, August 2006. doi: 10.1109/INDIN.2006.275767. ISSN: 2378-363X.

Hongyang Zhang, Maria-Florina Balcan, and David P Woodruff. Medical Missing Data Imputation by Stackelberg GAN. *Carnegie Mellon University*, 2018.

Shichao Zhang, Jilian Zhang, Xiaofeng Zhu, Yongsong Qin, and Chengqi Zhang. Missing Value Imputation Based on Data Clustering. In Marina L. Gavrilova and C. J. Kenneth Tan, editors, *Transactions on Computational Science I*, Lecture Notes in Computer Science, pages 128–138. Springer, Berlin, Heidelberg, 2008. ISBN 978-3-540-79299-4. doi: 10.1007/978-3-540-79299-4_7. URL https://doi.org/10.1007/978-3-540-79299-4_7.

Wen Zhang, Lingfei Deng, Lei Zhang, and Dongrui Wu. A Survey on Negative Transfer. *IEEE/CAA Journal of Automatica Sinica*, pages 1–25, 2022. doi: 10.1109/JAS.2022.106004.

Han Zhao, Shanghang Zhang, Guanhang Wu, José M. F. Moura, Joao P Costeira, and Geoffrey J Gordon. Adversarial Multiple Source Domain Adaptation. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018a. URL https://papers.nips.cc/paper/2018/hash/717d8b3d60d9eea997b35b02b6a4e867-Abstract.html.

Liang Zhao, Zhikui Chen, Zhennan Yang, Yueming Hu, and Mohammad S. Obaidat. Local Similarity Imputation Based on Fast Clustering for Incomplete Data in Cyber-Physical Systems. *IEEE Systems Journal*, 12(2):1610–1620, June 2018b. ISSN 1932-8184, 1937-9234, 2373-7816. doi: 10.1109/JSYST.2016.2576026. URL https://ieeexplore.ieee.org/document/7496925/.

Pengfei Zhu, Binyuan Hui, Changqing Zhang, Dawei Du, Longyin Wen, and Qinghua Hu. Multi-view Deep Subspace Clustering Networks. *arXiv:1908.01978 [cs]*, August 2019a. URL http://arxiv.org/abs/1908.01978. arXiv: 1908.01978 version: 1.

Xingquan Zhu and Xindong Wu. Class Noise vs. Attribute Noise: A Quantitative Study. *Artificial Intelligence Review*, 22(3):177–210, November 2004. ISSN 1573-7462. doi: 10.1007/s10462-004-0751-8. URL https://doi.org/10.1007/s10462-004-0751-8.

Yongchun Zhu, Fuzhen Zhuang, and Deqing Wang. Aligning Domain-specific Distribution and Classifier for Cross-domain Classification from Multiple Sources. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):5989–5996, July 2019b. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v33i01.33015989. URL http://arxiv.org/abs/2201.01003. arXiv:2201.01003 [cs].

Yongchun Zhu, Fuzhen Zhuang, Jindong Wang, Guolin Ke, Jingwu Chen, Jiang Bian, Hui Xiong, and Qing He. Deep Subdomain Adaptation Network for Image Classification. *IEEE Transactions on Neural Networks and Learning Systems*, 32(4):1713–1722, April 2021. ISSN 2162-237X, 2162-2388. doi: 10.1109/TNNLS.2020.2988928. URL http://arxiv.org/abs/2106.09388. arXiv:2106.09388 [cs].

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A Comprehensive Survey on Transfer Learning, June 2020. URL http://arxiv.org/abs/1911.02685. arXiv:1911.02685 [cs, stat].

Yukun Zuo, Hantao Yao, and Changsheng Xu. Attention-Based Multi-Source Domain Adaptation. *IEEE Transactions on Image Processing*, 30:3793–3803, 2021. ISSN 1057-7149, 1941-0042. doi: 10.1109/TIP.2021.3065254. URL https://ieeexplore.ieee.org/document/9380556/.

Appendix A

DIOS results

Contents

A.1 Comparative Study on an Imputation Task

Model	Metric	MCAR 25	MCAR 50	MCAR 75	MAR 25	MAR 50	MAR 75	MNAR 25	MNAR 50	MNAR 75
DIOS	RMSE	.147655 ±.001417	.165753 ±.002451	.184432 ±.005150	.152351 ±.007061	.162231 ±.006476	.175626 ±.007016	.153236 ±.002157	.166068 ±.001453	.185419 ±.003252
	ACC	**67.94%** **±0.99%**	**66.35%** **±1.29%**	62.94% ±0.85%	**69.30%** **±1.10%**	**67.71%** **±1.38%**	**66.68%** **±0.92%**	**68.21%** **±0.98%**	**66.35%** **±1.17%**	**63.43%** **±1.16%**
SUB	RMSE ≡	.161943 ±.001601 •	.161585 ±.000946 ∘	.162146 ±.000396 ∘	.168522 ±.006091 •	.164284 ±.005278 ≡	.163993 ±.005279 ∘	.167514 ±.001698 •	.163280 ±.000885 ∘	.162401 ±.000708 ∘
	ACC •	60.52% ±1.47% •	57.49% ±1.86% •	56.22% ±1.47% •	61.87% ±1.33% •	60.93% ±1.92% •	61.46% ±1.49% •	60.12% ±1.84% •	57.57% ±1.38% •	56.68% ±1.04% •
KNN	RMSE ∘	.141457 ±.001792 ∘	.147778 ±.000896 ∘	.160626 ±.000841 ∘	.144766 ±.006077 ∘	.145290 ±.004796 ∘	.148835 ±.005163 ∘	.146929 ±.001849 ∘	.150601 ±.001383 ∘	.163211 ±.001181 ∘
	ACC •	62.98% ±1.27% •	60.50% ±2.51% •	57.52% ±1.96% •	63.45% ±0.96% •	63.23% ±1.49% •	64.27% ±2.13% •	63.27% ±0.93% •	59.86% ±1.27% •	56.88% ±1.11% •
GAIN	RMSE •	.317680 ±.008408 •	.443963 ±.010126 •	.489011 ±.023880 •	.351021 ±.011060 •	.389780 ±.009296 •	.460428 ±.017003 •	.352910 ±.010999 •	.457647 ±.007529 •	.512846 ±.016024 •
	ACC •	60.03% ±1.49% •	55.28% ±2.24% •	54.16% ±2.43% •	60.19% ±1.83% •	57.25% ±2.64% •	57.25% ±2.88% •	58.18% ±1.69% •	56.60% ±2.12% •	54.03% ±2.08% •
MIDA	RMSE ∘	.140998 ±.001763 ∘	.150938 ±.001048 ∘	.159915 ±.000502 ∘	.146081 ±.006059 ≡	.151163 ±.005129 ∘	.158124 ±.005419 ∘	.147166 ±.001601 ∘	.153171 ±.000929 ∘	.160213 ±.000877 ∘
	ACC •	62.40% ±1.15% •	59.68% ±2.01% •	57.15% ±1.37% •	62.74% ±0.85% •	62.25% ±1.83% •	62.28% ±1.73% •	62.58% ±1.51% •	59.90% ±1.43% •	57.34% ±1.46% •
SOFT	RMSE •	.162199 ±.001267 •	.188729 ±.001550 •	.318306 ±.002893 •	.167524 ±.006668 •	.182695 ±.004686 •	.261668 ±.011991 •	.167500 ±.001411 •	.190355 ±.000792 •	.319813 ±.002293 •
	ACC •	60.70% ±1.25% •	57.40% ±2.35% •	54.74% ±1.99% •	62.12% ±1.25% •	60.84% ±2.12% •	60.85% ±1.74% •	60.83% ±1.41% •	58.07% ±2.03% •	54.11% ±1.01% •
MICE	RMSE ∘	.146057 ±.003959 ≡	.139764 ±.000973 ∘	.161761 ±.001185 ∘	.169247 ±.007783 •	.148322 ±.006285 ∘	.159073 ±.006032 ∘	.155295 ±.006698 ≡	.140114 ±.001137 ∘	.161093 ±.001602 ∘
	ACC •	64.24% ±1.08% •	63.73% ±1.93% •	59.73% ±2.22% •	63.58% ±1.03% •	63.98% ±1.49% •	64.20% ±1.92% •	64.04% ±1.05% •	62.49% ±1.15% •	61.30% ±1.25% •
SINK	RMSE ∘	.134969 ±.001879 ∘	.144074 ±.001081 ∘	.164820 ±.000555 ∘	.138898 ±.006151 ∘	.141512 ±.005241 ∘	.149139 ±.005583 ∘	.139490 ±.001210 ∘	.146041 ±.001143 ∘	.165661 ±.001388 ∘
	ACC •	64.71% ±1.33% •	64.38% ±2.11% •	61.97% ±2.02% ≡	64.31% ±1.15% •	64.31% ±1.89% •	65.04% ±0.85% •	64.51% ±1.06% •	65.22% ±2.02% ≡	60.51% ±2.05% •
MISS	RMSE ∘	**.108673** **±.002941** ∘	**.125300** **±.001736** ∘	**.155951** **±.001637** ∘	**.109202** **±.005189** ∘	**.117619** **±.005420** ∘	**.131325** **±.006977** ∘	**.113913** **±.001589** ∘	**.126153** **±.002062** ∘	**.153473** **±.002002** ∘
	ACC •	65.02% ±1.73% •	65.42% ±1.66% ≡	**63.63%** **±1.50%** ≡	64.48% ±1.01% •	64.90% ±1.10% •	65.55% ±1.28% •	65.00% ±0.48% •	64.16% ±1.19% •	62.94% ±2.04% ≡

DIOS is: • significantly better, ≡ equivalent, ∘ significantly worse, p-value: 0.05

Table A.1: Experimental results on an imputation task on the benchmark dataset ARRHYTHMIA with mechanisms Missing Completely At Random (MCAR)/Missing At Random (MAR)/Missing Not At Random (MNAR) at 25/50/75% missing rates.

Model	Metric	MCAR 25	MCAR 50	MCAR 75	MAR 25	MAR 50	MAR 75	MNAR 25	MNAR 50	MNAR 75
DIOS	RMSE	.354791 ±.009764	.359639 ±.007294	.364581 ±.009414	.364211 ±.034164	.365042 ±.032360	.365587 ±.032875	.356339 ±.008528	.361909 ±.008784	.367104 ±.010949
	ACC	81.52% ±1.03%	75.99% ±1.33%	**68.84%** **±2.28%**	**84.83%** **±2.40%**	**83.58%** **±4.22%**	**81.97%** **±5.57%**	82.83% ±0.94%	**78.48%** **±1.58%**	**71.97%** **±3.16%**
SUB	RMSE ∘	.352278 ±.006048 ≡	**.352346** **±.001980** ∘	.352639 ±.002130 ∘	.360654 ±.033761 ≡	.360363 ±.031252 ≡	.359879 ±.030958 ≡	.351967 ±.005062 ≡	**.353703** **±.004881** ∘	.353633 ±.002795 ∘
	ACC •	79.68% ±1.54% •	73.88% ±1.82% •	65.80% ±2.36% •	83.04% ±2.58% ≡	80.91% ±4.66% ≡	79.90% ±5.73% ≡	81.09% ±0.79% •	75.49% ±1.47% •	68.61% ±2.68% •
KNN	RMSE ≡	.355052 ±.010771 ≡	.368031 ±.004462 •	.373857 ±.004524 •	.358342 ±.031800 ≡	.361411 ±.031148 ≡	.367190 ±.030804 ≡	.364642 ±.011342 ≡	.392008 ±.019119 •	.373314 ±.004233 ≡
	ACC •	80.22% ±1.28% •	73.75% ±1.38% •	62.64% ±2.40% •	83.29% ±2.13% ≡	81.81% ±4.15% ≡	79.52% ±6.03% ≡	80.67% ±1.37% •	72.72% ±2.09% •	66.06% ±3.53% •
GAIN	RMSE •	.673642 ±.012690 •	.583580 ±.008805 •	.568528 ±.010932 •	.748863 ±.034236 •	.630108 ±.037070 •	.596207 ±.041304 •	.681309 ±.012488 •	.600566 ±.017407 •	.576864 ±.011416 •
	ACC •	78.30% ±2.07% •	71.54% ±2.54% •	62.65% ±3.27% •	82.14% ±2.68% •	79.81% ±3.76% ≡	77.94% ±7.07% ≡	78.83% ±2.27% •	71.32% ±2.26% •	64.83% ±2.87% •
MIDA	RMSE ∘	.353166 ±.009634 ≡	.353261 ±.004930 ∘	**.352348** **±.002132** ∘	.355059 ±.034699 ≡	**.358202** **±.030826** ≡	**.358725** **±.030234** ≡	.353342 ±.010923 ≡	.353716 ±.007260 ∘	**.353433** **±.003024** ∘
	ACC •	79.43% ±1.35% •	74.14% ±2.07% •	65.74% ±2.50% •	82.81% ±2.77% ≡	81.12% ±4.56% ≡	79.83% ±5.77% ≡	81.09% ±1.48% •	75.42% ±1.62% •	68.43% ±2.97% •
SOFT	RMSE •	.399457 ±.010849 •	.439985 ±.020144 •	.488128 ±.011124 •	.406362 ±.039287 •	.435463 ±.050641 •	.442456 ±.033540 •	.399789 ±.012094 •	.435826 ±.013690 •	.485905 ±.008625 •
	ACC •	80.33% ±1.30% •	72.41% ±1.59% •	62.97% ±2.22% •	82.96% ±1.99% ≡	81.07% ±4.22% ≡	79.13% ±5.65% ≡	81.54% ±0.91% •	75.45% ±1.63% •	66.33% ±3.73% •
MICE	RMSE ≡	.337901 ±.023228 ≡	.373411 ±.014028 •	.368016 ±.008155 ≡	.336300 ±.031014 ≡	.401930 ±.038811 •	.377346 ±.041223 ≡	.351152 ±.025579 ≡	.374875 ±.009909 •	.369924 ±.008788 ≡
	ACC •	81.42% ±1.06% ≡	75.88% ±1.43% ≡	66.14% ±1.51% •	83.75% ±1.86% ≡	82.22% ±3.22% ≡	80.19% ±5.06% ≡	82.49% ±1.19% ≡	77.45% ±1.09% ≡	69.04% ±2.48% •
SINK	RMSE •	.418757 ±.008199 •	.418634 ±.005683 •	.422992 ±.007348 •	.417165 ±.050458 •	.413836 ±.048295 •	.408270 ±.050617 •	.416659 ±.008568 •	.422191 ±.009417 •	.418687 ±.006926 •
	ACC •	78.87% ±1.66% •	71.94% ±1.51% •	61.52% ±1.32% •	83.20% ±2.43% ≡	79.83% ±4.70% ≡	78.74% ±6.95% ≡	80.17% ±1.46% •	73.10% ±2.05% •	63.29% ±2.88% •
MISS	RMSE ≡	**.320149** **±.008388** ∘	.366880 ±.005811 •	.426367 ±.007554 •	**.321893** **±.033098** ∘	.362349 ±.026773 ≡	.404480 ±.044979 ≡	**.316289** **±.005318** ∘	.359462 ±.005909 ≡	.412250 ±.011455 •
	ACC •	**82.35%** **±1.18%** ≡	**76.33%** **±1.34%** ≡	63.87% ±2.77% •	83.93% ±1.28% ≡	82.41% ±2.69% ≡	79.25% ±7.00% ≡	**83.10%** **±1.06%** ≡	77.91% ±2.04% ≡	68.52% ±3.29% •

DIOS is: • significantly better, ≡ equivalent, ∘ significantly worse, p-value: 0.05

Table A.2: Experimental results on an imputation task on the benchmark dataset STATLOG with mechanisms MCAR/MAR/MNAR at 25/50/75% missing rates.

Model	Metric	MCAR 25	MCAR 50	MCAR 75	MAR 25	MAR 50	MAR 75	MNAR 25	MNAR 50	MNAR 75
DIOS	RMSE	.136906 ±.000789	.159547 ±.002225	.206618 ±.004068	.134042 ±.002204	.149251 ±.001893	.173816 ±.002857	.137195 ±.001611	.159608 ±.002029	.205436 ±.002275
	ACC	**98.29% ±0.09%**	**98.12% ±0.07%**	96.91% ±0.19%	**98.31% ±0.09%**	**98.28% ±0.09%**	**98.14% ±0.13%**	**98.31% ±0.05%**	**98.23% ±0.08%**	97.02% ±0.20%
SUB	RMSE •	.319362 ±.000359 •	.319532 ±.000171 •	.319656 ±.000100 •	.323797 ±.003016 •	.324849 ±.003200 •	.326267 ±.003431 •	.322167 ±.000659 •	.323362 ±.000454 •	.324760 ±.000277 •
	ACC •	97.78% ±0.12% •	96.24% ±0.24% •	67.83% ±1.87% •	97.91% ±0.13% •	97.79% ±0.21% •	97.31% ±0.32% •	97.83% ±0.13% •	97.00% ±0.22% •	83.38% ±1.44% •
KNN	RMSE •	.171465 ±.000433 •	.182051 ±.000194 •	.213410 ±.000298 •	.173290 ±.002162 •	.181652 ±.002409 •	.196197 ±.002762 •	.172750 ±.000411 •	.184768 ±.000307 •	.220940 ±.000349 •
	ACC •	97.71% ±0.11% •	97.43% ±0.10% •	95.84% ±0.23% •	97.82% ±0.16% •	97.69% ±0.19% •	97.30% ±0.28% •	97.74% ±0.14% •	97.50% ±0.17% •	95.59% ±0.30% •
GAIN	RMSE •	.269801 ±.001455 •	.238402 ±.001020 •	.536074 ±.005791 •	.307802 ±.001784 •	.248680 ±.002288 •	.391710 ±.009960 •	.268669 ±.001205 •	.241067 ±.002235 •	.554727 ±.003461 •
	ACC •	97.57% ±0.13% •	97.40% ±0.20% •	79.17% ±1.60% •	97.73% ±0.11% •	97.58% ±0.13% •	96.81% ±0.27% •	97.68% ±0.15% •	97.47% ±0.16% •	86.66% ±0.91% •
MIDA	RMSE •	.289513 ±.004842 •	.318562 ±.000224 •	.319529 ±.000099 •	.282678 ±.005380 •	.307984 ±.006300 •	.324491 ±.004009 •	.291557 ±.003724 •	.322264 ±.000521 •	.324621 ±.000282 •
	ACC •	97.71% ±0.11% •	96.31% ±0.23% •	68.13% ±1.92% •	97.79% ±0.11% •	97.74% ±0.18% •	97.28% ±0.36% •	97.77% ±0.18% •	97.00% ±0.25% •	83.53% ±1.49% •
SOFT	RMSE •	.297167 ±.000474 •	.310272 ±.000676 •	.385783 ±.001054 •	.302148 ±.003227 •	.315895 ±.003425 •	.367079 ±.004706 •	.300616 ±.001117 •	.315058 ±.001206 •	.389065 ±.001178 •
	ACC •	97.78% ±0.13% •	96.04% ±0.32% •	69.45% ±2.16% •	97.97% ±0.11% •	97.73% ±0.19% •	97.27% ±0.32% •	97.80% ±0.14% •	96.74% ±0.30% •	84.80% ±1.18% •
MICE	RMSE ○	.124647 ±.000272 ○	.145737 ±.000250 ○	.206923 ±.000577 ≡	.127042 ±.001405 ○	.131368 ±.001965 ○	.157530 ±.003018 ○	.125361 ±.000201 ○	.146554 ±.000298 ○	.209269 ±.000740 •
	ACC •	98.00% ±0.17% •	97.84% ±0.11% •	**97.09% ±0.29%** ≡	98.03% ±0.09% •	98.02% ±0.13% •	97.84% ±0.17% •	98.08% ±0.13% •	97.86% ±0.10% •	**97.05% ±0.25%** ≡
SINK	RMSE •	.182632 ±.000293 •	.195835 ±.000346 •	.220215 ±.000355 •	.185622 ±.002000 •	.199430 ±.002217 •	.223433 ±.002465 •	.184920 ±.000370 •	.200091 ±.000473 •	.226424 ±.000519 •
	ACC •	97.77% ±0.19% •	97.03% ±0.24% •	92.90% ±0.45% •	97.90% ±0.19% •	97.59% ±0.18% •	97.24% ±0.26% •	97.74% ±0.09% •	97.18% ±0.22% •	93.49% ±0.54% •
MISS	RMSE ○	**.105803 ±.000459** ○	**.131395 ±.000367** ○	**.177283 ±.000749** ○	**.101257 ±.001610** ○	**.117162 ±.002386** ○	**.139133 ±.002813** ○	**.106212 ±.000529** ○	**.132246 ±.000835** ○	**.175603 ±.001144** ○
	ACC •	97.97% ±0.13% •	97.70% ±0.13% •	96.22% ±0.33% •	98.05% ±0.06% •	97.88% ±0.15% •	97.65% ±0.15% •	98.03% ±0.08% •	97.74% ±0.14% •	96.40% ±0.24% •

DIOS is: • significantly better, ≡ equivalent, ○ significantly worse, p-value: 0.05

Table A.3: Experimental results on an imputation task on the benchmark dataset MFEAT with mechanisms MCAR/MAR/MNAR at 25/50/75% missing rates.

Model	Metric	MCAR 25	MCAR 50	MCAR 75	MAR 25	MAR 50	MAR 75	MNAR 25	MNAR 50	MNAR 75
DIOS	RMSE	**.150895 ±.002641**	**.164328 ±.001631**	**.180357 ±.001074**	**.150250 ±.001979**	**.155772 ±.001988**	**.165047 ±.002045**	**.153728 ±.002247**	**.164754 ±.001048**	**.179688 ±.001020**
	ACC	**91.28% ±0.24%**	**91.17% ±0.16%**	**90.25% ±0.30%**	**91.33% ±0.23%**	**91.12% ±0.20%**	**91.00% ±0.46%**	**91.42% ±0.45%**	**90.95% ±0.31%**	**90.30% ±0.33%**
SUB	RMSE •	.221428 ±.000130 •	.221674 ±.000064 •	.222251 ±.000043 •	.225217 ±.000231 •	.225188 ±.000206 •	.226367 ±.000209 •	.224822 ±.000138 •	.224716 ±.000102 •	.225798 ±.000087 •
	ACC •	89.03% ±0.56% •	83.25% ±0.82% •	44.90% ±1.94% •	90.12% ±0.28% •	89.10% ±0.56% •	89.35% ±0.61% •	89.38% ±0.65% •	85.00% ±0.72% •	59.33% ±2.36% •
KNN	RMSE •	.161034 ±.000213 •	.171429 ±.000138 •	.188205 ±.000152 •	.165831 ±.000410 •	.176910 ±.000300 •	.192269 ±.000269 •	.165285 ±.000113 •	.176218 ±.000066 •	.192501 ±.000077 •
	ACC •	89.45% ±0.47% •	86.92% ±0.72% •	84.10% ±0.82% •	89.80% ±0.22% •	88.25% ±0.43% •	86.65% ±0.28% •	89.22% ±0.41% •	87.45% ±0.52% •	83.60% ±0.76% •
GAIN	RMSE	- -	- -	- -	- -	- -	- -	- -	- -	- -
	ACC	- -	- -	- -	- -	- -	- -	- -	- -	- -
MIDA	RMSE •	.222160 ±.000246 •	.224138 ±.000239 •	.226495 ±.000265 •	.225108 ±.000329 •	.226363 ±.000316 •	.228399 ±.000301 •	.225200 ±.000268 •	.226842 ±.000272 •	.229376 ±.000379 •
	ACC •	88.45% ±0.62% •	83.62% ±0.54% •	46.05% ±2.54% •	89.38% ±0.30% •	88.88% ±0.64% •	89.17% ±0.56% •	88.78% ±0.38% •	85.78% ±0.74% •	60.73% ±2.37% •
SOFT	RMSE •	.212770 ±.000139 •	.225273 ±.000370 •	.300132 ±.001366 •	.216074 ±.000177 •	.226427 ±.000238 •	.272862 ±.000319 •	.215795 ±.000127 •	.228053 ±.000344 •	.302223 ±.000150 •
	ACC •	89.00% ±0.58% •	82.12% ±1.10% •	24.55% ±0.95% •	89.08% ±0.23% •	88.90% ±0.44% •	88.62% ±0.82% •	89.17% ±0.63% •	84.33% ±0.97% •	34.40% ±1.21% •
MICE	RMSE	- -	- -	- -	- -	- -	- -	- -	- -	- -
	ACC	- -	- -	- -	- -	- -	- -	- -	- -	- -
SINK	RMSE •	.155776 ±.000291 •	.167637 ±.000307 •	.189442 ±.000627 •	.159341 ±.000382 •	.170726 ±.000316 •	.187439 ±.000259 •	.159377 ±.000316 •	.171885 ±.000361 •	.193548 ±.000439 •
	ACC •	90.12% ±0.68% •	88.47% ±0.74% •	76.20% ±1.82% •	89.80% ±0.61% •	89.65% ±0.68% •	89.28% ±0.94% •	89.95% ±0.58% •	88.53% ±0.85% •	76.72% ±1.97% •
MISS	RMSE	- -	- -	- -	- -	- -	- -	- -	- -	- -
	ACC	- -	- -	- -	- -	- -	- -	- -	- -	- -

DIOS is: • significantly better, ≡ equivalent, ○ significantly worse, p-value: 0.05

Table A.4: Experimental results on an imputation task on the benchmark dataset ORL with mechanisms MCAR/MAR/MNAR at 25/50/75% missing rates.

Model	Metric	MCAR 25	MCAR 50	MCAR 75	MAR 25	MAR 50	MAR 75	MNAR 25	MNAR 50	MNAR 75
DIOS	RMSE	.179357 ±.005146	.183667 ±.005556	.187247 ±.005486	.181048 ±.009643	.180440 ±.008267	.178010 ±.006844	.179684 ±.005332	.181208 ±.004697	.184389 ±.005870
	ACC	**73.10%** **±1.28%**	**70.04%** **±0.75%**	**67.10%** **±0.63%**	**74.41%** **±1.55%**	**72.97%** **±1.55%**	**71.21%** **±2.41%**	**73.42%** **±1.26%**	**71.12%** **±1.08%**	**68.69%** **±1.06%**
SUB	RMSE ○	.174523 ±.004774 ≡	**.173623** **±.001549** ○	**.173725** **±.000378** ○	.177390 ±.010051 ≡	.175049 ±.009208 ≡	.173129 ±.007534 ≡	.175769 ±.005101 ≡	**.176149** **±.003080** ○	**.176457** **±.002658** ○
	ACC •	70.24% ±1.07% •	66.72% ±1.80% •	63.62% ±1.93% •	71.05% ±1.49% •	69.19% ±2.02% •	67.71% ±3.28% •	71.07% ±2.00% •	68.03% ±1.49% •	66.08% ±2.04% •
KNN	RMSE ○	.174144 ±.004952 ○	.181033 ±.001311 ≡	.181461 ±.001623 ○	.167741 ±.011594 ○	**.171564** **±.011021** ≡	.174089 ±.010412 ≡	.189628 ±.012075 •	.188144 ±.003784 •	.180590 ±.002672 ≡
	ACC •	70.28% ±1.90% •	66.24% ±1.57% •	63.63% ±2.02% •	71.17% ±1.26% •	70.06% ±1.55% •	67.51% ±2.99% •	70.21% ±1.35% •	67.03% ±1.66% •	64.49% ±1.13% •
GAIN	RMSE •	.254705 ±.004902 •	.223014 ±.009764 •	.221218 ±.008795 •	.286876 ±.017126 •	.232661 ±.008550 •	.222677 ±.004746 •	.260687 ±.008514 •	.224324 ±.005695 •	.240855 ±.024878 •
	ACC •	70.70% ±0.82% •	66.20% ±1.23% •	62.89% ±1.97% •	70.68% ±1.57% •	69.05% ±1.78% •	66.72% ±3.11% •	70.48% ±1.33% •	67.90% ±1.66% •	63.79% ±1.62% •
MIDA	RMSE ○	.186314 ±.012573 ≡	.178359 ±.006692 ≡	.174592 ±.003432 ○	.180846 ±.012939 ≡	.174561 ±.009621 ≡	**.172014** **±.007672** ≡	.179897 ±.011177 ≡	.180030 ±.008561 ≡	.177930 ±.005142 ○
	ACC •	70.17% ±1.44% •	66.80% ±1.59% •	63.23% ±2.02% •	71.56% ±1.40% •	69.31% ±2.66% •	67.59% ±3.29% •	70.46% ±1.40% •	68.12% ±1.84% •	66.09% ±1.58% •
SOFT	RMSE •	.266956 ±.019727 •	.308334 ±.018194 •	.368544 ±.007549 •	.258113 ±.028138 •	.279084 ±.025456 •	.298164 ±.056538 •	.270694 ±.029011 •	.310358 ±.023320 •	.362265 ±.008335 •
	ACC •	70.16% ±1.12% •	67.50% ±1.20% •	63.21% ±1.40% •	70.74% ±1.67% •	68.98% ±2.52% •	67.83% ±2.45% •	70.82% ±1.63% •	67.77% ±2.53% •	64.10% ±2.37% •
MICE	RMSE ○	**.159770** **±.006494** ○	.180242 ±.002994 ≡	.179222 ±.001852 ○	**.160694** **±.009298** ○	.181529 ±.009714 ≡	.176488 ±.005994 ≡	.169201 ±.009142 ○	.182615 ±.004548 ≡	.184273 ±.004195 ≡
	ACC •	70.02% ±1.09% •	67.32% ±1.80% •	62.93% ±1.77% •	71.82% ±1.39% •	69.56% ±1.92% •	67.45% ±3.32% •	71.51% ±1.94% •	68.58% ±1.34% •	65.23% ±2.07% •
SINK	RMSE •	.179658 ±.003267 ≡	.190006 ±.002562 •	.204615 ±.002847 •	.173807 ±.012449 ≡	.180448 ±.011600 ≡	.187591 ±.010050 •	.178070 ±.005936 ≡	.190206 ±.003927 •	.204999 ±.002906 •
	ACC •	70.04% ±1.06% •	65.81% ±1.93% •	61.02% ±1.64% •	70.62% ±0.97% •	68.82% ±2.47% •	67.45% ±3.12% •	70.27% ±1.71% •	65.49% ±1.65% •	62.42% ±1.63% •
MISS	RMSE ≡	.165444 ±.005227 ○	.185626 ±.002332 ≡	.201445 ±.002156 •	.164886 ±.012820 ○	.179029 ±.013580 ≡	.197582 ±.013769 •	**.166607** **±.009190** ○	.185163 ±.004316 ≡	.199842 ±.006875 •
	ACC •	70.19% ±1.41% •	66.04% ±2.27% •	62.76% ±1.19% •	70.71% ±1.37% •	69.70% ±1.94% •	66.56% ±3.22% •	71.21% ±0.91% •	67.12% ±1.57% •	63.69% ±2.48% •

DIOS is: • significantly better, ≡ equivalent, ○ significantly worse, p-value: 0.05

Table A.5: Experimental results on an imputation task on the benchmark dataset PIMA with mechanisms MCAR/MAR/MNAR at 25/50/75% missing rates.

Model	Metric	MYOCARDIAL		NHANES		COVID	
DIOS	bACC	**77.91%** **±1.12%**		**64.17%** **±0.36%**		86.84% ±1.23%	
	AUC	**86.28%** **±0.42%**		**69.92%** **±0.56%**		**92.95%** **±0.93%**	
SUB	bACC	77.30% ±0.00%	≡	60.35% ±0.00%	•	85.91% ±0.00%	•
	AUC	85.09% ±0.00%	•	66.10% ±0.00%	•	91.20% ±0.00%	•
KNN	bACC	68.83% ±0.00%	•	63.00% ±0.00%	•	**88.08%** **±0.00%**	○
	AUC	78.94% ±0.00%	•	67.78% ±0.00%	•	91.53% ±0.00%	•
GAIN	bACC	63.89% ±2.21%	•	61.36% ±0.53%	•	85.14% ±0.91%	•
	AUC	74.22% ±1.11%	•	66.85% ±0.40%	•	91.36% ±0.73%	•
MICE	bACC	76.55% ±0.00%	•	61.70% ±0.00%	•	87.98% ±0.00%	○
	AUC	81.39% ±0.00%	•	67.30% ±0.00%	•	92.43% ±0.00%	≡
MISS	bACC	73.00% ±0.87%	•	61.40% ±1.03%	•	85.15% ±1.67%	•
	AUC	80.82% ±1.60%	•	66.48% ±0.90%	•	91.30% ±1.20%	•
SOFT	bACC	77.24% ±0.99%	≡	61.70% ±0.93%	•	84.48% ±0.78%	•
	AUC	84.88% ±0.77%	•	66.93% ±1.08%	•	91.12% ±0.85%	•
SINK	bACC	75.66% ±1.22%	•	60.77% ±0.98%	•	86.82% ±1.49%	≡
	AUC	83.26% ±1.01%	•	65.42% ±1.18%	•	91.48% ±1.13%	•
MIDA	bACC	75.09% ±0.70%	•	62.15% ±1.26%	•	85.55% ±1.12%	•
	AUC	82.87% ±0.78%	•	66.91% ±1.30%	•	91.67% ±0.62%	•

DIOS is: • significantly better, ≡ equivalent, ○ significantly worse, p-value: 0.05

Table A.6: Experimental results on an imputation task on three real-world medical mixed-type tabular datasets.

A.2 Comparative Study on an Imputation Task

Model	Metric	NOISE 0	NOISE 0.05	NOISE 0.1	NOISE 0.15	NOISE 0.2	NOISE 0.4	NOISE 0.6
DIOS	RMSE	.022217 ±.003364	**.094868 ±.001615**	**.117715 ±.001888**	**.132717 ±.002012**	**.145746 ±.001884**	**.201759 ±.003169**	**.265438 ±.005044**
	ACC	**69.09% ±1.09%**	**68.35% ±1.48%**	**67.04% ±0.88%**	**64.76% ±0.92%**	**63.96% ±1.56%**	**60.55% ±1.15%**	**59.41% ±0.60%**
NONE	RMSE •	**.000000 ±.000000** ○	.107722 ±.000852 •	.152856 ±.000889 •	.187011 ±.000792 •	.215826 ±.000864 •	.305352 ±.001080 •	.374148 ±.000717 •
	ACC •	63.92% ±0.00% •	60.10% ±2.52% •	59.78% ±1.64% •	58.41% ±2.35% •	57.54% ±1.06% •	55.59% ±2.52% •	52.17% ±2.35% •
SFIL	RMSE	- -	- -	- -	- -	- -	- -	- -
	ACC •	56.15% ±0.73% •	55.95% ±0.98% •	55.27% ±0.90% •	55.09% ±0.63% •	54.54% ±0.50% •	54.72% ±1.19% •	54.19% ±1.68% •
SPOL	RMSE •	.062775 ±.002273 •	.131134 ±.002834 •	.169402 ±.002560 •	.197156 ±.002424 •	.220914 ±.001850 •	.300011 ±.001453 •	.365156 ±.000946 •
	ACC •	60.76% ±1.03% •	54.06% ±3.20% •	47.70% ±1.67% •	48.07% ±2.66% •	47.30% ±1.88% •	48.74% ±1.24% •	49.49% ±2.17% •
PFIL	RMSE	- -	- -	- -	- -	- -	- -	- -
	ACC •	62.50% ±1.20% •	60.22% ±3.26% •	59.94% ±2.01% •	57.96% ±2.46% •	56.62% ±1.13% •	54.58% ±1.67% •	53.42% ±2.14% •
PPOL	RMSE •	.042325 ±.000245 •	.130772 ±.001476 •	.168084 ±.001590 •	.195269 ±.000960 •	.219575 ±.001124 •	.301512 ±.001295 •	.367876 ±.000736 •
	ACC •	63.21% ±1.24% •	60.99% ±3.11% •	60.14% ±1.99% •	56.71% ±2.33% •	54.96% ±2.18% •	52.50% ±2.40% •	51.77% ±1.91% •

DIOS is: • significantly better, ≡ equivalent, ○ significantly worse, p-value: 0.05

Table A.7: Experimental results on a correction task on the benchmark dataset ARRHYTHMIA with erroneous values rates 0/5/10/15/20/40/60%.

Model	Metric	NOISE 0	NOISE 0.05	NOISE 0.1	NOISE 0.15	NOISE 0.2	NOISE 0.4	NOISE 0.6
DIOS	RMSE	.114708 ±.002194	.137553 ±.001855	.162571 ±.005927	**.186799 ±.008194**	**.211346 ±.007440**	**.277133 ±.002640**	**.332875 ±.004374**
	ACC	**87.48% ±0.25%**	**86.29% ±0.54%**	**85.25% ±0.70%**	**83.29% ±0.97%**	**81.90% ±1.09%**	**75.78% ±1.75%**	**67.55% ±1.12%**
NONE	RMSE ≡	**.000000 ±.000000** ○	**.111065 ±.003219** ○	**.157520 ±.002874** ○	.193922 ±.001858 •	.223263 ±.001812 •	.311895 ±.003253 •	.383534 ±.003808 •
	ACC •	84.06% ±0.00% •	83.80% ±0.73% •	82.22% ±1.02% •	80.88% ±1.38% •	78.78% ±1.73% •	71.13% ±1.63% •	63.46% ±2.27% •
SFIL	RMSE	- -	- -	- -	- -	- -	- -	- -
	ACC •	85.33% ±0.67% •	84.10% ±0.58% •	82.78% ±1.25% •	81.03% ±1.96% •	79.33% ±1.51% •	72.93% ±0.94% •	66.19% ±1.94% ≡
SPOL	RMSE •	.130674 ±.000920 •	.168412 ±.002665 •	.199776 ±.002339 •	.226227 ±.002091 •	.249161 ±.002513 •	.320054 ±.003182 •	.378386 ±.004059 •
	ACC •	85.12% ±0.60% •	83.86% ±0.71% •	82.75% ±1.06% •	81.23% ±1.39% •	79.58% ±1.39% •	73.84% ±1.16% •	66.96% ±1.56% ≡
PFIL	RMSE	- -	- -	- -	- -	- -	- -	- -
	ACC •	84.75% ±0.61% •	83.46% ±0.76% •	82.07% ±0.84% •	80.29% ±1.05% •	78.43% ±1.64% •	71.25% ±1.55% •	62.78% ±1.27% •
PPOL	RMSE ≡	.103064 ±.000793 ○	.156244 ±.002162 •	.187602 ±.003251 •	.215673 ±.001799 •	.238741 ±.001937 •	.313380 ±.003438 •	.376681 ±.003125 •
	ACC •	84.87% ±0.66% •	83.77% ±0.63% •	82.03% ±0.55% •	80.41% ±1.49% •	78.49% ±1.43% •	70.90% ±1.12% •	62.28% ±1.24% •

DIOS is: • significantly better, ≡ equivalent, ○ significantly worse, p-value: 0.05

Table A.8: Experimental results on a correction task on the benchmark dataset STATLOG with erroneous values rates 0/5/10/15/20/40/60%.

Model	Metric	NOISE 0	NOISE 0.05	NOISE 0.1	NOISE 0.15	NOISE 0.2	NOISE 0.4	NOISE 0.6
DIOS	RMSE	.058886 ±.003538	**.091930 ±.000620**	**.114757 ±.000889**	**.131610 ±.001119**	**.146440 ±.001224**	**.206205 ±.001301**	**.261981 ±.001116**
	ACC	**98.29% ±0.03%**	**98.25% ±0.05%**	**98.14% ±0.07%**	**98.08% ±0.09%**	**97.95% ±0.09%**	**96.35% ±0.17%**	**86.87% ±0.91%**
NONE	RMSE •	**.000000 ±.000000** ○	.098293 ±.000295 •	.138905 ±.000343 •	.170232 ±.000340 •	.196622 ±.000310 •	.278053 ±.000229 •	.340514 ±.000240 •
	ACC •	98.15% ±0.00% •	97.99% ±0.10% •	97.83% ±0.14% •	97.86% ±0.14% •	97.59% ±0.15% •	96.30% ±0.26% ≡	85.75% ±0.43% •
SFIL	RMSE	- -	- -	- -	- -	- -	- -	- -
	ACC •	97.81% ±0.13% •	97.84% ±0.24% •	97.67% ±0.17% •	97.67% ±0.12% •	97.41% ±0.10% •	95.91% ±0.24% •	85.26% ±0.58% •
SPOL	RMSE •	.018439 ±.000342 ○	.100045 ±.000308 •	.139794 ±.000335 •	.170642 ±.000328 •	.196761 ±.000310 •	.276997 ±.000216 •	.334571 ±.000260 •
	ACC •	97.97% ±0.10% •	97.98% ±0.22% •	97.92% ±0.16% •	97.80% ±0.11% •	96.82% ±0.35% •	40.41% ±3.04% •	54.22% ±1.61% •
PFIL	RMSE	- -	- -	- -	- -	- -	- -	- -
	ACC	- -	- -	- -	- -	- -	- -	- -
PPOL	RMSE	- -	- -	- -	- -	- -	- -	- -
	ACC	- -	- -	- -	- -	- -	- -	- -

DIOS is: • significantly better, ≡ equivalent, ○ significantly worse, p-value: 0.05

Table A.9: Experimental results on a correction task on the benchmark dataset MFEAT with erroneous values rates 0/5/10/15/20/40/60%.

Model	Metric	NOISE 0	NOISE 0.05	NOISE 0.1	NOISE 0.15	NOISE 0.2	NOISE 0.4	NOISE 0.6
DIOS	RMSE	.072061 ±.001482	.097387 ±.001040	**.118795 ±.001724**	**.134184 ±.001355**	**.148178 ±.001044**	**.191014 ±.000643**	**.232878 ±.001250**
	ACC	**91.22% ±0.28%**	**91.12% ±0.28%**	**91.10% ±0.23%**	**91.05% ±0.33%**	**90.85% ±0.44%**	**89.72% ±0.61%**	**83.65% ±0.87%**
NONE	RMSE ≡	**.000000 ±.000000** ○	**.093711 ±.000108** ○	.132491 ±.000115 •	.162272 ±.000090 •	.187342 ±.000096 •	.264980 ±.000093 •	.324528 ±.000107 •
	ACC •	90.00% ±0.00% •	90.00% ±0.46% •	90.33% ±0.51% •	90.15% ±0.28% •	90.45% ±0.57% ≡	88.12% ±0.85% •	71.80% ±1.63% •
SFIL	RMSE	- -	- -	- -	- -	- -	- -	- -
	ACC •	78.65% ±1.09% •	78.62% ±1.65% •	78.47% ±1.21% •	78.95% ±1.33% •	78.15% ±1.56% •	74.08% ±1.80% •	51.75% ±2.27% •
SPOL	RMSE	- -	- -	- -	- -	- -	- -	- -
	ACC	- -	- -	- -	- -	- -	- -	- -
PFIL	RMSE	- -	- -	- -	- -	- -	- -	- -
	ACC	- -	- -	- -	- -	- -	- -	- -
PPOL	RMSE	- -	- -	- -	- -	- -	- -	- -
	ACC	- -	- -	- -	- -	- -	- -	- -

DIOS is: • significantly better, ≡ equivalent, ○ significantly worse, p-value: 0.05

Table A.10: Experimental results on a correction task on the benchmark dataset ORL with erroneous values rates 0/5/10/15/20/40/60%.

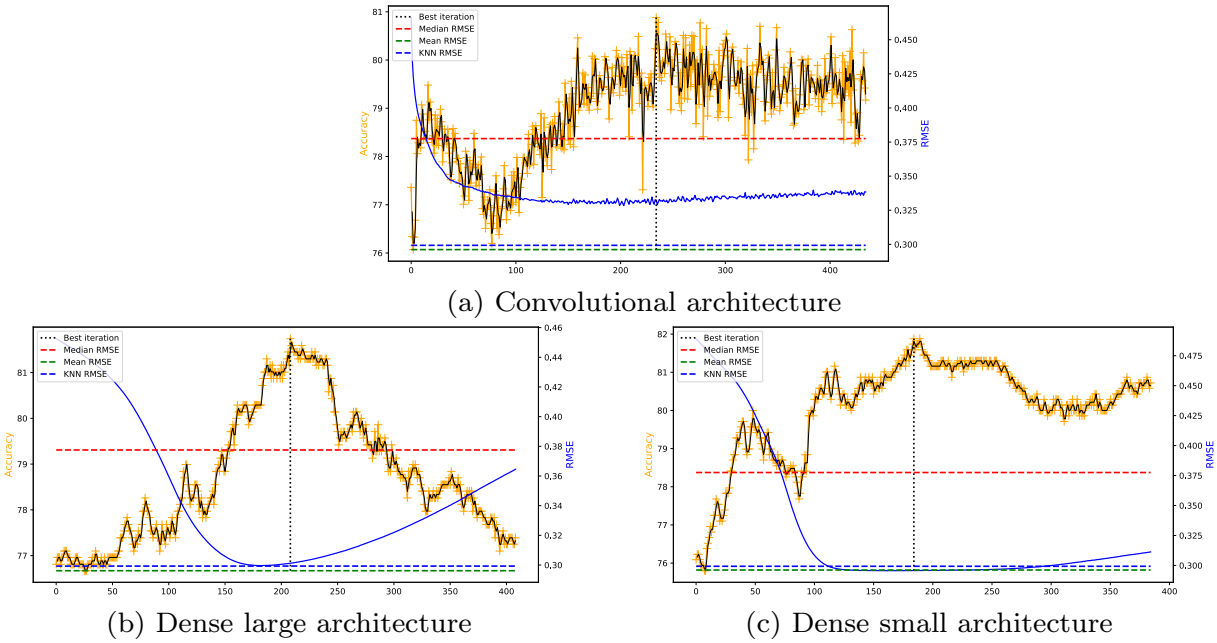
Model	Metric	NOISE 0	NOISE 0.05	NOISE 0.1	NOISE 0.15	NOISE 0.2	NOISE 0.4	NOISE 0.6
DIOS	RMSE	.031102 ±.005485	.095970 ±.002590	**.129322 ±.003464**	**.145087 ±.004204**	**.156343 ±.004610**	**.191861 ±.003404**	**.226214 ±.007582**
	ACC	**76.04% ±0.35%**	**74.57% ±0.84%**	**73.58% ±1.01%**	**73.07% ±0.73%**	**71.68% ±0.85%**	**67.92% ±0.72%**	**65.44% ±1.17%**
NONE	RMSE •	**.000000 ±.000000** ∘	**.095012 ±.003124** ≡	.134108 ±.003882 •	.164000 ±.004345 •	.189278 ±.004217 •	.266476 ±.002477 •	.325244 ±.002492 •
	ACC •	73.05% ±0.00% •	71.31% ±1.02% •	70.14% ±1.52% •	69.78% ±1.43% •	68.70% ±1.53% •	64.53% ±1.38% •	61.84% ±1.47% •
SFIL	RMSE	- -	- -	- -	- -	- -	- -	- -
	ACC •	73.58% ±0.90% •	72.42% ±0.98% •	71.45% ±1.29% •	70.93% ±1.17% •	69.45% ±1.07% •	66.76% ±0.96% •	64.73% ±1.19% ≡
SPOL	RMSE •	.094054 ±.002142 •	.131477 ±.003677 •	.155997 ±.003554 •	.178815 ±.003774 •	.198886 ±.003205 •	.261991 ±.003733 •	.309920 ±.003596 •
	ACC •	75.16% ±0.68% •	73.59% ±1.46% ≡	72.25% ±1.26% •	71.43% ±1.35% •	69.13% ±1.50% •	66.05% ±0.96% •	62.54% ±1.49% •
PFIL	RMSE	- -	- -	- -	- -	- -	- -	- -
	ACC •	71.49% ±0.63% •	71.17% ±1.09% •	69.74% ±1.73% •	69.79% ±1.57% •	68.61% ±1.09% •	64.23% ±1.46% •	62.24% ±1.18% •
PPOL	RMSE •	.084519 ±.000933 •	.116449 ±.002618 •	.143318 ±.003361 •	.165470 ±.003963 •	.184012 ±.003601 •	.249572 ±.002032 •	.304141 ±.002363 •
	ACC •	72.44% ±1.09% •	70.79% ±1.42% •	70.00% ±1.27% •	69.48% ±0.91% •	67.84% ±1.33% •	64.15% ±1.60% •	61.64% ±1.28% •

DIOS is: • significantly better, ≡ equivalent, ∘ significantly worse, p-value: 0.05

Table A.11: Experimental results on a correction task on the benchmark dataset PIMA with erroneous values rates 0/5/10/15/20/40/60%.

A.3 Technical Details

During our tuning experiments to design the best possible architecture for DIOS, we found that for datasets containing fewer features, using fully-connected architectures worked best. We found that convolutional architectures tend to yield very unstable results on datasets with few features compared to fully-connected architectures. Figure A.1 shows a comparison of the training phase of one convolutional and two fully-connected architectures on the STATLOG dataset containing 14 features. The convolutional architecture training is shown in (a), (b) is the large fully-connected architecture (dimensions: 64, 128, 32, 14), and (c) is the small one (dimensions: 5, 10, 14). In both fully-connected cases, the input is set as the original corrupted data $\tilde{X}$. While it takes essentially the same time to converge to satisfactory results for both fully-connected architectures, a smaller architecture often leads to better and more stable results, making it easier to find a good stopping point. We can see in this example that the bigger fully-connected architecture obtains slightly worse results than the smaller one in terms of Root-Mean-Square Error (RMSE), while the convolutional architecture obtains very bad results. In both fully-connected cases, the model ends up overfitting after a point but the smaller architecture takes more iterations to do so. This makes it easier to stop training at the right time, maximizing the results for both the RMSE and Accuracy metrics.



(a) Convolutional architecture

(b) Dense large architecture

(c) Dense small architecture

Figure A.1: Comparison between a convolutional (a), a large (b) and a small (c) fully-connected architecture on the STATLOG dataset in MCAR 25% setting. The x axis is the number of iterations (epochs), the yellow curve is the Accuracy evolution on the supervised training set, the blue curve is the RMSE, dashed lines are baseline RMSEs, and the dotted vertical line is the iteration at which we rolled back once the early-stop took place.

To obtain the best possible results it is important to stop the training before overfitting on the training data starts. To stop at the right time, we implemented an early-stopping

procedure. This stopping condition is defined as a degradation of the loss, accuracy, or another metric defined by the user, for a set number of iterations. Once the training process is stopped, we can simply rollback to the iteration that yielded the best result depending on the chosen metric. In figure A.1 we can see the iteration at which we rolled back to in each case, signified as a dotted line. In this case, the selection of the best imputation was based on the obtained accuracy on the supervised training set. In the case of the convolutional architecture the results are very unstable, resulting in an imputation of bad quality. However, on both fully-connected architectures we can see that the maximum accuracy is obtained roughly at the same step as the lowest RMSE. This shows that in a, at least partially, supervised setting, using the best obtained accuracy as a stopping point leads to very satisfactory results and often almost to the best possible results.

Depending on the dimensions of the dataset to correct, it is important to define an adapted architecture. As described earlier, we used either convolutional or fully-connected architectures depending on the amount of features in the dataset. For the STATLOG dataset we obtained the best results by using a fully-connected architecture with 3 layers of dimensions 5, 10, and 14. For the PIMA dataset we found best results using a fully-connected architecture with 3 layers of dimensions 8, 16, and 8. For the COVID dataset a small fully-connected architecture with 2 layers of dimensions 80 was used. We used the exact same convolutional architecture for ARRHYTHMIA, NHANES, MYOCARDIAL and MFEAT datasets, the only difference being that we added more filters at each layer for the MFEAT dataset (which contains more features). In those four cases, the architecture contained 4 layers and was based on the generic convolutional architecture showed in figure A.2. For the ORL dataset containing 14150 features, we used a deeper version of the convolutional architecture by adding 3 additional hidden layers to capture deeper knowledge about the data. While it is important to define a good architecture to obtain optimal results, we found that very few changes to our generic architecture were needed to obtain the best possible results on any dataset.

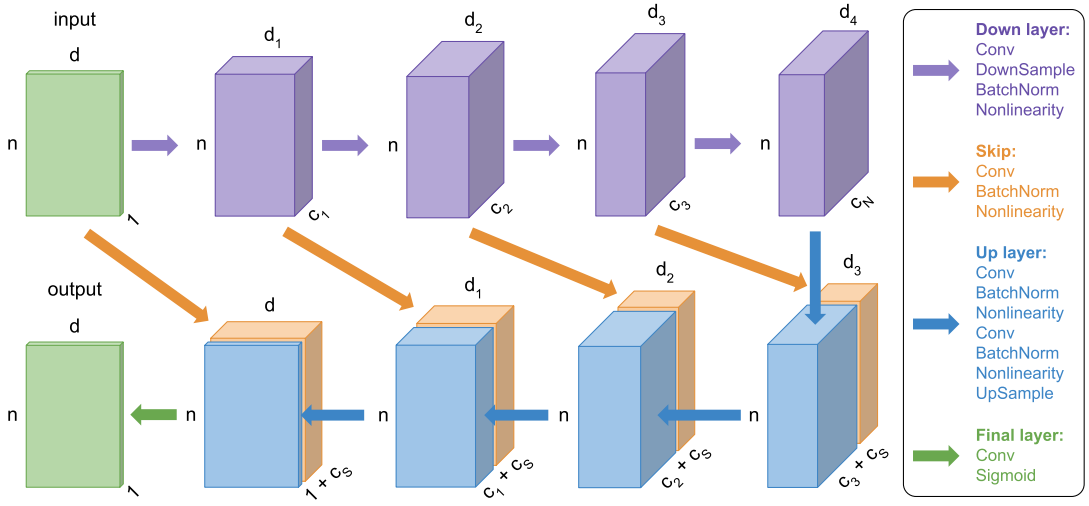


Figure A.2: Generic convolutional encoder-decoder architecture with skip connections.

As can be seen in Figure A.2, the generic convolutional architecture we defined is organized as a U-Net, in an encoder-decoder manner: with down layers followed by up layers,

and skip connections linking encoder with decoder layers so that higher and lower levels of feature extractions can be merged and exploited together during the decoding phase. In our architecture we used 1D convolutions, which allow for feature reduction, and thus, extraction of knowledge and abstractions from the features, while keeping the initial element dimension and organization from the original dataset. We reverse the process using up-sampling layers to return to the original dataset dimensions. Skip connections make it simpler for the model to keep track of the initial features without losing information, while dimension reduction captures deeper knowledge. Extracted knowledge is then used in combination with skip connections outputs to obtain corrected values, and impute values that are missing depending on known values.

As can be seen in Figure A.2, down layers are composed of a convolutional layer, a down-sample step, a batch-normalization, and a non-linearity. In our case, we used the natural down-sample of convolutional layers by setting a stride higher than 1. We found that this simple mechanism gave better results than using a separate down-sample step such as an average or max pooling. We found that using a batch-normalization layer at each level in our architecture allowed us to reach more stable and more consistent results. As for non-linearity, we used LeakyReLU, which gave better results than ReLU or other activation functions. We set all our paddings as zero paddings. Skip connections are based on the same model as down layers, except that they do not perform any down-sampling as we want to keep the same dimensions as the input. Up layers take the concatenation on the channel dimension of the previous layer and the corresponding skip connection as inputs. The up-sampling step is performed at the end of the layer: we used the up-sample nearest neighbor mode. Finally, last layer gives the model output: this layer is composed of a convolutional layer followed by a Sigmoid. Sigmoids are often used in the last reconstruction layer of encoder-decoders to yield stable and consistent results. The number of channels in the encoder part $(c_1, ..., c_N)$ can expand at each level to capture deeper and more abstract knowledge about the data. We found that setting a fixed number of channels for the skip-connections gave the best results, usually between 4 to 8 channels. During our experiments, we noted that to obtain the best possible results it seems important to minimize the dimension at the bottleneck part of the architecture so that the filters at this particular layer cover the entirety of the feature map. For example, in the case of an architecture with 4 layers such as in figure A.2 we want $d_3 = d_4$ with filters of dimension d_3. In that way, we make sure that the whole feature map is covered, which allows knowledge to be extracted and shared between features that are not in a close neighborhood.

As discussed earlier, we set the net input Z as pure random noise when we use a convolutional architecture, and we set $Z = \tilde{X}$, where $\tilde{X}$ is the original corrupted data, when we use a fully-connected architecture. In any case, we used a noise-based regularization that we apply to the net input Z at each training iteration. Thus, Z is slightly modified from its original value at each iteration, which prevents the model from overfitting or getting stuck during its learning phase. We used the Adam optimizer to train our models, all our experiments were performed using the PyTorch deep learning library.

We used deep neural networks, either fully-connected or convolutional, for our experiments. Training such models requires a large number of matrix multiplication operations:

Graphics Processing Units (GPUs) are better suited to execute these operations than Central Processing Units (CPUs). We ran all our experiments on a Tesla V100 PCIE 32GB, which allowed us to drastically speed-up our training times, thus, enabling us to run all our experiments for 10 runs in each setting.

Using the GPU described above, we were able to train our method for any used dataset in a relatively short amount of time. Running DIOS on datasets with few features such as the STATLOG dataset takes a few seconds to minutes at most, and the training phase on the ORL dataset containing 14150 features could take up to 1 hour depending on chosen parameters. Considering the fact that DIOS is a preprocessing method that will usually be used only once before predictive methods, those training times seem perfectly acceptable for most real-world applications.

Appendix B

S-HOT and M-HOT results

Contents

B.1 Experimental Results on Benchmark Datasets

Dataset	Pattern		*SI*		*S-HOT*		*MI*		*M-HOT*	
IRIS	MCAR	10%	0.9972681	(4)	**0.9973865**	(3)	0.9976699	(2)	**0.9976913**	(1)
		15%	0.9942281	(4)	**0.9944422**	(3)	0.9946880	(2)	**0.9947552**	(1)
		25%	0.9835323	(4)	**0.9839692**	(3)	0.9841479	(2)	**0.9843596**	(1)
	MAR	10%	**0.9975236**	(3)	0.9974886	(4)	0.9978336	(2)	**0.9978845**	(1)
		15%	**0.9970188**	(3)	0.9970006	(4)	**0.9973150**	(1)	0.9973097	(2)
		25%	0.9941960	(4)	**0.9943554**	(3)	0.9943643	(2)	**0.9943890**	(1)
	MNAR	10%	0.9972209	(4)	**0.9972343**	(3)	**0.9973548**	(1)	0.9973011	(2)
		15%	0.9937917	(4)	**0.9938451**	(3)	0.9941570	(2)	**0.9941622**	(1)
		25%	0.9891964	(4)	**0.9902221**	(3)	0.9905233	(2)	**0.9907251**	(1)
STAT	MCAR	10%	0.9105582	(4)	**0.9106386**	(3)	**0.9132492**	(1)	0.9132475	(2)
		15%	0.9060337	(4)	**0.9066862**	(3)	0.9091019	(2)	**0.9091584**	(1)
		25%	0.9047888	(4)	**0.9059597**	(3)	0.9077214	(2)	**0.9078041**	(1)
	MAR	10%	**0.9127799**	(3)	0.9127328	(4)	**0.9144097**	(1)	0.9142985	(2)
		15%	**0.9079620**	(3)	0.9078667	(4)	**0.9093156**	(1)	0.9092031	(2)
		25%	0.8959471	(4)	**0.8968849**	(3)	0.8990894	(2)	**0.8992881**	(1)
	MNAR	10%	0.9070192	(4)	**0.9071004**	(3)	0.9095510	(2)	**0.9095576**	(1)
		15%	0.9040681	(4)	**0.9042151**	(3)	**0.9061872**	(1)	0.9060699	(2)
		25%	0.8951658	(4)	**0.8967363**	(3)	0.8992252	(2)	**0.8992736**	(1)
WINE	MCAR	10%	0.9987736	(4)	**0.9988008**	(3)	0.9989672	(2)	**0.9989811**	(1)
		15%	0.9955454	(4)	**0.9956407**	(3)	0.9960062	(2)	**0.9960307**	(1)
		25%	0.9910498	(4)	**0.9914148**	(3)	0.9919927	(2)	**0.9923485**	(1)
	MAR	10%	0.9961058	(4)	**0.9961142**	(3)	0.9965056	(2)	**0.9965060**	(1)
		15%	0.9977720	(4)	**0.9978677**	(3)	**0.9982010**	(1)	0.9981809	(2)
		25%	0.9952116	(4)	**0.9959576**	(3)	0.9965157	(2)	**0.9968702**	(1)
	MNAR	10%	**0.9987205**	(3)	0.9987058	(4)	**0.9988244**	(1)	0.9988131	(2)
		15%	0.9974746	(4)	**0.9974850**	(3)	0.9976465	(2)	**0.9976683**	(1)
		25%	0.9808498	(4)	**0.9822719**	(3)	0.9841291	(2)	**0.9844587**	(1)
PIMA	MCAR	10%	0.8193054	(4)	**0.8196586**	(3)	**0.8211340**	(1)	0.8211193	(2)
		15%	0.8073739	(4)	**0.8078378**	(3)	0.8095505	(2)	**0.8095824**	(1)
		25%	0.8029002	(4)	**0.8043589**	(3)	0.8060367	(2)	**0.8065611**	(1)
	MAR	10%	0.8238900	(4)	**0.8242472**	(3)	**0.8257089**	(1)	0.8256414	(2)
		15%	0.8045918	(4)	**0.8061503**	(3)	0.8080830	(2)	**0.8083194**	(1)
		25%	0.8017568	(4)	**0.8025144**	(3)	**0.8041203**	(1)	0.8040062	(2)
	MNAR	10%	0.8280685	(4)	**0.8284115**	(3)	0.8302729	(2)	**0.8303076**	(1)
		15%	0.8279577	(4)	**0.8283792**	(3)	0.8298929	(2)	**0.8300464**	(1)
		25%	0.8005008	(4)	**0.8021749**	(3)	0.8043448	(2)	**0.8047250**	(1)
ABAL	MCAR	10%	0.8737739	(4)	**0.8740180**	(3)	0.8748059	(2)	**0.8749393**	(1)
		15%	0.8714861	(4)	**0.8717831**	(3)	0.8725539	(2)	**0.8726250**	(1)
		25%	0.8663833	(4)	**0.8666186**	(3)	0.8674332	(2)	**0.8675645**	(1)
	MAR	10%	0.8742551	(4)	**0.8743399**	(3)	0.8751972	(2)	**0.8753671**	(1)
		15%	0.8720722	(4)	**0.8721856**	(3)	0.8731502	(2)	**0.8731739**	(1)
		25%	0.8697060	(4)	**0.8699619**	(3)	0.8708046	(2)	**0.8709102**	(1)
	MNAR	10%	0.8760444	(4)	**0.8760447**	(3)	0.8768033	(2)	**0.8769350**	(1)
		15%	0.8753217	(4)	**0.8754605**	(3)	0.8763271	(2)	**0.8764456**	(1)
		25%	0.8683507	(4)	**0.8690281**	(3)	0.8696780	(2)	**0.8697714**	(1)
Average rank			3.8889		3.1111		1.7556		1.2444	

Table B.1: Area Under the Curve (AUC) results when applying imputation frameworks Single-Imputation (SI), Multiple-Imputation (MI), Single-Hotpatching (S-HOT) and Multiple-Hotpatching (M-HOT) using the MISSFOREST imputation method on benchmark datasets. Bold values are the best results between pairs of frameworks SI vs S-HOT, and MI vs M-HOT.

Dataset	Pattern		*SI*		*S-HOT*		*MI*		*M-HOT*	
IRIS	MCAR	10%	0.9799937	(4)	**0.9817654**	(3)	0.9833994	(2)	**0.9840372**	(1)
		15%	0.9768163	(4)	**0.9808358**	(3)	0.9823092	(2)	**0.9831680**	(1)
		25%	0.9669967	(4)	**0.9722641**	(3)	0.9761056	(2)	**0.9770136**	(1)
	MAR	10%	0.9957063	(4)	**0.9961878**	(3)	0.9965493	(2)	**0.9966640**	(1)
		15%	0.9942239	(4)	**0.9952689**	(3)	0.9960484	(2)	**0.9961207**	(1)
		25%	0.9893513	(4)	**0.9910596**	(3)	0.9915209	(2)	**0.9920205**	(1)
	MNAR	10%	0.9936567	(4)	**0.9950805**	(3)	0.9956014	(2)	**0.9956776**	(1)
		15%	0.9724648	(4)	**0.9761291**	(3)	0.9802625	(2)	**0.9819184**	(1)
		25%	0.9412312	(4)	**0.9464214**	(3)	0.9571874	(2)	**0.9600941**	(1)
STAT	MCAR	10%	0.9080227	(4)	**0.9102585**	(3)	0.9134026	(2)	**0.9137060**	(1)
		15%	0.8998590	(4)	**0.9028497**	(3)	0.9064267	(2)	**0.9067930**	(1)
		25%	0.8908900	(4)	**0.8956605**	(3)	0.8980645	(2)	**0.8989371**	(1)
	MAR	10%	0.9084245	(4)	**0.9090188**	(3)	**0.9112693**	(1)	0.9111061	(2)
		15%	0.9051054	(4)	**0.9063680**	(3)	0.9078594	(2)	**0.9078908**	(1)
		25%	0.8991662	(4)	**0.9027334**	(3)	0.9058215	(2)	**0.9061238**	(1)
	MNAR	10%	0.9030340	(4)	**0.9040986**	(3)	0.9067615	(2)	**0.9071034**	(1)
		15%	0.9035329	(4)	**0.9052432**	(3)	0.9079736	(2)	**0.9082962**	(1)
		25%	0.8905727	(4)	**0.8944715**	(3)	0.8966984	(2)	**0.8972140**	(1)
WINE	MCAR	10%	0.9981833	(4)	**0.9983207**	(3)	0.9985619	(2)	**0.9985971**	(1)
		15%	0.9962062	(4)	**0.9971960**	(3)	0.9976257	(2)	**0.9976910**	(1)
		25%	0.9904914	(4)	**0.9934990**	(3)	**0.9943642**	(1)	0.9942573	(2)
	MAR	10%	0.9943647	(4)	**0.9948008**	(3)	0.9955126	(2)	**0.9957826**	(1)
		15%	0.9966424	(4)	**0.9970709**	(3)	**0.9974876**	(1)	0.9974503	(2)
		25%	0.9937316	(4)	**0.9952213**	(3)	0.9956761	(2)	**0.9961109**	(1)
	MNAR	10%	0.9986522	(4)	**0.9989077**	(3)	**0.9989886**	(1)	0.9989613	(2)
		15%	0.9970764	(4)	**0.9974153**	(3)	0.9976771	(2)	**0.9976784**	(1)
		25%	0.9868731	(4)	**0.9913767**	(3)	**0.9929763**	(1)	0.9929660	(2)
PIMA	MCAR	10%	0.8064355	(4)	**0.8108632**	(3)	0.8126446	(2)	**0.8147203**	(1)
		15%	0.7905122	(4)	**0.7956762**	(3)	0.7985624	(2)	**0.8004567**	(1)
		25%	0.7636735	(4)	**0.7701991**	(3)	0.7740310	(2)	**0.7771393**	(1)
	MAR	10%	0.8059250	(4)	**0.8086973**	(3)	0.8113151	(2)	**0.8127776**	(1)
		15%	0.7872220	(4)	**0.7918551**	(3)	0.7931241	(2)	**0.7958718**	(1)
		25%	0.7785481	(4)	**0.7832708**	(3)	0.7866454	(2)	**0.7903447**	(1)
	MNAR	10%	0.8172368	(4)	**0.8209724**	(3)	0.8234770	(2)	**0.8245369**	(1)
		15%	0.8045201	(4)	**0.8093813**	(3)	0.8113161	(2)	**0.8130785**	(1)
		25%	0.7636845	(4)	**0.7714932**	(3)	0.7734528	(2)	**0.7806867**	(1)
ABAL	MCAR	10%	0.8640126	(4)	**0.8656192**	(3)	0.8664509	(2)	**0.8665868**	(1)
		15%	0.8574107	(4)	**0.8602595**	(3)	0.8606084	(2)	**0.8607058**	(1)
		25%	0.8414577	(4)	**0.8459436**	(3)	0.8464660	(2)	**0.8465174**	(1)
	MAR	10%	0.8655842	(4)	**0.8664084**	(3)	0.8670096	(2)	**0.8670916**	(1)
		15%	0.8589554	(4)	**0.8604022**	(3)	0.8613812	(2)	**0.8614189**	(1)
		25%	0.8487023	(4)	**0.8521187**	(3)	**0.8532367**	(1)	0.8530732	(2)
	MNAR	10%	0.8646802	(4)	**0.8657062**	(3)	0.8670122	(2)	**0.8670726**	(1)
		15%	0.8592630	(4)	**0.8613171**	(3)	0.8625236	(2)	**0.8625561**	(1)
		25%	0.8424408	(4)	**0.8449781**	(3)	**0.8481797**	(1)	0.8481437	(2)
Average rank			4.0000		3.0000		1.8444		1.1556	

Table B.2: AUC results when applying imputation frameworks SI, MI, S-HOT and M-HOT using the SOFTIMPUTE imputation method on benchmark datasets. Bold values are the best results between pairs of frameworks SI vs S-HOT, and MI vs M-HOT.

Dataset	Pattern		*SI*		*S-HOT*		*MI*		*M-HOT*	
IRIS	MCAR	10%	0.9857773	(4)	**0.9895060**	(3)	**0.9925170**	(1)	0.9924744	(2)
		15%	0.9875209	(4)	**0.9899045**	(3)	0.9909667	(2)	**0.9916338**	(1)
		25%	0.9836152	(4)	**0.9869794**	(3)	0.9881549	(2)	**0.9888826**	(1)
	MAR	10%	0.9964254	(4)	**0.9972233**	(3)	0.9977066	(2)	**0.9977583**	(1)
		15%	0.9959845	(4)	**0.9967155**	(3)	0.9971628	(2)	**0.9973015**	(1)
		25%	0.9941741	(4)	**0.9948820**	(3)	0.9949109	(2)	**0.9952507**	(1)
	MNAR	10%	0.9931132	(4)	**0.9951754**	(3)	0.9960869	(2)	**0.9961488**	(1)
		15%	0.9882231	(4)	**0.9913868**	(3)	0.9923640	(2)	**0.9925256**	(1)
		25%	0.9751727	(4)	**0.9801958**	(3)	0.9824279	(2)	**0.9839101**	(1)
STAT	MCAR	10%	0.9020429	(4)	**0.9123281**	(3)	0.9160128	(2)	**0.9161932**	(1)
		15%	0.8856269	(4)	**0.9045924**	(3)	0.9079333	(2)	**0.9081124**	(1)
		25%	0.8769033	(4)	**0.8993412**	(3)	0.9020794	(2)	**0.9025791**	(1)
	MAR	10%	0.9087615	(4)	**0.9145394**	(3)	0.9173822	(2)	**0.9180998**	(1)
		15%	0.8998852	(4)	**0.9082186**	(3)	**0.9114664**	(1)	0.9108804	(2)
		25%	0.8807372	(4)	**0.8956080**	(3)	**0.8983544**	(1)	0.8977701	(2)
	MNAR	10%	0.8935386	(4)	**0.9006735**	(3)	0.9037203	(2)	**0.9044389**	(1)
		15%	0.8939280	(4)	**0.9030125**	(3)	0.9062339	(2)	**0.9063437**	(1)
		25%	0.8601279	(4)	**0.8868338**	(3)	0.8908618	(2)	**0.8913759**	(1)
WINE	MCAR	10%	0.9983811	(4)	**0.9986075**	(3)	**0.9987367**	(1)	0.9987216	(2)
		15%	0.9966068	(4)	**0.9976643**	(2)	0.9976465	(3)	**0.9979061**	(1)
		25%	0.9915570	(4)	**0.9946475**	(3)	0.9947619	(2)	**0.9951303**	(1)
	MAR	10%	0.9965180	(4)	**0.9975478**	(3)	**0.9977893**	(1)	0.9977813	(2)
		15%	0.9970171	(4)	**0.9975442**	(3)	0.9978439	(2)	**0.9978859**	(1)
		25%	0.9962614	(4)	**0.9975461**	(3)	0.9977948	(2)	**0.9979197**	(1)
	MNAR	10%	0.9983833	(4)	**0.9986169**	(3)	0.9987648	(2)	**0.9987725**	(1)
		15%	0.9975738	(4)	**0.9977293**	(3)	0.9979271	(2)	**0.9979903**	(1)
		25%	0.9910734	(4)	**0.9929808**	(3)	0.9934225	(2)	**0.9936716**	(1)
PIMA	MCAR	10%	0.8127648	(4)	**0.8193582**	(3)	0.8216261	(2)	**0.8223632**	(1)
		15%	0.8058630	(4)	**0.8116789**	(3)	0.8131427	(2)	**0.8132622**	(1)
		25%	0.8003100	(4)	**0.8066228**	(3)	**0.8081799**	(1)	0.8080758	(2)
	MAR	10%	0.8155870	(4)	**0.8189119**	(3)	**0.8213601**	(1)	0.8208527	(2)
		15%	0.8036819	(4)	**0.8082194**	(3)	0.8095054	(2)	**0.8097639**	(1)
		25%	0.7948673	(4)	**0.8006539**	(3)	**0.8024029**	(1)	0.8016825	(2)
	MNAR	10%	0.8203723	(4)	**0.8231400**	(3)	**0.8253893**	(1)	0.8252170	(2)
		15%	0.8199156	(4)	**0.8240971**	(3)	**0.8259666**	(1)	0.8256314	(2)
		25%	0.7939407	(4)	**0.7999276**	(3)	**0.8020002**	(1)	0.8018077	(2)
ABAL	MCAR	10%	0.8648877	(4)	**0.8670693**	(3)	0.8674506	(2)	**0.8674665**	(1)
		15%	0.8629564	(4)	**0.8653915**	(3)	**0.8658764**	(1)	0.8658450	(2)
		25%	0.8567035	(4)	**0.8608724**	(1)	**0.8607069**	(2)	0.8606392	(3)
	MAR	10%	0.8657632	(4)	**0.8673467**	(3)	**0.8679247**	(1)	0.8678653	(2)
		15%	0.8644014	(4)	**0.8662844**	(3)	**0.8666815**	(1)	0.8666551	(2)
		25%	0.8593164	(4)	**0.8617823**	(3)	**0.8621055**	(1)	0.8619164	(2)
	MNAR	10%	0.8678301	(4)	**0.8695650**	(3)	0.8699069	(2)	**0.8699575**	(1)
		15%	0.8660991	(4)	**0.8684580**	(3)	**0.8690043**	(1)	0.8689099	(2)
		25%	0.8569403	(4)	**0.8595820**	(3)	**0.8600201**	(1)	0.8599013	(2)
Average rank			4.0000		2.9333		1.6444		1.4222	

Table B.3: AUC results when applying imputation frameworks SI, MI, S-HOT and M-HOT using the GAIN imputation method on benchmark datasets. Bold values are the best results between pairs of frameworks SI vs S-HOT, and MI vs M-HOT.

Dataset	Pattern		*SI*		*S-HOT*		*MI*		*M-HOT*	
IRIS	MCAR	10%	**0.9554177**	(3)	0.9554087	(4)	0.9580418	(2)	**0.9589565**	(1)
		15%	0.9530381	(4)	**0.9537060**	(3)	0.9550508	(2)	**0.9556869**	(1)
		25%	0.9453838	(4)	**0.9467725**	(3)	0.9483980	(2)	**0.9495824**	(1)
	MAR	10%	0.9905962	(4)	**0.9908503**	(3)	0.9912635	(2)	**0.9913792**	(1)
		15%	0.9895670	(4)	**0.9901444**	(3)	0.9906208	(2)	**0.9911103**	(1)
		25%	0.9852364	(4)	**0.9855719**	(3)	0.9863382	(2)	**0.9874122**	(1)
	MNAR	10%	0.9845493	(4)	**0.9847597**	(3)	0.9857842	(2)	**0.9862155**	(1)
		15%	0.9555301	(4)	**0.9561772**	(3)	0.9571348	(2)	**0.9576568**	(1)
		25%	0.9308975	(4)	**0.9339321**	(3)	0.9372261	(2)	**0.9381131**	(1)
STAT	MCAR	10%	0.9077575	(4)	**0.9082601**	(3)	**0.9109495**	(1)	0.9108722	(2)
		15%	0.9007673	(4)	**0.9010883**	(3)	**0.9037409**	(1)	0.9035844	(2)
		25%	0.8931136	(4)	**0.8936269**	(3)	**0.8962443**	(1)	0.8960913	(2)
	MAR	10%	**0.9077300**	(3)	0.9076973	(4)	**0.9099291**	(1)	0.9097665	(2)
		15%	0.9034708	(4)	**0.9036366**	(3)	**0.9054812**	(1)	0.9053064	(2)
		25%	0.8998750	(4)	**0.9000970**	(3)	**0.9030175**	(1)	0.9029715	(2)
	MNAR	10%	0.9049275	(4)	**0.9052633**	(3)	**0.9073643**	(1)	0.9073239	(2)
		15%	0.9028127	(4)	**0.9029637**	(3)	0.9057320	(2)	**0.9058654**	(1)
		25%	0.8902191	(4)	**0.8907920**	(3)	**0.8927232**	(1)	0.8926771	(2)
WINE	MCAR	10%	0.9960665	(4)	**0.9964714**	(3)	0.9966659	(2)	**0.9967312**	(1)
		15%	0.9920738	(4)	**0.9927122**	(3)	0.9935617	(2)	**0.9935720**	(1)
		25%	0.9851423	(4)	**0.9866744**	(3)	**0.9874086**	(1)	0.9872919	(2)
	MAR	10%	0.9956389	(4)	**0.9962256**	(3)	0.9965696	(2)	**0.9965773**	(1)
		15%	0.9918686	(4)	**0.9925073**	(3)	0.9929713	(2)	**0.9931399**	(1)
		25%	0.9945250	(4)	**0.9948903**	(3)	**0.9952987**	(1)	0.9952949	(2)
	MNAR	10%	0.9978923	(4)	**0.9981631**	(3)	0.9982540	(2)	**0.9982850**	(1)
		15%	0.9974959	(4)	**0.9975922**	(3)	**0.9978945**	(1)	0.9978659	(2)
		25%	0.9875513	(4)	**0.9883172**	(3)	**0.9894476**	(1)	0.9893522	(2)
PIMA	MCAR	10%	0.8019355	(4)	**0.8023523**	(3)	0.8049024	(2)	**0.8057123**	(1)
		15%	0.7873334	(4)	**0.7881910**	(3)	0.7913116	(2)	**0.7926936**	(1)
		25%	0.7720143	(4)	**0.7724599**	(3)	0.7782346	(2)	**0.7813148**	(1)
	MAR	10%	0.8005373	(4)	**0.8008901**	(3)	0.8035979	(2)	**0.8042388**	(1)
		15%	0.7866671	(4)	**0.7867853**	(3)	0.7896857	(2)	**0.7906564**	(1)
		25%	0.7748337	(4)	**0.7761880**	(3)	0.7792370	(2)	**0.7808444**	(1)
	MNAR	10%	0.8129813	(4)	**0.8135142**	(3)	0.8156111	(2)	**0.8163256**	(1)
		15%	0.7984465	(4)	**0.7989641**	(3)	0.8015427	(2)	**0.8024684**	(1)
		25%	0.7695292	(4)	**0.7704833**	(3)	0.7753233	(2)	**0.7778510**	(1)
ABAL	MCAR	10%	**0.8468388**	(3)	0.8465451	(4)	0.8497579	(2)	**0.8499176**	(1)
		15%	**0.8397988**	(3)	0.8393014	(4)	0.8425319	(2)	**0.8427391**	(1)
		25%	**0.8263714**	(3)	0.8257296	(4)	0.8295212	(2)	**0.8295821**	(1)
	MAR	10%	**0.8482142**	(3)	0.8476160	(4)	0.8498965	(2)	**0.8501366**	(1)
		15%	**0.8410048**	(3)	0.8408962	(4)	0.8424528	(2)	**0.8425750**	(1)
		25%	**0.8348346**	(3)	0.8345719	(4)	0.8368085	(2)	**0.8369419**	(1)
	MNAR	10%	**0.8455356**	(3)	0.8445578	(4)	0.8500263	(2)	**0.8501793**	(1)
		15%	**0.8443605**	(3)	0.8437819	(4)	0.8463715	(2)	**0.8464826**	(1)
		25%	0.8247781	(4)	**0.8252351**	(3)	0.8283600	(2)	**0.8283827**	(1)
Average rank			3.7778		3.2222		1.7333		1.2667	

Table B.4: AUC results when applying imputation frameworks SI, MI, S-HOT and M-HOT using the MIDA imputation method on benchmark datasets. Bold values are the best results between pairs of frameworks SI vs S-HOT, and MI vs M-HOT.

Dataset	Pattern		*SI*		*S-HOT*		*MI*		*M-HOT*	
IRIS	MCAR	10%	0.9958323	(4)	**0.9968544**	(3)	**0.9973218**	(1)	0.9972525	(2)
		15%	0.9947010	(4)	**0.9961452**	(3)	0.9967011	(2)	**0.9967655**	(1)
		25%	0.9879101	(4)	**0.9920251**	(3)	0.9931490	(2)	**0.9933714**	(1)
	MAR	10%	0.9977831	(4)	**0.9980598**	(3)	**0.9983718**	(1)	0.9983512	(2)
		15%	0.9964883	(4)	**0.9966728**	(3)	0.9969672	(2)	**0.9970707**	(1)
		25%	0.9934700	(4)	**0.9937325**	(3)	0.9940230	(2)	**0.9940345**	(1)
	MNAR	10%	0.9968828	(4)	**0.9969866**	(3)	0.9975337	(2)	**0.9975402**	(1)
		15%	0.9922183	(4)	**0.9939836**	(3)	0.9953051	(2)	**0.9953053**	(1)
		25%	0.9800315	(4)	**0.9851027**	(3)	**0.9893560**	(1)	0.9882223	(2)
STAT	MCAR	10%	0.9004228	(4)	**0.9099691**	(3)	**0.9122272**	(1)	0.9119025	(2)
		15%	0.8893594	(4)	**0.9026763**	(3)	0.9048134	(2)	**0.9050871**	(1)
		25%	0.8766682	(4)	**0.8956989**	(3)	0.8979225	(2)	**0.8987079**	(1)
	MAR	10%	0.9075065	(4)	**0.9135148**	(3)	**0.9157279**	(1)	0.9153541	(2)
		15%	0.9002249	(4)	**0.9072489**	(3)	0.9083954	(2)	**0.9085098**	(1)
		25%	0.8919708	(4)	**0.9009516**	(3)	0.9031956	(2)	**0.9033104**	(1)
	MNAR	10%	0.8966214	(4)	**0.9043305**	(3)	0.9067652	(2)	**0.9076346**	(1)
		15%	0.8953022	(4)	**0.9049765**	(3)	0.9072963	(2)	**0.9077530**	(1)
		25%	0.8724639	(4)	**0.8917990**	(3)	0.8940123	(2)	**0.8946876**	(1)
WINE	MCAR	10%	0.9986694	(4)	**0.9987253**	(3)	0.9988537	(2)	**0.9988668**	(1)
		15%	0.9975506	(4)	**0.9976749**	(3)	**0.9979885**	(1)	0.9979830	(2)
		25%	0.9947814	(4)	**0.9956436**	(3)	0.9958397	(2)	**0.9958428**	(1)
	MAR	10%	0.9963918	(4)	**0.9966461**	(3)	0.9968344	(2)	**0.9969354**	(1)
		15%	0.9978826	(4)	**0.9984737**	(3)	0.9986410	(2)	**0.9986593**	(1)
		25%	0.9966551	(4)	**0.9973227**	(3)	0.9976426	(2)	**0.9977587**	(1)
	MNAR	10%	0.9990950	(4)	**0.9991137**	(3)	**0.9991486**	(1)	0.9991412	(2)
		15%	0.9981937	(4)	**0.9982292**	(3)	0.9984342	(2)	**0.9984523**	(1)
		25%	0.9901818	(4)	**0.9940507**	(3)	0.9948135	(2)	**0.9948292**	(1)
PIMA	MCAR	10%	0.8183348	(4)	**0.8207717**	(3)	0.8225089	(2)	**0.8225461**	(1)
		15%	0.8060807	(4)	**0.8103984**	(3)	**0.8123862**	(1)	0.8123352	(2)
		25%	0.7932394	(4)	**0.8020177**	(3)	0.8040140	(2)	**0.8055621**	(1)
	MAR	10%	0.8201186	(4)	**0.8218285**	(3)	**0.8235582**	(1)	0.8235392	(2)
		15%	0.8078931	(4)	**0.8127261**	(3)	**0.8143046**	(1)	0.8142421	(2)
		25%	0.7989430	(4)	**0.8058399**	(3)	**0.8072073**	(1)	0.8071192	(2)
	MNAR	10%	0.8250540	(4)	**0.8278898**	(3)	0.8299199	(2)	**0.8303720**	(1)
		15%	0.8137489	(4)	**0.8189948**	(3)	0.8211264	(2)	**0.8213390**	(1)
		25%	0.7916129	(4)	**0.8011662**	(3)	0.8035721	(2)	**0.8036587**	(1)
ABAL	MCAR	10%	0.8655977	(4)	**0.8707402**	(1)	**0.8707151**	(2)	0.8707068	(3)
		15%	0.8604767	(4)	**0.8680851**	(1)	**0.8674447**	(2)	0.8673406	(3)
		25%	0.8495787	(4)	**0.8621626**	(1)	**0.8596208**	(2)	0.8592872	(3)
	MAR	10%	0.8665143	(4)	**0.8711025**	(1)	**0.8708308**	(2)	0.8708024	(3)
		15%	0.8613696	(4)	**0.8689452**	(1)	**0.8678045**	(2)	0.8676311	(3)
		25%	0.8518054	(4)	**0.8642873**	(1)	**0.8612268**	(2)	0.8608408	(3)
	MNAR	10%	0.8697805	(4)	**0.8746136**	(1)	0.8743051	(3)	**0.8743183**	(2)
		15%	0.8651282	(4)	**0.8728212**	(1)	**0.8716987**	(2)	0.8715489	(3)
		25%	0.8518229	(4)	**0.8642183**	(1)	**0.8614111**	(2)	0.8609479	(3)
Average rank			4.0000		2.6000		1.7778		1.6222	

Table B.5: AUC results when applying imputation frameworks SI, MI, S-HOT and M-HOT using the SINKHORN imputation method on benchmark datasets. Bold values are the best results between pairs of frameworks SI vs S-HOT, and MI vs M-HOT.

B.2 Experimental Results on Medical Datasets

Dataset	Metric	*SI*	*S-HOT*	*MI*	*M-HOT*
COVI	bACC	**88.6725 ±2.5332**	88.6654 ±2.7242	88.7365 ±2.557	**88.7617 ±2.5724**
	AUC	0.9614745 ±0.0121358	**0.9620376 ±0.0119127**	0.9627426 ±0.0115929	**0.9628588 ±0.0115969**
	F1	88.7026 ±2.5649	**88.7527 ±2.7286**	88.7615 ±2.6008	**88.7998 ±2.571**
MYOC	bACC	70.7693 ±2.6293	**70.9092 ±2.7726**	71.3911 ±2.5436	**71.6052 ±2.6053**
	AUC	0.8243456 ±0.0221242	**0.8247766 ±0.0217767**	0.8487178 ±0.0190603	**0.8489546 ±0.019136**
	F1	**85.7213 ±1.3505**	85.7094 ±1.4744	**86.4856 ±1.3564**	86.4783 ±1.3795
NHAN	bACC	63.5868 ±1.6318	**63.7206 ±1.7016**	64.3151 ±1.5493	**64.3741 ±1.6**
	AUC	0.7016346 ±0.0171187	**0.7023302 ±0.0175341**	**0.7172432 ±0.0152297**	0.7170467 ±0.0158433
	F1	63.3669 ±1.7352	**63.4867 ±1.7925**	64.0853 ±1.6477	**64.1244 ±1.7684**

Table B.6: Experimental results when applying imputation frameworks SI, MI, S-HOT and M-HOT using the MISSFOREST imputation method on three real-world medical datasets. Bold values are the best results between pairs of frameworks SI vs S-HOT, and MI vs M-HOT.

Dataset	Metric	*SI*	*S-HOT*	*MI*	*M-HOT*
COVI	bACC	87.8585 ±2.4947	**88.3732 ±2.5208**	88.1767 ±2.5231	**88.3682 ±2.5538**
	AUC	0.954733 ±0.0140975	**0.9575914 ±0.0135124**	0.9580374 ±0.0133241	**0.9583938 ±0.0133435**
	F1	87.9294 ±2.4903	**88.4383 ±2.5006**	88.2291 ±2.5398	**88.4105 ±2.5544**
MYOC	bACC	68.9386 ±2.6463	**69.1718 ±2.6207**	69.694 ±2.6266	**69.8195 ±2.715**
	AUC	0.8031829 ±0.0243522	**0.8034473 ±0.0228805**	**0.8330273 ±0.0188834**	0.8321686 ±0.0192249
	F1	84.6802 ±1.4062	**84.7812 ±1.3634**	**85.6378 ±1.4017**	85.6106 ±1.4561
NHAN	bACC	62.9123 ±1.6569	**63.1646 ±1.673**	63.9159 ±1.4439	**64.0496 ±1.574**
	AUC	0.6919003 ±0.0173845	**0.6973462 ±0.0168397**	0.7115084 ±0.0147607	**0.7135331 ±0.0152922**
	F1	62.7138 ±1.764	**62.9228 ±1.7481**	63.7059 ±1.5399	**63.796 ±1.7449**

Table B.7: Experimental results when applying imputation frameworks SI, MI, S-HOT and M-HOT using the SOFTIMPUTE imputation method on three real-world medical datasets. Bold values are the best results between pairs of frameworks SI vs S-HOT, and MI vs M-HOT.

Dataset	Metric	*SI*	*S-HOT*	*MI*	*M-HOT*
COVI	bACC	87.0045 ±2.6473	**88.4374 ±2.5974**	87.6717 ±2.6013	**88.4832 ±2.4591**
	AUC	0.947926 ±0.0160487	**0.9593048 ±0.0139753**	0.9542347 ±0.015257	**0.9583125 ±0.0143513**
	F1	87.0508 ±2.6829	**88.4577 ±2.5767**	87.6887 ±2.6213	**88.4995 ±2.466**
MYOC	bACC	68.8914 ±2.5702	**69.518 ±3.0158**	69.5851 ±2.6378	**70.0704 ±2.6058**
	AUC	0.8073393 ±0.0232012	**0.8144668 ±0.0226134**	0.8474765 ±0.018576	**0.848819 ±0.0191143**
	F1	84.8043 ±1.3901	**84.9436 ±1.4807**	85.8864 ±1.3565	**85.9842 ±1.3692**
NHAN	bACC	62.608 ±1.8379	**63.8408 ±1.5626**	64.7467 ±1.616	**64.8621 ±1.6765**
	AUC	0.6860431 ±0.0191444	**0.7065677 ±0.0165336**	0.7220923 ±0.0160951	**0.7268964 ±0.0163592**
	F1	62.452 ±1.8951	**63.5959 ±1.6702**	64.6042 ±1.6478	**64.672 ±1.8169**

Table B.8: Experimental results when applying imputation frameworks SI, MI, S-HOT and M-HOT using the GAIN imputation method on three real-world medical datasets. Bold values are the best results between pairs of frameworks SI vs S-HOT, and MI vs M-HOT.

Dataset	Metric	*SI*	*S-HOT*	*MI*	*M-HOT*
COVI	bACC	88.1085 ±2.5899	**88.2243 ±2.6737**	**88.4101 ±2.5023**	88.3932 ±2.4882
	AUC	0.959175 ±0.0131421	**0.9597053 ±0.0127948**	0.9606792 ±0.0127068	**0.9607641 ±0.0126721**
	F1	88.1598 ±2.6211	**88.3403 ±2.6449**	**88.473 ±2.4939**	88.4681 ±2.4804
MYOC	bACC	69.6988 ±2.7511	**69.8614 ±2.746**	70.0331 ±2.6288	**70.2846 ±2.5858**
	AUC	0.8126983 ±0.0228478	**0.8143542 ±0.0210367**	0.8419529 ±0.0187064	**0.8424803 ±0.0185702**
	F1	84.9747 ±1.4696	**84.9862 ±1.4871**	85.6655 ±1.4259	**85.7106 ±1.3629**
NHAN	bACC	63.4962 ±1.6211	**63.5197 ±1.738**	**64.2441 ±1.6441**	64.1943 ±1.6845
	AUC	0.6982824 ±0.0167148	**0.6984358 ±0.0174299**	**0.7141273 ±0.0156205**	0.7137177 ±0.0155668
	F1	63.2441 ±1.7827	**63.2592 ±1.8971**	**63.9746 ±1.767**	63.9232 ±1.8481

Table B.9: Experimental results when applying imputation frameworks SI, MI, S-HOT and M-HOT using the MIDA imputation method on three real-world medical datasets. Bold values are the best results between pairs of frameworks SI vs S-HOT, and MI vs M-HOT.

Dataset	Metric	*SI*	*S-HOT*	*MI*	*M-HOT*
COVI	bACC	88.0521 ±2.7099	**88.7577 ±2.4153**	88.1914 ±2.444	**88.5625 ±2.4207**
	AUC	0.954824 ±0.0147369	**0.9625726 ±0.0124882**	0.9624413 ±0.0123389	**0.9630812 ±0.012202**
	F1	88.0857 ±2.7305	**88.8315 ±2.3945**	88.2002 ±2.4686	**88.5704 ±2.4137**
MYOC	bACC	69.8555 ±2.7178	**70.0475 ±2.6415**	70.3021 ±2.5083	**70.4734 ±2.5403**
	AUC	0.8158175 ±0.0219278	**0.8188904 ±0.0214235**	0.8447799 ±0.0186636	**0.8456001 ±0.0184988**
	F1	85.1389 ±1.3661	**85.1642 ±1.412**	85.9508 ±1.3636	**85.9812 ±1.3924**
NHAN	bACC	63.1886 ±1.5701	**63.3785 ±1.5872**	**64.0025 ±1.5978**	63.9914 ±1.6632
	AUC	0.6926347 ±0.0176282	**0.6965183 ±0.0174091**	0.709166 ±0.0152069	**0.7095322 ±0.0156614**
	F1	62.9453 ±1.6887	**63.0783 ±1.6842**	**63.7281 ±1.7522**	63.7012 ±1.8955

Table B.10: Experimental results when applying imputation frameworks SI, MI, S-HOT and M-HOT using the SINKHORN imputation method on three real-world medical datasets. Bold values are the best results between pairs of frameworks SI vs S-HOT, and MI vs M-HOT.