Mary-Morstan : a multi-objective modular framework to automatically configure machine learning algorithms

Laurent Parmentier

UNIVERSITÉ DE LILLE

ÉCOLE DOCTORALE MADIS-631: MATHÉMATIQUES, SCIENCES DU NUMÉRIQUE ET DE LEURS INTERACTIONS

Mary-Morstan: A Multi-objective Modular Framework to Automatically Configure Machine Learning Algorithms

---

Mary-Morstan : un framework modulaire et multiobjectif pour la configuration automatique d'alrogithmes de machine learning

---

Laurent PARMENTIER

Thèse préparée et soutenue publiquement le 06/04/2022, en vue de l'obtention du grade de Docteur en Informatique et Application.

Composition du jury:

Pr. Pierre CHAINAIS	Ecole Centrale Lille	*Président, Examinateur*
Pr. Germain FORESTIER	Université de Haute-Alsace	*Rapporteur*
Pr. Edward KEEDWELL	University of Exeter	*Rapporteur*
Pr. Jalal FADILI	ENSICAEN	*Examinateur*
Pr. Marius LINDAUER	Leibniz University Hannover	*Examinateur*
Pr. Laetitia JOURDAN	Université de Lille	*Directrice de Thèse*
MCF HDR Marie-Eléonore KESSACI	Université de Lille	*Co-Directrice de Thèse*
Dr. Olivier NICOL	OVHCloud	*Co-Encadrant de Thèse*

Abstract

The growing usage of machine learning solutions (movie recommendation, speech recognition, fraud detection and so on) creates the demand for having more efficient tools to build them. Indeed, building a machine learning model is a tedious task. The practitioner is required to preprocess the data, build the features, select the machine learning algorithm and tune its hyper-parameters. Historically, these steps are handmade, but more recent tools called AutoML for Automatic Machine Learning have blossomed, and propose to perform these tasks automatically. Thus, AutoML eases the research of models and permits a gain of time for the experts, but also aims to help the non-experts to build a model without having to understand all the underlying mechanisms. In this work, we analyze the best known optimization methods used by the AutoML tools, and notice that among these methods, the evolutionary algorithms are very promising when it comes to improve the research of models. Indeed, evolutionary algorithms ease the tuning of the exploration versus exploitation trade-offs, are inherently capable of handling any sort of candidates (fixed and variable sizes), can tackle multiple objectives and can be easily parallelized. However, they have been barely studied in the AutoMLs, especially when it concerns the choice of the components such as the mutations or the algorithms. In this work, we first define a modular AutoML and a range of new components designed to study their impacts when used to automatically solve classification problems. Then, we come up with a method to accelerate all the optimization processes based on evolutionary algorithms for large datasets. Finally, we propose a solution to automatically tackle time series classification problems which, to the best of our knowledge, have never been studied before.

Résumé

L'utilisation grandissante de solutions d'apprentissage automatique (recommandation de films, reconnaissance du texte, détection de la fraude et ainsi de suite) crée une demande pour avoir des outils plus efficaces. En effet, construire un modèle d'apprentissage automatique est une tâche laborieuse. Le praticien doit formater les données, construire les attributs, sélectionner l'algorithme d'apprentissage automatique adéquat, et régler ses hyperparamètres. Historiquement ces étapes sont manuelles, mais des outils récents appelés AutoML, Automatic Machine Learning, ont vu le jour et proposent de réaliser ces tâches automatiquement. Ainsi, l'AutoML facilite la recherche des modèles et octroie un gain de temps aux experts, et permet également aux non-experts de construire un modèle sans avoir à comprendre les mécanismes sous-jacents. Dans ces travaux, nous analysons les méthodes d'optimisations les plus connues et utilisées par les outils d'AutoML. Lors de notre analyse, nous avons remarqué que parmi ces diverses méthodes, les algorithmes évolutionnaires semblent prometteurs dans la recherche des modèles. Notamment, ils facilitent la configuration de la phase de compromis d'exploration versus exploitation, sont intrinsèquement capables de manipuler toute sorte de candidats (taille fixe ou variable), peuvent aborder plusieurs objectifs et, sont facilement parallélisables. Cependant, ces algorithmes évolutionnaires restent très peu étudiés dans les AutoMLs, en particulier quand cela concerne le choix des composants tels que les mutations ou les algorithmes. Dans ces travaux, nous définissons un framework d'AutoML modulable avec de nouveaux composants. L'objectif est d'étudier l'impact de ces derniers quand ils sont utilisés pour résoudre des problèmes de classification. Par la suite, cela nous a menés au développement d'une méthode qui accélère l'ensemble du processus d'optimisation basé sur les algorithmes évolutionnaires devant traiter d'importants volumes de données. Pour finir, nous proposons une solution qui résout automatiquement le problème de classification des séries temporelles qui, d'après nos connaissances, n'a jamais été étudié auparavant.

Acknowledgments

I want to thank so many people who contributed in different manners to the spirit of this thesis.

I start with Germain Forestier and Edward Keedwell, the reviewers of my work, for reading my works with the greatest care. My appreciation is extended as well to the other members of my thesis jury for the great interest they took in my work.

Olivier Nicol, my company advisor, but also my trustful friend. He constantly challenged me on my certainty, and despite my obstinacy and the misconceptions I had, he never let me down. Our diverse conversations helped me to improve my reasoning when solving research problems.

Laetitia Jourdan and Marie-Eléonore Kessaci, my two thesis supervisors. They both supported my ideas, gave me precious advices on my work. Their given freedom allowed me to explore a wide range of hypotheses and contributed to understand my own flaws.

OVHCloud and its Machine Leaning Services team leads by Guillaume Salou. Despite my fresh arrival, they quickly trust my capacity to tackle a research subject. Aurélien Tanière for its continuous follow up on the administrative procedures and Alain Fiocco who pushed some blocking process to initiate the thesis.

My love Aurélie Perez and, my child Noah. Both bring me joy, love, and happiness in my daily life. Along the whole time of these works, they helped me to keep my motivation up.

The support of my parents, sister and its family. Their way of sharing love and their conception of the family inspired my life, and helped me to minimize the difficulties I went through.

My closest friends that share my life with all the fun moments as well as tough ones: Alexandre, Fabrice, Quentin, Angel, Louis, Marcello, Julien, Guillaume, Robin.

My coworkers, for their constant sympathy and positivity: Raphael Glon, Clement Bataille, Pauline Wauquier, Sophie Lennon, Mael Le-Gal, Eric Audam, Oualid Bouchair, Christophe Rannou, Steeven Vendecappelle, Julien Fayeulle, Audrey Plantureux.

My colleagues from the laboratory for their friendly talks and their exchange of knowledge: Lucien Mousin, Camille Buge, Szczepanski Nicolas, Nadarajen Veerapen, Julie Jacques,

Weerapan Sae-Dan, Clarisse Dhaenens.

Many thanks to all other people that have been part of my life at a moment or another and contributed in a way or another to the inspiration of these works. I'm thinking of my two previous managers that I considered as very good friends, for their inspiring methods of governing teams and projects: Eric Roth and David Cournoyer. My previous teachers that showed true passion in their jobs and a great respect for their students: Valérie Jaeger, François Brisoux, Patrick Lacharme, Max Chlebowski, Yann Secq, Bruno Beaufils, Christophe Rosenberger, Jalal Fadili, Stéphanie Molins. My distant friends: Yigit, Jonathan, André, Danny, Emilie, Matthew, Alexe, Raphael, Flavie, Jason. I'm also thinking of the people that were involved in the following names: Start Stunt Team, Equipage des 7 mers, Neo's project, While42's group.

Contents

List of Figures

List of Tables

Glossary

HPC High Performance Computing

DSOP DataSet Open Platform; see Section 1.3.1

AutoML Automatic Machine Learning; see Chapter 2

CASHAP Combined Algorithm Selection, Hyperparameter optimization And Preprocessing selection; see Section 2.1.3

MOP Multi-objective Optimization Problem; see Equation 2.1.4

SMBO Sequential Model-Based Optimization; see Section 2.2.2

EA/EAs Evolutionary Algorithm(s); see Section 2.2.3

MCTS Monte-Carlo Tree Search; see Section 2.2.4

SH Successive-Halving; see Section 4.1.2

TSC Time Series Classification; see Section 5.1.4

Introduction (French)

L'apprentissage automatique, ou machine learning en anglais, est omniprésent de nos jours. Parmi les dernières applications, cet apprentissage automatique est utilisé pour les voitures autonomes dans le but de reconnaître les objets qui défilent en cours de route et pour prendre des décisions telles qu'augmenter ou réduire la vitesse d'accélération du véhicule. Cette technique est également employée dans de nombreux services (site web de shopping, applications de streaming vidéos) pour recommander [89] des items (vidéos, musiques, livres et tout autre produit). D'autres applications moins évoquées sont l'estimation du prix d'une maison, la détection d'anomalies (surchauffes dans les datacentres), la reconnaissance des codes postaux écrits à la main, la reconnaissance des catégories de déchets dans un centre de tri etc.

Cet intérêt grandissant pour l'apprentissage automatique est dû à plusieurs raisons.

Premièrement, alors que la création d'une fonction dans un processus classique dépend principalement du code réalisé par un être humain, l'apprentissage automatique, lui, construit une fonction (également appelée modèle) à l'aide d'un algorithme qui utilise les données comme source de connaissances (Figure 0-1). En déléguant la création de la fonction à un algorithme et à des sources externes, le travail nécessaire pour la construire est réduit. Ceci est particulièrement vrai pour les fonctions très compliquées qui, face à un environnement changeant, ont constamment besoin d'être mises à jour pour fonctionner avec précision. Créer ou mettre à jour manuellement une telle fonction requiert un temps considérable pouvant facilement rendre obsolètes ses résultats. En effet, imaginons implémenter une fonction qui reconnaît les objets dans une image. Si chaque objet a des centaines de variables (formes, couleurs, contrastes, dimensions), ainsi que des millions d'images à considérer tels que l'on distingue l'ensemble des caractéristiques de chaque objet, cela représenterait un temps de travail considérable pour la construire avec précision. En fait, la situation se complexifierait encore si de nouvelles images et de nouvelles variables s'ajoutent. Il serait alors nécessaire de travailler à nouveau l'ensemble des règles précédemment implémentées afin de s'assurer que la fonction continue de prédire avec précision. En d'autres termes, la force de l'apprentissage automatique se situe dans sa capacité à construire des fonctions complexes en

un temps raisonnable via l'extraction de connaissances résidant dans les données. De plus, ces algorithmes peuvent être relancés indéfiniment de telle manière que les nouvelles données (c'est-à-dire la connaissance) soient constamment prises en considération dans la nouvelle fonction reconstruite. Ainsi, le laborieux processus humain de coder et de maintenir à jour une fonction est réduit et devient réalisable.

Deuxièmement, les techniques ont beaucoup progressé au cours de la dernière décennie. En effet, la convergence des algorithmes existants [68] est devenue plus efficace. Il y a de nouveaux algorithmes plus performants [18, 67, 109], et il en est de même pour les méthodes de préprocessing des données [23, 118]. Ainsi, ces techniques ont permis de gagner en performance en comparaison aux algorithmes plus classiques résolvant des problèmes similaires. Par conséquent, elles ont accéléré l'adoption de l'apprentissage automatique.

Enfin, la prolifération d'établissements (data centres) capables de stocker une large quantité de données avec toujours plus de puissance de calculs (clusters, clouds), et le fait que nous monitorons bien plus d'informations à tous les niveaux de nos sociétés, contribuent également à l'adoption des processus d'apprentissage automatique.

A OVHCloud, nous avons une grande quantité de données à travers le monde et nous proposons une large gamme de produits, allant de l'enregistrement des noms de domaines à la location de serveurs dédiés. Par conséquent, le comportement de nos clients diffère d'un produit à un autre et il en est de même pour les pays. Comprendre ce changement de comportement pourrait permettre l'amélioration de l'expérience utilisateur pour nos clients, ainsi qu'améliorer nos processus internes. En effet, parmi les cas d'usages internes, il y a la possibilité de réduire les pannes avec la maintenance préventive, d'optimiser le stock des composants informatiques, et de réduire la consommation d'énergie des infrastructures. Ainsi, pour toutes ces raisons, OVHCloud a développé de l'intérêt à utiliser l'apprentissage automatique.

L'entreprise a commencé à utiliser l'apprentissage automatique pour s'attaquer à une problématique concernant la fraude sur les services OVHCloud Public Cloud. Initialement, les fraudeurs étaient identifiés avec des règles statiques directement implémentées dans le code. Cependant, leur comportement change constamment et a rendu la solution rapidement inadaptée. Cela a plusieurs répercussions sur l'entreprise. La perte d'argent. Le fait

que les fraudeurs consomment des produits empêche les utilisateurs légitimes d'accéder aux ressources. En effet, les ressources d'OVHCloud Public Cloud fonctionnent en flux tendu pour des principes écologiques et pour minimiser les coûts. La fraude participe également à la détérioration des composants (exemple des mineurs en cryptomonnaie). De plus, les fraudeurs ont tendance à stocker des données illicites, ce qui peut mettre à mal l'image de l'entreprise. Ainsi, nous avons décidé d'appliquer des solutions d'apprentissage automatique plus adaptées au changement de comportement, ce qui a permis de résoudre l'ensemble des conséquences évoquées. Néanmoins, de nouvelles complications sont apparues, notamment la difficulté à mettre en place et à maintenir l'enchaînement des processus impliqués dans l'apprentissage automatique. Citons également la complexité d'extraire des données qui produisent de bons modèles, l'expertise requise pour correctement sélectionner et configurer les algorithmes d'apprentissage automatique, et le manque d'outils pour servir un modèle via un microservice que tout le monde peut utiliser intuitivement.

Ainsi, nous avons décidé de construire notre propre outil capable de faciliter l'ensemble des processus présents dans l'apprentissage automatique. Au cours de notre démarche, nous avons remarqué que des outils similaires existaient dans la sphère de recherche publique. Ces outils sont communément appelés AutoML, qui signifie l'Automatisation du Machine Learning, et essaient de résoudre les différentes problématiques rencontrées par les praticiens. Souhaitant développer nos connaissances dans ce domaine, nous avons décidé d'étudier ces outils, ce qui nous a également menés sur le sujet de cette thèse et ces travaux.

Pour entreprendre cette tâche, OVHCloud s'est associé avec ORKAD via une procédure CIFRE. ORKAD est une équipe de recherche faisant partie du laboratoire CRIStAL. Leur domaine d'expertise concerne la résolution de problèmes d'optimisation avec des optimisations métaheuristiques et de l'extraction de connaissances. Cela concorde parfaitement avec le coeur des méthodes utilisées par les outils d'AutoML pour sélectionner et configurer les algorithmes d'apprentissage automatique afin de maximiser une fonction d'objectif.

Dans ces travaux, nous avons développé un nouvel outil d'AutoML capable d'étudier des algorithmes évolutionnaires, des méthodes d'optimisation étudiées sur d'autres problématiques (N-Queens), mais à peine explorés sur les AutoMLs. Nos travaux ont débouché sur une première proposition, une technique permettant aux algorithmes évolutionnaires d'être

plus rapides quand appliqués sur de larges jeux de données. Et, une seconde proposition permettant de résoudre automatiquement le problème de la classification sur des séries temporelles.

Le manuscrit décrivant nos travaux est organisé comme suit:

Chapitre 1 introduit l'entreprise OVHCloud et son intérêt pour l'IA, justifié par la présence de fraudeurs qui a donné naissance au développement d'un AutoML en plus de cette thèse (1.2.1). Ensuite, nous mettons en lumière les produits d'IA qui ont été développés par l'entreprise, chacun d'entre eux prenant une place comme un composant dans l'AutoML. Le premier est Prescience (1.2.2), une plateforme globale d'AutoML utilisable au travers d'une interface web. Ensuite, OVHCloud Serving Engine (1.2.3), un service qui permet de déployer n'importe quel modèle d'apprentissage automatique. Enfin, OVHCloud AI Training (1.2.4), une plateforme avec un environnement de GPU/CPU extensible et qui peut se connecter avec un Jupyter notebook. D'autre part, nous évoquerons trois projets développés en parallèle de la thèse: DSOP (1.3.1), une plateforme qui permet de faciliter la récupération de jeux de données, Interpretability Engine (1.3.2), une librairie qui facilite l'interprétation de modèles déployés sur OVHCloud Serving Engine et, Slurm-PCi (1.3.3), un cluster de haute performance évolutif basé sur OVHCloud Public Cloud pour lancer les expériences de recherche.

Chapitre 2 donne un arrière-plan sur l'apprentissage automatique (2.1.1) et les difficultés sous-jacentes qu'il cause, ce qui nous amène au développement du nouveau domaine d'étude appelé AutoML (2.1.2). L'AutoML peut être défini comme un problème d'optimisation (2.1.3). Nous énumérons et étudions les solutions de l'état de l'art (2.2). Cette étude nous a permis de mettre en avant les parties sous-exploitées dans l'AutoML (2.3), ce qui a conforté nos choix pour le développement d'un nouvel outil.

Chapitre 3 introduit Mary-Morstan, un nouvel outil d'AutoML. Cet outil est basé sur les algorithmes évolutionnaires, que nous motivons par les nombreux avantages qu'ils présentent (3.1). En effet, les algorithmes évolutionnaires sont adaptés pour traiter des candidats avec des tailles variables, ce qui en fait une solution idéale pour traiter le problème de l'AutoML (2.1.3). Ils sont nativement capables de traiter plusieurs objectifs, une fonctionnalité prometteuse pour le futur des AutoMLs. Et, ils sont également conçus pour être

facilement configurables, ce qui permet de jouer avec la phase d'exploration versus exploitation lors de l'optimisation. Il s'avère que ce dernier aspect a prouvé avoir un impact sur les performances d'optimisation d'autres problèmes de recherche (N-Queens). Etant donné qu'il n'a pas été étudié sur le problème de l'AutoML, nous proposons de le faire en y ajoutant de nouveaux composants qui permettent d'étudier l'impact sur les performances de l'AutoML. Dans ce chapitre, nous commençons par donner une vue globale (3.2) de notre outil en le comparant avec l'état de l'art dans le but de mettre en avant ces nouveaux composants. Ensuite, nous détaillons le fonctionnement des différents composants (3.3) en plus des impacts attendus sur les modèles d'apprentissage automatique (3.3.3.3). Pour évaluer les performances de notre AutoML, nous réalisons deux différentes expériences, une première pour valider notre solution (c'est-à-dire que les performances de l'état de l'art soient atteintes), et une seconde pour voir si l'introduction de nos composants surpasse l'état de l'art (3.4). Durant nos expériences, nous avons remarqué une faiblesse majeure des AutoMLs basés sur les algorithmes évolutionnaires qui, ont des difficultés à gérer des jeux de données conséquents (3.5).

Chapitre 4 propose une technique permettant aux AutoMLs basés sur les algorithmes évolutionnaires de gérer des jeux de données conséquents (4). La technique est basée sur Successive Halving, introduite sur le problème des Multi-armed Bandits (4.1.2). Une fois Adaptée pour les AutoMLs basés sur les algorithmes évolutionnaires (4.1.3), nous réalisons une expérience sur de petits jeux de données et sur de grands jeux de données afin d'observer son comportement (4.3). Comme attendu, sur des petits jeux de données, l'emploi du Successive Halving n'apporte pas de valeur ajoutée. Par contre, sur des jeux de données plus conséquents, l'amélioration par rapport à l'état de l'art est très significative, résolvant complètement le problème entrevu dans le chapitre précédent.

Chapitre 5 continue les travaux sur l'aspect modulable initié avec Mary-Morstan. Suite à l'émergence de nouveaux algorithmes d'apprentissage automatique dédiés pour des problèmes de classification avec des données temporelles (2.1.1.3), et non étudiés par les AutoMLs, nous avons décidé de le résoudre avec notre solution en tirant profit des nouveaux composants proposés. Nous commençons par introduire le problème de classification avec des séries temporelles, et motivons l'intérêt qu'il y a pour le résoudre avec une solution

d'AutoML (5.1). Nous décrivons ensuite les modules de Mary-Morstan permettant de se pencher sur ce problème (5.1.5). Enfin, nous proposons une séries d'expériences sur des jeux de données classiques de classification de séries temporelles (5.2). Expériences qui ont démontré un intérêt significatif de notre approche sur une large partie d'entre eux, comparée à des étalons standard définis au préalable (5.3).

Chapitre 6 résume l'ensemble des contributions et donne un panorama sur les futurs travaux pouvant être réalisés.

Introduction

Machine Learning is omnipresent nowadays. Among the latest applications, we find its presence in autonomous cars to recognize the different objects encountered along the road as well as to take a decision such as accelerating the car, or reducing its speed. We also find it in services (shopping websites, video streaming applications) to recommend [89] different items (movies, songs, books, and any other products). Many other applications that are less mentioned exist. For example, estimating the price of a house, detecting the anomalies (e.g. overheating in data centers), recognizing handwritten zip codes, categorizing and recognizing waste in sorting centers, and so on.

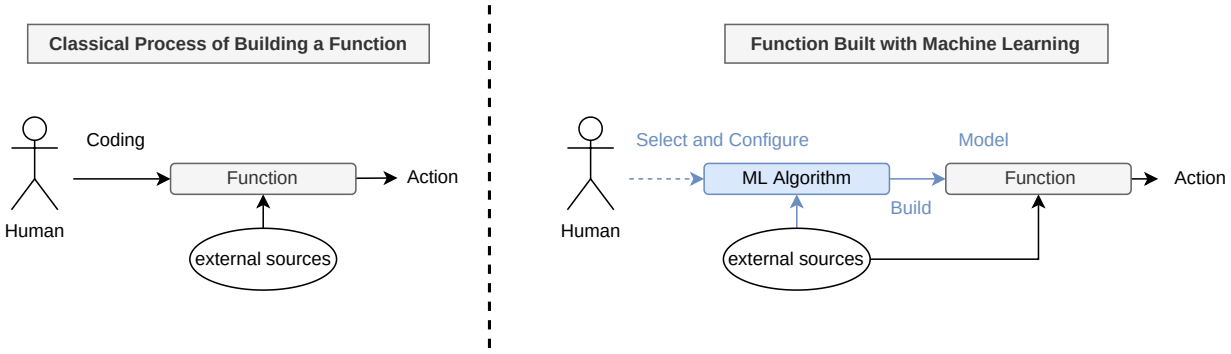


Figure 0-1: Traditional Process of Building a Function versus with Machine Learning

This growing interest in machine learning is caused by different reasons.

First, unlike a classical process where the creation of a function mainly depends on the coding made by a person, a function (also called a model) in machine learning is built via a dedicated algorithm which uses external data as a source of knowledge (Figure 0-1). By delegating the creation of the function to an algorithm and to external sources, the necessary work to build it is reduced. This is especially true for very complex functions that see the structure of their input or output evolve over the time, and need to be updated in order to work accurately. Manually creating or updating such a function would take an unreasonable amount of time if done by humans. For example, imagine you want to implement a function that recognizes the objects in a picture. Given that each object can be described by hundreds of variables (e.g. forms, colors, contrasts, dimensions) and that millions of pictures should be considered in order to distinguish all the characteristics of each object, it would take a

huge amount of time to accurately build such a function. In fact, it becomes even more complicated if new pictures and new variables are introduced. You would have to pass all along the already implemented rules (add, remove, or update them) to ensure that the function still predicts accurately. In other words, the strength of machine learning resides in its capacity to build a complex function within a reasonable time by extracting the knowledge that lies in the data. Moreover, these algorithms can be re-performed indefinitely in such a way that new data (i.e. knowledge) are consistently taken in consideration within the given function that is rebuilt. Thus, the laborious human process of coding and maintaining an updated function is reduced and becomes more achievable.

Second, the progress of the techniques in the domain, by improving the convergence of the existing algorithms [68], proposing new faster and more efficient algorithms [18, 67, 109], and new preprocessing methods [23, 118], permits to beat classical algorithms that solve similar problems and by consequence accelerates the adoption of machine learning.

Third and last, the proliferation of facilities (e.g. data centers), capable of storing large quantities of data with more computational power (e.g. clusters, clouds) and the fact that we monitor much more information at all the levels of our societies, also contribute to the adoption of machine learning processes for taking advantage of all that data by hand would simply be intractable.

At OVHCloud, we have a lot of data centers around the world, and we provide a wide range of products going from the registration of domain names to the rental of baremetal servers. Therefore, the behaviors of our customers completely differ from one product to another but also depending to the countries, the type of company, etc. Understanding this change of behavior might improve the user experiences of our customers. We can also leverage machine learning for many of our internal processes. Among others, we can mention the possibility to reduce hardware failures with preventive maintenance, to optimize our stock of hardware components or to reduce the energy consumption of our infrastructures. Thus, for all these reasons, OVHCloud has an interest in using machine learning.

The company started to use machine learning processes by tackling fraud on OVHCloud Public Cloud services. The initial solution consisted in catching the fraudsters with static rules directly implemented in the code. However, the behavior of the fraudsters constantly

changed and made the solution inadequate. This had multiple bad consequences, such as the loss of money for the company but not only. The fact that fraudsters consume the products tends to obstruct the legitimate users to have access to the resources. Indeed, the resources of OVHCloud Public Cloud services work in almost just-in-time production, which permits to respect some ecological principles, as well as to minimize our costs. Fraud also participates to deteriorate the components (e.g. cryptocurrency mining softwares). Moreover, the fraudsters tend to store illicit data which might harm the corporate image. Thus, we decided to involve a solution of machine learning, more adapted to the constant changes of behavior, which solved all the enumerated issues. However, it also brings new complications. Among these complications, there is the difficulty to set up and maintain an efficient and productive pipeline which is due to many reasons. To name a few, there is the complexity of extracting the data such that the model gives good results, the expertise required to correctly select and configure the machine learning algorithms, and the lack of tools to set up a model as an intuitive microservice usable by other teams.

Hence, we decided to build our own tool, capable of easing all the involved processes within machine learning. Along our road, we noticed the existence of similar tools in public research. These tools commonly called AutoML, for Automatic Machine Learning, try to solve the different issues encountered by their practitioners. Our lack of knowledge and our motivation to know more on that field pushed us forward to investigate these tools, which also led us on the subject of this thesis and this work.

To undertake this task, OVHCloud associated with ORKAD through a CIFRE[1] procedure. ORKAD is a research team, part of CRIStAL laboratory and the University of Lille. Their domain of expertise concerns the optimization problems solved with metaheuristic optimizations and with the extraction of knowledge. This perfectly fits with the core methods used by the AutoML tools to select the machine learning algorithms and to tune their hyper-parameters, which is like maximizing a complex objective function.

In this thesis work, we developed a new AutoML tool capable of studying the evolutionary algorithms, a known optimization method well studied on other problems (e.g. N-Queens), but barely explored in AutoML. Our works led us to a first proposition, which is a new

[1] https://kutt.parmentier.io/cifre

technique permitting the evolutionary algorithms to perform faster on large datasets without using any parallelism methods. And, to a second proposition which automatically solve the time series classification problems.

The manuscript that describes our work is organized as follows:

Chapter 1 introduces the company OVHCloud and its interest in AI, notably caused by the presence of fraud that leads to the development of an AutoML and this thesis work (1.2.1). Then, we give some insight concerning the AI products that were developed by the company, each of which being a component of AutoML: Prescience (1.2.2), a global web-ui AutoML platform, OVHCloud Serving Engine (1.2.3), a service to deploy any machine learning model, and OVHCloud AI Training (1.2.4), a scalable GPU/CPU platform that can be plugged with a Jupyter notebook. Finally, we list three sideline projects that were developed along with the thesis: DSOP (1.3.1) a platform to ease the retrieving of datasets, Interpretability Engine (1.3.2) a library that eases the interpretation of black-box models deployed on OVHCloud Serving Engine, and Slurm-PCi (1.3.3) a scalable HPC platform on top of OVHCloud Public Cloud to run research experiments.

Chapter 2 gives background on machine learning (2.1.1) and the underlying problems it causes, which leads to the study of a new field called AutoML (2.1.2). AutoML can be defined as an optimization problem (2.1.3). We finally propose a classification to categorize all these approaches (2.2). This classification also permits to highlight the least studied aspects of the problem (2.3) which contributed to guide our investigations while developing our new AutoML.

Chapter 3 introduces Mary-Morstan, a new AutoML tool. In a nutshell, the tool is based on evolutionary algorithms which is motivated by their multitude of advantages (3.1). Indeed, these algorithms are adapted to manage candidates with variables sizes, which makes it an ideal solution to solve the AutoML problem (2.1.3). They are natively capable of managing multiple objectives, a promising feature for the future of AutoMLs. And, they are also designed to be easily tuned, which permits to play with the exploration versus exploitation trade-off during the optimization. This last aspect has been well studied on different research problems and impacts the optimization performance. However, it has not been studied on the AutoML problem, which is why we proposed new components permitting

to study its impact on the performance of the latter. We give a general overview of our tool (3.2) and compare it with the state-of-the-art in order to emphasize the new components (3-2). Then, we detail all the different components (3.3) and give some of the expected impacts (3.3.3.3) on machine learning models. Finally, to evaluate our AutoML, we run two different experiments. A first one to validate our solution (i.e. that it meets the state-of-the-art performance) and a second one to see if the new components surpass the state-of-the-art (3.4). During our experiments, we noticed that a major weakness of EAs based AutoML is that they struggle to tackle large datasets (3.5).

Chapter 4 proposes a technique to handle large datasets with Evolutionary-based AutoML solutions (4). The technique is based on Successive Halving, and was introduced on the Multi-armed Bandits problem (4.1.2). After adapting it to EAs-based AutoMLs (4.1.3), we perform an experiment (4.2) on small and large datasets to observe how it behaves (4.3). On small datasets, there is no significant improvement. However, on large datasets, our solution converges much faster than the state-of-the-art, while maintaining roughly the same final performance.

Chapter 5 continues the modular work initiated with Mary-Morstan. With the emergence of new machine learning algorithms dedicated to time series classification problems (2.1.1.3), we decided to solve it with our AutoML, which to the best of our knowledge has never been tackled. We first introduce the problem of time series classification (5.1). We then describe the components of Mary-Morstan permitting to tackle the problem (5.1.5). Finally, we propose an experiment on classical datasets used to benchmark the time series classification algorithms (5.2). It shows that our solution significantly outperforms existing AutoMLs straightforwardly adapted to the time series classification problem (5.3).

Chapter 6 summarizes all the above contributions (6.1), and gives a panorama for the future works (6.2).

Chapter 1

Industrial Context

In this chapter we introduce the company OVHcloud and how the company started to have an interest in developing Artificial Intelligence products, which led to the development of these thesis works. Then, we enumerate three sideline projects that we developed to support this thesis work.

1.1 OVHCloud: the company

OVHCloud (previously named OVH) is a company founded by Octave Klaba and his family in 1999. The company initially provided few services (web hosting, domain name registrar) and expanded to many more offers since (dedicated server, public cloud, private cloud, VPS, VOIP, database as a service, AI/GPU environment, etc.)[1].

The company is known to be among the larger ones in the world, and the largest cloud provider in Europe [1]. This can be easily shown through its presence[2] in the world (see Figure 1-1). With **33 data centers** located in **12 locations** across 4 continents, a **bandwith network of 22 TBPS**, and more than **400,000 servers** running, it is also a world-wide cloud competitor. Indeed, the company totalizes **1.6 million customers** across **140 countries**.

OVHCloud is also an **innovative company**. As a matter of fact, most of the **servers**

[1] see https://www.ovhcloud.com for more details
[2] data center, office, Point of Presences (PoPs)

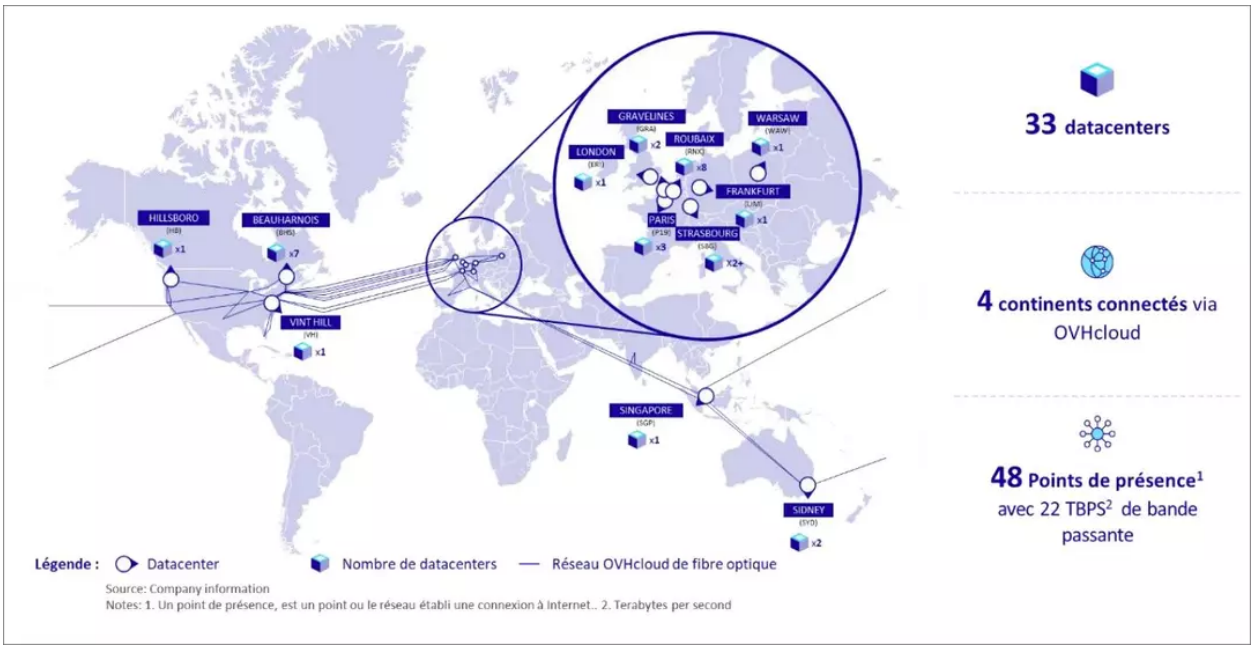


Figure 1-1: Presence of OVHCloud in the World (2021)

are **built** and **mounted** in a **dedicated factory**[3] (see Figure B-1), which permits to build on demand architectures[4] and to reduce the production costs.

Its strong track record of **innovation** continues in the data center with the **racking** and the **cooling** part where multiple patents have been deposited. The racks have been designed to be quickly mounted in production (instant power, network, and cooling), and oriented for the maintenance. The proprietary water-cooling technology reduces the need for energy, and consequently the cost of electricity for the final customer. Compared to most of the data centers that have air conditioning, it is much more ecological to use water-cooling.

More recently, **Artificial Intelligence** is another key innovation for the development of efficient technologies [110]. To continue its quest of innovation, OVHCloud contributed in this field in various ways: development of **new products** for the customers (**Prescience, ML Serving, AI training**), resolution of **internal problems** (**fraud detection, network overheat monitoring**[5]**, water block anomaly detection**), and the **research work of this thesis in the field of AutoML** (**international papers and patents**).

[3]https://blog.ovh.com/fr/blog/nouveau-site-de-croix-une-usine-world-class-pour-la-production-dovh/
[4]type of cpu, amount of memory, type of storage: hdd, ssd, nvme
[5]https://blog.ovhcloud.com/network-devices-overheat-monitoring/

In the next section, we first focus on the Artificial Intelligence related products which, then lead to the development of this thesis along with its sideline projects.

1.2 Solutions Related to Artificial Intelligence

In this subsection, we first motivate the interest of OVHCloud in Artificial Intelligence and more specifically in Machine Learning, then we give an overall view of the different products that have been made from this interest.

1.2.1 Motivation

The usage of Artificial Intelligence at OVHCloud took its root in 2017 with the **Machine Learning Services (MLS)** team composed of two people: Guillaume Salou and Christophe Rannou.

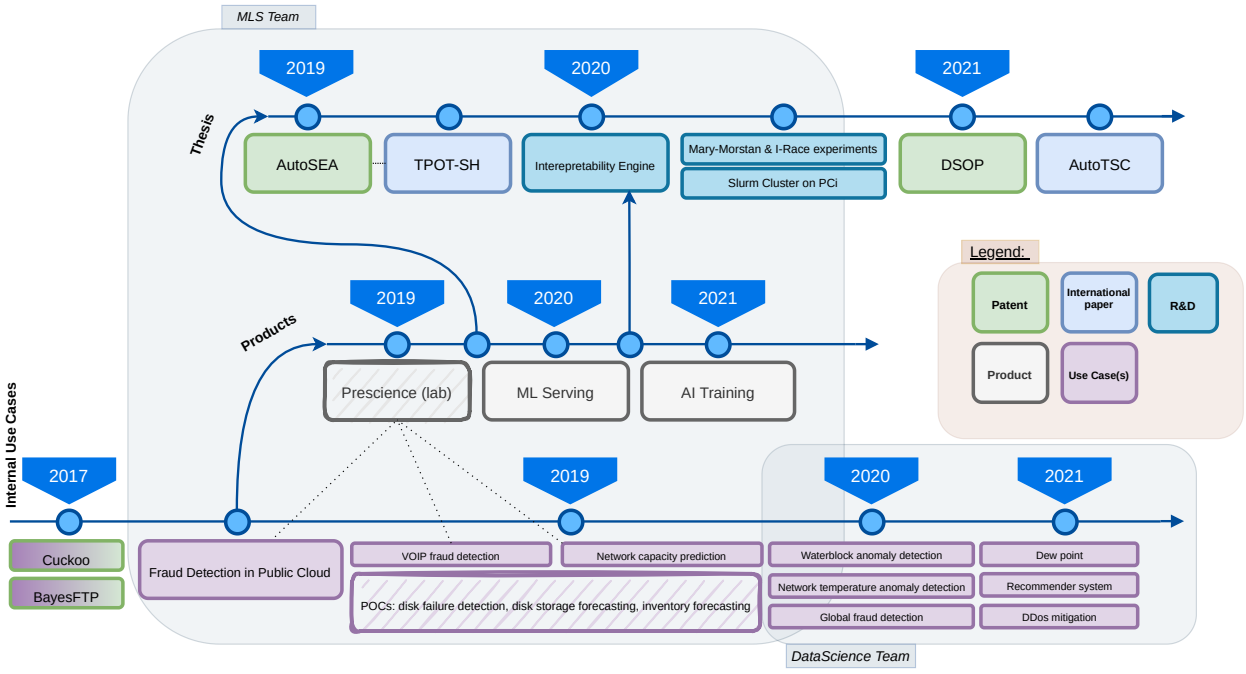


Figure 1-2: OVHCloud contributions and usage of Artificial Intelligence

A timeline of OVHCloud contributions is provided in Figure 1-2.

The MLS team noticed that the developers of the Public Cloud (PCi) product had

difficulties to catch the fraudsters. The product allowed the customers to consume the infrastructures (compute, memory, storage) instantly on credit and they generally paid at the end of the month. Thus, the fraudsters found an interest in consuming the service for free until the end of the month. As a countermeasure, the developers of PCi implemented some rules, e.g. if a new customer consumes more than 200$ within the first few days, then he might be suspicious. However, these rules were too generic to catch the different profiles of fraudsters, and required a lot of updates (push code in production) to be constantly efficient. For this reason, MLS started to develop a new service (API) based on Machine Learning.

Machine Learning has the capacity of building a model thanks to a dynamic function which adapts itself to the input data (see 2.1.1 for more details). In other words, for this case, the model can catch the different profiles. Once built, it can serve to predict if a consumer is a fraudster (or not) with a certain probability.

The setup of the new ML service was a success. It considerably reduced the number of fraudster on PCi as well as the lost expenses. However, it required a lot of effort to be efficiently maintained as well. It happened that new models, when pushed in production, incured an unexpected performance drop or worse, completely failed to make predictions, leaving an open door for the fraudsters. The issues were due to many reasons. Sometimes, it was due to modifications made in the information system (e.g. move of data to another place, the appearance of new data which were not correctly formatted). Sometimes, the pre-processing phases were wrongly built. And sometimes, the ML algorithm used was incapable of giving great results with the selected features.

Following these observations, the MLS team realized how laborious it is to maintain and put a Machine Learning model in production. Thus, they started to have an interest in building an automatic tool that eases the use of Machine Learning. At the same time, they found out that a related research field called AutoML (Automatic Machine Learning) existed. However, the tools were very research oriented and therefore not ready for productive environments. That was how MLS team started to build a new tool, called **Prescience**, which was much more production ready[6]. **At the same moment, we also initiated the work of the thesis, which consisted in studying the tools present in public**

[6]available at https://labs.ovh.com/machine-learning-platform

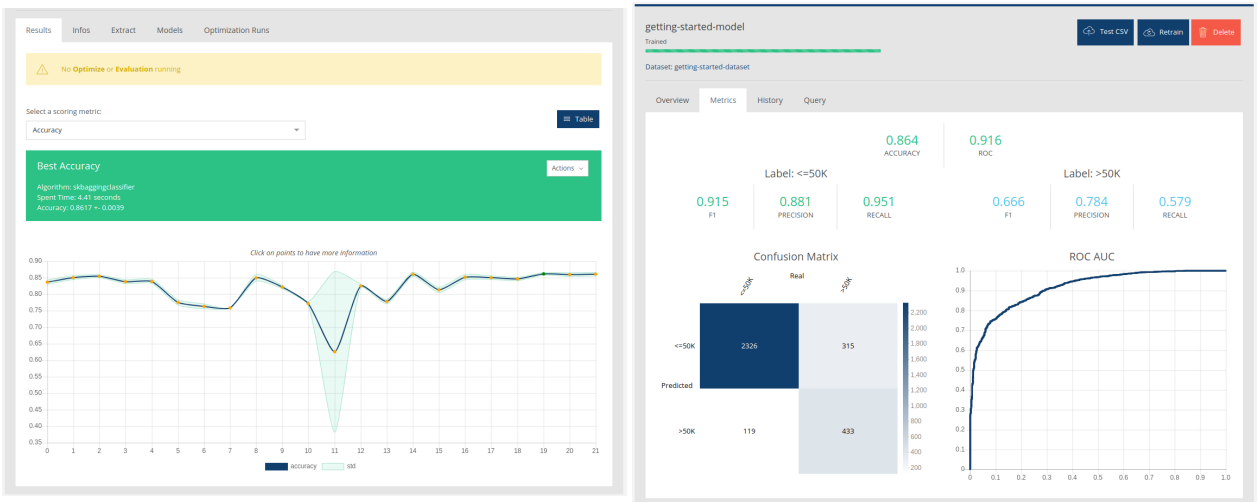
research.

Note that along the time, the MLS team grew (from two to a dozen) and the major objectives started to differ. For this reason, the team was split in 2021 such that the MLS team focused on the development of AI products and that new Data Science team took charge of internal use cases related with AI.

1.2.2 Prescience

Prescience[7] is a Machine Learning platform that can be used through a **web user interface**, or a command-line interface. The user simply deposits their dataset in the allowed format (CSV, Parquet, Warp10) and selects the problem to solve (regression, classification, forecasting), and all the remaining steps (preprocessing the data into features, finding a model) are automatically performed.

The **strength** of the platform lies in its **straightforward usage**, its ability to distribute the compute, which **accelerates the optimization** process to **find a model**[8] (see Figure 1-3a), and in its **visualization** of the performance for a given model (confusion matrix, ROC curve, and all other metrics: F-1, precision, recall: see Figure 1-3b).



(a) Optimization that looks for the best model.

(b) Model's performance wt. different metrics.

Figure 1-3: Prescience Web-UI interface.

[7]https://blog.ovhcloud.com/prescience-introducing-ovhs-machine-learning-platform/

[8]selection of the ML algorithm and tuning its hyper-parameter. It is based on a distributed version of SMAC (Bayesian Optimization)

While it might be straightforward to use at first-hand for a non-expert, it remains difficult to understand the final performance indicators, or how to properly use a deployed model. On the other hand, an expert might be frustrated by the lack of methods and customization (e.g. preprocessing phase), which limits the performance of the optimization and by consequence the performance of the deployed model. Therefore, the platform has not been proposed as a product yet, and is only available on the lab of OVHCloud.

Through the development of this platform, MLS remarked that building all the steps of an AutoML in detail is a massive job. For this reason, the MLS team started to focus on one particular step of Prescience that showed a great interest: the usage of a model as a service. This led to the development of a new product called **ML Serving**[9].

1.2.3 ML Serving

ML Serving[10] consists in deploying a **model as a service**, in other words, as an API on top of a model (Figure 1-4), with the benefit of the cloud (**scalability, availability, pay-as-you-go**). By doing so, the **practitioner** is free to **build** the **model with their preference (language, library)**, as long as it can be exported in one of the formats allowed by the platform (PMML, ONNX, HDF5, Tensorflow SavedModel). These formats aim to export/import a model from a platform to another.

Once the model built, the practitioner can deploy its model by **choosing the closest region** to its customers (Europe or America), the **hardware configuration** (scalability: **memory, CPU**) which permits to **adapt** to the **requirements** of the **model** (e.g. size and complexity to predict) and to the **needs of the customers** (e.g. how often the model will be called). The product also integrates a versioning feature which permits the customer to choose which model to deploy on the service. This is especially useful when a previous version of the model had better performance than a new one deployed.

Along with the enumerated features, there are the advantages of the cloud. The **customer does not have to constantly secure** its **service**, it is done by the team that updates the different internal services (e.g. Nginx proxy), **nor to maintain the infras-**

[9]https://www.ovhcloud.com/en/public-cloud/machine-learning-serving/
[10]https://blog.ovhcloud.com/serving-engine-a-cloud-based-tool-to-deploy-machine-learning/

tructure. If a part of the hardware is broken, the model is not impacted and is automatically deployed on another machine (high availability).

Another great advantage is the pay-as-you-go, which allows the customer to only pay the time that the model is deployed. Thus, it avoids the customer to buy all the equipment which is costly and that usually requires some maintenance. Neither does he have to rent a service during the whole month while it is partially used along this period of time.

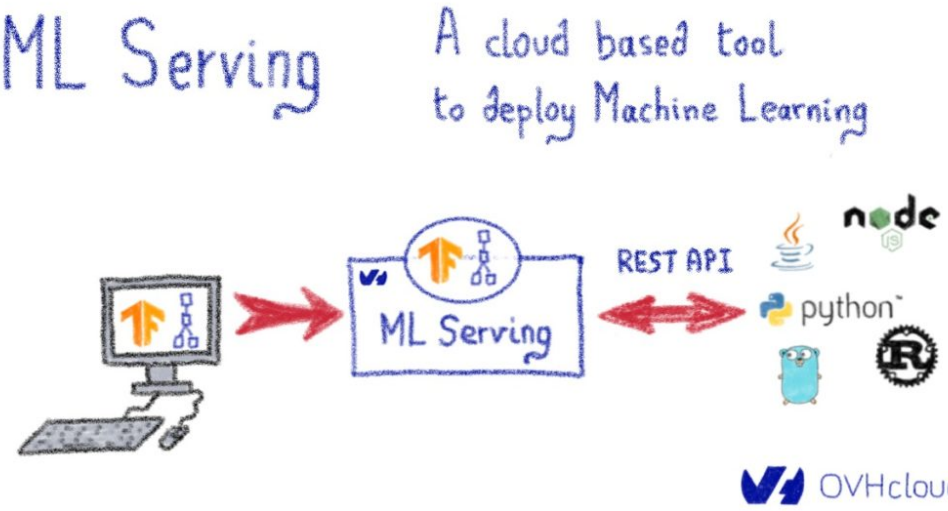


Figure 1-4: ML Serving. A model is locally build and then exported in the service. Finaly, any user can interact with the model through the service with their favorite language.

1.2.4 AI Training

This decade, Machine Learning is in high demand [110], notably because of its great performance, especially to handle a large quantity of data that have specific structures (images, videos, and sounds). In accordance with the market, comes **AI Training**[11], a GPU environment that permits to quickly **train Deep Learning models without having to deal with the installation and the configuration of the drivers (neither to maintain the hardware)**.

Like ML Serving, it includes all the benefits of the cloud. So it is possible to **scale** the platform **(choose the number of GPUs)** while keeping an highly available service (no

[11] https://www.ovhcloud.com/en/public-cloud/ai-training/

failure, possibility to choose a region). The goal of the platform is to ease the tasks for the data scientists by letting them focus on their primary job which is to build accurate models and not to manage all the trimmings.

Moreover, the proposed GPU (Nvidia Tesla V100s) is quite expensive to acquire, but with the pay-as-you-go principle, the customers can rent it at their demand. By doing so, a customer can optimize their costs and keep the advantage of having a powerful compute environment.

AI Training is not limited to the GPU, and can also be scaled with CPU, which is quite useful for more classical algorithms. Besides, a web-ui notebook (Jupyter) can be linked to the platform, and permits to directly develop the models on it.

1.2.5 Conclusion

To conclude this section, OVHCloud took advantage of its internal use cases to build innovative and visionary products that perfectly line up with the needs of the market (see Figure 1-5). By going from the storage of the data (OVHCloud Object Storage), to the deployment of a model (OVHCloud ML Serving), plus the in between products (OVHCloud Data Processing, OVHCloud AI Training), the whole process of extracting value from the data can be chained with all the benefits of the cloud.

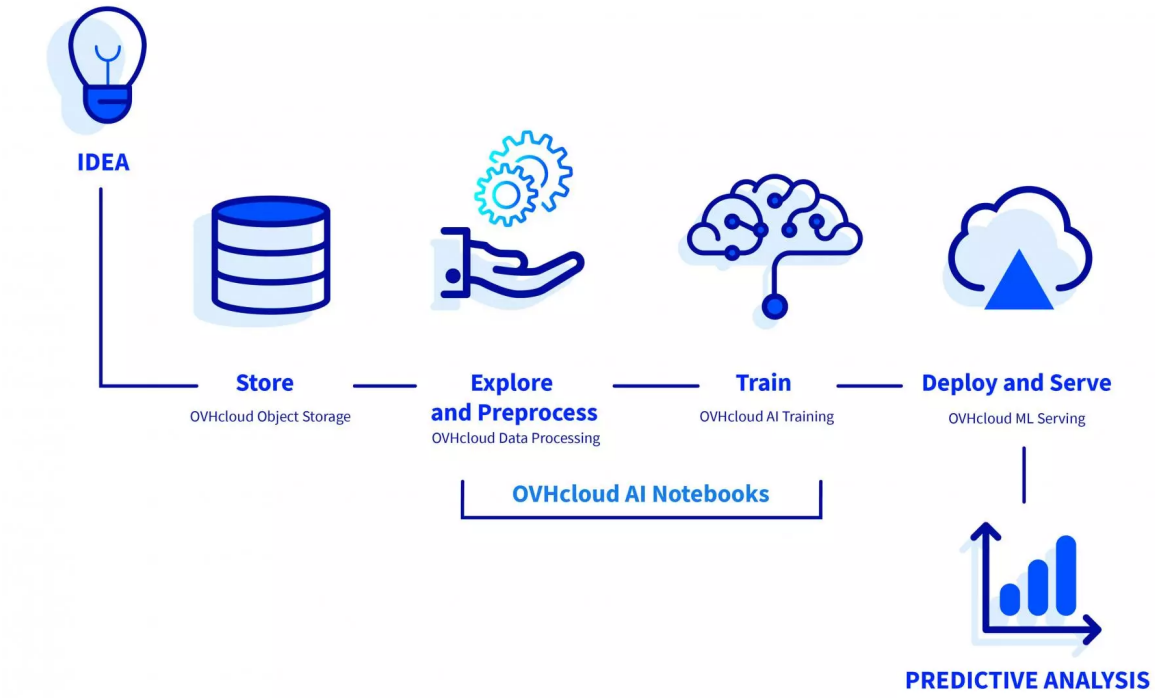


Figure 1-5: OVHCloud AI products aligned with a Data Scientist journey.

1.3 Sideline Projects

This subsection introduces **three projects** that have been developed in parallel of the thesis. There is **DataSet Open Platform (DSOP)**, a platform to ease the retrieval of datasets. **Interpretability Engine**, a library that eases the interpretation of Machine Learning models deployed on OVHCloud ML Serving. And **Slurm PCI**, a set of DevOps tools to manage Slurm Clusters on top of the OVHCloud Public Cloud product which has served to run all the experiments of the thesis.

1.3.1 DSOP: DataSet Open Platform

DataSet Open Platform (DSOP) is the subject of two patents[12]. The project comes from simple observations that Machine Learning **practitioners exchange** and **use** their **datasets with no standards**. This has multiple **consequences**.

First, it is common that practitioners transform the data to their need (e.g. fit with a model, improve the performance), and along the time, these **transformed data** might be **shared** and re-shared with no trace of their origin. If such a dataset is used during a **research experiment**, it could lead to **biased results**. Indeed, either the results are better thanks to the replacement of information, or worst due to the loss of information. The **obscure transformation** of the data has another consequence, the **difficulty of being reproducible**, which is in contradiction with the principle in research.

Second, practitioners tend to use multiple datasets (e.g. for different use cases, to merge different datasets as a final one). These **datasets** are usually **present on different platforms** (UCI, OpenML.org, Dataturks, datahub.io to name few public ones). **Each platform** has its **own encoding** (raw, CSV, JSON, tar.gz, etc.) of the data, and its **own interface of communication** (library, REST API). This **extra layer** of tasks takes the **practitioner away from its main task**.

Third, and partially related with the previous point, **a same dataset** might be **present on multiple platforms**. Which is great if one **platform is down**, another can be used (**failover**) or to get the data from the closest point (**accelerate the download**). On the

[12]EP21305706

other hand, as announced earlier, the process of getting the data might completely change from a platform to another (e.g. encoding, interface of communication).

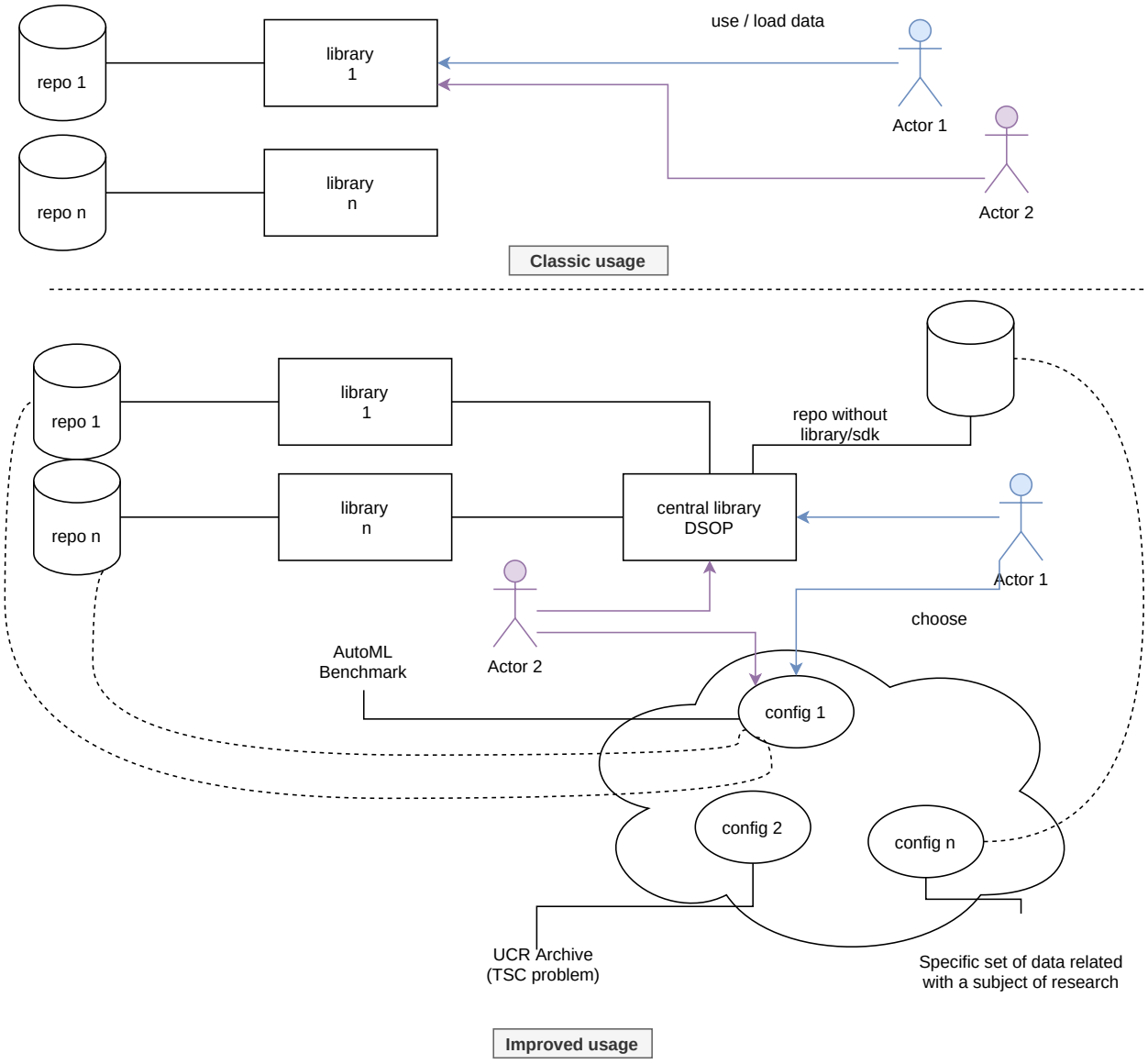


Figure 1-6: DSOP overall view (classical usage versus improved).

To **solve** all the **above problems**, we noticed that no tools exist so we decided to develop our own that we called DataSet Open Platform (DSOP). **DSOP** (Figure 1-6) eases the process of getting the data and works as follows: the user defines a configuration file (see the example in Figure B-2), which can be later shared with others. This file contains a list of datasets where for each dataset a backend is specified with its parameters (url,

authentication) and optionally an extractor (the process of decoding and/or transforming the data). The controller then reads this file and loads the needed data into a general class representation with a common interface. Thus, we have a tool that traces all the history of the data, ensures that the transformation of the data is reproducible, and can even use a backup platform to maintain a productive environment.

To highlight the advantages of DSOP in practice, we provide two examples of code (with and without our tool) in Figure B-3.

1.3.2 Interpretability Engine

There are more and more different **Machine Learning algorithms**, and each of them **extracts the knowledge** from the data (builds the model) in **different manners**. Some models are pretty obvious to read, e.g. Decision Trees, while some others are more complex, e.g. Deep Neural Networks. However, even if some algorithms have a predisposition to be easily readable, they might become complex as well when they are fed with a large quantity of data. Thus, a new field of research which consists in **explaining (or interpreting) complex models** has emerged [6, 41, 78, 84, 86, 104, 117]. This bunch of algorithms aims to help practitioners to **debug** their **models**, and non-practitioners to **understand and trust** the **decisions made by the models** which are generally considered as **black-box functions**.

With the development of **ML Serving**, we thought it would be **great** if customers could get more **insight on their models**. To do so, we[13] first analyzed the existing methods in order to get an idea of their functionalities and their complexities (see 1-7).

It appeared that Partial Dependence Plot (PDP) [41] was the most interesting method to integrate in a first place. This method has a low complexity and gives a global insights on the model. We started to implement Interpretability Engine[14], a library capable of using PDP and other methods in the future with the models that are deployed on ML Serving.

Interpretability Engine works as follows. You first define the model to query (token authentication and deployment url), select the method of interpretation, and specify the

[13] this work has been conjointly done with Etienne Levecque
[14] https://github.com/ovh/interpretability-engine

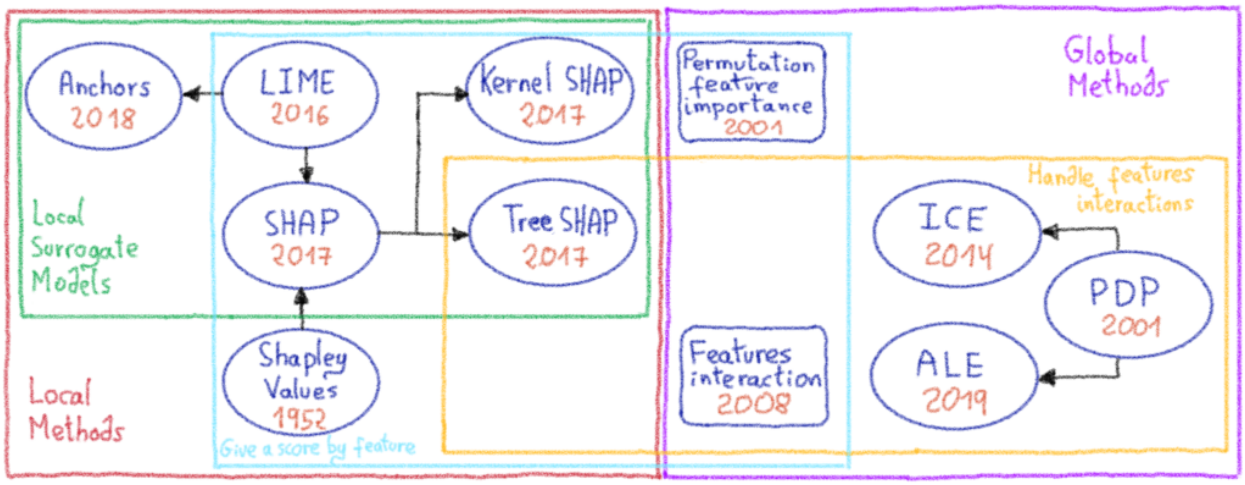


Figure 1-7: A taxonomy on the intepretability methods.

feature(s) to interpret. Note that ML Serving only stores the model, not the data, that is why we require a sample (it can be the training set, or test set) to query. Then, based on the method, the model will be challenged and each of its response analyzed to draw a final result (either in a pdf file, or directly on the screen).

We provide an example of CLI usage (Appendix B) and its partial result (PDP for one feature: sepal width) on Figure 1-8 for the Iris dataset. The dataset is composed of three flower species (Setosa, Vergicolor or Virginica) and four features (sepal length, sepal width, petal length and petal width). As we can see on the figure, higher is the sepal width, the higher is the probability of predicting a Setosa species. The lower is the sepal length, the higher is the probability of predicting a Virginica species.

Thanks to the tool, we now understand how the model reads the features to take its decisions, which becomes much less obscure for practitioners.

1.3.3 A scalable HPC (Slurm) setup on the Public Cloud

It is common for **scientific researchers to need** a **laboratory** to **run their experiments**. In **our case**, the experiments of the thesis consist in **running stochastic algorithms (AutoML)**, which require multiple runs to get significative results. In other words, we **need multiple CPU** to **distribute the runs** and **get the results** in a **reasonable time**.

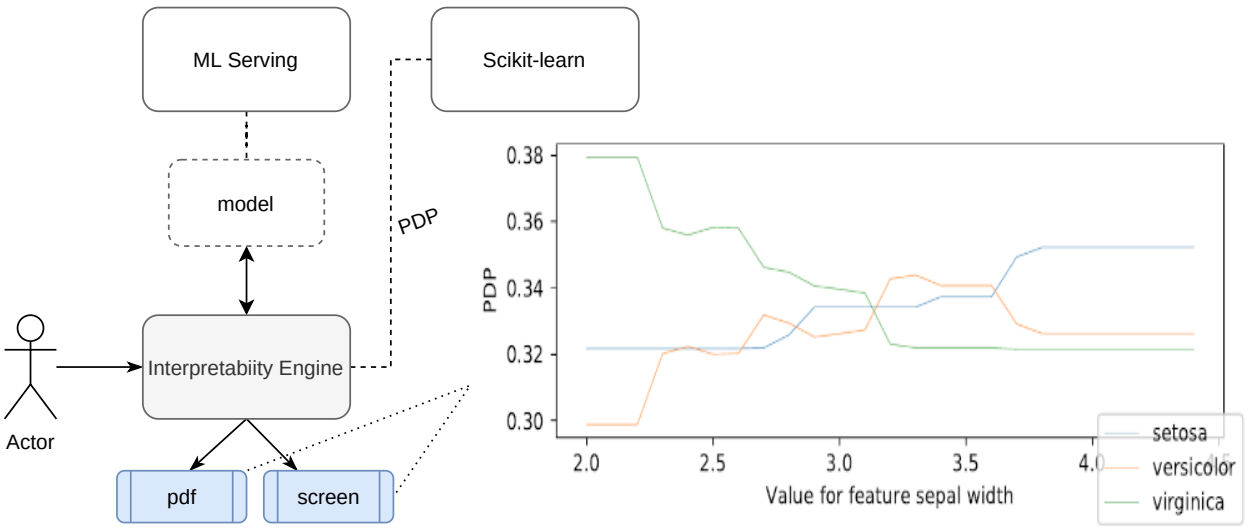


Figure 1-8: Interpretability Engine workflow with Iris dataset as an example and its related PDP result (see B).

There are multiple infrastructures that are candidate to solve the above problem:

- ORKAD's Slurm cluster, composed of 112 cores.
- Grid5000 [9], a Slurm cluster composed of 15 000 cores.
- And OVHCloud (Public Cloud, Baremetal), the number of cores depending on the budget. No HPC solution.

Before choosing the infrastructure, we need to remind the characteristics of an AutoML. It is known to be very costly. It requires a long run time as well as a lot of resources (CPU and memory per job). To give an example, a run of an AutoML tool easily takes a dozen minutes if not hours to see its performance converge on a single dataset (see experiments of the thesis, e.g. 3.4.2). And it easily goes to multiple days with more complex datasets (number of samples, number of features) or more complex search spaces (Machine Learning algorithms that are costly to train).

On ORKAD's Slurm cluster, we are multiple users, and the AutoML experiments would easily monopolize the cluster. Moreover, AutoML solutions are written in different languages, with different libraries, and require a lot of adaptation with the current setup of the cluster. Also, the AutoML benchmarks take a lot of resources in terms of space (memory and hard

drive). Lastly, and a major reason we wanted to avoid this cluster, is its instability. It happened that the cluster went down due to moves or blackouts, which is very problematic for AutoML experiments that typically need long runs, and the majority of the AutoML in the literature are incapable of starting an optimization from where it stopped.

Concerning Grid5000, there is a policy that tends to limit the number of jobs, the resources per user, and the time that a job can run (wall time)[15], which makes the cluster not praticable for AutoML experiments.

For these reasons, I decided to take benefit of my DevOps expertise along with the products of OVHCloud to have an HPC platform without these limitations. The choice was rapidly made: OVHCloud Public Cloud (internally abbreviated PCi) and Slurm. Hence the project name **Slurm-PCi**.

OVHCloud Public Cloud is known for its nice features:

- Add and remove instances in seconds (no wait to add resources such as CPU, memory, and hard drive storage).
- Choose the category of instances (memory usage, compute usage, in between) and its region (Europe, USA, Asia, etc..).
- Secure the communications with VRack[16] (plus the possibility to interconnect with other internal products, e.g. OVHCloud Cloud Databases, a database that scales for the needs of Slurm).
- The possibility to communicate with the API (permits to automatize the setup of the instances with Terraform).
- Pay-as-you-go: control the expenses (no idle nodes).

Slurm is open source and aligned with the value of the company. It is also well known to be robust, and runs on one of the biggest HPC clusters, the Tianhe-2 with 16 000 nodes and 3.1 million cores.

[15] https://www.grid5000.fr/w/Grid5000:UsagePolicy
[16] similar to a VLan

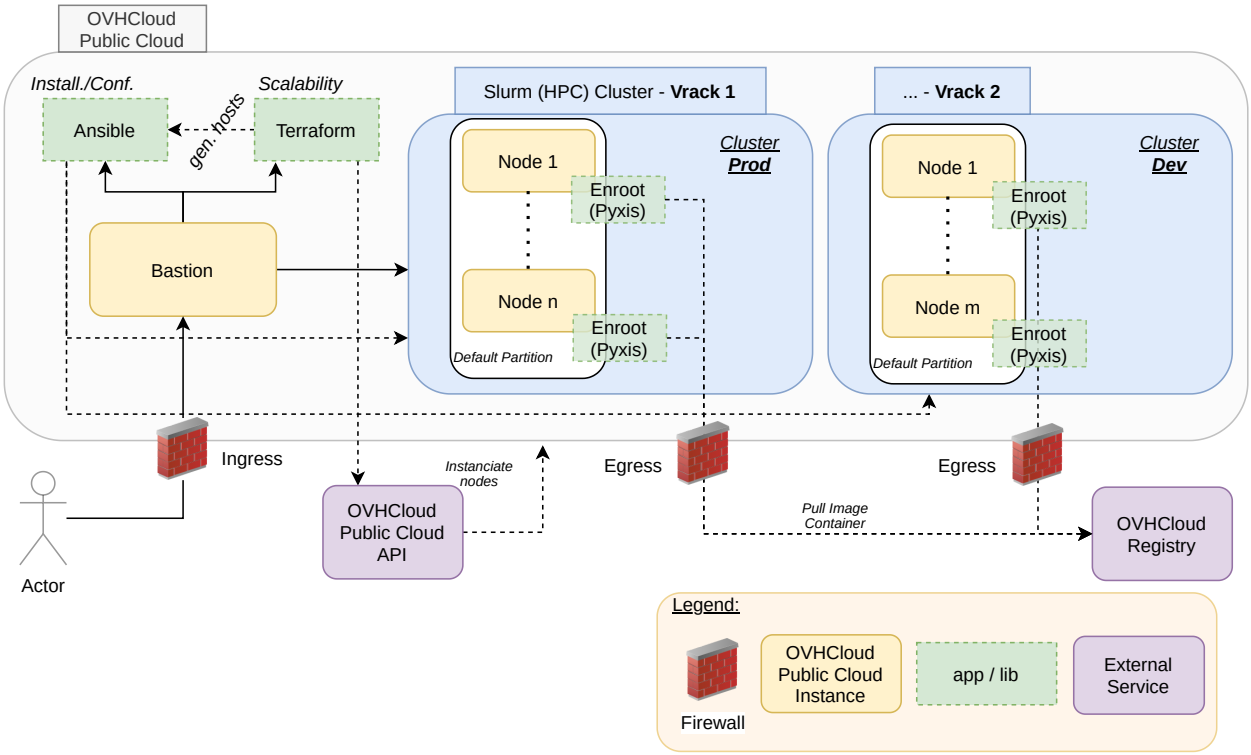


Figure 1-9: Architecture of a Slurm (HPC) on top of OVHCloud Public Cloud.

Slurm-PCi[17] works as follows (see Figure 1-9): there is a bastion node that controls the ingress traffic and serves to manage the Slurm cluster(s) thanks to Terraform and Ansible.

- **Terraform interacts with the OVHCloud Public Cloud API to set up the instances** (Operating System, hardware requirements, VRack ID), and generates the files later used by Ansible in order to know the different nodes.
- **Ansible installs and configures Slurm on the nodes along with the different libraries**.

Due to the differences of requirements between the AutoML tools, I decided to have a container capable solution[18]. Among Docker, Singularity, Podman and Enroot [3], I chose the latter for its native support of Nvidia GPUs which might be a great advantage for future experiments with Deep Learning models (e.g. AutoDL).

[17] will be published on https://github.com/ovh/
[18] https://slurm.schedmd.com/containers.html

An example that adds a node is given in Figure B with its associated command in Figure B. It shows how easy (almost no human intervention) it is to scale the HPC cluster.

As we can see, the combination of a Cloud structure with Slurm makes a great laboratory for researchers. **It permits to have a redundant, highly available, scalable, and costly manageable HPC Cluster**.

Note that some limitations exist and might make some researcher reluctant to use such a solution. For example, there is no shared memory between nodes, which is sometimes used by MPI[19] practitioners. Indeed, it requires specific equipments and it is not needed by a majority of cloud customers for now, which is why there is no such a feature yet in this project.

A closing remark concerns Kubernetes. Along with the thesis, I tried[20] the platform to run some experiments, which is pretty nice to have features that Slurm does not have, e.g. Web-UI that monitors all the resources, integration with other microservices. Nevertheless, it was not practical to run thousands of jobs with different resource requirements. For example, it was not possible to connect a CPU to a specific job all along its run, which started to bias the experiments due unbalanced resources. To go further on this topic, the reader may refer to [103].

[19]https://www.open-mpi.org/
[20]https://kutt.parmentier.io/wqZKhk

Chapter 2

Automated Machine Learning

This chapter motivates the interest to study the field of the Automated Machine Learning (AutoML). We first introduce the concepts of Machine Learning and highlight the problems that are encountered by their practitioners. These problems are then summed up under one unique combinatorial problem which leads to what we call the Combined Algorithm Selection, Hyperparameter optimization And Preprocessing selection problem (abbreviated CASHAP). We then enumerate the potential solvers for CASHAP along with the ones that have been studied in the literature. Finally, we depict the state-of-the-art with their pros and cons which shall help us to get a direction on the development of a new AutoML solution called Mary-Morstan.

2.1 Background

In this section we first recall what Machine Learning (ML) is intended for and the way it is usually handled by its users. Its usage reveals some tedious tasks that we formally define as the CASHAP problem and that can be automated, hence the birth of the term: AutoML.

2.1.1 Machine Learning

2.1.1.1 Concept

Machine Learning (ML) is a domain where mathematical **function**s are **dynamically built** from data in order to find relationships.

The construction of these functions is usually separated in **two phases** as shown in Figure 2-1: a first one called *training* which consists in fitting a *model* from the *features*, and a second one called *testing* that compares predictions made by the *trained model* on data not used for training, to the actual data (or values). Features[1] are preprocessed[2] data that can be efficiently handled by ML algorithms.

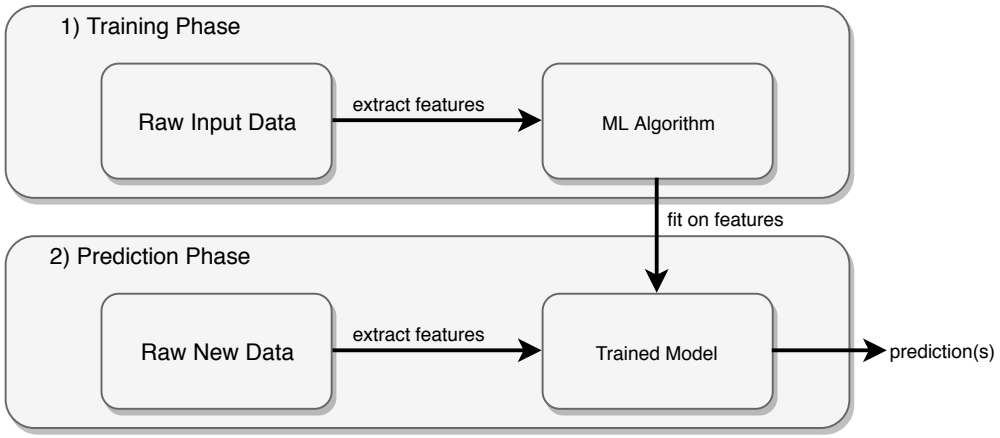


Figure 2-1: Simplified Machine Learning workflow

We distinguish two different approaches to *learn* and *predict*:

- **Supervised learning**: involves part of data used as *known predictions*[3].
  Predictions can be contrasted in two classes:
  - Classification: when the target variable (called the label) is categorical[4]
  - Regression: when the target variable is numerical

[1] also known as attributes, or variables
[2] data which has been selected, cleaned and encoded
[3] also called *target variable*
[4] i.e. finite set of values

- **Unsupervised learning**: when no variables are used as a *target*. This kind of algorithm regroups the closest points together into *clusters* and predicts in which cluster belongs a new observation[5].

Note: There are other subfields of ML not represented here: reinforcement learning [113], semi-supervised learning [25], and transfer learning [126]. Like supervised and unsupervised learning, these subfields of Machine Learning rely on the data to shape the content of the function (the model).

Machine learning algorithms present a considerable **advantage** because they **do not require to be explicitly programmed** and are able to construct complex structures which would normally take a lot of time by hand. To illustrate its advantage, we draw a Decision Tree (DT) in Figure 2-2, a famous ML algorithm which can easily be compared to a set of conditions programmable in most of the programming languages.

While a program needs to be updated by hand when the rules have changed, a ML algorithm just need to be retrained on data. This is the case for the DT that will automatically rebuild the whole structure of the conditions and gives a great gain of time, especially if the function has hundreds if not thousands of variables.

However, this considerable advantage of ML algorithms needs some **pre-requirements**, such as the **amount of data** [32, 48], the **quality of the data** [111, 119], and the **right setup of hyper-parameters** in order to get accurate models.

Thanks to new methods of computation, larger storage (e.g. cloud for big data [36]) and different ways of collecting data (e.g. monitoring tools, sensors) we are capable of constructing models that respect some of these requirements [79].

Nonetheless, the **configuration** of **ML algorithms** and the maintenance of the models **remain** a current **issue** that we discuss in section 2.1.2, just after the summary of the best known classification approaches in the two below subsections.

2.1.1.2 Classical Machine Learning Approaches for Classification

Here is a list of classical Machine Learning algorithms used for classification:

[5] also called an instance

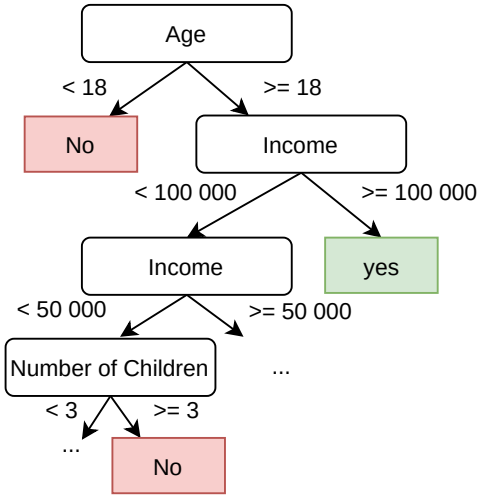


Figure 2-2: Example of a Decision Tree that predicts if a bank can give a loan to buy a house. Each box represents a variable, and each arrow a condition to respect. The leaves represent the possible decisions returned. In that case, different financial parameters tend to change along the years (e.g. inflation), which by consequence require the conditions to be updated. E.g. if the market price of the houses increases, a loan for buying a house would certainly require a higher income.

- **Decision Tree** consists in building a tree of rules where the decision (target value) is given by the leave that returns a decision. Each rule is represented as a node with two branches where each node serves as a variable and its branches as the conditions (left branch if it is less or equal to a determined value, right branch if it is greater). To name a few implementations: ID3, C4.5 [107], CART.

- **Random Forest** [50] builds a multitude of decision trees (ensemble). To get the final decision, it applies a majority voting between the target values returns by each tree.

- k-**Nearest Neighbors** [39] looks for the k closest observations by computing a distance between the variables of a test sample and the variables of the observations. The test sample will be classified as the most present target values among the k closest observations.

- **Support Vector Machine (SVM)** [18]: constructs a hyperplane according to a defined kernel (e.g. linear, polynomial, sigmoid, gaussian radial) such that it maximizes the separation of the targets. Each kernel completely changes the performance of the

model, but also has an importance on the complexity of building it.

- **Boosting** [109] methods iteratively train weak classifiers which are then unified as a final strong one (ensemble). A weak classifier can be viewed as a very simple rule that splits the data on a unique variable.

To get an idea on how the models shape the separation of the data, we draw the Figure 2-3[6].

For further references, the reader may refer to [16, 123].

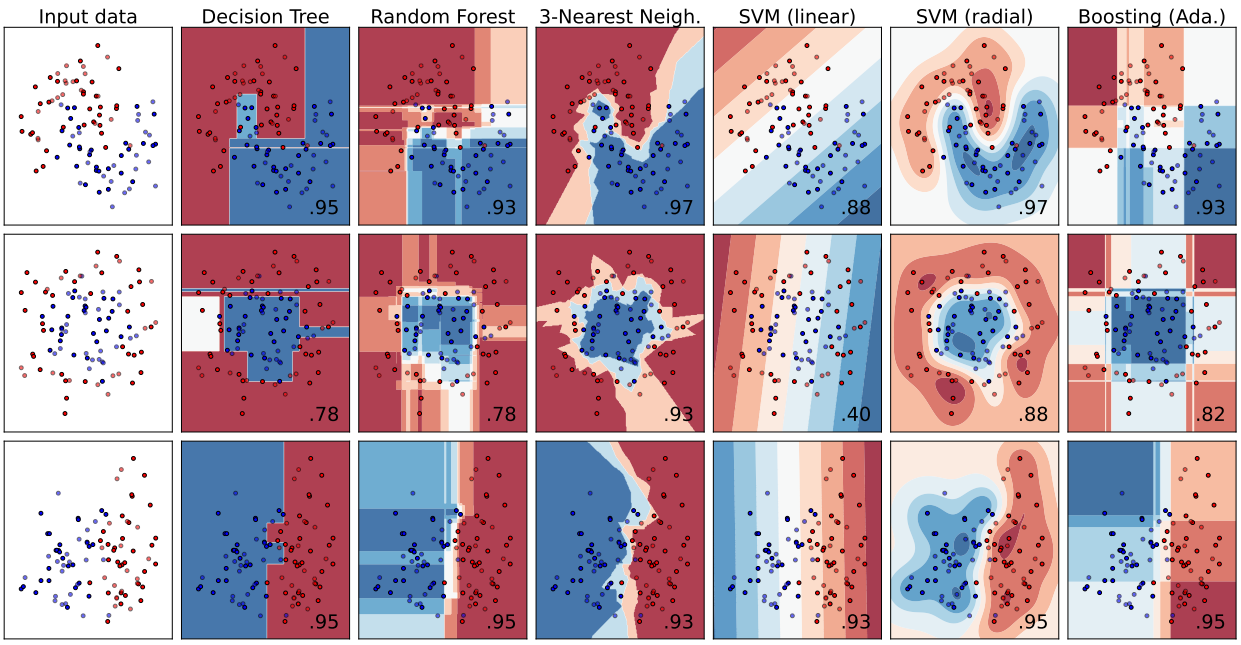


Figure 2-3: A comparison of classifiers trained on three different binary datasets. The datasets have been generated such that the ideal frontier between the two target values respectively looks like a moon, a circle, and a line.

2.1.1.3 Machine Learning Approaches for Time Series Classification

Here is a list of Machine Learning approaches oriented to solve time series classification problems (see section 5.1):

- The **interval-based** [31] approach simply splits the different time series into random intervals and extracts statistical features (e.g. mean, standard deviation) for each

[6]The example as been subsampled from from: https://scikit-learn.org/stable/auto_examples/classification/plot_classifier

interval. These statistical features are then used as tabular variables with a classical ML algorithm.

- The **distance-based** [98] approach consists in computing distances between the series. One of the most used algorithms is the k-NearestNeighbors (k-NN) with the Dynamic Time Wrapping (DTW) distance. The method finds the centroid and computes the DTW distance for each time series within a cluster. It is then possible to train a k-NN algorithm with accurate results.

- The **shapelet-based** [125] approach finds the most representative sub-shape(s) to discriminate the classes. These sub-shapes are called shapelets and serve to extract the main features (e.g. distance to a shapelet) to train a classifier (e.g. decision tree).

- The **dictionary-based** [108] approach consists in building a dictionary of words. Each word represents a shrunk part of the time series. Bag-of-SFA-Symbols (BOSS) is a reduction-noise algorithm that explores this mechanism. BOSS splits the time series into sliding windows. Those windows are represented by words which are sequences of symbols extracted by the Symbolic Fourier Approximation. The repetitive and consecutive words are discarded by the *numerosity reduction*. The remaining words are used to construct a histogram of numbers of apparitions that characterizes the time series. Then a prediction can be made by using a 1-NN, where a distance is measured between the histograms.

- The **kernel-based** [30] approach uses kernels to detect different patterns in time series. The author of ROCKET [30] uses a lot of random kernels with random length, weights, bias, dilation, and padding to extract the features which are then used to train a linear classifier.

2.1.2 Issues encountered by Machine Learning practitioners

In order to properly use the ML algorithms, the practitioners need to go through a series of steps that we depict in the Figure 2-4.

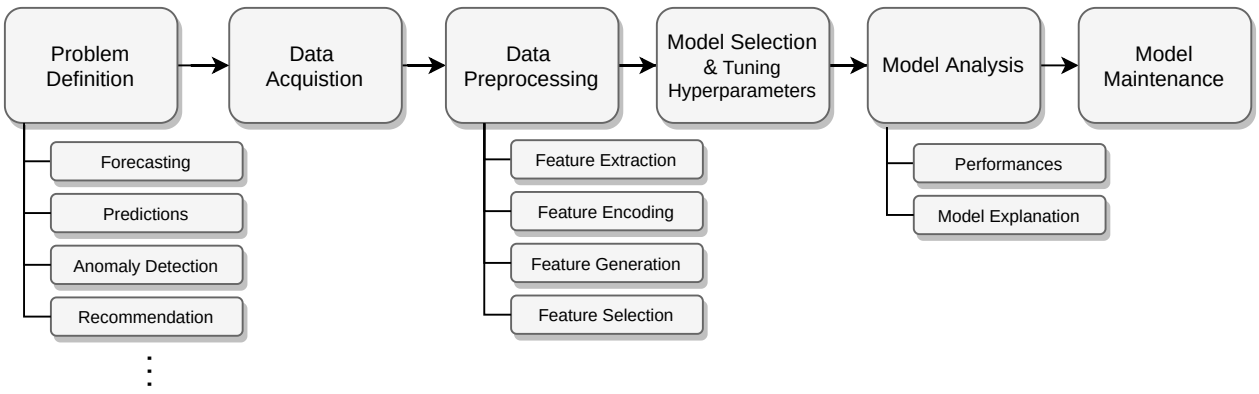


Figure 2-4: Practical Machine Learning workflow.

Among these steps, they first define the **initial problem** that will be solved (forecasting, anomaly detection, etc...). This will determine the data to acquire and the algorithms that are capable to tackle the initial problem. Then, the **data** are **acquired** from the different sources (e.g. relational database, Hadoop file system, sensors). It is followed by a **preprocessing** part which transforms the raw data into **features** (data that can be efficiently used by ML algorithms). We divide the preprocessing phase in four parts:

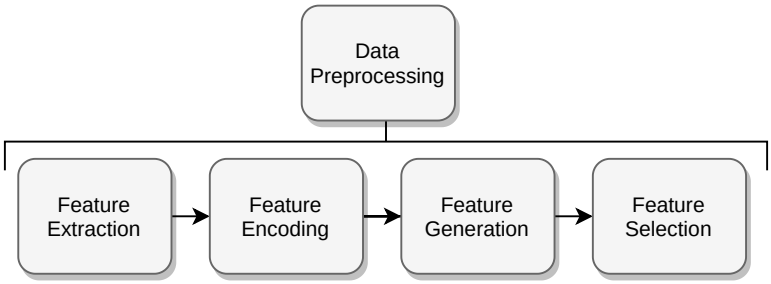


Figure 2-5: Regular Data Preprocessing Workflow in Machine Learning. **Note** that all these steps can be combined in different orders through a pipeline.

- *Feature extraction* consists in retrieving the relevant data to the problem. This part might also include some cleaning, e.g. remove absurd values given by a sensor (negative values that should not appear).

- *Feature encoding* encodes the extracted features such that they are suitable for ML algorithms. For example, it imputes the missing values [57] with the mean, or transforms the categorical features as a one-hot numeric array [100].

- *Feature generation* adds new features [97] by doing some computation such as aggregating the features (e.g. multiply two features), or changing the scale of the data (e.g. logarithm, exponential, polynomial, normalization, standardization). This step helps the ML algorithms converge faster and could avoid overfitting.

- *Feature selection* is applied at the end, just before giving the final features for the learning phase. It consists in keeping the most relevant features or removing some unnecessary ones such that the loss of information is minimized. Indeed, the model still needs to be accurate. This technique helps to reduce the training time, notably by avoiding the curse of dimensionality [120]. It can also be used to ease the explanation of the models by having fewer variables. Another way to reduce the number of features, is to use techniques that build new, more informative features based on the existing ones. The most famous techniques is the Principal Component Analysis (PCA) [94].

Once that practitioners have the features, they need to **select** the most adequate **ML algorithm** that solves the initial problem and **tune** its **hyper-parameters** such that it maximizes one (or more) performance indicators (e.g. precision, recall). After that, they analyze the model for different reasons:

- **model explanation**: gives explanation/interpretation of the results given by the model [84, 87, 104, 117]. It is useful to debug a model and understand the cause of the results. It can also be used to gain trust from non-practitioners who see a model and its results as a black-box function.

- **performance**: one usually checks the performance of the model by evaluating it on a test set which has not been seen during the training phase. In other words, one verifies that the decisions that are taken by the model match with the targets of the unseen samples. This step might also monitor the performance along the time. Indeed, depending on the problem, the data might change (new samples, or known samples with different values in the variables) and might cause a degradation in the performance of the model (bad predictions visible through the false positive rate or false negative rate). The monitoring may serve later to make different actions.

Finally, the **model maintenance** uses the monitored performance of the model to run some actions, e.g. alert the users, train a new model, replace the degraded model by the most suitable one (if a previous one exists), retrain the current one with new samples or with other hyper-parameters.

As we can see, properly using ML algorithms requires to chain multiple steps. Among these steps, some are relatively complex. It is the case with the selection of the ML algorithm which by nature has an unpredictable outcome when not trained and might even act unexpectedly once trained (e.g. Boosting methods are stochastic). This, plus the plethora of hyper-parameter possible values, makes the overall task tedious. While it might seem obvious to settle on small datasets (see Figure 2-3), it is not the case with larger (more than hundred variables and thousands of samples) and scattered ones (see Figure B-4).

Thereby, new tools called **AutoML (Automated Machine Learning)** have emerged [53, 116] in order to automatically solve **the selection of the algorithm** and the **optimization of the hyper-parameters**. Note that very early contributions exist, but they only tackle a unique step (either the selection of a model [19] or the optimization of hyper-parameters [11]), while the concept of AutoML is to aggregate and automatizes multiple steps like stated above.

The development of AutoML tools also leads to AutoML challenges that put them in competition [46, 47, 73] in order to compare the performance of the solutions.

In the next subsection we formally define the AutoML problem that can also be called the CASHAP problem.

2.1.3 Definition of Combined Algorithm Selection, Hyperparameter optimization And Preprocessing selection (CASHAP)

The AutoML problem has been formally defined by Thornton for the first time as the **CASH** problem in AutoWEKA [116]. CASH stands for Combined Selection and Hyperparameter Optimization and has been stated as follows:

$$\mathrm{A}^{\star}_{\lambda^{\star}} \in \underset{\mathrm{A}^{(\mathrm{i})} \in \mathcal{A}, \lambda \in \Lambda^{(i)}}{\arg\min} \mathcal{L}(\mathrm{A}^{(\mathrm{i})}_{\lambda}(\mathcal{D}_t), \mathcal{D}_v) \tag{2.1}$$

Where:

- $\mathcal{A}$ is a set of n ML algorithms
- $\Lambda^{(1)}, ..., \Lambda^{(n)}$ are hyper-parameter spaces respective to each ML algorithm $\mathrm{A}^{(1)}, ..., \mathrm{A}^{(n)}$
- $\mathcal{D}$ is a dataset split in two parts $\mathcal{D}_t$ and $\mathcal{D}_v$
- $\mathcal{L}(\mathrm{A}^{(\mathrm{i})}_{\lambda}(\mathcal{D}_t), \mathcal{D}_v)$ is a loss function for $\mathrm{A}^{(\mathrm{i})}$, using $\lambda \in \Lambda^{(i)}$, trained on $\mathcal{D}_t$ and evaluated on $\mathcal{D}_v$ for $i \in \{1, ..., n\}$

However, this definition omits a major component, the preprocessing phase. Here we introduce a new definition of the AutoML problem where the preprocessing part is included in the definition. We formally define **Combined Algorithm Selection, Hyperparameter optimization And Preprocessing selection (CASHAP)** [92] as the following problem:

$$(\rho^{\star}, \mathrm{A}^{\star}_{\lambda^{\star}}) \in \underset{\rho \in \Phi, \mathrm{A}^{(\mathrm{i})} \in \mathcal{A}, \lambda \in \Lambda^{(i)}}{\arg\min} \mathcal{L}(\mathrm{A}^{(\mathrm{i})}_{\lambda}(\rho(\mathcal{D}_t)), \rho(\mathcal{D}_v)) \tag{2.2}$$

Where Φ is the space containing all different preprocessing methods, s.t. $\forall \rho_1, \rho_2 \in \Phi, \rho_1 \circ \rho_2 \in \Phi$. All others symbols are previously defined in the CASH problem.

In other words, with CASHAP, a solution or a candidate is a ML pipeline represented as a combination of preprocessing method(s) and a single ML algorithm with its associated hyper-parameters.

2.1.4 Metaheuristics Optimization And Definition of Multi-objective Problems

Before going through the state-of-the-art of the AutoML tools (CASHAP solvers), we first give an overall view of the optimization solutions that have the potential to solve the AutoML problem in Figure 2-6. Then, we define the multi-objective optimization problem, tackled by our framework Mary-Morstan (Chapter 3) and some other tools (see 2.2).

AutoML is an optimization problem and can be theoretically solved by exact approaches, i.e. returning the best solution. Nevertheless, the AutoML problem encompasses costly

black-box function(s), and is a combinatorial problem (see 2.1.3) which makes the problem too computationally intensive to be solved in a reasonable time with these approaches. Thereby, the AutoML tools are focused on heuristic approaches, by looking for good solutions. To the best of our knowledge, all the solutions rely on metaheuristic [26] methods that are capable to handle any problem, and are all global-based methods. Contrarily to local methods that only focus on exploitation, i.e. they use unknown regions, the global search methods will also use the unknown regions (called exploration) and will play with a trade-off between the exploitation and the exploration.

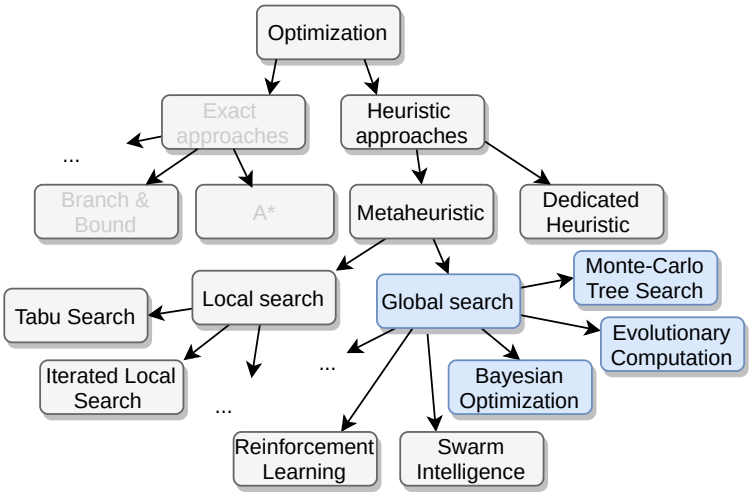


Figure 2-6: Distinction of optimization methods in AutoML (2021). The presence of AutoML tools is fill in blue.

The depiction in Figure 2-6 should help the reader to better situate the AutoML tools in optimization, and might eventually help for the direction of future works, by notably looking at non-exploited (e.g. Local Search) and promising methods.

In the state-of-the-art, some tools can handle multiple objectives like our proposed tool. Multi-Objective is also an optimization Problem (MOP) with n objectives, each of them associated to a function $f_i \mid i \in [1..n]$ to optimize (minimize or maximize), and mathematically defined as follows:

$$(\text{MOP}) = \begin{cases} optimize & F(x) = (f_1(x), f_2(x), ..., f_n(x)) \\ s.t. & x \in X \end{cases} \tag{2.3}$$

With X the feasible set that represents the *search space*. We also denote Z the set of solutions that represents the *objective space*:

$$Z = \{F(x), x \in X\} \tag{2.4}$$

To have a better representation of these spaces, the Figure 2-7 provides an intuitive mapping of two objectives.

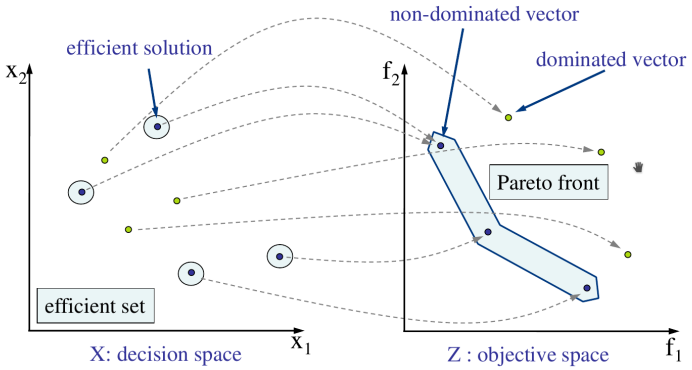


Figure 2-7: Bi-objective representation where f_1, f_2 are the objectives to minimize, and x_1, x_2 are two variables characterizing the solutions.

To compare feasible solutions in the *search space*, the notion of *Pareto dominance* needs to be introduced.

Definition 1 *(Dominance relation of Pareto) Let $x, x' \in X$, a solution x dominates x' if $f_i(x) \leq f_i(x')\ \forall i \in \{1, ..., n\}\ \wedge \exists i \in \{1, ..., n\} : f_i(x) < f_i(x')$. This relation is denoted by $x \prec x'$.*

Definition 2 *(Pareto optimal) A solution $x^* \in X$ is pareto optimal if $\nexists x \in X : x \prec x^*$.*

Definition 3 *(Pareto optimal set) Set denoted $P = \{x^* \in X\}$ containing all pareto optimal solutions.*

Definition 4 *(Pareto front) Set denoted $PF = \{F(x) | x \in P\}$. containing all the images of pareto optimal set through the function F.*

The target of multi-objective optimization algorithms is to get the best Pareto optimal set represented, with a well-converged and well-diversified Pareto front such as shown in Figure B-7. The ideal point is at the origin, and the nadir point is at the opposite.

2.2 State-of-the-art

In this section we enumerate all the different AutoML solutions that tackle the CASH or the CASHAP problem as defined in section 2.1.3.

As depicted in Figure 2-8, we can see the interest of tackling the problem.

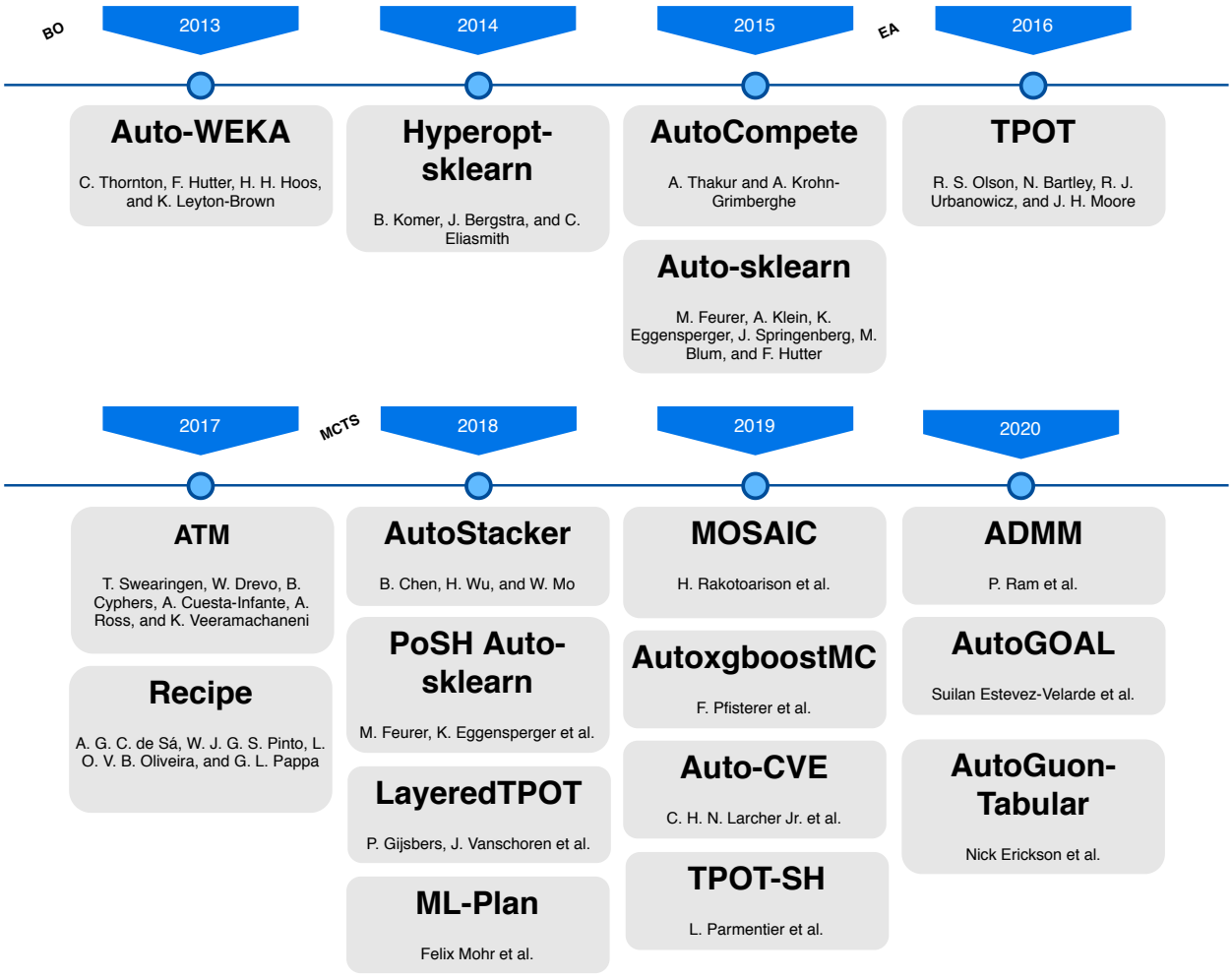


Figure 2-8: AutoML implementations along the years.

To better distinguish the solutions, we regroup them by approach. An approach is characterized by its main core algorithm (metaheuristic) used to optimize the combinatorial

problem. We also order the approaches according to their first appearance in publications. To the best of our knowledge, there are **4 main approaches** that **tackle the AutoML problem**:

- Non-adaptive
- Sequential Model-Based Algorithms (SMBO) [116]
- Evolutionary Algorithms (EAs) [91]
- Monte-Carlo Tree Search (MCTS) [83]

2.2.1 Non-adaptive

The non-adaptive methods are pure exploration methods that do not make any exploitation and by consequence do not take the previous candidates found in consideration. The two most famous methods are **Grid Search** and **Random Search**.

Gird Search consists in enumerating a grid of ML algorithms and hyper-parameters which by association forms a candidate. Then each candidate is trained until the grid is consumed. The method is greedy and can only be achieved on a small search space.

Random Search randomly trains candidates defined in a search space until a budget (e.g. number of iterations, elapsed time, convergence) is reached. Compared to Grid Search, the method demonstrates to be more promising [13]. As depicted in Figure 2-9, a Random Search having the same budget (here the number of trials) as a Grid Search, shows that Random Search spreads better in the search space of configurations. Therefore, it increases the probability to have good configurations.

Both methods are relatively easy to implement and are currently present in two AutoML: **Hyperopt-sklearn** [66] and **AutoGoal** [34].

More recently a new method called **Hyperband** [69] based on Random Search shows that investigating the resources like data samples and features permits to speedup the optimization. The idea has been explored in the SMBO approach [35, 37] that we discuss in the next subsection, and in the EAs [92].

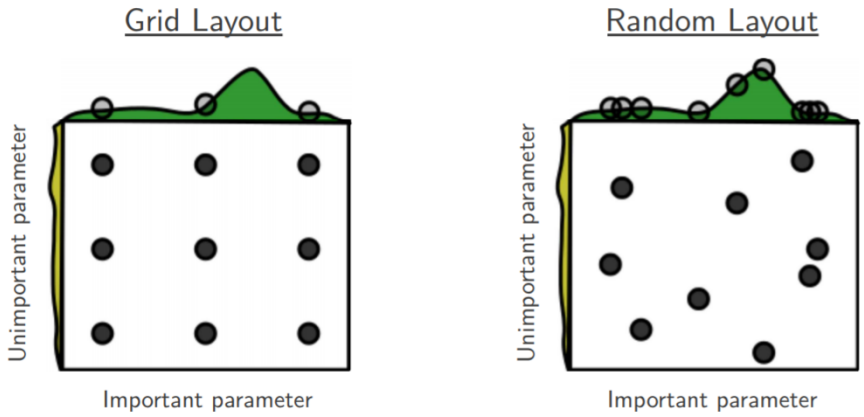


Figure 2-9: Grid Search (left) and Random Search (right) representations on two parameters with their associated distribution performance.

2.2.2 Sequential Model-Based Optimization (SMBO)

The Sequential Model-Based Optimization (SMBO) [51] approach is a Bayesian Optimization (BO) that leans on an **acquisition function** plus a **surrogate model** that represents the posterior. The posterior consists in capturing beliefs from the previous evaluations of the function. The acquisition function uses the posterior to estimate the next promising candidates to consider. These candidates are then evaluated and their performance used to update the posterior.

An example is given Figure 2-10. At $t = 3$, we can see that the posterior returns a large standard deviation between two points that are far from each other and an elevated mean if the observations are higher than the others. On the other hand, if two points are relatively close and low, the standard deviation will be tight and the mean inferior to the rest. As a consequence, when the acquisition function reads the posterior, it will make a distribution with high density on regions with high standard deviations, which can be understood as unknown regions due to the lack of observations. This part helps to promote the exploration. The acquisition function will act similarly on regions that have a high average and few observations, but promotes exploitation of promising and known regions. Exploitation is more visible at $t = 4$, with the new observation from the previous step that updates the posterior and by consequence the distribution from the acquisition which is in favor of the surrounded region.

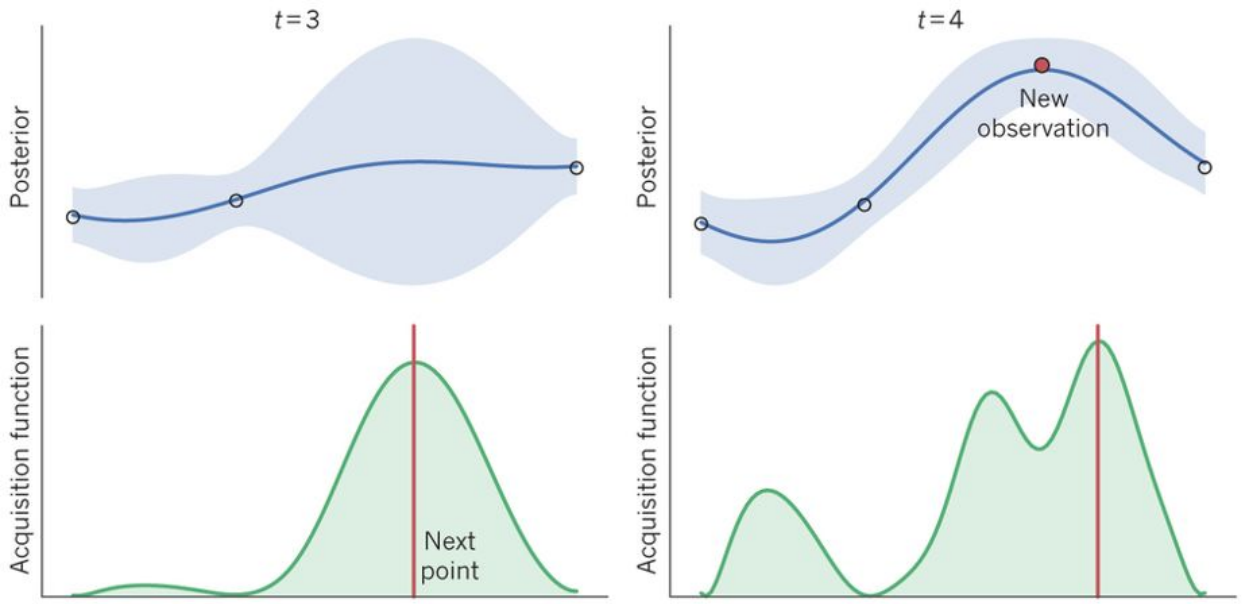


Figure 2-10: Example of a SMBO with 3 (left) and 4 (right) observations. A Gaussian Process is represented as a posterior and the Expected Improvement has been used as the acquisition function. The blue line represents an estimated mean and the blue frame represents an estimated standard deviation.

To give an idea on how the acquisition function balances between the exploration and the exploitation, we provide one of the most famous equations, the Expected Improvement [61] (EI) in 2.5.

$$E[I(x)] = \underbrace{(y^\star - \hat{y}(x))\,\overbrace{\Phi_{0,1}\left(\frac{y^\star - \hat{y}(x)}{s(x)}\right)}^{\text{Penalized by area under the curve}}}_{\text{Promote exploitation}} + \underbrace{s(x)\;\phi_{0,1}\left(\frac{y^\star - \hat{y}(x)}{s(x)}\right)}_{\text{Promote exploration}} \tag{2.5}$$

With $y^\star$ the current best solution known, $\hat{y}(x)$ the kriging prediction (e.g. Gaussian Process), $s(x)$ the standard deviation, Φ denotes the cumulative distribution function, and ϕ denotes the probability density function.

Different implementations of SMBO exist. Each of them differ in their representation of the posterior and acquisition function. Moreover, they tend to add some extra procedures to solve the different cons of Bayesian optimizations. Indeed, the Bayesian optimizations are known to be costly when updating the posterior with many observations. This is notably why the posterior has been implemented with surrogate models. Surrogate models tend to

be less costly when updated and evaluated. The following SMBO methods are currently used in the AutoML:

- **Sequential Model-Based Optimization for General Algorithm Configuration (SMAC)** [52] used by **Auto-sklearn** [38] and **Auto-Weka** [116]. SMAC uses a decision tree regression as a surrogate model, and an intensification mechanism that ensures that the performance of the best candidate is competitive on a growing set of instances against the other candidates.

- **Tree-Parzen Estimator (TPE)** [12] used by **Auto-Weka** [116]. The strength of the method relies in its linear scalability in terms of number of configurations (number of candidates and the number of hyper-parameters) which makes the optimization fast and cheap. To do so, they use $p(x|y)$ with two density distributions and the authors demonstrate that Expected Improvement is equivalent to a ratio between the two densities. The drawback of this technique is that configurations should be uncorrelated to have it work as expected, which is not the case for certain hyper-parameters.

- **Bayesian Optimization and HyperBand (BOHB)** [35] used by **PoSH Auto-sklearn** [37]. A TPE serves as a surrogate model. The novelty of this technique is to handle even more configurations by adding the Hyperband [69]. Hyperband eliminates candidates that perform poorly and gives more resources (dataset subsampling, feature subsampling) to the promising ones at the same time.

- **Bayesian Tuning and Bandits (BTB)** [45] introduced by **ATM** [114]. The idea is to distribute the costly part (cubic complexity in terms of observations) of the Gaussian Processes (GPs). To tackle this issue, they introduce the notion of hyperpartition, which delimits the specificity of each ML algorithm (e.g. of hyperpartition: Decision Trees, Support Vector Machines (SVM) with polynomial kernels, SVM with sigmoid kernels), and build a GPs model per hyperpartition. Then, they use a Multi-Armed Bandit (MAB) algorithm (UCB), to know which hyperpartition should be selected and by consequence which surrogate model should be used. In this manner, they limit the complexity of GPs, except if the majority of candidates fall in the same surrogate model, i.e. an hyperpartition dominates the optimization.

2.2.3 Evolutionary Algorithms (EAs)

The Evolutionary Algorithms [33] have a pretty simple workflow that encompasses the main following steps: **Evaluation**, **Selection**, and **Reproduction** within a loop. A global illustration is given Figure 2-11 and a simple algorithm in Appendix 3. Like any other optimization, it generally starts with an initialization phase which creates some individuals according to a distribution. The individuals are then evaluated. In Machine Learning it generally consists in running an evaluation strategy (e.g. holdout, K-fold). Some extra evaluations can be made in order to measure an individual's performance from other aspects [4, 60, 81]. The evaluations are then used by the selection process in order to select the individuals that pass to the next generation. Lastly, the eligible individuals go through the reproduction process. This process includes variation operators that plays the role of exploration versus exploitation by modifying individual attributes. The operators are commonly called mutations for the ones that perform on one individual, and crossovers for those who perform on two or more individuals. Once the reproduction process is done, it goes back to the evaluation phase and continues until a termination criterion is met.

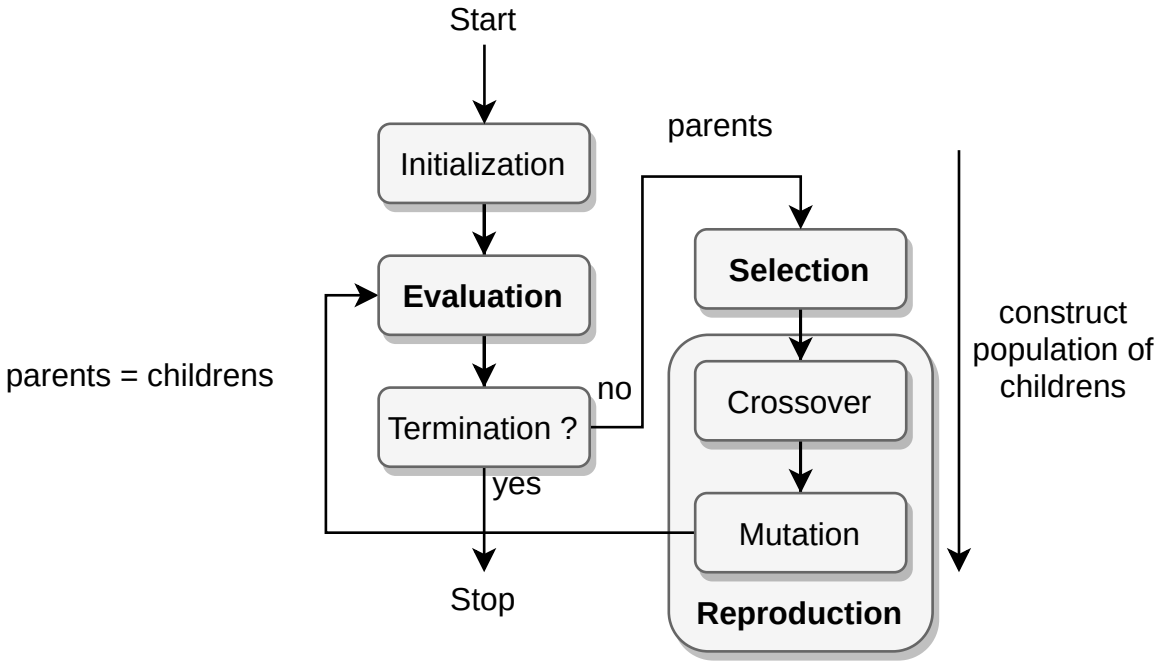


Figure 2-11: Evolutionary Algorithm workflow.

Before listing the different AutoML based on EAs, we present a relation table (2.1)

Table 2.1: Relation table between Optimization, Evolutionary Algorithms and AutoML

Optimization	**EA Metaphor**	**Transcription AutoML**
Optimization problem	Environment	AutoML
Solution	Individual/Candidate	ML Pipeline (tree)
Objective function	Fitness	ML metrics*
Element of the solution	Locus/Attribute	Node (ML algorithm, Preprocessing method, HPs)
Value of the element	Allele	Instance (ML algorithm or Preprocessing method) / HPs

* in majority but not only.

that helps to understand the different but same meaning terms used between Evolutionary Algorithms, AutoML and Optimization. We also explain the **Genetic Programming (GP)** principle, which according to the literature is the only one used with the EAs-based AutoML. This is explained by their capacity to handle complex individuals where the attributes might be mathematical functions and not only values.

In Genetic Programming, each solution or candidate is represented as a tree composed of nodes and leaves (see B-8). Each internal node is called a primitive and each leaf a terminal. A primitive can be seen as a function (also called an operator), and a terminal as an argument or a constant. Primitives and terminals are defined depending on the problem we solve.

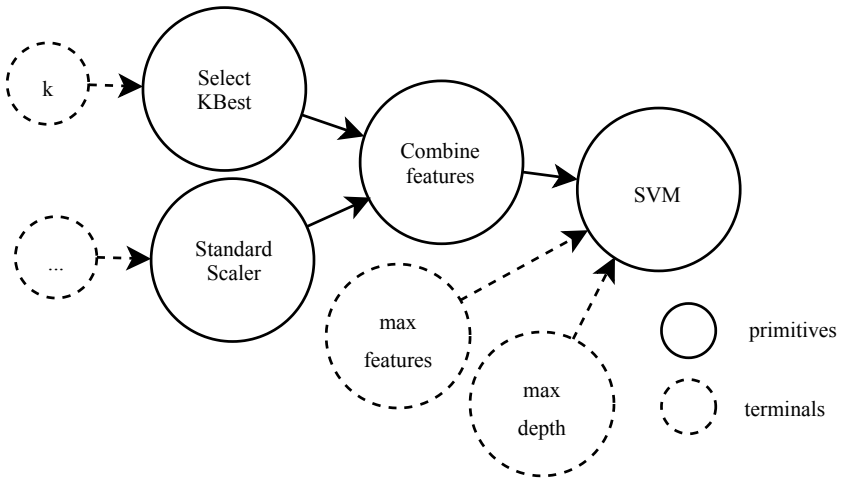


Figure 2-12: Representation of an AutoML candidate (Genetic Programming).

In the case of AutoML, the primitives are the ML algorithms or the preprocessing methods and terminals are the hyper-parameters (see 2-12). Note that the root node of the tree is necessarily a ML algorithm used as a final estimator and other nodes act as operators

taking data as input and returning the transformed data. Trees can be composed of different branches where two branches are merged through a special primitive called "Combine features" which has been introduced by TPOT [90]. It simply combines the output from two previous nodes.

To the best of our knowledge, the AutoML based on EAs, and more specifically tackled with Genetic Programming are the following:

- **REsilient ClassifIcation Pipeline Evolution (RECIPE)** [28], a **Grammar-based Genetic Programming (GGP)**. It is a GP with a grammar that constrains the choice of attributes which constitutes the individual. In machine learning it can be translated by having a logical ML pipeline additionally to their capacity to be trained. Indeed, some models cannot be trained, e.g. logistic regression with non-numerical values, while some others do not make sense, e.g. having consecutively the same algorithm. Thus, the grammar helps to make valid pipelines and theoretically reduces the original search space which improves the optimization.

- **Autostacker** [24]. The particularity of their solution is to stack the outputs given by the non-final ML algorithms in the pipeline. Their inspiration comes from stacking methods [122], which permits to extract new features thanks to the decision process from other models.

- **Tree-based Pipeline Optimization Tool (TPOT)** [90] is built on top of DEAP [40], a modular framework helping to build processes based on Evolutionary Algorithms (EAs). The implementation used by TPOT is a GP with a $(\mu+\lambda)$-ES strategy [14] that we detail in our implementation (section 3.3.2). TPOT uses a clever technique (see algorithm B) from the Python language (decorator) and the scikit-learn [95] library. When a candidate is subject to a mutation, it is trained on a very small subsample from the training set, which permits to train it really fast. If a Python exception is raised during the training phase, i.e. the candidate cannot be trained, the mutation is again called, and it continues recursively until a valid candidate is given. This permits their optimization to be as efficient as the GGP present in RECIPE. Indeed, through this technique, they reproduce a virtual "Grammar" that is induced from the

rules that have been implemented in the different algorithms. While it is a benefit in terms of performance for the optimization, it complicates the analysis of the process to understand the roles of the operators. TPOT also introduces a concept of synthetic features, which are added to the original dataset. This happens when a ML algorithm is present in the in-between nodes. In that case, the predictions from these algorithms are used as features. Thus, it includes a similar behavior to the stacking methods present in Autostacker. Thanks to the two elaborated features above, TPOT performs as well as the two previous AutoML combined.

2.2.4 Monte-Carlo Tree Search (MCTS)

Monte-Carlo Tree Search approaches (MCTS) [83] can be represented as a **tree** (Figure 2-13) where each node represents a partial or a complete solution to the problem. The optimization is composed of **four main components**: a **Selection**, an **Expansion**, a **Simulation** (or Sampling), and a **Backpropagation** that are iteratively called and repeated according to a termination criterion. The main idea is to find an interesting path in the tree that builds good solutions to the problem. Firstly, the selection determines a path along the nodes that will be manipulated by the expansion. The selection of an edge in the tree can be viewed as a Multi-Armed Bandit problem. Secondly, the expansion adds a new node to the selected path and also drives the content of the node. Thirdly, the simulation evaluates the added node. Finally, the backpropagation propagates the reward (evaluation) given by the simulation to the previous nodes. This reward will then be served during the selection of the next iteration.

In AutoML, each node of the tree is a ML pipeline. To facilitate the explanation, the Figure 2-14 is depicted from **ML-Plan** [83]. The idea behind the optimization is to find the regions with partial pipelines giving good performances. Examples of uncompleted pipelines are the ones with missing pre-processing methods. The selection/expansion mimics the practitioner's behavior which generally consists in iteratively adding new transformers to the data and should improve the results made by the estimator. Such demeanor plays in favor of exploitation. Other examples of uncompleted pipelines are the ones with specific estimators but none or very few tried pre-processing methods. Going on such a node would

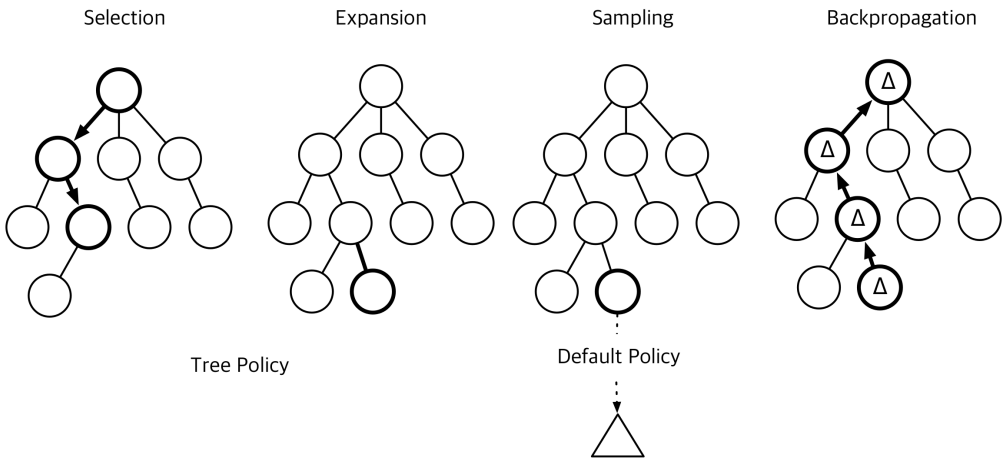


Figure 2-13: Representation of the four main phases in a Monte-Carlo Tree Search.

promote the exploration.

In the literature two solutions based on this approach exists: **ML-Plan** [83], and **MOSAIC** [101] widely inspired from the first.

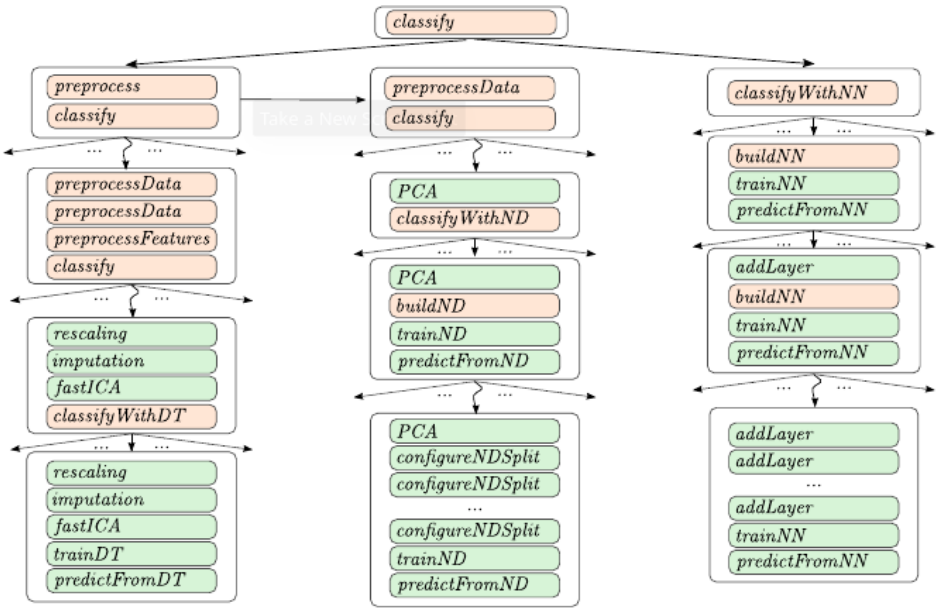


Figure 2-14: Example of a Monte-Carlo Tree Search with ML-Plan. Inside each node there are tasks, where a task is either a primitive or a complex. A primitive task is in green, and is instantiated (algorithms selected or configured). An orange task is not complete and needs to be refurnished (by adding a primitive or another complex task). A complete solution in the leaf node is only composed of primitive tasks.

2.3 Limitations of the existing solutions

This section distinguishes the above implementations by enumerating their pros (summarized in Figure 2-15) and cons written below.

Concerning the **non-adaptive** methods [13], they are the simplest methods to implement, but their lack of exploitation make them unguided and limit the efficiency of the optimization. While recent methods (Hyperband) [69] improve the optimization speed by taking in consideration the specificity of the AutoML (costly due to the complexity of the algorithms and induced by the dataset size), they remain erratic. In consequence, these methods have been transposed on methods considering the past [35] and proved to work even better.

The **SMBO** is the most studied approach in AutoML. Indeed, we can easily find a lot of publications [37, 38, 99, 114, 116] that tackle the problem with distinct surrogate models i.e. Decision Tree, TPE, multiple Gaussian Processes, and show good achievements in terms of speed to handle many candidates despite the costly part induced by BO. SMBO also encompasses a lot of additional features that improve the optimization performance (ensembles, meta-learning, end-to-end, multi-objective). Ensembles [22] permits to aggregate the decisions made by multiple trained models. Meta-learning [20] aims to warm-start the optimization by using the already good known configurations tried on similar instances. The similitudes are found by computing the closest distances between statistical measures on the datasets (e.g. number of instances, average value for features that have the same name). End-to-end deals with the whole process of AutoML (e.g. the capability to connect on different databases and directly get the data on raw formats). This aspect does not improve the performance of the models but makes the AutoML tools easier to use. Therefore, such a feature has more interest in industries than for the progress in research. Multi-objective consists in having models evaluated with a multitude of objectives instead of having just one metric, it might be to minimize the pipeline size like it is done in TPOT or to maximize the interpretability as done with AutoxgboostMC. Note that ADMM [72] differs a little bit, the objectives are cumulated through a penalty process, which does not let the user select the model and also change how the optimization is guided.

While a lot of aspects have been studied in SMBO, its major limitation concerns its capability to handle candidates that have variable sizes. Indeed, investigate the ML pipeline components is an elementary function to properly solve the CASHAP problem (2.1.3). As discussed in the section 2.1.2, CASHAP represents the closest issue encountered by ML practitioners when compared to CASH that omits the preprocessing phase. Typically, the practitioners improve their pipelines by adding and removing pre-processing methods. SMBO is not the most adequate solution to handle such a feature during the optimization. As a matter of fact, SMBO relies on a fixed number of variables (representing the posterior space) to select the next promising candidates. Adding or removing the variables during the optimization would make the space impracticable.

The **EAs**, are more recent but less studied compare to the SMBO. To the best of our knowledge, end-to-end and meta-learning have never been studied for EAs. Only the stacking [24] method, which acts closely to the ensembles have been studied. There is also a bi-objective aspect in TPOT, but the second objective has been fixed by the AutoML itself, in order to minimize the size of the pipeline. Thus, the user is limited to specify the first objective only, making the solution equivalent to the others with a mono-objective. While the SMBOs have been configured with different surrogate models and acquisition functions, the EAs have not been tuned at all. There are always the same operators, selection method, algorithm and parameter values. Consequently, the EAs have not been so much investigated despite its great potential to be adaptable and their inherent capacity to handle flexible pipelines [10, 33]. Indeed, the EAs are capable to change what constitutes a candidate, which make them perfect applicants to solve the CASHAP problem.

MCTS is the most recent approach. Like the EAs, it is capable to solve flexible ML pipelines [121], therefore, it is also adequate to solve the CASHAP problem. Like some SMBO implementations, it has been investigated on various aspects: meta-learning, ensembles, multi-objective. Nonetheless, like the EAs, the parameters related to the optimizer have been quietly tuned. The selection which is one of the major components that performs the trade-off between exploration and exploitation has been implemented with two different methods only: a randomized depth-first search for ML-Plan [83] and the AlphaGo Zero criteria for MOSAIC [101]. The same remark could be done on the other phases (expansion

and back-propagation).

A last remark concerns the parallelism in the EAs, which compared to SMBO and MCTS can be easily applied. By design, a given iteration with EAs includes multiple independent candidates that can be trained at the same time, while SMBO and MCTS usually require the result from the previously trained candidate to find and train the next one. However, some methods [54, 71, 82] exist to overcome this limitation.

To summarize, the SMBO approach has already been well studied on the problem of the AutoML, while the EAs and the MCTS approaches have been less considered.

In the next chapter, we introduce the development of a new AutoML which is notably motivated by a majority of the above elements.

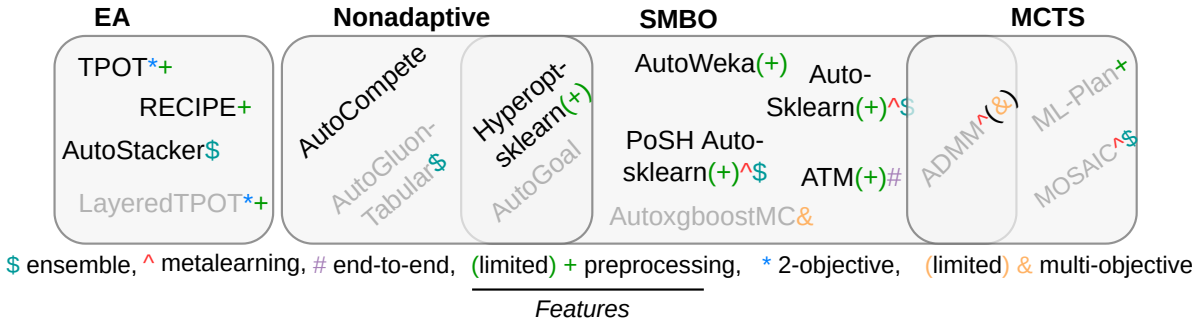


Figure 2-15: Optimization approach(es) and main feature(s) implemented in the AutoML. The unpresent solutions at the beginning of this manuscript works are written in grey.

Chapter 3

Mary-Morstan: a Multi-Objective and Modular AutoML Framework

In this chapter we present Mary-Morstan [1], a novel AutoML framework based on evolutionary algorithms (EAs) which is modular and permits to entirely study what constitutes the EAs. We first give our motivation to build such a solution, followed by a global overview of our tool along with the processes of the new components. Then we perform two experiments, a first one to ensure that our tool meets the state-of-the-art performance, and a second one to see the impact of the new components on classification problems.

3.1 Introduction

In the previous chapter, we enumerated the state-of-the-art AutoML solutions. By looking at them, we noticed that all the **solutions** based on **evolutionary algorithms** [24, 28, 90] have been **empirically configured** and the authors **barely** have **motivated** their choice for the **settings**. The operators (i.e. mutations and crossovers) are always the same, and the few parameters used during their experiments are similar (e.g. number of individuals per generation, the mutation rate, the crossover rate, the selection methods, the distribution used to select the operators).

[1] it is a reference to the wife of Watson, the name used by the Artificial Intelligence software built by IBM. The name Mary-Morstan is the continuity of my Master's degree's final project

Since it is well known that **tuning EAs** on a given problem has an **impact** on the **performance** [33], we decided to develop a new AutoML framework capable to be tuned in order to observe if it is the case when applied on machine learning pipelines.

Our choice to focus on the EAs is not only due to the lack of configurations. We also notice that the **multi-objective** aspect plays a significant role [60, 99] for the success of the future AutoMLs. Indeed, an **ideal AutoML** should return **models** that are **interpretable**, **unbiased**, and **robust**. In other words, we should be capable to understand the decision made by a model. The decision should not be unfair, e.g. discriminative. And the performance should not decrease along the time. Other objectives could have an importance, e.g. minimizing the prediction latency [72]. In order to have such an AutoML, the different facets should be measurable and handled by the optimization. Recent studies proved that there is a growing interest to quantify [17, 64, 84, 87, 130] the related aspects. **By design** the **EAs** are **capable of handling multiple measures (objectives)** during the optimization, which by consequence should participate in the successful development of the AutoML.

The multi-objective capacity is not the only benefit of EAs. **EAs** also have the **capacity of handling individuals that change** (e.g. variable size) along the optimization. It is very common for a ML pipeline to be expanded by adding or removing preprocessing methods in order to transform the data which increases its accuracy. Not all the optimization approaches are flexible, it is the case for the SMBOs that struggle to handle individuals that have changes in their structure.

Lastly, another great advantage of EAs is their possibility to be **easily parallelized** on multiple threads. It is not the case with the SMBOs and MCTS approaches which wait for each evaluation to select the next candidate. Even if we did not study this part, we have to mention it to highlight the future work that can be done with EAs that tackle the AutoML problem.

To summarize, the lack of experiments to configure the EAs, and the enumerated pros above has reinforced our confidence to focus the development of the AutoML using this approach.

In the following sections, we detail how our proposed solution operates and we present the different available components. We designed the framework to be modular when it comes to

integrating new elements such as operators (i.e. variations), algorithms, selection methods and objectives.

3.1.1 Individual Representation

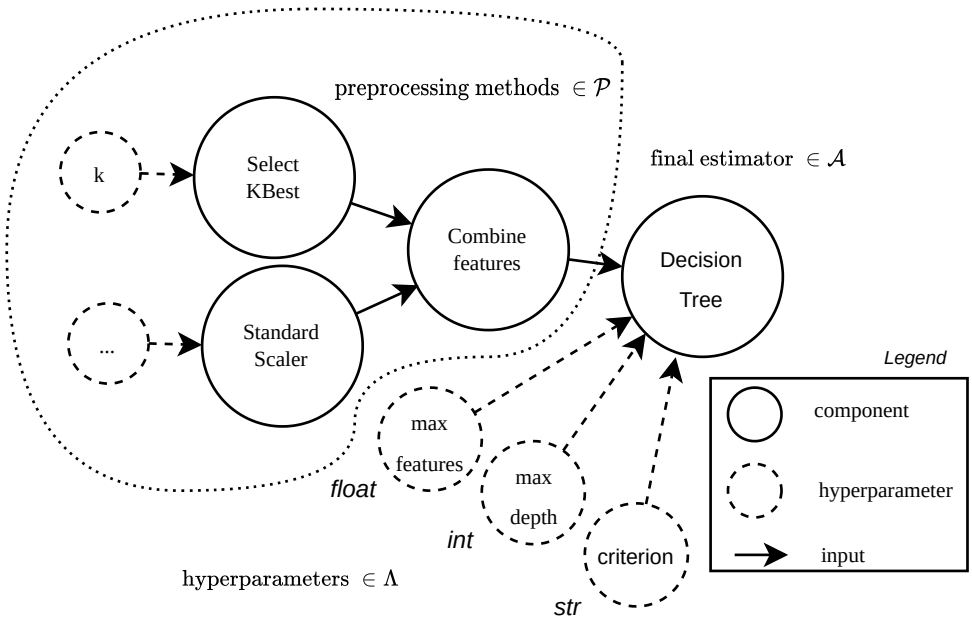


Figure 3-1: Exemplar representation of an individual in Mary-Morstan, also called candidate or ML Pipeline.

For a better comprehension along the different procedures presented later, we provide a representation of an individual in Figure 3-1 that we also call a ML Pipeline or a candidate. A ML Pipeline can be depicted as a tree with nodes where each node represents an attribute. We distinguish three types of nodes.

- The estimator node, always at the root of the tree, used to take the final decision.
- The preprocessing nodes present in between the root node and the leaves, used to preprocess the input data.
- The hyper-parameter nodes, present at the leaves of the tree and representing the parameters of the different algorithms.

Note that a specific method called Combine Features has been implemented. Similarly to TPOT, the method allows to create complex trees by merging the output of two nodes.

3.2 General Overview

As shown in Figure 3-2, Mary-Morstan starts with a phase of initialization, which includes three parts.

- The selection of a Machine Learning space, containing all the algorithms and their associated parameters. This dictionary space is very common in other AutoML solutions.
- The selection of an **EA space**, specifying and configuring all the different EAs components. To the best of our knowledge, there is no such a feature in the current AutoML solutions.
- The generation of initial ML pipelines.

Passed the initialization phase, the framework starts an EA loop process (see 2.2.3) where the ML pipelines are subject to variations, evaluations and selection until a budget is exhausted. The budget can be implemented in different manners. Usually it is represented as a fixed number of iterations (generations). An alternative to the budget can be an amount of time, or when there is no more progress (convergence).

In the Figure 3-2, we highlight in green the new components that we propose in comparison to TPOT.

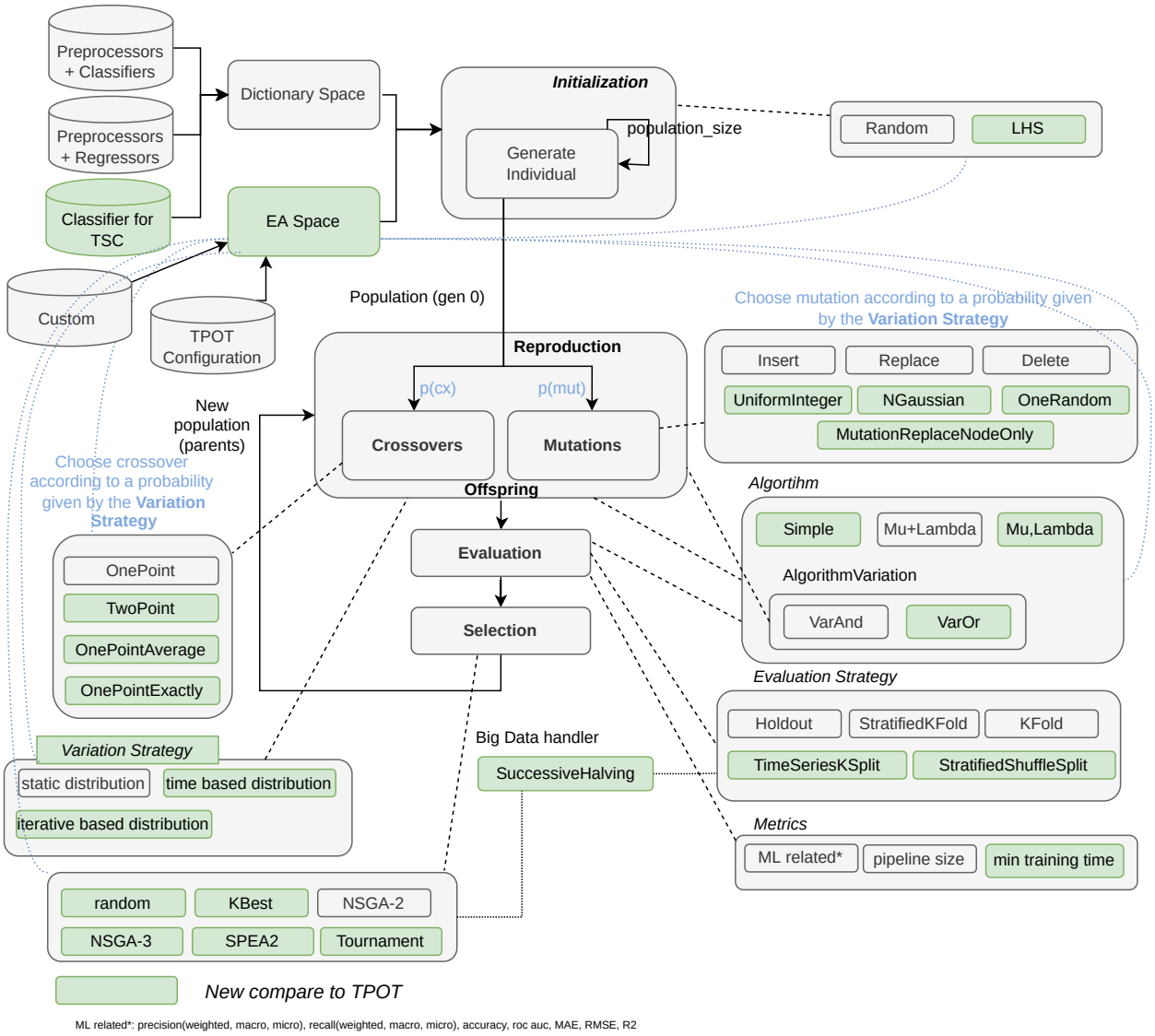


Figure 3-2: Architecture of Mary-Morstan

Table 3.1: Mary-Morstan parameters

Parameter	Values
Population size $\mathcal{P}$	$[1..\infty]$
Generations G	$[1..\infty]$
Initialization	{random, LHS}
Initialization individual min/max size	$\{\min, \max\} \in \mathbb{R} \mid \min \leq \max$
Per-individual mutation rate	[0,1]
Per-individual crossover rate	[0,1]
Algorithm	{simple, mu+lambda, mu,lambda, mu+lambda+kappa}
AlgorithmVariation	{VarOr, VarAnd}
Mutation	{insert, replace, delete, uniformInteger, gaussian($\sigma \in [0, 1]$), oneRandom}
Crossover	{onePoint, twoPoint, onePointAverage, onePointExactly}
Candidate evaluation	{holdout, KFold, stratifiedKFold, stratifiedshufflesplit timeSeriesKSplit}
Selection	{random, KBest, best, **NSGA-II, NSGA-III, SPEA2**}
Maximum evaluation time per candidate	$[1..\infty]$ (minutes)

Multi-objective selection methods.

3.3 Components

3.3.1 Generate the Initial Population

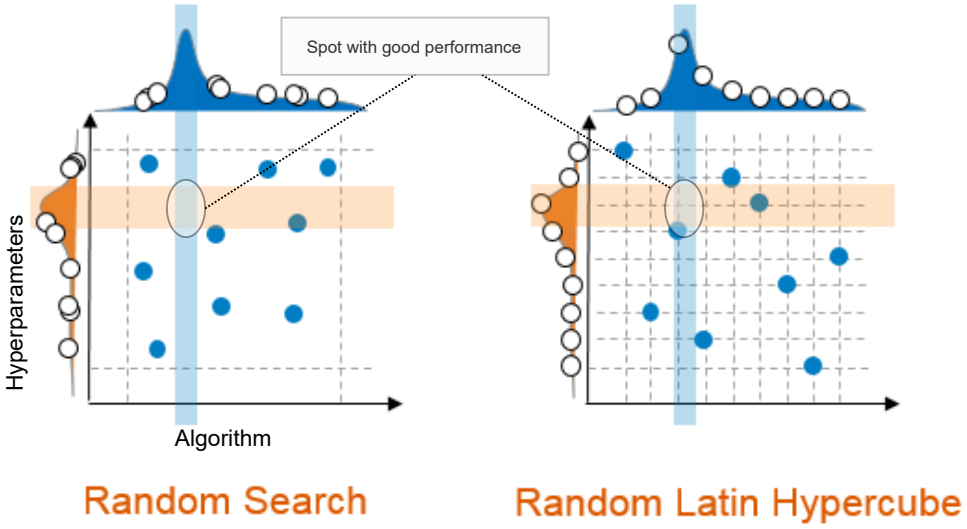


Figure 3-3: Procedures to initialize the population at generation 0 with Mary-Morstan.

During the initialization phase, $\mathcal{P}$ individuals are generated. Each individual is generated with a random size (i.e. the number of nodes) picked between a minimum and a maximum.

The root node is chosen among the estimator group (i.e. a set of algorithms defined in the ML Search Space as estimators). The in between nodes are chosen from the preprocessing group. An option can be specified to include the estimators in the preprocessing group.

In order to ensure a better diversity of candidates at generation zero, we propose an alternative to a random initialization, the Latin Hypercube Sampling (LHS) strategy. As shown in Figure 3-3, the method maximizes the number of algorithms used, and then uses a pseudo-random generator to select the hyper-parameters per algorithm. The idea with LHS is to generate $\mathcal{P}/n$ individuals per estimator (n represents the number of estimators in the search space).

3.3.2 Algorithms

The algorithm defines the general behavior of the EAs (see 2.2.3). We implement the most famous ones in order to give more choice to the practitioner. Here is a list of the four algorithms present in Mary-Morstan:

- Simple (Alg. 3), which follows a traditional evolutionary algorithm workflow.
- $(\mu + \lambda)$-ES (Alg. 1) which contrarily to the simple EA, generates λ offsprings and then performs a selection of μ individuals among the offspring plus the parents.
- (μ, λ)-ES (Alg. 4) is similar to $(\mu + \lambda)$ except that the selection is realized on the offspring individuals only.
- $(\mu + \lambda + \kappa)$-ES (Alg. 5) is similar to $(\mu + \lambda)$-ES with κ individuals that are randomly generated and not subject to any variations. The κ individuals are subject to the μ selection.

3.3.2.1 Procedures

Even if a procedure globally changes the behavior of the optimization, from an algorithmic point of view, they slightly differ in the code. For this reason, we only detail the procedure $(\mu + \lambda)$-ES below and provide the remaining ones in the appendix (3).

Algorithm 1 $(\mu + \lambda)$-ES

Input: population size, $\mathcal{P}$; wall time, T; offspring size, λ; selection size, μ; distribution function, P; training set, $\mathcal{D}_t$
Output: best candidate

$\alpha \leftarrow$ generate($\mathcal{P}$)
$\mathfrak{P}_p \leftarrow \{(\alpha_k, \text{evaluate}(\alpha_k, \mathcal{D}_t)), k = 1, ..., \mathcal{P}\}$
while elapsed_time $< T$ **do**
    $\alpha \leftarrow$ clone($\mathfrak{P}_p$, λ)
    $\alpha \leftarrow$ variation(α, P(elapsed_time))
    $\mathfrak{P}_o \leftarrow \{(\alpha_k, \text{evaluate}(\alpha_k, \mathcal{D}_t)), k = 1, ..., \lambda\}$
    $\mathfrak{P}_p \leftarrow$ select $(\mathfrak{P}_p \cup \mathfrak{P}_o, \mu)$
end while
return $\alpha^\star \in \text{opt.}\mathcal{L}$

In Algorithm 1, we denote α the list of non-evaluated candidates, $\mathfrak{P}_p$ the parent population, and $\mathfrak{P}_o$ the offspring population. The procedures are the following:

- **generate**(p): generates a list of p candidates.
- **evaluate**(c, d): evaluates the candidate c on dataset d.
- **clone**(p, λ): clones λ candidates by selecting them randomly in the population p.
- **variation**$(p,\ P(\text{elapsed\_time}))$: applies on each individual of the population p a variation operator, e.g. a mutation or a crossover, according to probability that follows the distribution P. The distribution P can be changed along the time, thanks to the elapsed_time that is passed as a parameter.
- **select**(p, μ): Select μ best candidates from the population p. In the $(\mu + \lambda)$-ES, the population p is composed with the parent population $\mathfrak{P}_p^{(i)}$ and the offspring $\mathfrak{P}_o^{(g)}$.

3.3.2.2 Algorithm Variation

Currently two kinds of mechanisms are implemented concerning the variations: VarAnd and VarOr. For a given generation, VarAnd permits the algorithm to apply the mutations as well as the crossovers on a candidate while VarOr only permits to apply to one of the two. In the following subsections, we detail how the different variations modify the ML pipeline.

3.3.3 Variations

In this section we enumerate all the variations and depict for each of them an example on how it changes the pipeline. We first begin with the mutation procedures and finish with the crossovers. A mutation consists in changing attributes on a selected individual, while a crossover mixes attributes between two or more candidates.

3.3.3.1 Mutation Procedures

The **MutationInsert** procedure, Figure 3-4, randomly picks an algorithm from the ML search space, and also randomly generates its associated hyper-parameters according to the restrictions specified by the user. Then, a random node is selected and used to insert the new generated node. The insertion only happens on nodes below the root node. This mutation tends to increase the pipeline size along the generations.

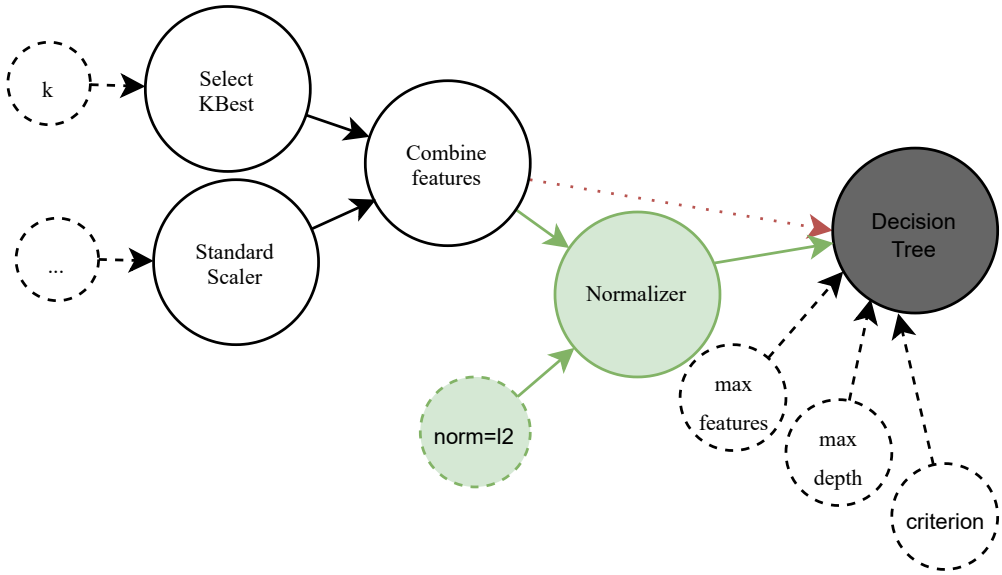


Figure 3-4: Example of individual subject to MutationInsert .

The **MutationDelete** procedure randomly picks a node in the tree, excluding the root node and hyper-parameters, and removes it from the tree. In the example of the Figure 3-5, the Combine Features node is removed, which enforces a chaining of nodes. This mutation tends to reduce the pipeline size along the generations.

The **MutationReplace** procedure in Figure 3-6, randomly picks a node which is replaced by a generated node. The type of the generated node is similar to the previous one. If it

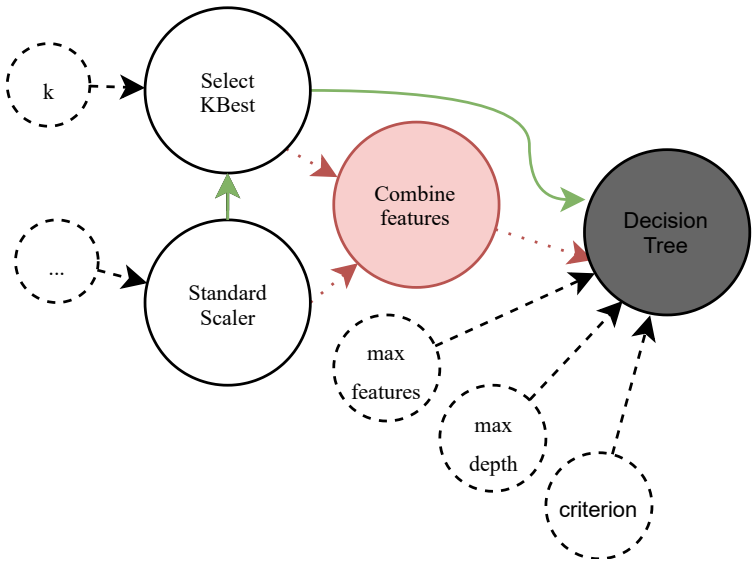


Figure 3-5: Example of individual subject to MutationDelete.

is a hyper-parameter, a new value is uniformly chosen from the search space. If it is a component node, a new component is uniformly picked from the ML search space, and its hyper-parameters are also randomly generated. Note that in this example the Combine Features node is replaced, causing a chaining of the two previous nodes.

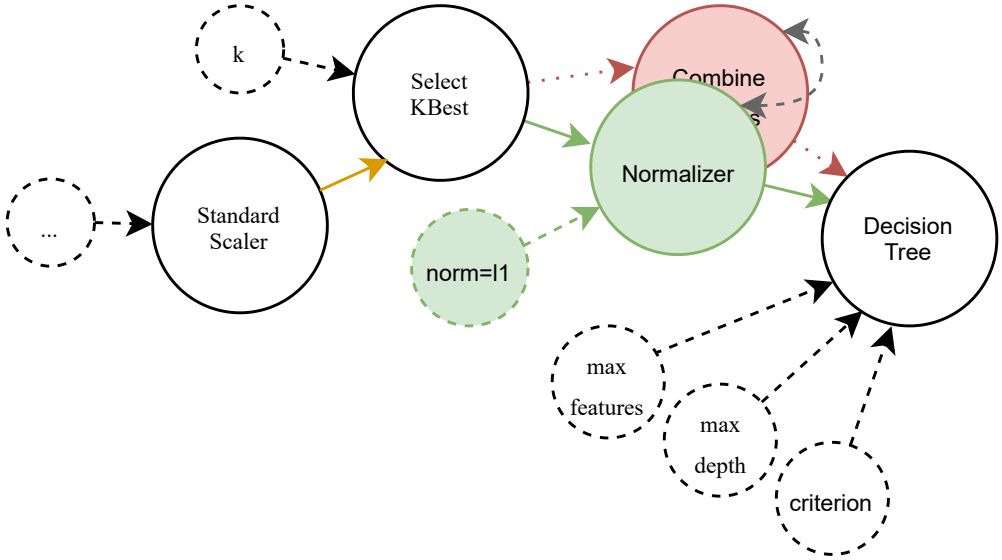


Figure 3-6: Example of individual subject to MutationReplace.

The **MutationOneRandom**, Figure 3-7, randomly chooses one hyper-parameter node and changes the value according to a uniform distribution within the range specified in the

ML search space.

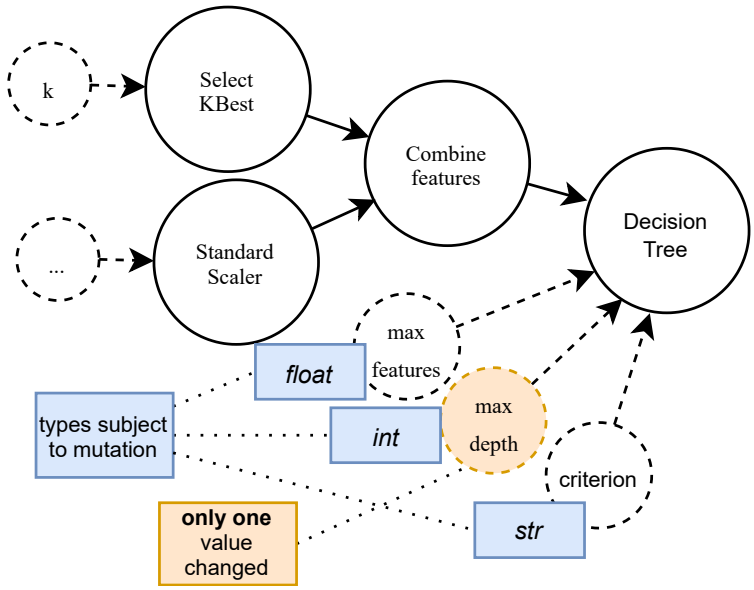


Figure 3-7: Example of individual subject to MutationOneRandom.

The **MutationUniformInteger**, Figure 3-8, adds noise around the current values based on a uniform distribution for the hyper-parameter nodes of real and integer types. The uniform distribution is drawn according to the lowest value between the current value minus the lower bound (defines in the search space) and the upper-bound minus the current value.

The **MutationGaussian**, Figure 3-9, adds noise according to a normal distribution for all the hyper-parameter nodes with a real type. This mutation has three parameters, the mean which is 0 by default, the variance undefined by default, and the sigma ratio defined to 10%. When the variance is undefined, it will take the current value of the node multiplied by the sigma ratio. The advantage of such a mutation lies in its aptitude to be instantiated with different sigma ratios, which allows to play with the amplitude of the noise applied on the parameters.

3.3.3.2 Crossover Procedures

The **CrossoverOnePoint**, Figure 3-10, switches the subtrees based on a common index node in the tree. The index is randomly determined between zero (i.e. root node) and the lowest pipeline size (i.e. the candidate having the lowest number of attributes). Root nodes are excluded since exchanging them would not make any change. Please note that if

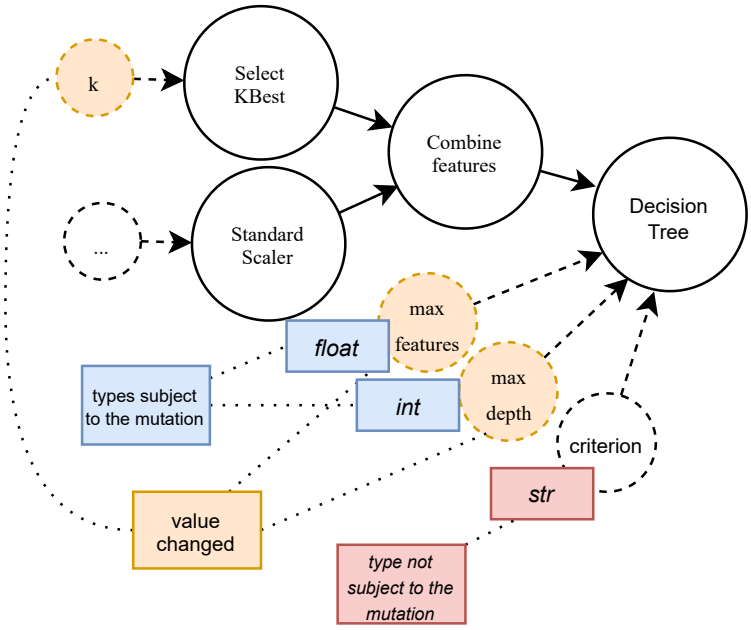


Figure 3-8: Example of individual subject to MutationUniformInteger.

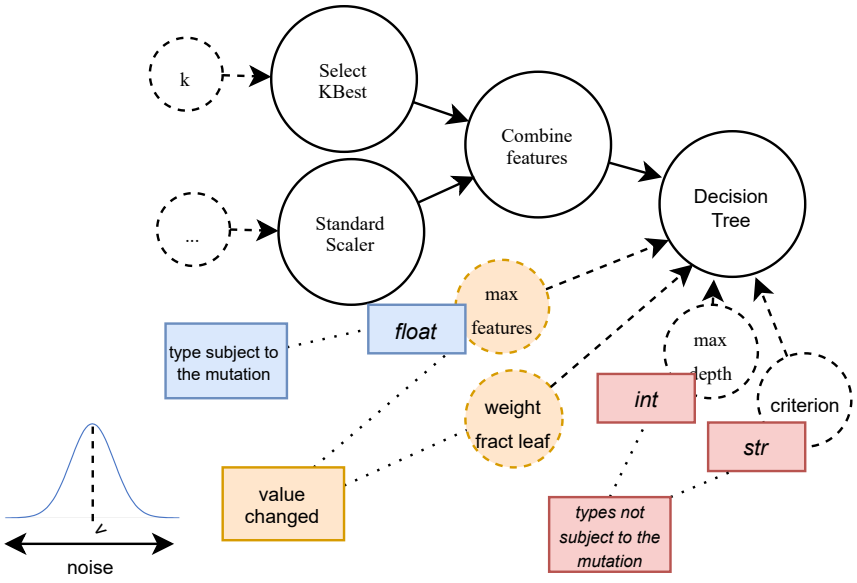


Figure 3-9: Example of individual subject to MutationGaussian.

two pipelines have the same size, and the latest nodes are exchanged, it is equivalent to a CrossoverOnePointExactly on the latest index. There is an exception that if an individual is only composed of one estimator, and the second one has an estimator and at least one preprocessor, the subtree can be exchanged between the two individuals. This permits a candidate to inherit the whole preprocessing phase from another candidate.

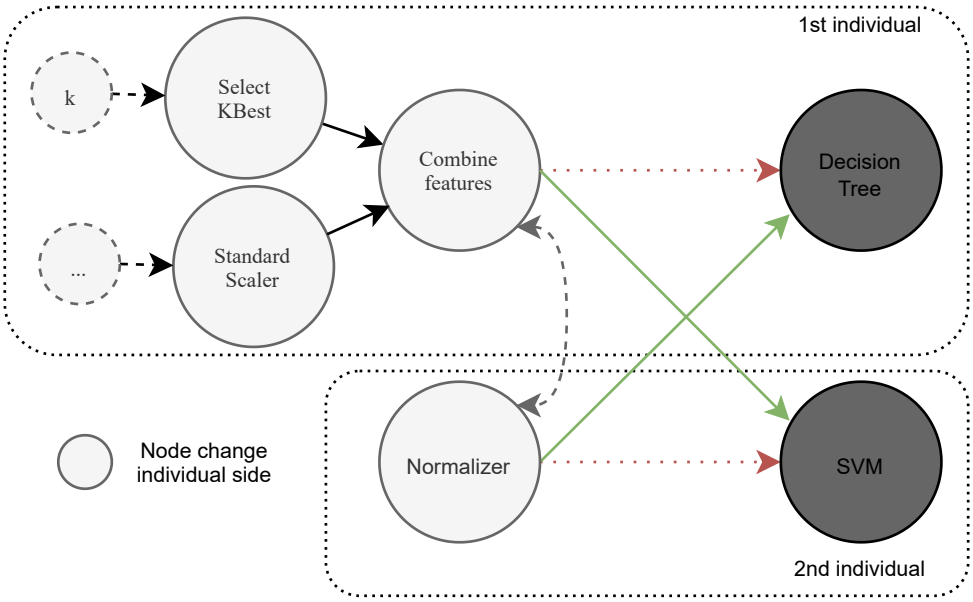


Figure 3-10: Example of individuals subject to CrossoverOnePoint.

The **CrossoverTwoPoint**, Figure 3-11, exchanges two subtrees, distinguished by two common index nodes in the tree. The index node is picked between zero and the lowest pipeline size. This crossover changes the in between structure for the ML pipelines selected. Note that both pipelines should have two preprocessing nodes at least. In the case that the two pipelines have two preprocessing nodes exactly, it is equivalent to a CrossoverOnePointExactly.

The **CrossoverOnePointExactly**, Figure 3-12, aims to exactly change one node between two individuals. The nodes are chosen on an index number determined between zero and the lowest pipeline size.

The **CrossoverOnePointAverage**, Figure 3-13, selects two similar hyper-parameter nodes, and changes the value between one of them by the mean of the two.

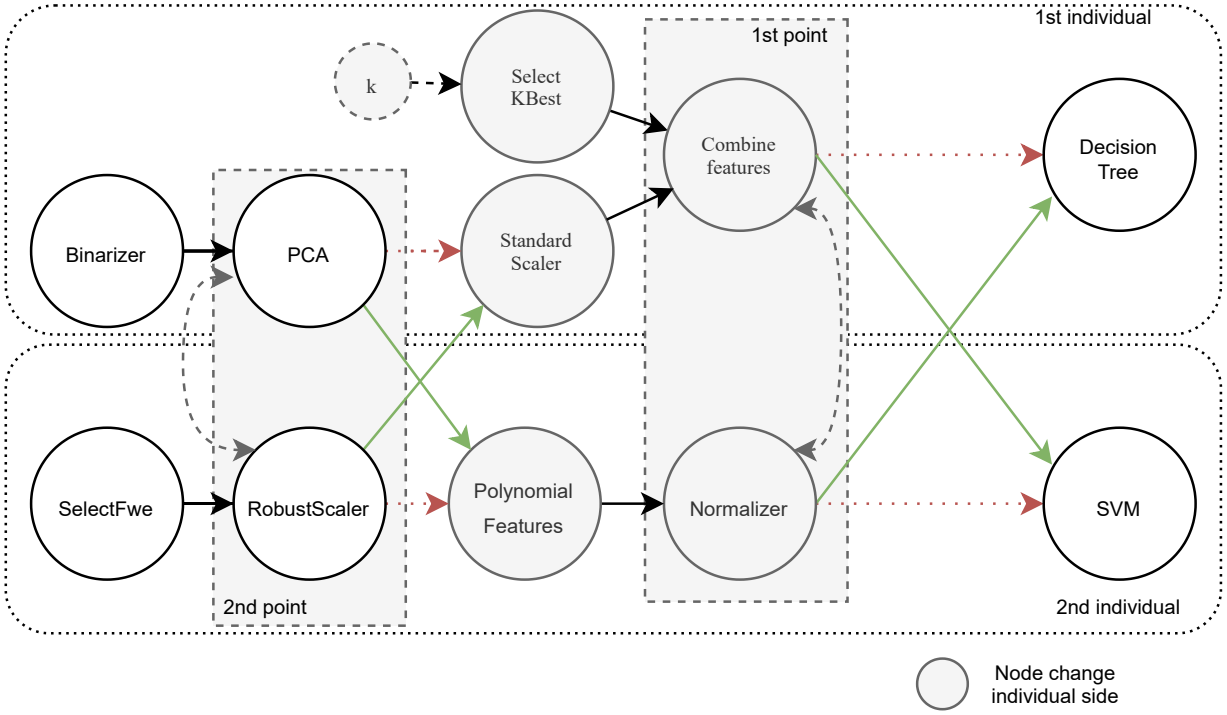


Figure 3-11: Example of individuals subject to CrossoverTwoPoint.

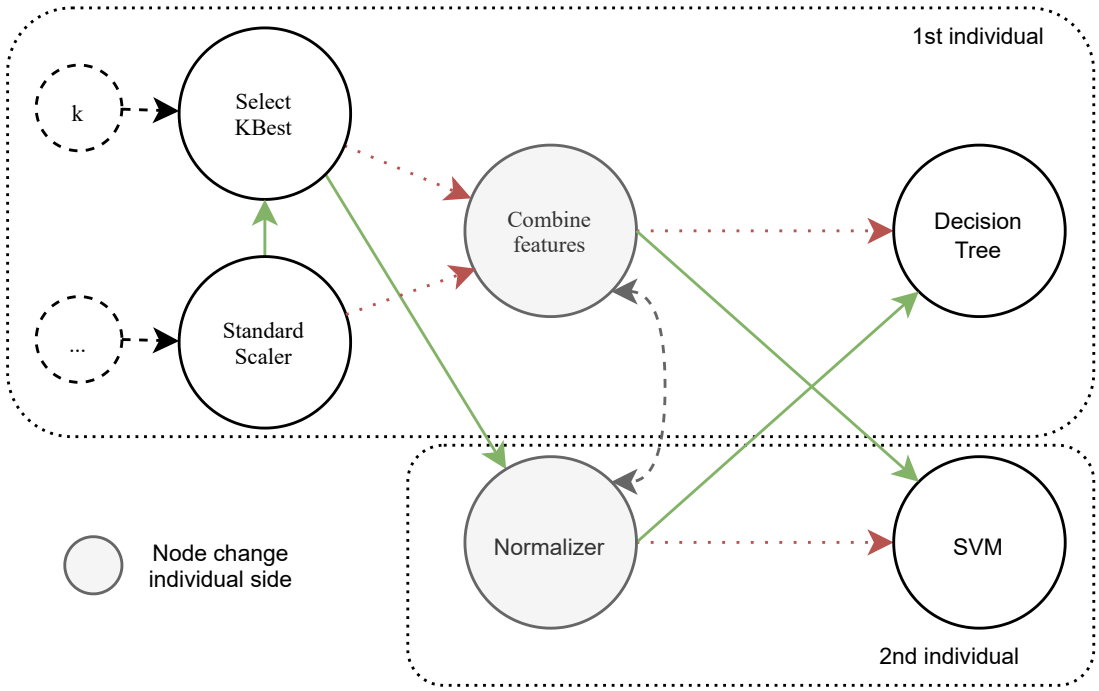


Figure 3-12: Example of individuals subject to CrossoverOnePointExactly.

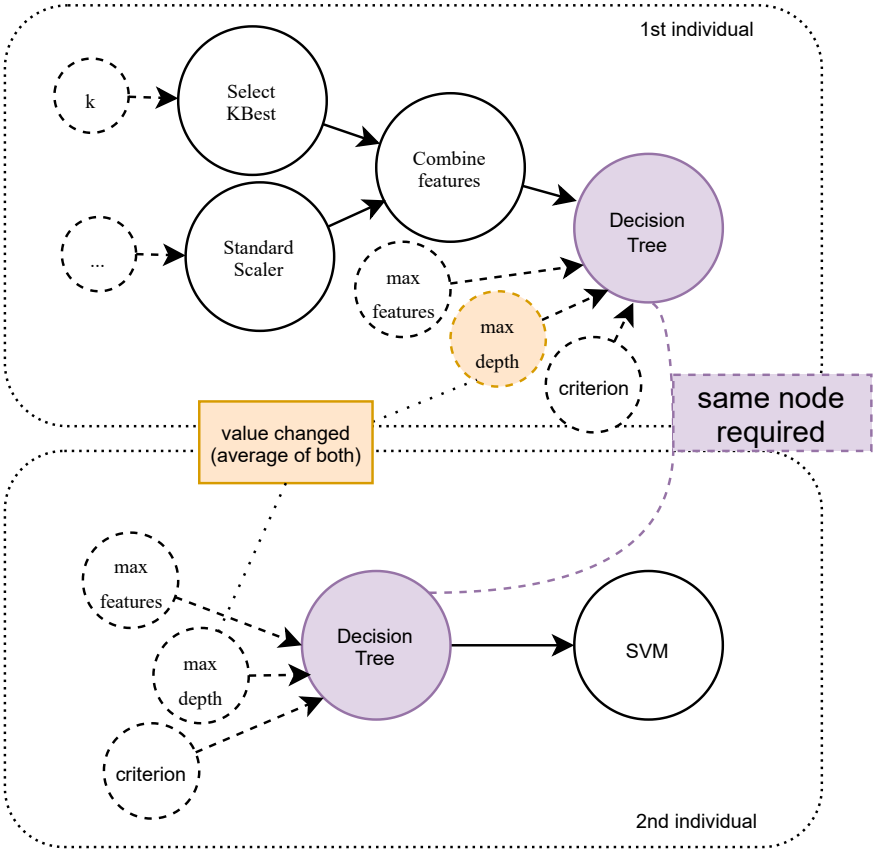


Figure 3-13: Example of individuals subject to CrossoverOnePointAverage.

3.3.3.3 Expected Impacts

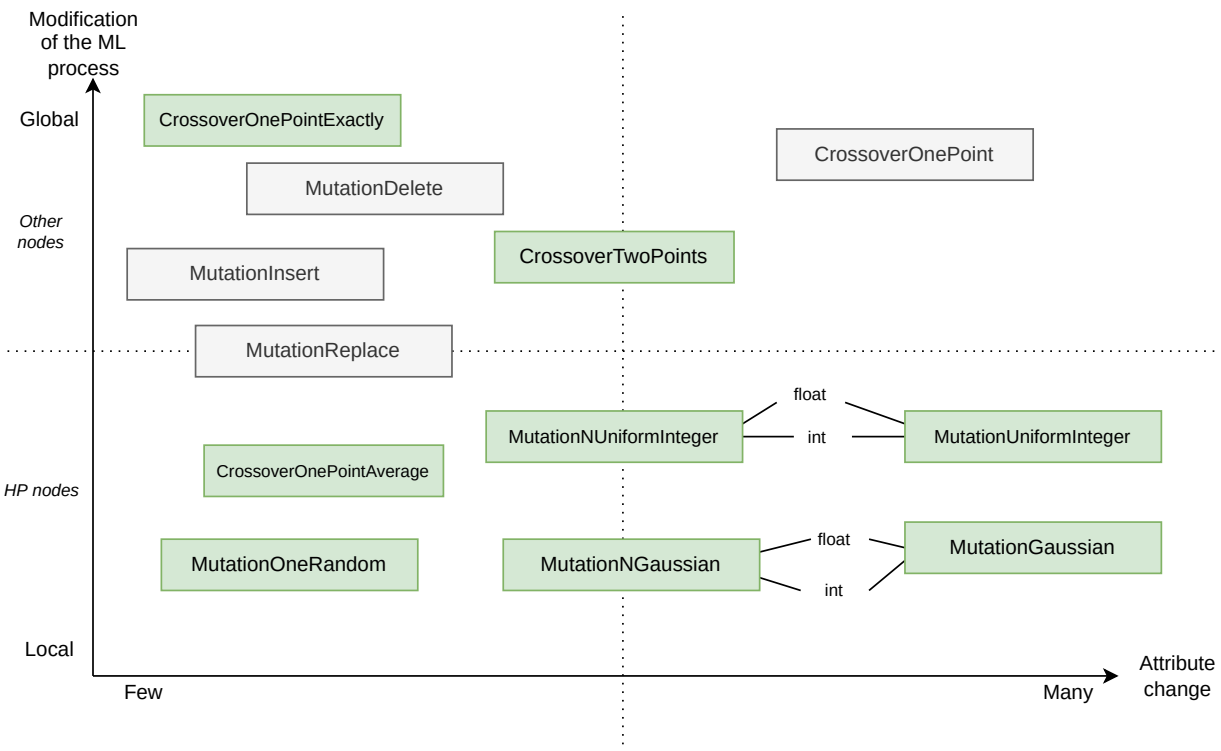
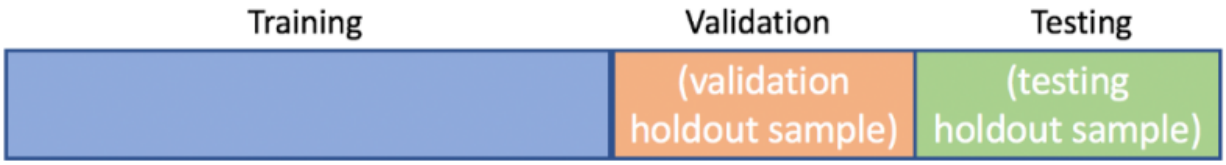


Figure 3-14: Taxonomy of reproduction procedures with Mary-Morstan v0.14+.

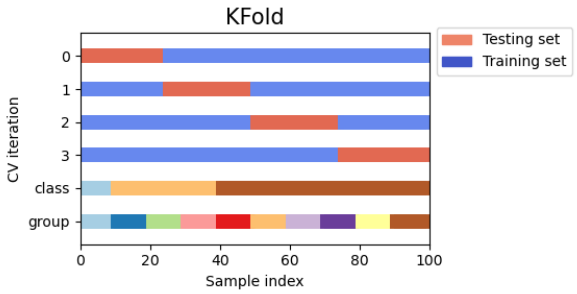
The choice of the different procedures have not been made randomly. They have been designed in such a manner that they differently change the pipeline with the objective of building heterogeneous populations. To the best of our knowledge, this degree of freedom to configure the optimization of an evolutionary algorithm in AutoML has never been proposed. It aims to improve the performance of the returned candidate by configuring the AutoML to the needs of the problem, or to the specificity of the dataset. To better understand how the procedures impact an individual, we propose a taxonomy in Figure 3-14.

3.3.4 Evaluation Strategies

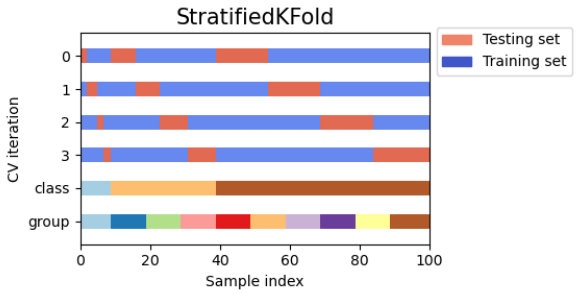
Mary-Morstan includes the most common evaluation methods used in Machine Learning. Among them, we find the Holdout validation, the K-Fold cross-validation, the Stratified Shuffle Split validation, the Stratified K-Fold cross-validation, and the Time Series Split validation.



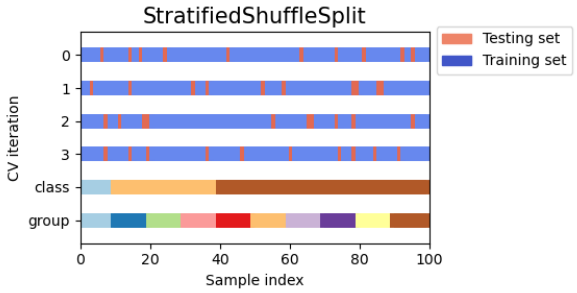
(a) Holdout validation.



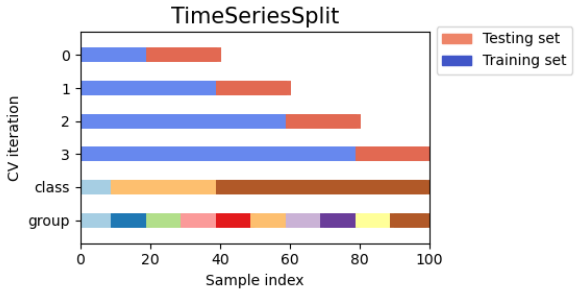
(b) K-Fold cross-validation.



(c) Stratified K-Fold cross-validation.



(d) Stratified Shuffle Split cross-validation.



(e) Time Series Split validation.

On the main lines, the holdout validation simply split the whole data set in two parts. The first part, called training set, feeds the machine learning algorithm, and the second part, called validation set, is used to evaluate the trained model. The problem with the holdout validation is that some observations will be seen in the training set but unseen during the evaluation. By consequence, a model can be discarded due to bad performance while it was effectively a good candidate that can handle specific observations. To avoid such a case, smarter methods like K-Fold cross-validation have emerged. The method splits the dataset into K folds, where each fold splits the dataset in two parts (train set and validation). Contrarily to the holdout, we now have multiple splits at different points. The ML algorithm is individually trained on each fold (train set), and then evaluated on the validation set related to the fold, which permits the algorithm to glimpse the whole dataset. The final result is obtained by aggregating (e.g. average, median) the performance from the validation folds. Other variants of the K-Fold exist (Stratified, Shuffle Split), and they permit to better handle the different specificity of the datasets, for example when a dataset has unbalanced classes. To better represent how the different evaluation strategies perform, we provide the Figure 3-15a. For further details, the reader may refer to [65, 124].

3.3.5 Selection methods

The following selection methods are present in Mary-Morstan:

- **selectKRandom**: randomly selects K individuals.
- **selectKBest**: selects K individuals according to the ascending order of the fitness values.
- **selectTournament** [33]: picks the k best individuals from a portion of individuals selected from the whole population and repeats this process until there are K selected individuals.
- **selectNSGA2** [29]: consists in selecting K individuals according to the non-dominated Pareto fronts and to the crowding distance when it reaches the last front. The crowding distance maximizes the diversity of the candidates, i.e. avoids to select candidates that

have too close fitness values in a given front. The figure 3-16 gives an overall picture of this process.

- **selectNSGA3** [55]: improves the NSGA-2 selection by notably tackling the many objectives problem. It works similarly to NSGA-II, but replaces the crowding distance by a *niching* procedure that selects individuals not well represented in the selected fronts based on reference points defined by the user.

- **selectSPEA2** [129]: selects K individuals from a fitness assignment strategy which incorporates density information.

The last three selection methods are capable and adapted to deal with multi-objective problems (candidates with multiple fitness values), while the remaining ones are not. In order to keep our framework agnostic, the selectKBest and selectTournament methods will use the first value of the fitness if multiple values are present. This mechanism might be useful for further experiments, e.g. comparing the performance of two different selection methods (e.g. a mono-objective and a multi-objective) in a post-hoc analysis.

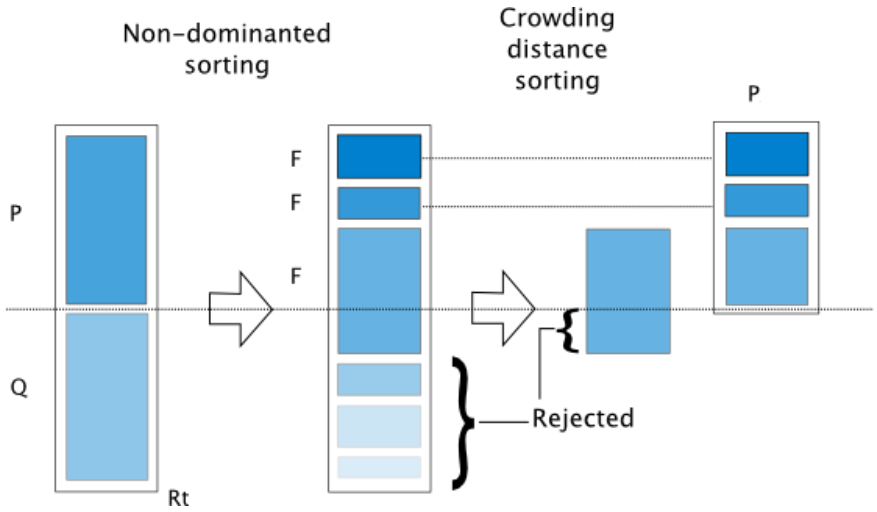


Figure 3-16: NSGA-2 selection. P represents the selected individuals from the non-dominated fronts F plus the crowding distance in the last front.

3.4 Experiments on Classification

In this section we detail the experiments that we run with Mary-Morstan. We perform **two main experiments**. The **first experiment compares** our proposed solution **Mary-Morstan with TPOT** [90], that is currently and to the best of our knowledge the most efficient AutoML based on EAs. Such an experiment can be done by using the same configuration as TPOT, which permits to demonstrate that our framework is an AutoML capable of having similar performance to the state-of-the-art. The **second** experiment consists in **looking for better EA configurations**, which might help to accelerate the performance of the optimization by investigate the exploration versus exploitation trade off.

3.4.1 Mary-Morstan Instantiated Like TPOT

To compare Mary-Morstan with TPOT, we similarly configure our solution to TPOT. We use their variation procedures, the same evolutionary parameters (Tables A.4), and the same ML search space (see Appendix A.2 and Appendix A.3).

3.4.1.1 Protocol

Each algorithm is stochastic and by consequence are run 30 times on a Slurm [127] cluster for each dataset that are described in Table A.1. The datasets come from an article [43] that describes how to benchmark an AutoML framework.

Similarly to the TPOT [90] experiment, 75% of the original dataset have been used for the training data (with a 5-fold cross-validation) and the 25% remaining have been used to test the best candidate found at the end of the optimization process. The balanced accuracy metric has been chosen for the diversity of the datasets (presence of balanced, unbalanced, binary, and multi-class datasets).

In order to ensure fair environments, we used enroot and pyxis [3], two related technologies that permit to encapsulate each algorithm with the same libraries. Thus, we avoid the biases induced by recent versions of the dependencies that generally include better optimizations. Also, we limited each run to 3500 MB of memory, and one single CPU.

Table 3.2: TPOT vs M-M (TPOT) average performance on the test scores over 30 runs. According to the Mann-Whitney U Test with a 5% of significance, there is no statistical difference between both algorithms.

Dataset	**TPOT**	**M-M (TPOT)**
sylvine	96.07 (± .66)	96.2 (± .59)
numerai28.6	52.08 (± .28)	52.02 (± .32)
blood-transfusion	66.59 (± 3.28)	66.76 (± 4.04)
credit-g	69.18 (± 4.6)	70.5 (± 2.7)
kc1	73.78 (± 2.88)	74.10 (± 2.66)
Australian	86 (± 2.27)	85.77 (± 2.33)
vehicle	88.85 (± 1.94)	88.93 (± 2.18)
phoneme	89.43 (± 1.07)	89.57 (± 1.03)
Shuttle	98.51 (± 1.83)	97.35 (± 3.42)
jasmine	49.73 (± 4.81)	48.37 (± 5.24)
Amazon_employee_access	77.72 (± 5.5)	78.81 (± .85)
bank-marketing	86.93 (± .3)	87.04 (± .53)
jungle_chess_2pcs_raw_endgame_complete	98.04 (± 1.09)	97.52 (± 1.5)
adult	83.16 (± 2.94)	83.73 (± 1.29)
connect-4	81.09 (± .85)	80.44 (± .87)
car	98.62 (± 4.65)	98.25 (± 6.78)
segment	96.61 (± .39)	96.02 (± 1.5)
kr-vs-kp	99.45 (± .26)	99.51 (± .27)
mfeat-factors	99.08 (± .09)	99.16 (± .12)

3.4.1.2 Results

We report the final performance on two different supports:

- Table 3.2 compares the final performance (obtained after 100 generations or after 5 days) on the test score between TPOT and M-M.

- Figure B-9 draws the average performance of the validation score for each algorithm.

Among the 39 datasets, both algorithms performed on 19 datasets only. The remaining ones were too large to be trained with the computational environment that we have defined (memory limit per run).

The final results that we obtained (Table 3.2), have shown that Mary-Morstan and TPOT perform similarly on the different datasets. To prove this statement, we used the Mann Whitney U Test with a significance of 5%. It shows that no algorithm statistically outperforms the other on each dataset.

The convergence results also show interesting results (Figure B-9). While both algorithms seem to perform similarly for most of the datasets, there are some datasets where TPOT converges faster than Mary-Morstan (blood-transfusion) and vice versa (phoneme). However, regarding the statistical results on the test set, the hypothesis of a significant difference can be discarded for the final results (end of optimization) which is also the part with the most noticeable gap on the validation scores. In other words, for some datasets, the algorithms slightly overfit on the train set, but not significantly enough to confirm that one is better than another.

To conclude this experiment, we have shown that our proposed solution reaches the state-of-the-art performance in the field of the AutoML based on Evolutionary Algorithms.

3.4.2 Tuning Mary-Morstan Evolutionary Space

By introducing new algorithms (3.3.2) and new variation procedures (3.3.3) we increase the number of possibilities to configure Mary-Morstan. Indeed, by only considering the algorithms and the procedures with static parameters (4 crossovers, 6 mutations, $4*2$ algorithms when multiplied with their variations), we have $2^{18} \approx 262$ thousands possible combinations of configurations. If we also take in consideration the parameters present in some variations, .e.g, sigma_ratio, or the parameters of the mu+lambda algorithms, we can have an infinity of possibilities. Knowing that for the smallest datasets, e.g. blood-transfusion (Fig. 3-17), a single run of Mary-Morstan easily takes 2 hours, it would roughly take 15 years[2] to try the 2^{18} configurations. In fact, it would be even more, because one run of Mary-Morstan is stochastic, meaning that multiple runs are required. And even if this time could be reduced by using parallel computing, it would not be enough.

Indeed, let's assume we have a cluster with 3000 cores with each run performed on a single core. Since a run is stochastic, we generally need at least 30 runs per configuration. With a grid search, this would give 218 days[3] to know which algorithm is the best for this dataset. While it sounds more reasonable, we considered a reduced EA search space, on the smallest dataset, with a lot of cores available. If only three instances of MutationGaussian are added

[2] $(2^{18} * 2)/(24 * 360) \approx 60$
[3] $((2^{18} * 2 * 30)/3000)/24 \approx 218$

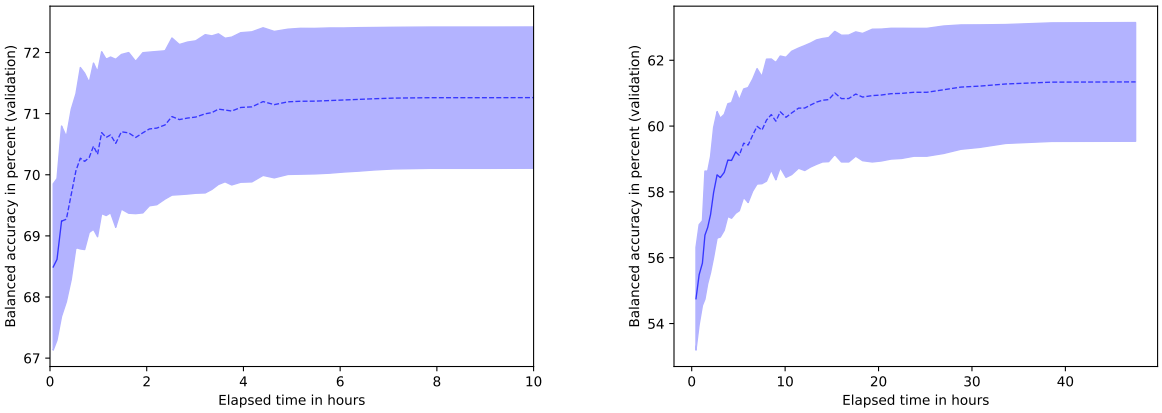


Figure 3-17: Convergence of Mary-Morstan with TPOT parameters for the dataset blood-transfusion (left), and jasmine (right). The line represents the average performance over 30 runs on the validation set, and is framed by the standard deviation.

(with a different sigma_ratio for each), and ran on a medium dataset size (.e.g, jasmine Fig. 3-17), it would take at least 10 hours to converge. This would throw our projection to 24 years[4] which, is not reasonable anymore, even with parallelism. Still, adding three parameters on the EA search space is limited, and plenty of other datasets remain to be benchmarked [8, 43] with even more samples and features.

Yet, there is no public research that studies the impact of the exploration versus exploitation trade-off for the evolutionary algorithms in AutoML. This experiment aims to check if there is any interest to tune the EAs of the AutoML for the classification problems.

3.4.2.1 Protocol

As previously introduced, finding the best EA search space is a combinatorial problem with a costly black-box function that depends on the dataset. For this reason, we first use I-Race [76], known as a metaheuristic optimization that looks for the best configuration of a black-box function on a given set of instances. We then run Mary-Morstan with the best configuration found by I-Race 30 times and use these results to compare with the "best known" state-of-the-art configuration, i.e. the results from Mary-Morstan instantiated as TPOT.

[4] $((2^{18+3} * 10 * 30)/3000)/(24 * 360) \approx 24$

I-Race works as follows: it first samples new configurations according to a particular distribution, then selects the best configurations from the newly sampled ones by means of racing, and finally updates the sampling distribution in order to bias the sampling towards the best configurations. These three steps are repeated until a termination criterion is met.

A global view of the architecture is provided in Figure 3-18. In our experiment, the target runner is the framework Mary-Morstan, the parameter space is the EA search space that we defined in Table 3.3, and the configuration scenario has been set with a budget (termination criterion) of 5000 experiments. We also let I-Race know that our target runner is stochastic, and by consequence should be run multiple times. Since it would be too expensive to compute the test score for each generation, we staged the optimization to the 30th and the 60th generation. In other words, I-Race is run two times per dataset, one run to find the best configuration at generation 30 and another for the best configuration at generation 60. In this way, through the unseen sets we can verify the presence of a significant progress along the optimization process.

To maximize our chances of having good results, we optimize the parameter space on each instance and not on a set of instances. The instances used are picked up from the Table A.1, and we exclude the expensive datasets that require too much computational resources. We also run I-Race through the Slurm batch mode in order to accelerate the optimization. Like the previous experiment, we used enroot and pyxis [3] and limit each run to 3500 MB of memory, and one single CPU.

Along with all our experiments, 75% of the original dataset has been used for the training data (with a 5-fold cross-validation strategy) and the 25% remaining have been used to test the best candidate found at the end of the optimization process. Due to the diversity of the datasets, the balanced accuracy metric was the most adapted metric to be used.

3.4.2.2 Results

We report the final performance on two different supports:

- Table 3.4 compares the final performance on the test set at the generation 30 and at the generation 60 for each algorithm.

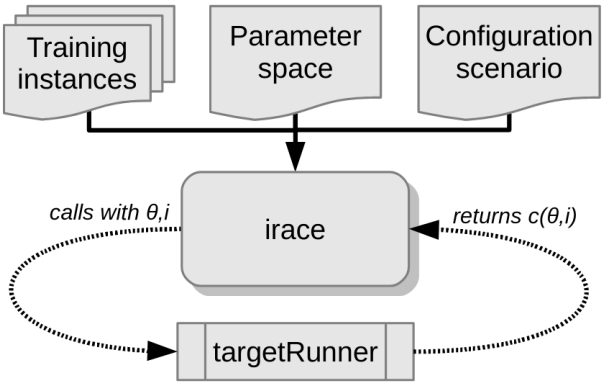


Figure 3-18: Scheme of irace flow information.

Table 3.3: EA search space.

Parameter	**Values**
algorithm	$\in$ {simple, mu+lambda, mu,lambda}
population	$\in [2..100]$
lambda	$\in [10..100]$
mu	$\in [10..100] : mu > lambda$
mut_portion	$\in [0, 1]$
cx_portion	$\in [0, 1]$
sigma_ratio	$\in [0.1, .5]$
CrossoverOnePoint	$\in$ {included, not_included}
CrossoverOnePointAverage	$\in$ {included, not_included}
CrossoverOnePointExactly	$\in$ {included, not_included}
CrossoverTwoPoint	$\in$ {included, not_included}
MutationDelete	$\in$ {included, not_included}
MutationGaussian	$\in$ {included, not_included}
n_ratio_gaussian	$\in [.1, .5]$ if MutationNGaussian = included
MutationInsert	$\in$ {included, not_included}
MutationNGaussian	$\in$ {included, not_included}
MutationNUniformInteger	$\in$ {included, not_included}
MutationOneRandom	$\in$ {included, not_included}
MutationReplace	$\in$ {included, not_included}
MutationUniformInteger	$\in$ {included, not_included}
n_ratio_uniform	$\in [.1, .5]$ if MutationNUniformInteger = included

- Figure B-10 draws the average performance on the validation score for each algorithm.

Among the 19 datasets, both algorithms have results on 12 datasets. The remaining one have unexpected errors with I-Race (batch of experiments with "infinite" score), and by consequence are not completely optimized by I-Race. To avoid biased results, we exclude them.

Table 3.4: Average of the test scores over 30 runs for M-M* (best configuration found by I-Race) versus M-M (TPOT configuration) at generation 30 and 60. The bold type means a statistical difference between M-M* and M-M (TPOT) for a given generation according to the Mann-Whitney U Test with 5% of significance.

Dataset	generation 30		generation 60	
	M-M*	M-M (TPOT)	M-M*	M-M (TPOT)
blood-transfusion	66.64 (± 5.23)	66.73 (± 3.42)	66.71 (± 3.93)	66.79 (± 3.63)
credit-g	70.25 (± 2.8)	70.54 (± 2.98)	69.11 (± 2.42)	70.04 (± 2.92)
kc1	73.45 (± 3.32)	73.24 (± 3.02)	73.49 (± 2.83)	73.78 (± 2.86)
Australian	85.77 (± 2.59)	86 (± 2.37)	85.91 (± 2.44)	85.98 (± 2.29)
jasmine	48.3 (± 4.5)	48.45 (± 4.75)	49 (± 5.27)	49.41 (± 5.38)
Amazon_employee_acc.	77.84 (± 1.41)	78.09 (± 1.27)	75.74 (± 3.2)	**78.69** (± .84)
bank-marketing	**87.05** (± .44)	86.58 (± .79)	85.1 (± 6.91)	**87.02** (± .50)
jungle_chess_2pcs	94.23 (± 2.18)	94.38 (± .84)	92.44 (± 1.31)	**97.34** (± .34)
adult	82.54 (± 1.59)	82.79 (± 1.44)	83.76 (± 1.21)	83.50 (± 1.42)
car	99.28 (± .86)	98.80 (± 1.53)	99.24 (± .86)	99.11 (± 1.51)
sylvine	96.07 (± .56)	96.13 (± .63)	95.54 (± .51)	96.2 (± .59)
segment	**96.51** (± .79)	94.02 (± .94)	**96.57** (± .4)	93.93 (± .9)

We observe interesting but not promising results on the convergence curves (Fig. B-10). We clearly see that the convergence varies from an instance to another. However the configuration M-M(TPOT) seems to converge faster or equivalently to the configurations found by I-Race for most of the datasets: credit-g, kc1, Australian, car, Sylvine, jasmine, jungle_chess, segment. Only few datasets remain (blood-transfusion, Amazon_employee_access, bank-marketing, adult) with M-M* that converges faster (at some steps). While these results on validation sets highlight the difference between the solutions with a preference for M-

M(TPOT) configuration, the results on test sets slightly change the scene.

On the test set (Table 3.4), I-Race was able to finding a configuration M-M* that outperforms the M-M(TPOT) configuration on two datasets (bank-marketing and segment) at the generation 30, and is statistically equivalent for the remaining datasets. However, at generation 60, M-M* and M-M(TPOT) are equivalent for the dataset bank-marketing. In fact, the M-M(TPOT) instance works even better at generation 60 as it is highlighted by the statistical test which demonstrates a significant difference for three datasets (Amazon, bank-marketing, jungle_chess), while M-M* only outperforms one dataset (segment).

In consequence, it seems that the convergence results were mostly overfitting on the training set, which is confirmed by the statistical test performed on the test set at different steps of the optimization process. Finding a better configuration that statistically brings a faster convergence might be interesting but remains computationally expensive (it is a third layer optimization) with a very small benefit (we have a small gain of performance for two datasets during the early phase of optimization). Indeed, we had to use an optimizer (I-Race) which is a costly process to optimize the EA space for only one specific dataset and limits his purpose, i.e. find a configuration that works better for a set of instances.

To conclude this experiment, finding an EA configuration that statistically outperforms all along the optimization process is not promising at first-sight. We explain these unexpected and bad results by different reasons. First, I-Race has been mostly used, and designed for non-expensive black-box functions, which is not the case for AutoML. Evaluating one solution easily takes seconds if not minutes. This leads I-Race with too few results regarding the number of candidates evaluated (number of experiments) and the search space it is looking for. Second, our EA space is relatively vast and surely not optimized enough. Some operators might act too similarly and should by consequence be removed to have a smaller EA space. Finally, Machine Learning is already an optimization process, AutoML adds another level of optimization, and on top of that we add I-Race, again another level of optimization. At the end, there are three layers of optimizations, which becomes a little bit too tricky to solve classification problems.

Even if the results were not promising for classification problems, it still might be interesting to continue the investigations on other problems such as on Time Series Classification,

or Multi-objective problems. Indeed, these other problems require a different approach of optimization, i.e. a different EA space, and by consequence the experiment remains challenging and could lead to more promising results.

3.5 Conclusion

To conclude this chapter, we proposed a state-of-the-art AutoML solution capable of handling classification problems and regression problems. We demonstrate that our performance reaches the state-of-the-art with the latest AutoML Benchmark [43]. We also designed our AutoML to be modular, in such a way that the exploration versus exploitation process can be tuned through the specification of an Evolutionary Algorithm (EA) space. The introduction of the EA space eases research to observe how the performance is impacted when the space is tuned. Despite our unpromising results on classifications problems, we remain convinced that tuning the EA space might impact positively the performance on others problems (e.g. regression, time series classification, forecasting, anomaly detection, and so on). In fact, in later research that we expose in Chapter 5, it demonstrated a real interest. Thanks to the modular aspect of Mary-Morstan, we easily tuned the EA space to solve time series classification problems. Such a benefit might also be useful with multi-objective problems that usually search in a multitude of directions, which tends to slow down the optimization when the objectives are in contradiction.

Through all our experiments, we noticed a major problem. The time to evaluate a machine learning pipeline is relatively costly. It takes multiple seconds for small datasets, and up to multiple hours for large ones. Thus, we decided to tackle this problem in the next chapter by proposing a technique based on Successive Halving that we adapted to the evolutionary algorithms in order to accelerate our AutoML process.

Chapter 4

Mary-Morstan-SH: a Technique to Tackle Large Datasets and Accelerate the Optimization

In this chapter we propose a new technique that permits any AutoML solution based on evolutionary algorithms to accelerate its optimization process. We first introduce our motivations, the principle of the technique, and how to integrate it with any evolutionary algorithm. Then we elaborate the experimental setup, followed by the associated results.

This chapter has been the subject of a publication in the proceedings of the 31st International Conference on Tools with Artificial Intelligence (ICTAI) [92], and two patents[1] with OVHCloud.

4.1 Introduction

4.1.1 Motivation

Most of the AutoML are slow when they deal with large datasets. Indeed, the training set size and the number of candidates to evaluate are substantial.

[1] US20200272909A1 / EP3702974A1

This issue has been noticed during an AutoML challenge [46], particularly by the authors of Auto-SKlearn who improved their solution by integrating a more adapted exploration method called Successive-Halving (SH) [63], which reduces the complexity of the AutoML trained on large datasets (PoSH [37]). Broadly speaking, SH consists in progressively exhausting a defined budget that we detail in the next section (4.1.2).

During our experiments of Mary-Morstan on classification (3.4), we notice the same issue. Running our AutoML on large datasets is costly. Thus, we think that integrating SH in our process would help us solve the problem. However, SH has been adapted to run on SMBO approaches, not on evolutionary algorithms which is the core method used by our tool. For this reason, we propose to adapt the method such that Mary-Morstan and any other AutoML based on EAs can use it.

The integration of such a method would also permit to reduce the costs in both computational power and memory.

In the following sections, we describe the concept of SH which inspires the solution that we propose: Mary-Morstan-SH (MM-SH).

4.1.2 Successive-Halving principle

Sequential Halving [63] also called Successive Halving [56] (SH) is a technique introduced in the Multi-Armed Bandits problem. The strategy consists in eliminating the worst half of the arms as well as increasing the number of times that surviving arms are pulled during the iterations of the process.

4.1.3 Include Successive-Halving in Evolutionary Algorithms

Our proposal is to adapt the two main components of SH in the evolutionary process. We replace the number of arms by the population size (4.1) and the number of times that arms are pulled by the sample size used by a candidate to be evaluated (4.2) that we call the budget. The interest of such a method is to spend little time on as many candidates as possible, pruning bad ones and then spend more time on promising ones.

$$p_i = \mathcal{P}/2^i \tag{4.1}$$

$$b_i = b \times 2^i \tag{4.2}$$

We denote p_i and b_i the population size and the budget at i^{th} generation. We initialize $\mathcal{P}$ with the initial population size specified in the evolutionary algorithm and b as a certain percentage of the training set size.

In the classical use of the SH, the reduction of the number of arms and the increase of the budget are applied at each step of the loop. However, applied on Evolutionary Algorithms (EAs), population size quickly decrease to one individual only as well as the budget is totally consumed (i.e. budget is equal to the dataset size). Also EAs need several successive generations to find interesting regions of the search space. Therefore we adapt the original formulas and end with the following equations (4.6) and (4.7) which respectively reduces the population size and increases the budget at some step of the generations.

To find these equations we replace i by $i \times a$, where a is a coefficient that controls the point when the population is divided by two. So for the population size p_i, we have:

$$p_i = \mathcal{P}/2^{i \times a} \tag{4.3}$$

To find the coefficient, we need to be able to define the number of iterations needed such as there are m individuals left in the population:

$$\mathcal{P}/2^i = m \implies i = \log_2(\mathcal{P}/m) \text{ iterations} \tag{4.4}$$

Then we distribute the ratio on the G generations, thus give:

$$a = \log_2(\mathcal{P}/m)/G \tag{4.5}$$

We use the floor to the whole equation to keep a natural number that represents the population size. We also use the floor to the exponent to ensure that division of the population

is performed by a factor 2. Finally, we insert "+1" to balance the equation such that the number of iterations that happen before the divisions are equals. It yields to:

$$p_i^\star = \lfloor \mathcal{P} / (2^{\lfloor i \times \lfloor (\log_2(\mathcal{P}/m)+1) \rfloor / (G+1) \rfloor}) \rfloor \tag{4.6}$$

Where m is the minimal number of individuals to keep in the parent population, G and $\mathcal{P}$ are respectively the number of generations and the initial population size specified in the evolutionary algorithm.

Concerning b_i, we applied the same process to replace the exponent.

$$b_i^\star = b \times 2^{\lfloor i \times \lfloor (\log_2(B/b)+1) \rfloor / (G+1)) \rfloor} \tag{4.7}$$

Where b is the initial budget and B the maximum budget. Fig. 4-1 allows to have a better representation of the new defined functions $p_i^\star$ and $b_i^\star$.

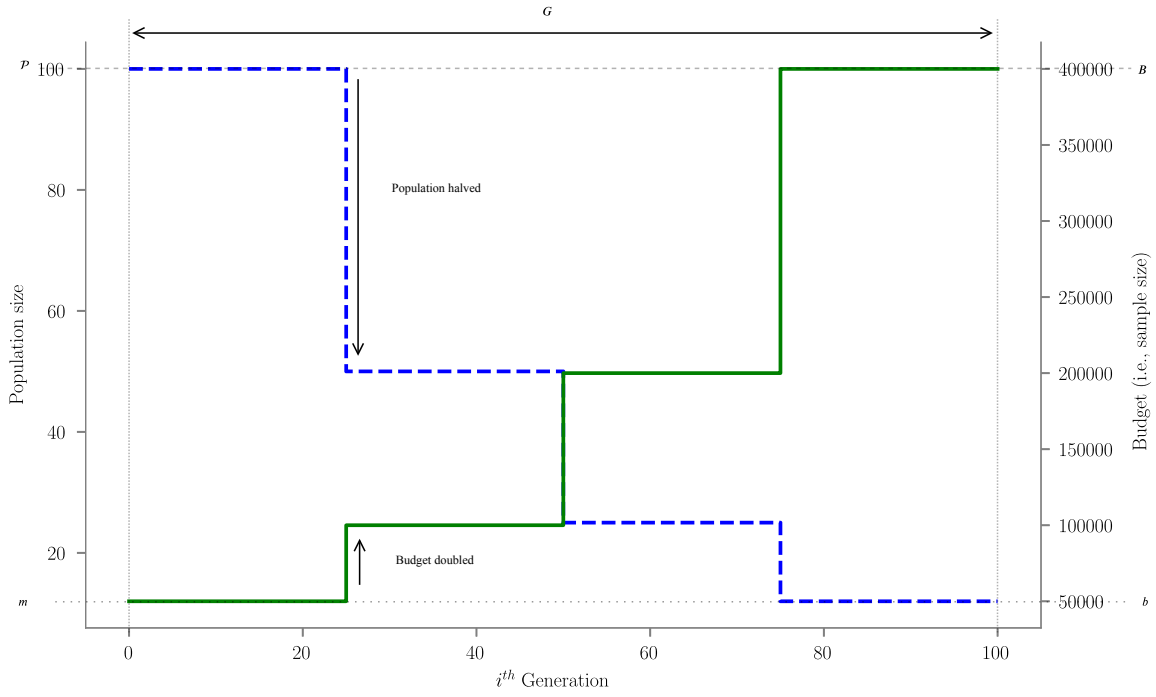


Figure 4-1: Representation of $p_i^\star$ in blue dot line and $b_i^\star$ in green solid line. The horizontal axe represents the generations. The vertical left axe and the vertical right axe respectively represent the population size $p_i^\star$ and the budget $b_i^\star$ at i^{th} generation.

The implementation of our tool uses Mary-Morstan with the $(\mu + \lambda)$-ES (see 1) in such a way that at i^{th} generation, values $\mu = \lambda = p_i^\star$ and that first objective of the fitness function F is evaluated on $b_i^\star$ samples.

Procedure of MM-SH is provided in Algorithm 2.

Algorithm 2 Mary-Morstan-SH

Input: population size, $\mathcal{P}$; minimum of individuals, m; initial budget, b; maximum budget, B; number of generations, G; training set, $\mathcal{D}_t$
Output: best candidate from the Pareto front

$b_0 \leftarrow b$
$p_0 \leftarrow \mathcal{P}$
$\alpha^{(0)} \leftarrow \text{generate}(\mathcal{P})$
$\mathcal{D}_0 \leftarrow D' : D' \subseteq \mathcal{D}_t, |D'| = b_0$
$\mathfrak{P}_p^{(1)} \leftarrow \{(\alpha_k^{(0)}, \text{evaluate}(\alpha_k^{(0)}, \mathcal{D}_0)), k = 1, ..., \mathcal{P}\}$
for $i := 1$ **to** G **do**
  $b_i \leftarrow b \times 2^{\lfloor b \times \lfloor (\log_2(B/b)+1) \rfloor / (G+1)) \rfloor}$
  $\mathcal{D}'' \leftarrow D' : D' \subseteq \mathcal{D}_t, |\mathcal{D}' \cup \mathcal{D}_{i-1}| = b_i, D' \cap D_{k-1} = \emptyset$
  $\mathcal{D}_i \leftarrow \mathcal{D}_{i-1} \cup D''$
  $\alpha^{(i)} \leftarrow \text{clone}(\mathfrak{P}_p^{(i)}, p_{i-1})$
  $\alpha^{(i)} \leftarrow \text{variation}(\alpha^{(i)})$
  $\mathfrak{P}_o^{(i)} \leftarrow \{(\alpha_k^{(i)}, \text{evaluate}(\alpha_k^{(i)}, \mathcal{D}_i)), k = 1, ..., p_{i-1}\}$
  $p_i \leftarrow \lfloor \mathcal{P} / (2^{\lfloor i \times \lfloor (\log_2(\mathcal{P}/m)+1) \rfloor / (G+1) \rfloor}) \rfloor$
  $\mathfrak{P}_p^{(i+1)} \leftarrow \text{select } (\mathfrak{P}_p^{(i)} \cup \mathfrak{P}_o^{(i)}, p_i)$
end for
return $\alpha^\star \in \text{opt}.F$

We denote $\alpha^{(i)}$ the list of non-evaluated candidates, $\mathfrak{P}_p^{(i)}$ the parent population, $\mathfrak{P}_o^{(i)}$ the offspring population and $\mathcal{D}_i$ the i^{th} subset of the training set $\mathcal{D}_t$. The procedures are the following (each one being instantiated according to the experiments presented in Section 4.3):

- **generate**(p): generates a list of p candidates. In our case each candidate is a tree whose nodes are picked up from primitives and terminals from the leaves.

- **evaluate**(c, d): evaluates the candidate c on dataset d. In the experiments, a 5-fold cross-validation is used on the dataset d. The objective value assigned to a candidate is the performance computed on the validation set.

- **clone**(p, λ): clones λ candidates by selecting them randomly in the population p.

- **variation**(p): applies on each individual of the population p a variation operator, i.e. a mutation or a crossover, according to the rate in Tab. A.4.

- **select**(p, μ): Select μ candidates from the population p. TPOT used the efficient selection from NSGA-II algorithm. In the $(\mu + \lambda)$-ES, the population p is composed with the parent population $\mathfrak{P}_p^{(i)}$ and the offspring $\mathfrak{P}_o^{(g)}$

4.2 Experiments

In this section, we describe our experimental protocol that allow us to compare the optimization performance of MM-SH versus TPOT.

4.2.1 Protocol

Evolutionary algorithms are stochastic and have to be run several times in order to measure their performance. In our experiments, we run TPOT and MM-SH 30 times each. We keep the original settings from the article of TPOT [90] as described in Tab. A.4. For large datasets, we reduce the number of generations and for TPOT, add a supplementary termination criterion based on time. Indeed, TPOT would use too much time to provide results on large datasets. By observing early convergence around 25 generations during the experiments on small datasets, we fix the number of generations to 25 for the large ones with MM-SH. The termination criterion is fixed to the time needed by MM-SH to stop naturally (i.e. after the 25 generations).

Contrarily to TPOT protocol [90] which presents the results from the performance of the best candidate (candidate maximizing accuracy in the Pareto front at the 100^{th} generation, i.e. the last one), we measure the performance of each individual in the population at each generation. This measure allows us to know how the optimization process converges.

To evaluate our experiments, we used a sample of 75% of the original dataset as training data and test the candidates on the remaining 25%. The split is shuffled in a stratified way with a different seed per run, and are identical for both algorithms with the aim to have comparable results (same training and test set for each paired run).

By design, at generation i, the number of samples seen by MM-SH depends on the budget $b_i^\star$, which is a percentage of instances taken from the 75% of the training set. This budget serve to perform a cross-validation on candidates. In the case of TPOT, the whole training set serve along the generations to perform the cross-validation of the candidates. Hence MM-SH is evaluated on fewer and different instances than TPOT. Thus it makes the results from the cross-validation less accurate. To solve this problem, we use an identical test set (25% remaining) between TPOT and MM-SH that fairly evaluate and compare both algorithms.

In order to compare the performance of TPOT and MM-SH, we compute the average elapsed time between generations, and for all generations, the average performance of the ML pipelines.

Table 4.1: Own settings of MM-SH

Parameter	**Value**
Initial budget b	30% of samples from training set
Maximal budget B	100% of samples from training set
Minimum individuals m	10

MM-SH has its own parameters that are fixed as described in Tab. 4.1. We naturally set the maximum budget to 100% in order to make available the whole training set in the latest iterations of the optimization process.

4.2.2 Dataset Corpus

Both TPOT and MM-SH are run on 8 datasets (see Tab. 4.2). The small datasets have been picked up from the experiment in the article of TPOT [90] in an attempt to validate our experimental setup. We additionally provide 4 large datasets (composed of hundred thousand instances) from an AutoML benchmark [43].

4.2.3 Computational environment

We conduct the experiments on OVH Public Cloud with different clusters depending on our requirements. For small datasets, we use two C2-120 virtual machines (VM) composed of 32

Table 4.2: Classification datasets

Dataset	# Inst.	# Attr.	# Class.	Majority class
wine-quality-red	1599	12	10	43%
car-evaluation	1728	6	4	70%
spambase	4601	57	2	60%
wine-quality-white	4898	12	10	45%
miniboone	130064	51	2	72%
kddcup99	494020	41	23	57%
airlines	539383	8	2	55%
covertype	581012	54	7	49%

denote the cardinality.

cores of 3.1Ghz and 120GB RAM each. The first VM is used to run TPOT and the second one for MM-SH. This way we ensure fair computation environments for both algorithms. The large datasets require more memory, thus we use multiple R2-240 VMs with 240GB RAM and 16 cores of 2.3Ghz. For example, each run of TPOT for the airlines dataset took up to 20GB of memory. With only 12 runs, the machine was out of memory. In order to prevent this problem, we set up multiple VMs and fairly balance runs on clusters by keeping fair computation environments for each algorithm.

4.3 Results

In this section we describe the results of our experiments. Firstly, we compare the performance of MM-SH and TPOT on small datasets, and then on the large datasets.

4.3.1 Small Datasets Results

Fig. 4-2 presents the results on the small datasets. MM-SH does not perform well on these small datasets. Indeed, the performance of MM-SH is always below TPOT over the 100 generations.

The results are explained by the small number of instances in each dataset at the beginning of the run of MM-SH that leads to overfitting. However, as mentioned in our motivation, MM-SH has not been designed for small datasets, but to validate our experimental setup.

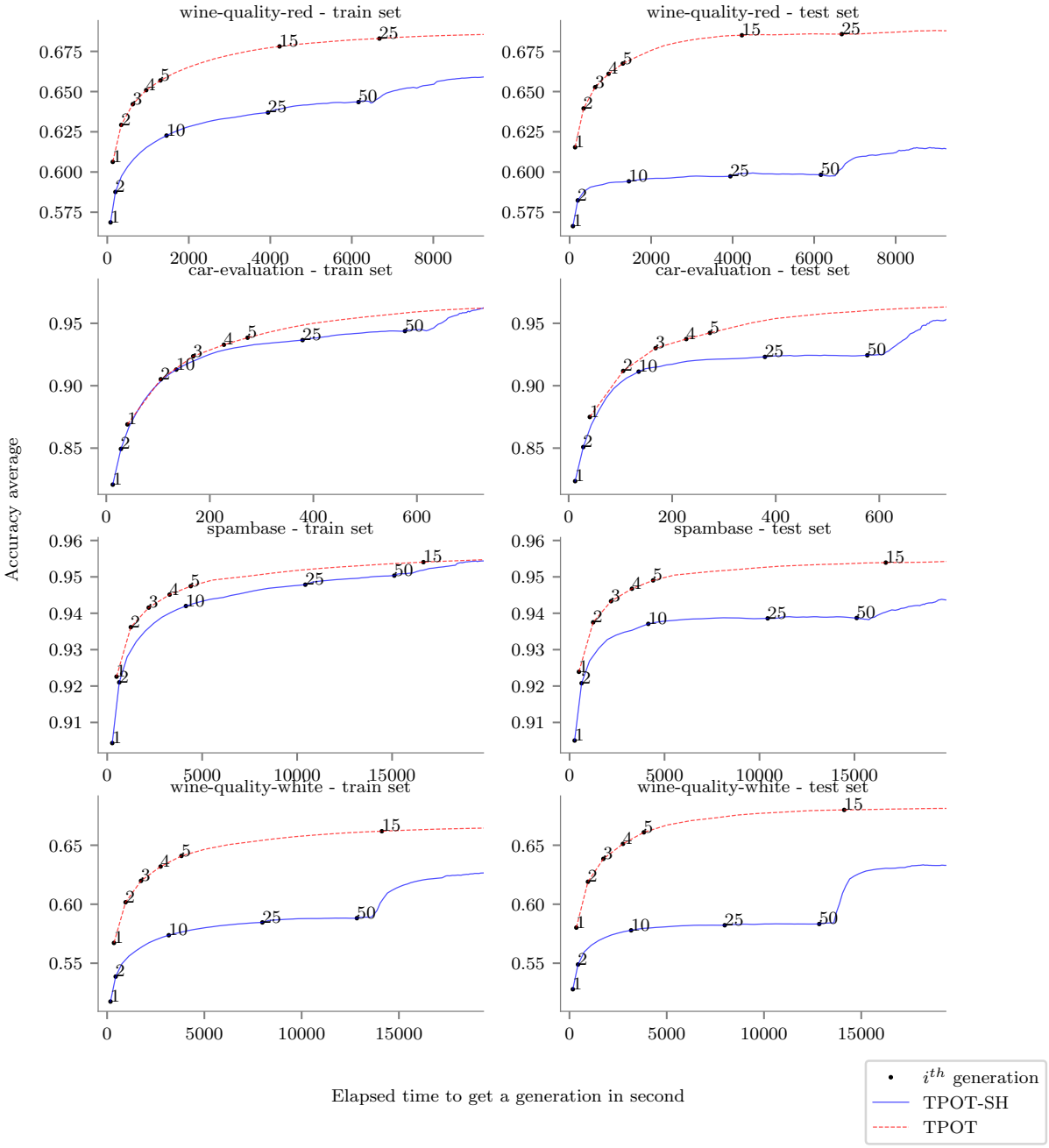


Figure 4-2: Results for small datasets. Panes on left side are performance from cross-validation on training set, panes on right side are performance from test set. Each horizontal axe represents the elapsed time in seconds and each vertical axe represents the accuracy. TPOT is represented as a red dot line and MM-SH as a blue solid line. Each black point represent when a generation is completed, i.e. all individuals of the population are evaluated.

Table 4.3: Accuracy on test set at different time where T1 and T2 respectively represent the time that MM-SH and TPOT obtained the first generation and T3 is the time that MM-SH obtained the latest generation.

Dataset	**T1**		**T2**		**T3**	
	TPOT	MM-SH	TPOT	MM-SH	TPOT	MM-SH
wine-quality-red	72 seconds		180 seconds		2.22 hours	
	N.A.	56.6%	61%	58.26%	68.47%	62.1%
car-evaluation	72 seconds		212 seconds		1.73 hours	
	N.A.	82.79%	87.45%	85.92%	97%	95.15%
spambase	360 seconds		810 seconds		7.54 hours	
	N.A.	90.74%	92.39%	92.1%	95.4%	94.6%
wine-quality-white	165 seconds		432 seconds		5.22 hours	
	N.A.	52.75%	57.9%	55.12%	68.2%	63.2%
miniboone	2.61 hours		6.17 hours		46 hours	
	N.A.	89.4%	83.5%	91.8%	93.3%	94%
kddcup99	6.57 hours		18.6 hours		72 hours	
	N.A.	98%	96.2%	99.9%	99.9%	99.9%
airlines	2.32 hours		8.8 hours		53.5 hours	
	N.A.	63.8%	61.4%	65.6%	66.2%	66.6%
covertype	7.5 hours		12.15 hours		101.5 hours	
	N.A.	71.2%	61.8%	75.2%	79.45%	96.2%

N.A. for Not Available.

We can notice that the MM-SH's convergence line is irregular with some bumps. It is explained by the population size which is divided at some generations. It implies that the performance of the population is not homogeneous. Indeed, when the population size decreases between two successive generations, the average is computed on less and better candidates.

As expected, we observe that MM-SH obtained the first generation before TPOT, since the size of the training set, controlled by SH, is very small. This benefit is insignificant for small datasets but become more significant for large datasets.

4.3.2 Large Datasets Results

Fig. 4-3 presents the computed results for large datasets. We clearly observe that MM-SH performs better than TPOT on the training set as well as on the test set. MM-SH finds in average better candidates and faster. In order to have a better comprehension, we detail the results for miniboone dataset and provide an overview of the results for all datasets in Tab. 4.3.

At the first generation of miniboone dataset, MM-SH obtains an accuracy of 89.6% on training set and 89.4% on test set in 2.61 hours. TPOT obtains the first generation in 6.17 hours with an accuracy of 78.3% on training set and 83.2% on test set. Thus, we divide the time of getting first generation by two, and at the same time the accuracy is increased by 6% on test set. It would take a total of 14 hours from the initial run, i.e. five times longer to TPOT to reach a similar level of performance as the one MM-SH reaches. After a few generations, the population quickly converges and we clearly observe that TPOT stays below MM-SH. After 46 hours of run, TPOT converges to a performance of 93.3% that is already achieved by MM-SH 29 hours earlier.

At first reading, these results can confuse machine learning practitioners, indeed, TPOT has more knowledge, i.e. 75% of training set, compared to MM-SH during the first generations (e.g. 30% from the 75% of the training set). Hence TPOT should give better performance, however, TPOT has a timeout of 5 minutes by default to evaluate a candidate. If the time is passed, the candidate is discarded. This explains why good candidates that take too much time to be evaluated are not included in the computation. To verify this

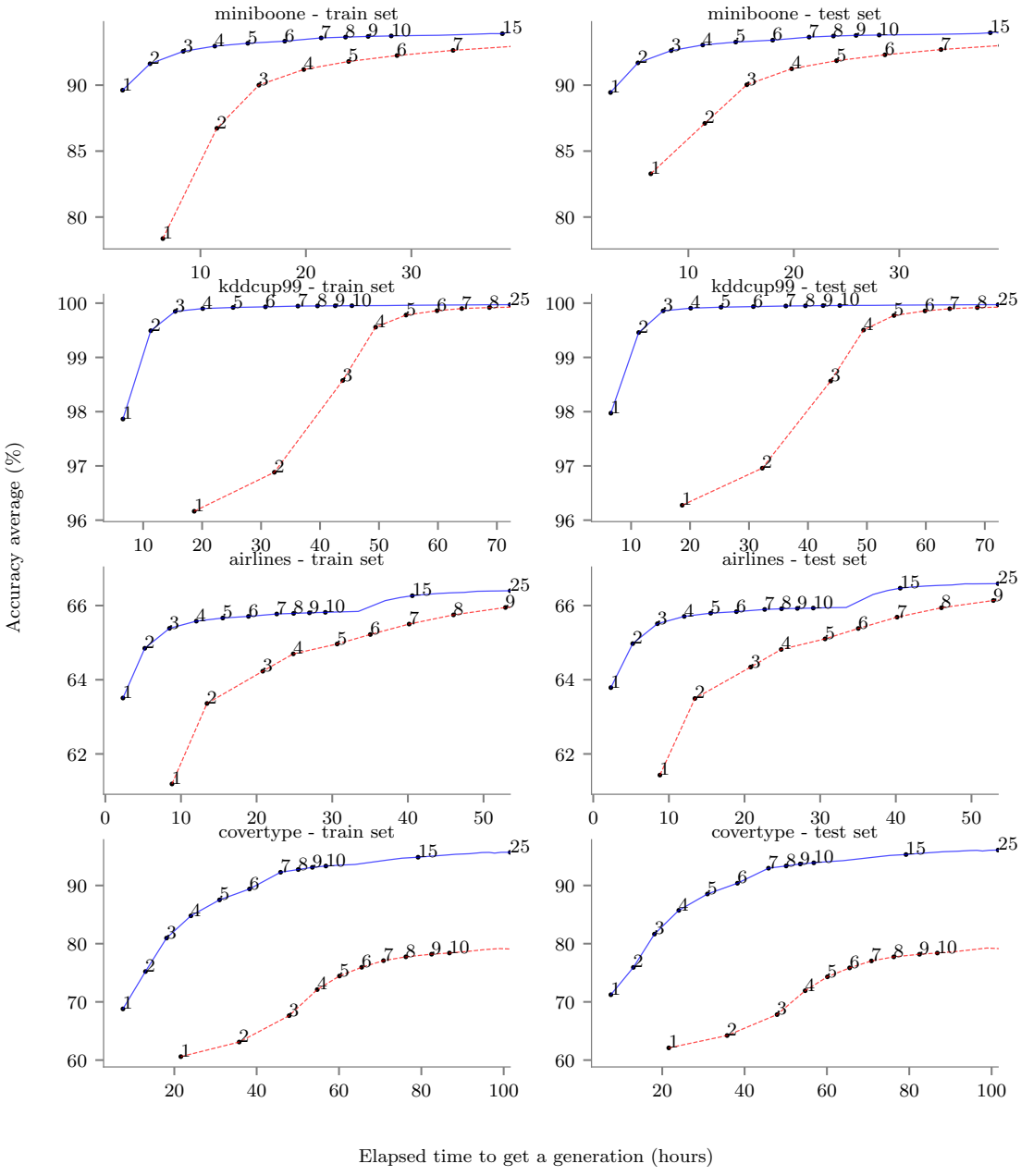


Figure 4-3: Results for large datasets. Panes on left side are performance from cross-validation on training set, panes on right side are performance from test set. Each horizontal axe represents the elapsed time in hours and each vertical axe represents the accuracy. TPOT is represented as a red dot line and MM-SH as a blue solid line. Each black point represent when a generation is completed, i.e. all individuals of the population are evaluated.

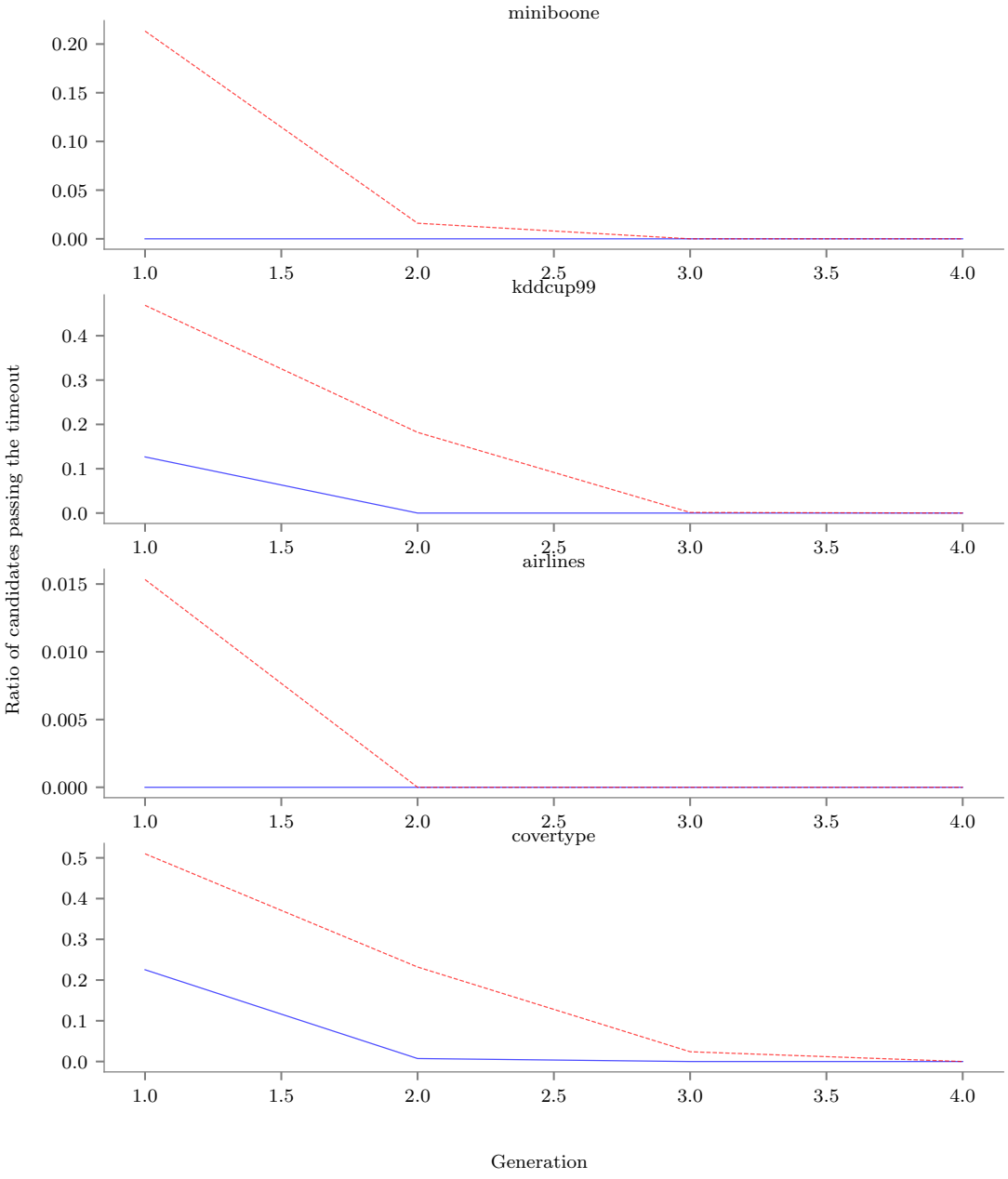


Figure 4-4: Unevaluated candidates on large datasets. TPOT is represented as a red dot line and MM-SH as a blue solid line.

hypothesis, we draw the ratio of non-evaluated candidates per generation in Fig. 4-4. We can observe that timeout is reached for almost half of the candidates at the first generation in TPOT (see for example covertype and kddcup99 datasets, the largest datasets in terms of instances $\times$ features), but then decrease to zero after two or three generations. Even if it is less significant, this phenomenon also happens with MM-SH. We did not include results after generation four because it is just a flat line with no candidates having a timeout. Concerning small datasets they never reached the timeout regardless of the generation. One manner to solve this issue is to increase the timeout, that would certainly be a benefit for TPOT on early generations as well as for MM-SH. Nonetheless it would also be a disadvantage in term of time to get the first generation. Hence, increasing the timeout does not solve the problem of accelerating the optimization process. Moreover, the number of non evaluated candidates is not significant for airlines dataset which has good and similar performance to other large datasets. By this fact, we can be quite confident that timeout procedure does not bias our results. In consequence, having a smaller subset of the training set that represents the dataset well enough to train ML algorithm leads to faster and better performance.

Note that dataset airlines has an irregularity. This abrupt change is not visible in other large datasets because the performance of the candidates is homogeneous. When the performance of the candidates is not homogeneous, our selection process that decreases the population size impacts the average performance. Indeed, we start with a population of 100 candidates which falls to 25 individuals at generation 13. This is significant when there is variance in the performance of the population.

Another remark concerns the performance of training set versus test set. They often do not have a lot of differences regardless of the generation and the dataset. This is explained by the stratified cross-validation scheme that keeps a training set representative to the test set. In this way, based on our results, we could think that comparing performance on training set or test set is the same. However we insist that keeping the test set as a measure is a good thing to ensure reliable and comparable results when the sample size varying as we propose through the budget process.

As we can see, our algorithm is far ahead from TPOT on large datasets. The first generation is always obtained much faster. This can be significantly important for application

requiring results in a limited amount of time. Moreover, we obtain better results. The strength of our solution mainly resides on the budget, which on large datasets take a subset part that represents the dataset well enough to train good ML pipelines.

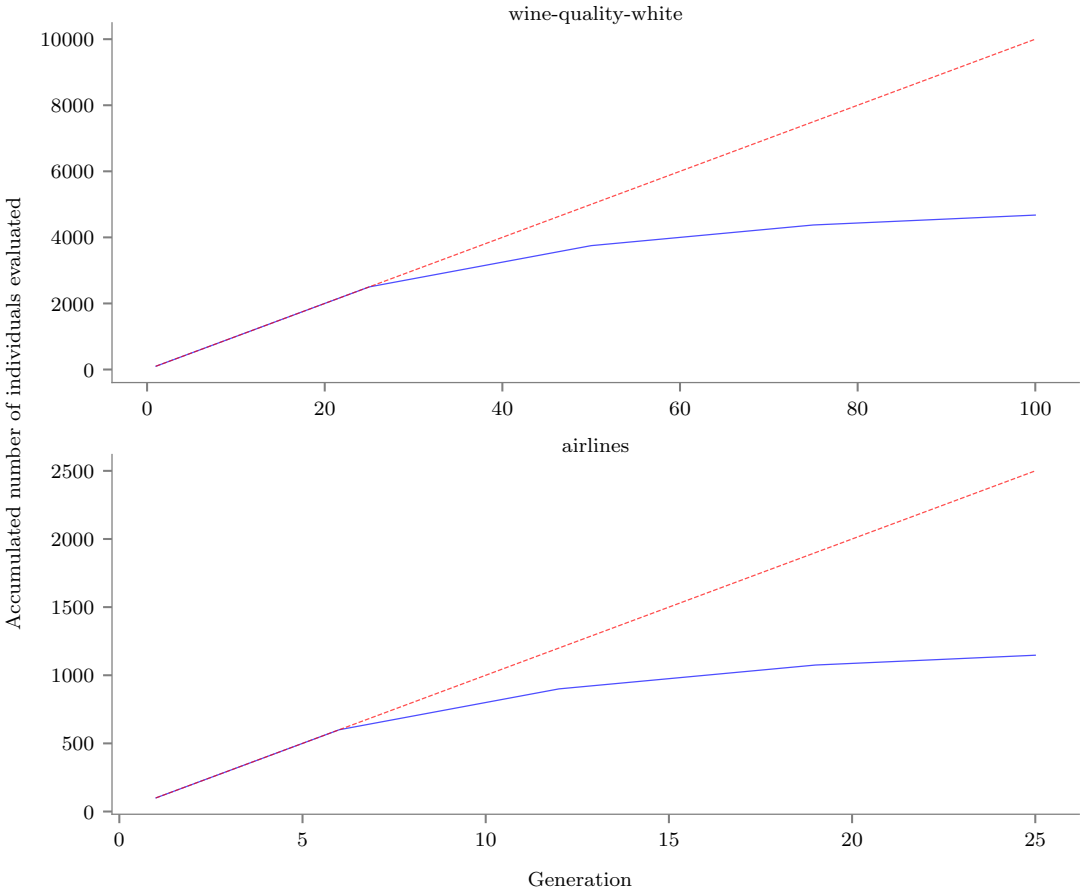


Figure 4-5: Total of accumulated evaluations per generation. TPOT is represented as a red dot line and MM-SH as a blue solid line.

4.3.3 Discussion

Here we discuss about our results and provide some hint for future works on a short term in an attempt to improve MM-SH.

First of all, we expected better results for small datasets size. One way to improve the current results is to se tup a higher initial budget in such a way that subset will be represen-

tative enough for the problem. However, when a dataset is very small, it is complicated to find such a subset. It would be interesting to study some statistical methods to know what percentage of the training set size would represent the dataset well enough. This percentage would serve as an initial budget in equation (4.7). Another solution to consider is to change the principle of doubling the budget by multiplying it with another factor than two in equation (4.2) and this could also be done for the population size in equation (4.1). Changing the factor would not only be an improvement for the performance on small datasets but also for the performance on larger ones. Studying different factors is another lead to follow. We did not explore this aspect because we initially planned to respect the concept of successive halving.

In complement to the results on the optimization convergence, we extracted the total number of evaluated candidates per generation for two datasets in Fig. 4-5, a small one and a large one. In total, TPOT evaluates 10100 candidates, i.e. 100 pipelines times 100 generations plus 100 candidates from generation 0 for the small dataset, and 2600 for the large one. In the case of MM-SH, it evaluates 4763 candidates for the small dataset and 1235 candidates for the large dataset. MM-SH evaluates approximately two times less candidates compared to TPOT. This behavior depends on the parameter m, representing the minimum number of individuals in equation (4.6). These curves are interesting because it shows that MM-SH evaluates the same number of candidates than TPOT during the first generations and obtains faster as well as better results. In consequence, we deduce that budget in equation (4.7) is the main component influencing the performance at the beginning. Also, these curves express how MM-SH does not explore the search space as much as TPOT. Meaning that MM-SH has the potential to improve the performance by starting with more candidates at the initialization of the population size in such a way that the total number of evaluations is fairly equal to TPOT evaluations. This gain of candidates should give more diversities and more choices during the successive selections and thus a better overall optimization performance. As a side effect, starting with more candidates would increase the time needed to obtain results due to the evaluation of more candidates. However this would only happen at the beginning of the optimization process because the population size decrease later. Moreover this side effect can be reduced by having a low initial budget. Since

the focus of our work was the speed of the optimization process, we did not investigate on higher population size. Studying different population sizes with different budgets at initialization of MM-SH would be one more leading point for enhancements.

Another perspective of amelioration indirectly related with our solution but where we questioned ourselves during the experiments is how to specify the number of generations. We followed the parameters from the article of TPOT [90]. However there are no explanations why they are using 100 generations. This issue could be simply solved by comparing the performance from a generation to another and automatically considered when the optimization process has converged to decide when to stop. Similarly, there are no explanations why they are using 100 individuals per generation. These parameters have been chosen arbitrarily. It would be interesting to study different population size to see how the optimization process performs. Hopefully, as seen on results from small datasets, 100 generations and 100 individuals are enough to converge, but it will not necessarily be the case for all dataset types, e.g. very large ones. Please note that studying the initial population size from TPOT is different than it is in MM-SH. Indeed, while TPOT keeps it constant, MM-SH varies the population size along the generations.

Lastly, an interesting aspect we did not explore is to experiment our solution on bigger datasets with more than millions of samples. We are pretty confident that our method would perform pretty well on such dataset size thanks to the notion of maximum budget in equation (4.7). Indeed, this aspect avoid the optimization process to take too much time. TPOT maximum evaluation time would not be equivalent to this aspect, because when this timeout is reached, the candidate is simply discarded, that is not the case with maximum budget. So the maximum budget not only reduces the optimization time but also permits complex candidates to be evaluated and kept along the runs.

To summarize, MM-SH could be enhanced by trying different factors in equations (4.2) and (4.1). By exploring different population sizes, initial budgets and maximum budget, it should give even better results and handle larger datasets in a reasonable time.

4.4 Conclusion and future works

To conclude this chapter, we proposed a solution permitting evolutionary algorithms to solve the AutoML problem faster on large datasets with better results. The implementation of the solution is pretty simple and does not increase the complexity of the evolutionary algorithms. Also, we did not insist on this point but we notice that our solution requires two times less memory space at least, along the whole optimization process. This can be a considerable gain for infrastructures with hardware constraints.

This improvement could serve to accelerate the optimization in other problems, e.g. regression, time series classification. It also gives the advantage to train more candidates, and by consequence increase the diversity of candidates, which can be beneficial for multi-objective AutoML solutions [99].

Chapter 5

AutoTSC: an Instance of Mary-Morstan Dedicated to the Time Series Classification Problem

In this chapter, we first motivate our choice to develop an AutoML solution called AutoTSC that automatically solves the Time Series Classification (TSC) problem. Then we detail the experimental setup we used, is followed by the results demonstrating the advantage of our solution. Finally, we give some perspectives for futur research.

This chapter has been the subject of a publication in the proceedings of the 33rd International Conference on Tools with Artificial Intelligence (ICTAI) [93].

5.1 Introduction

In the previous chapter, we approached machine learning algorithms on datasets without temporal information. However, in some use cases, the data are subject to changes that depend on preceding data, or on time. Most of the classical ML algorithms (Table. A.2) assume that the features are independent, and do not consider the temporal dependence. In consequence, those algorithms will not perform efficiently on such data.

5.1.1 Illustration

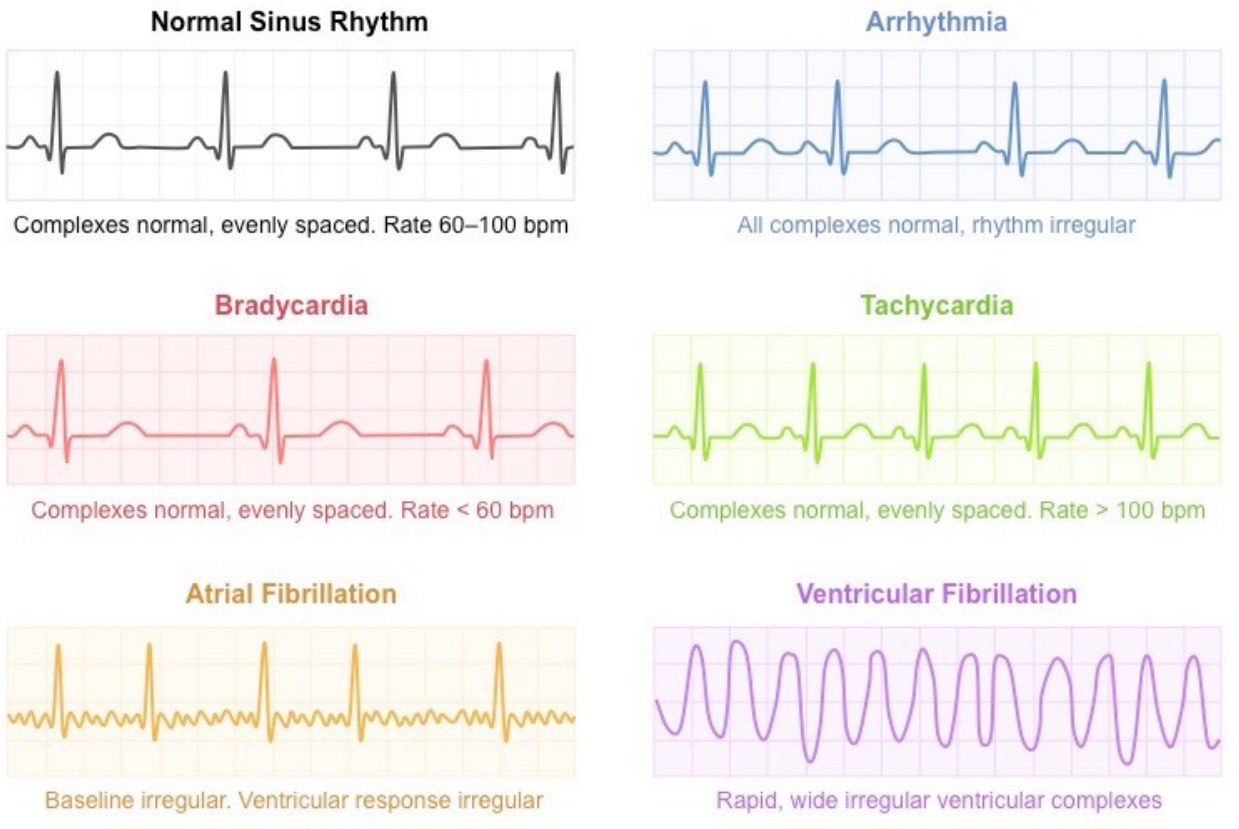


Figure 5-1: Different Cardiac Rhythm Diagnoses.

To better understand the importance of these notions (dependence on time or on preceding data), we illustrate the problem through a basic use case in the field of Healthcare. Let's imagine we build a model that finds abnormal heartbeats [62] thanks to the annotated data measured by an Electrocardiogram (ECG) (Fig. 5-1).

To distinguish the different anomalies, we use an interpretable model, a Decision Tree (Fig. 5-2). First, we need to define the features. Apparently, taking the Beats Per Minute (BPM) would be enough to discriminate some of them, e.g. the Normal Sinus Rhythm, the Bradycardia, and the Tachycardia. However, if we include the Arrhythmia, the BPM varies, and our model would produce false positives which can be dramatic for the health of the patient. Taking the length between the different segments, e.g. between the waves and the QRS complex (see Fig. B-11 for the references) would result in the same issues. Indeed, the length varies as well as the BPM from a step to another. In fact, at some step

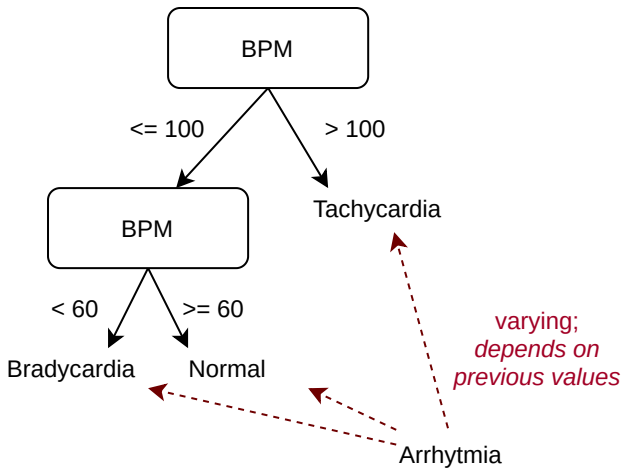


Figure 5-2: Biased Decision Tree model to distinguish the heartbeat anomalies.

we recognize the Bradycardia pattern, while at some others we observe the Tachycardia or even the Normal Sinus Rhythm.

To better discriminate the heartbeat anomalies, we need to extract the information differently, by including the change from a step to another, which makes the features more dependent. In this way, saying that the BPM hange from 60 to 100, and then from 100 to 60 and so on along the time, would certainly better recognize the Arrhythmia. A similar reasoning can be made on the Atrial Fibrillation and Ventricular Fibrillation anomalies.

This very simple illustration highlights the importance of taking the time steps in consideration during the learning process of a ML algorithm, or the model will give incorrect results.

5.1.2 Use Cases Plurality

We can cite a lot of other use cases that rely on the temporal order of the data.

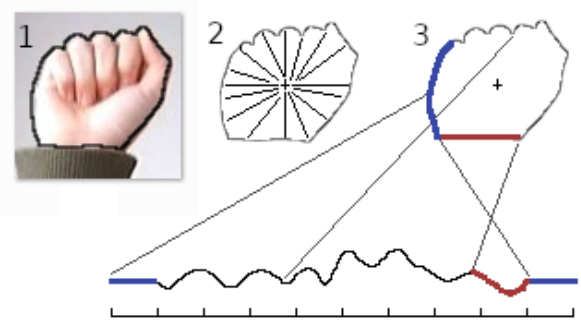


Figure 5-3: Extraction of the time series from a picture representing the letter A in sign language. Step 1 consists in delimiting the edge of the hand. Step 2 defines a base (usually the center of the edge). Step 3 measures the length between the base and each point of the of the edge and report them in a time series.

In Computer Vision, to recognize the sign language [106] and for example help people to better understand those with disabilities. An overview of the process to extract the time series from an image representing the 'A' letter in sign language is given Figure. 5-3.

It has been used for a while now in our most advanced smartphone for speech recognition [42] by making the actions from our voice, e.g. playing a specific song, doing a research on internet, or purchasing online products. It is also used to monitor our activities [21] by guessing the way we walk or sit.

In Geology [7] to classify hearth quakes in order to improve the long-term health monitoring systems used by civil engineer when they build vital engineering structures (bridges, towers, offshore platforms). This example can be spread to any data that rely on natural phenomena (e.g. windstorms, volcanic eruptions, flooding).

All these scenarios emphase the diversity of the data that rely on successive events. For these reasons, new ML algorithms that are more suited for these problematics have blossomed, and they try to minimize the loss of information in the time steps.

5.1.3 Emergence of the Solutions and the Underlying Problem

Among the libraries capable of tackling the time series classification problem, we can cite sktime [75], and tslearn [115] making the use of machine learning algorithms dedicated for TSC more accessible. Since our work were focused on classical machine learning algorithms,

we did not study the solutions based on Deep Learning methods such as TensorFlow [2]. We discuss this part in the last chapter: conclusion and perspectives.

Sktime seems the most promising library by including a lot of different approaches (Table. A.5). To name a few, there are: the interval-based [31] approaches, the distance-based [98] approaches, the shapelet-based [125] approaches, the dictionary-based [108] approaches, the kernel-based [30] approaches, and more recently[1] the signature-based approaches [88] (see 2.1.1.3 for more details).

Nevertheless, these new techniques also introduce a lot of hyper-parameters which have an impact on the performance of the trained models. Accurately choosing the algorithm and tuning the related hyper-parameters take time for two reasons. The first reason concerns the number of possible choices (see Table A.5), and the second one is related to the time of training which depends on the algorithm's complexity and the dataset's characteristics (e.g. number of instances, sequence length).

This brings the problem to be equivalent to the AutoML problem. However the search space of the algorithms changes, and the preprocessing phase seems useless since it is already as part of the algorithm itself.

5.1.4 Formal Definition

The Time Series Classification (TSC) problem consists in training a classifier on input variables that are an ordered set of real values. It can be formally defined as a trained model $\mathcal{M}$, that maps a time series X to a probability distribution (or a label prediction) $\mathbb{D}$ over the labels:

$$\mathcal{M} : X \rightarrow \mathbb{D} \tag{5.1}$$

Where:

- $X = [x_1, x_2, ..., x_t]$ is a time series of length t and $x_i \in \mathbb{R}$
- $\mathcal{M} = \mathrm{A}_\lambda(\mathcal{D})$ is a trained model, with A_λ a ML-TSC algorithm associated with its

[1] sktime v0.6+

hyper-parameters, and $\mathcal{D} = \{(X_1, Y_1), ..., (X_N, Y_N)\} : (X_i, Y_i) \in (X, \mathbb{N})$ is a dataset, i.e. a collection of a time series paired with labels

In this chapter we only focus on the Machine Learning algorithms oriented for the Time Series Classification. In the rest of the reading, we will use the term ML-TSC to designate these algorithms (Table. A.5).

5.1.5 Our Proposition

As previously said, the preprocessing phase is included in the ML-TSC algorithm. By consequence, it changes how the process of exploration versus exploitation should be performed.

Indeed, in a classic AutoML approach, such as it is done with Mary-Morstan (see section 3.4), or with TPOT (see Figure 2.2.3) for classification problems, the presence of some operators, e.g. MutationInsert (3.3.3.1), and CrossoverOnePoint (3.3.3.2), that changes the structure of the ML pipeline, i.e. the preprocessing phase, is not required anymore. In consequence, we only use the operators that are adapted for a pipeline of size 1 (see Tab. A.7).

Moreover, during some empirical experiments, we notice that running a pure random phase along the first generations were beneficial to accelerate the convergence. In this way, we use the distribution P in Algorithm 1, such that during a quarter of the optimization time the MutationReplaceNodeOnly (pure exploration) will be used, and then other operators prone to exploitation continue the optimization process.

Thanks to the modular framework that we have built, we can easily change the EA space in order to fit these expectations, and we prove their importance through the results of our experiments.

To summarize:

- We define a ML-TSC search space (Table A.5).
- We use adapted operators (Tab. A.7) for the CASH problem (no preprocessing).
- And we change the distribution P (Tab. A.7) over the time of the optimization.

In order to clearly understand what this instance of Mary-Morstan is intended for, we named it AutoTSC for Automatic Time Series Classification.

5.2 Experimental Setup

In this section we detail the whole experimental setup used to demonstrate the performance of the proposed solution AutoTSC, which is an instance of Mary-Morstan with a Time Series Classification search space additionally to a dedicated evolutionary algorithm configuration.

5.2.1 Baselines

Since there was no work that addresses the problem of AutoML for ML-TSC, there is no off-the-shelf baseline to compare our work to. Therefore, to validate its interest, we defined two baselines:

1. a Random Search, that randomly explores the space of ML-TSC algorithms described in Table A.5.

2. TPOT[2], a famous standard AutoML solution based on EAs. Since TPOT has been shown to be closely equivalent to the other known state-of-the art AutoML solutions [43], we excluded the other libraries to save computational resources.

5.2.2 Dataset Corpus

A UCR archive was proposed in 2015 and updated in 2018 [8]. It gathers most of the TSC datasets used in the literature so far (128 datasets). Since then, many ML-TSC articles base their experiments on this archive [30, 108, 128] and so did we. However, because testing an AutoML is time consuming, we designed a procedure to discard datasets for which the best solution is trivial given our search space and only consider "hard" datasets, but for which a model can be trained in less than 5 minutes. This procedure is as follows:

- Run a random search of 1 hour through our search space with a wall time of 5 minutes per evaluation.

- Discard the datasets for which we did not obtain any results or for which we observed more than 50% of timeout (too long to evaluate).

[2] v0.11.5 https://github.com/EpistasisLab/tpot/releases/tag/v0.11.5

- Discard the datasets with a standard deviation of the balanced accuracy (on the validation set) that is less than 0.03. For 40 datasets we even observed 0 variance. In that case we considered the task too easy to be solved by automated ML.

5.2.3 Protocol

Evolutionary Algorithms and Random Search are both stochastic and have to be run several times in order to measure their performance. In the experiments, each algorithm is run 30 times on each dataset on the training set with a wall time of 1 day. This time has been chosen to enable convergence even for large datasets. The algorithms are run on 24 datasets (see Tab. A.6) present in the UCR archive [8] and selected as the "hardest" by the aforementioned procedure.

The settings used for this experiment within AutoTSC are provided in Tab. A.7. TPOT uses a very similar configuration [90], except for the selection and the variation operators i.e. the mutation and crossover. Indeed, as explained in the previous section, we reworked these operators in order to be more adapted to the problem. Moreover, in TPOT, P cannot be modified and does not evolve over time. In our solution, we also adapted P to the specificities of TSC and made it time-dependent.

At the end of each run, the best candidate is evaluated with the balanced accuracy on the test set (provided by the UCR archive). Our dataset corpus being quite heterogeneous, the choice of this metric has been motivated by the fact that it is significant on a wide variety of classification tasks (balanced or unbalanced classes, binary or multi-class).

5.2.4 Computational Environment

We conduct the experiments on OVH Public Cloud using Slurm [127] on C2-120 virtual machines (VM) composed of 32 cores with 3.1Ghz and 120GB RAM. We set up cgroup and Pyxis+ENROOT [3] to keep fair computation environments for each run.

5.3 Results

In this section we describe the results of our experiments. Firstly, we analyze the distribution of the test scores obtained at the end of the wall time (Fig. 5-4). Then we compare the convergence of the validation scores (Fig. 5-5). Finally, we further discuss the results and consider secondary indicators measured during the experiment (Fig. 5-6 and Fig. 5-7).

5.3.1 Final Results on Test Set

The results on the test set are depicted on Fig. 5-4. For each dataset, a Friedman [96] test with a significance of 5 percent is performed. If the null hypothesis cannot be rejected, we write "EQUIVALENTS" to signify that all optimizers perform similarly. If it is not the case, we secondly conduct a pairwise post-hoc analysis by using the Wilcoxon signed-rank [112] test with the same significant percentage. If a given optimizer statistically beats the other two, we write its name on the figure. If there are two winners, both are written.

Among the 24 datasets (Fig. 5-4), AutoTSC statistically outperforms both Random Search and TPOT on 12 datasets while TPOT only outperforms AutoTSC and Random Search on 2 datasets. AutoTSC performs equivalently to Random Search but better than TPOT on 5 other datasets and the three optimizers were considered equivalents on the last 5 ones. This clearly demonstrates the interest of using a smarter optimization process rather than just a Random Search or a classical AutoML solution in particular when computational resources are limited.

We notice that TPOT was unable to run on 2 datasets: ShapesAll and FiftyWords, which are the biggest in terms of classes. That can be explained by the pretest mechanism included in TPOT. It consists in training every candidate on a small subsample of the data to avoid wasting time on degenerated pipelines. Yet, due to a technicality, it is incapable of doing so when the number of classes is greater than the subsample size.

By looking at the characteristics of the datasets, we can observe other interesting results. For example, the number of classes, i.e. binary versus multiclass. AutoTSC seems better at handling multiclass classification for it wins 10 of the 18 multiclass datasets. However, AutoTSC only wins 2 of the 6 datasets on the binary ones. The size of the dataset does

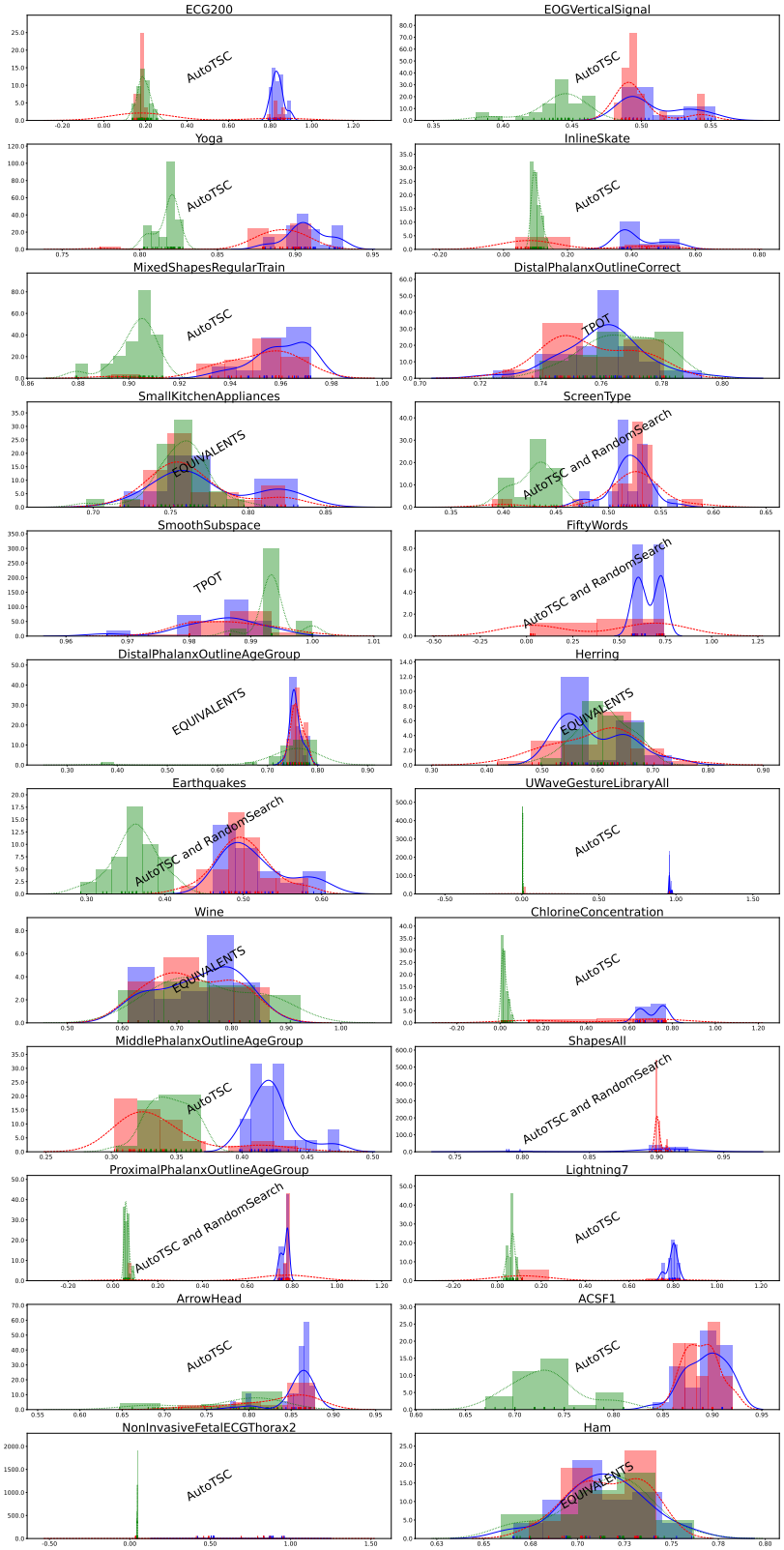


Figure 5-4: Test score distribution for each dataset after a day of run. AutoTSC is represented in solid blue lines. RandomSearch is represented in red dashed lines, and TPOT in green dotted lines. The horizontal axis represents the balanced accuracy value obtained from the test set and the vertical axis represents the density. A diagonal text highlight if an optimizer statistically outperforms.

not seem to matter though. On small datasets (i.e. less than 150 instances), AutoTSC wins 5 of the 8 datasets and on large datasets, AutoTSC wins 7 times out of 16. The same observation can be made on the length of the time series. AutoTSC wins 7 of the 13 small datasets (less than 500 time steps), and 5 of the 11 large datasets. What is interesting however is that AutoTSC dominates on the 4 largest datasets in terms of number of instances multiplied by the sequence length - EOGVerticalSignal, MixedShapesRegularTrain, UWaveGestureLibraryAll, NonInvasiveFetalECGThorax2. This and AutoTSC's superiority on multiclass problems show that the more difficult the classification tasks, the more one has interest to use AutoTSC.

5.3.2 Evolution of the Validation Score over Time

Fig. 5-5 presents the convergence curves of the balanced accuracy. The curves clearly show that AutoTSC converges faster than Random Search. Indeed, for half of the datasets (12 datasets: ECG200, MixedShapesRegularTrain, SmallKitchenAppliances, SmoothSubspace, DistalPhalanxOutlineAgeGroup, Earthquakes, Wine, MiddlePhalanxOutlineAgeGroup, ScreenType, Herring, Lightning7, Ham), the solid blue line corresponding to AutoTSC is significantly above the red dashed line that corresponds to Random Search. For the other half, random search converges as fast as AutoTSC, indicating that several areas of the search space can achieve a near-optimal performance.

5.3.3 Discussion

The most important remark concerns the fact that TPOT was surprisingly able to naively outperform AutoTSC and Random Search on two datasets (Fig. 5-4): SmoothSubSpace and DistalPhalanxOutlineCorrect. It is interesting that an AutoML configured for standard classification problems is capable to handle TSC problems so well. We see two main reasons for this to happen.

Firstly, in the introduction, we argued that using a classical ML algorithm by considering time steps as features is generally not a good idea. Indeed the main assumption under such models is the fact that features are independent. Even algorithms that do not make this

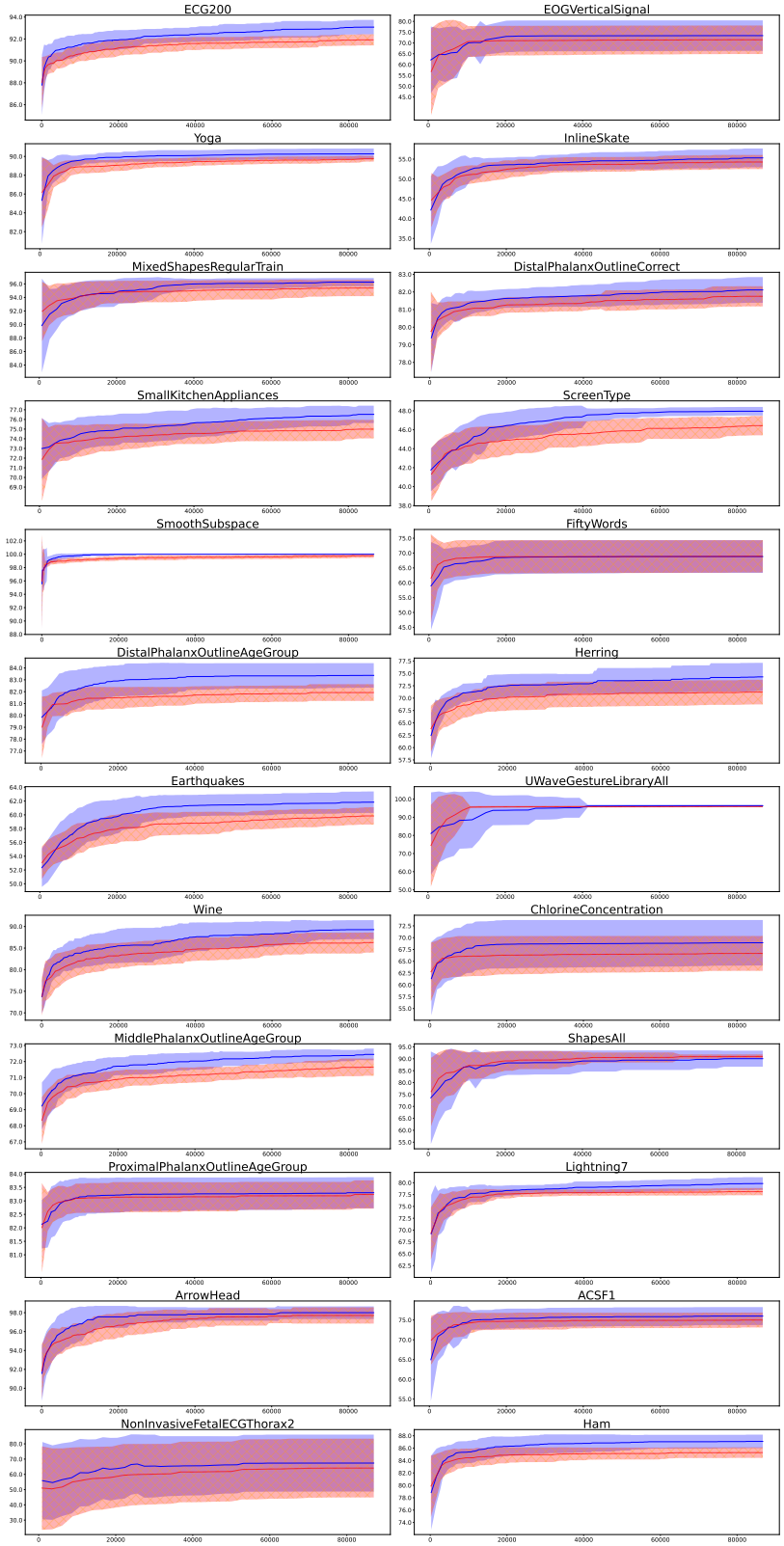


Figure 5-5: Convergence of the balanced accuracy per dataset for AutoTSC and Random Search. AutoTSC is represented in solid blue lines and RandomSearch is represented in red dashed lines. The horizontal axis represents the elapsed time in seconds and the vertical axis represents the average of the best scores obtained in validation. Each line is framed by the standard deviation, to better distinguished both, the Random Search has been hatched.

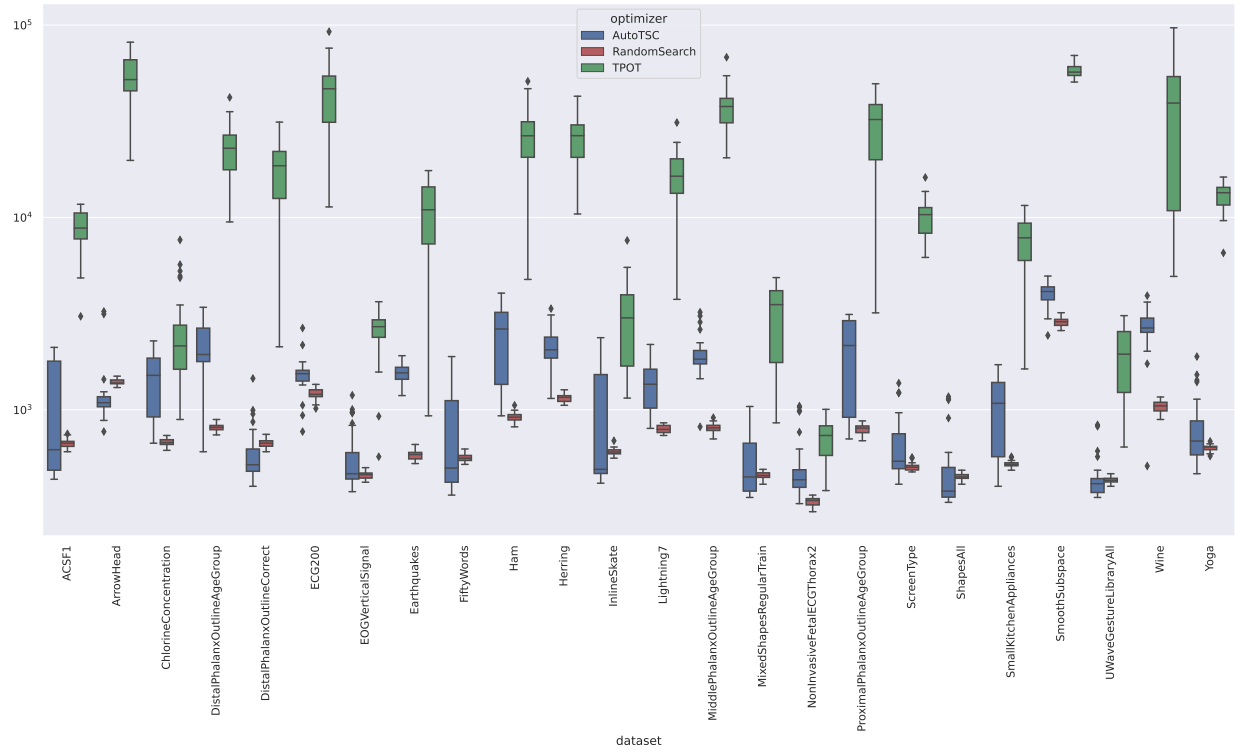


Figure 5-6: Log scale distribution of candidates evaluated for each optimizer. Per dataset are respectively plotted the optimizers AutoTSC, Random Search and TPOT. Each box correspond to the quartiles.

assumption (e.g. random forests) will have a hard time adapting to temporal variation in the data. Yet in some cases, classical ML algorithms could work well anyway and in that case, TPOT has a great advantage. Classical ML algorithms are much faster to train than ML-TSC algorithms so TPOT can try much more candidates than AutoTSC during an allocated time (see Fig. 5-6, TPOT evaluates ten times more candidates on average). It clearly seems to be the case for the dataset SmoothSubSpace. Moreover, its sequences only have a length of 15 and in half of the runs, TPOT's pipeline has an average size of 1 where the best trained candidate is a Gaussian Naive Bayes, which is well known for its independence assumption.

Secondly, most of the ML-TSC algorithms we included in the search space consist in a TS adapted preprocessing and a classical ML algorithm. In some cases however, more complex models could be required to fit the data, even after smart TS-oriented preprocessing. TPOT is able to assemble models and build complex pipelines (e.g. synthetic features). In this first version of AutoTSC, however, we did not include that possibility since the goal was to build a first proof of concept. On DistalPhalanxOutlineCorrect, we measure an average pipeline size of 5 for TPOT even though TPOT has for the second objective to minimize that size. This is not a hard proof because the TS preprocessing is meant to replace this model complexity but we can reasonably assume that for this dataset, the various preprocessing in AutoTSC did not match TPOT's ability to build a complex model.

Another interesting point is the frequency of the ML-TSC algorithms (Fig. 5-7) that differs from a dataset to another, showing that none of them is always the best. For example, the CompsableTimeSeriesForestClassifier works very well on the dataset SmoothSubspace, while BOSSEnsemble works very well on ArrowHead. Thus, the different ML-TSC approaches (e.g. dictionary-based, interval-based) demonstrate a true interest in the impact of the performance from a dataset to another. We also notice that some ML-TSC algorithms are more widely represented. It is the case of MUSE, ROCKETClassifier and WEASEL. MUSE is present on 17 datasets, ROCKETClassifer on 15 datasets and WEASEL on 12 datasets. On the other hand, some ML-TSC algorithms are completely absent: ShapeletTransformClassifier, ContractableBOSS, ProximityTree, ShapeDTW, DecisionTreeClassifier. Most of these methods are known to be computationally expensive and were most likely discarded with the wall time of five minutes. Anyhow, this diversity of models among the best can-

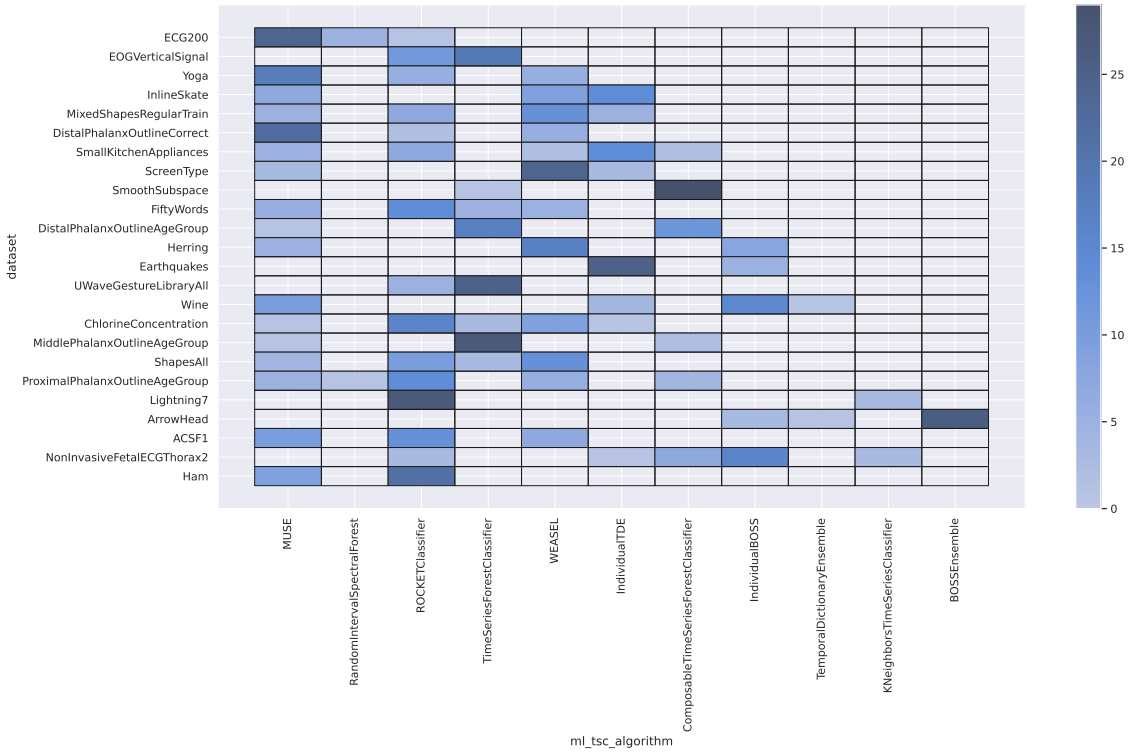


Figure 5-7: Heatmap of the best ML-TSC algorithms according to the validation scores present among the 30 runs per dataset at the end of the optimization.

didates selected by AutoTSC justifies the research for an automated process with a limited amount of time.

5.4 Conclusion and perspectives

AutoTSC, is capable of addressing the AutoML problem on timeseries classification. This solution consists in an evolutionary algorithm with variations specifically designed to handle this problem additionally to an associated search space. This work also demonstrated the interest of this solution on a standard set of TSC datasets from the UCR Archive. As such this solution constitutes a good and already usable first step toward automated ML for timeseries classification. Nonetheless, the careful study of the results conducted shows that this work opens a lot of perspective to go further.

First, contrary to what intuition would suggest, it may happen that standard ML algorithms perform well without any preprocessing on some datasets. Including them as well as standard preprocessing techniques in the search space could probably help handle a wider variety of cases. Moreover, TPOT's ability to stack algorithms could be very interesting even in the context of TSC. Adding this feature that can stack several TS preprocessing methods, followed by standard preprocessing methods and finally train several complementary models would certainly be beneficial in some complex situations. Obviously, in that case it also needs a regularization mechanism such as TPOT's minimization objective of the pipeline size.

Following this idea of considering a wider range of models, finding a way to adapt the wall time in order to be able to train costly models such as ShapeletTransformClassifier [49]. Yet this cannot be at the cost of slowing down the optimization process too much. A solution could be to try and minimize the training time.

Moreover we empirically defined a ML-TSC search space in Table. A.5, that certainly lacks expertise and could be improved by better tuning the hyper-parameter intervals. New algorithms could also be added, such as SignatureClassifer [88] that was recently added in sktime[3].

[3]https://github.com/alan-turing-institute/sktime/releases/tag/v0.7.0

Another set of perspectives is related to the very optimization process. We realized in this work that the amount of exploration needed (number of randomly generated candidates) depends on the dataset. We also realized that the optimization process benefited from exploration even after the first generations. This is why we introduced the function P that controls the distribution of the variation used at each generation. Note that P not only controls exploration but all the different types of evolutionary variations. Studying all this, possibly finding ways to make it adaptive to the nature of the dataset or the way the optimization goes would certainly be interesting.

Finally, our solution can by nature be made multi-objective. Among these objectives we want to add a measure that quantifies the interpretability of the models. A first AutoML that takes this kind of objective in consideration has been proposed [99], however it is not adapted for TSC problems. Indeed, given the specificities of the TS preprocessing, interpreting models is more difficult than for classical ML. Nevertheless, some methods have been recently developed to better interpret the ML-TSC models [44] and integrating them in our solution would be an interesting challenge.

Chapter 6

Conclusion

In this section we first summarize all the contributions present in the above chapters of this manuscript and then give perspectives for future works.

6.1 Summary of the Contributions

Our **first contribution** is a succinct **comparison of the state-of-the-art solutions and a way to classify them (section 2.3)**. At the beginning of the thesis works, there was no study on the difference between the approaches[1] of the AutoML tools. Thanks to our study, we emphasized the pros and cons of each of them which helped us to choose our development of our new tool Mary-Morstan based on the evolutionary algorithms, mainly motivated by the lack of study in this domain (algorithms and operators).

Our **second contribution** is the **study** of the **evolutionary algorithm components** and their **impact** when used by an **AutoML to solve classification problems (Chapter 3)**. We elaborated new operators and included known EA processes that were not studied in the other AutoML tools. Then we ran an optimizer (I-Race [76]) to find if a combination of our new components statistically outperformed the state-of-the-art performance. Unfortunately, it was unsuccessful, but the experiments remain interesting and might permit future researchers to use the same protocol on different problems (e.g. regression) or on different classification datasets which might be successful.

[1] Non-Adaptive, SMBO, EAs, MCTS

The **third contribution** is the **acceleration of the optimization** process for the **EA-based AutoML (Chapter 4) [92]**. During the experiments to study the impacts of EA components impacts, we noticed that optimizing large datasets was extremely slow and the same observation has been made by the authors of Auto-SKLearn, another AutoML tool. They came up with a solution called PoSH based on Successive Halving (SH) [56], a technique introduced in the Multi-armed Bandits problem which permits their tool to tackle large datasets. However, their technique called PoSH was not adapted for EA-based AutoML. In our contribution, we proposed a technique to make the SH work with any EA-based AutoML tool. Through our experiments we demonstrated a considerable gain on large datasets, with very few modifications in the optimization process.

The **fourth contribution** is a first proposition of solution capable of **automatically tackling the Time Series Classification (TSC) problem (Chapter 5) [93]**. With the emergence of new problems of classification under a format of time series [8] which requires specific preprocessing methods, we observed a proliferation of dedicated algorithms (2.1.1.3). These algorithms have the same underlying problems of classical classification methods, that is to select and configure an algorithm so that it gives the best performance. We noticed that these algorithms require no preprocessing, because it is included in the algorithm itself. Thanks to the development of our modular AutoML Mary-Morstan, we easily adapted it for this specific problem and showed an improvement of the performance when compared to a classical approach.

The **fifth contribution** is a **set of tools** developed along the thesis. These contributions are enumerated in the introduction (1.3): DSOP (1.3.1), a platform subject to two patents that eases the sharing of datasets while preserving the process used to transform the data, Intepretability Engine (1.3.2), a library that eases the interpretation of models deployed on OVHCloud Serving Engine, Slurm-PCi (1.3.3), a set of tools which makes the HPC scalable thanks to the use of OVHCloud Public Cloud.

6.2 Perspectives

To conclude this work, we propose a multitude of perspectives. Some of the perspectives have already been laid out along the chapters (4.4 and 5.4).

A **first** important **perspective** of amelioration concerns the **exploitation of the adaptable aspect developed within Mary-Morstan**. The tool has been designed to be tuned from different levels, and even if it was not a success on our experiments for classification (3.4) it worked for the time series classification problems with our instance called AutoTSC (5). From this fact, we remain convinced that tuning the EA space will impact the performance of other supervised problems, such as regression but also unsupervised problems. There are also new AutoML solutions dedicated to Deep Learning architectures, generally called AutoDL [27,58,59,70,102] which stands for Automatic Deep Learning, which face new challenges [74]. These solutions differ from the classic approach, because they try to find an architecture of neural network, commonly called NAS for Neural Architecture Search, instead of selecting an algorithm and tuning its hyper-parameters like done in CASHAP (2.1.3). Most of the AutoDL tools are based on EAs, mostly explained by their capacity of handling candidates of variable sized (e.g. adding or removing nodes in the architecture) without disturbing the optimization. Our tool being based on EAs, it makes a great applicant to study the impact of the EA operators on the NAS problem.

A **second** perspective is **the study and integration of new measures**. We are deeply convinced that the future of AutoML will rely on a diversity of metrics, and not just on those that evaluate the accuracy of the model (e.g. precision, recall, F-score) which "ideally" results in no false positives and/or no false negatives in classification. For example, it is great to have an autonomous car that takes decision thanks to deep learning models with almost no accident, but when an accident happens, it requires a certain amount of expertise to understand the thought process caused of the model. This complexity could be reduced by adding penalties or objectives within the evaluation/selection such that **AutoML** tools **build interpretable models [84, 85]**. This would ease the diagnoses for experts as well as non-experts (e.g. insurance company). Interpretability is not the only sort of measure to consider. Machine learning models tend to be unfair (or biased) due to the data made

by our intuitions and our environment. The problem has been highlighted [5, 105], notably with COMPAS, a software based on a ML models that helps judges predict if a criminal will reoffend once released. According to the models, there is more chance than white individuals will not reoffend when compared to black individuals. However, in practice, the reoffend rate between the two groups appears to be similar. Thus, the models are certainly biased by the data they have been feed with. It might therefore be important to integrate some mechanisms that **maximize fairness** in order to have less bias in the models returned by AutoML tools. Other interesting facets are not detailed here, but might also be included by the final users (maximize the **robustness** of a model, minimize the **prediction latency [72]**). **Integrating all these measures might make the optimization slower and more difficult**. Indeed, if we consider an interpretable model as a model with the fewer features possible, it will certainly reduce its accuracy due to the lack of information to represent a label. In other words, accurate models tend to be less interpretable and vice versa. The same reasoning can be made with other objectives. **When the objectives contradict each other, more evaluations are required to explore the space of each objective** (and their combination for a trade-off). As demonstrated on other problems (e.g. CEC09) solved thanks to EAs, the **performance of a multi-objective optimization depends upon the selection of the operators** such as the crossovers and the mutations [15, 80]. Our tool Mary-Morstan has been designed to be easily tuned and studied on such parameters. Therefore, studying different EA parameters of an AutoML with a combination of different objectives to observe if it has an impact on the performance of the optimization (hypervolume of the Pareto front, and diversity of the solutions) would constitute a great experiment with Mary-Morstan. A first study has been initiated by Pfisterer [99] on a multitude of objectives (intepretability and fairness), but does not investigate the tuning of the optimization.

The **third** perspective of **amelioration** is related to our proposition **Mary-Morstan-SH**. We did accelerate the optimization process on large datasets that have more than a hundred thousand samples. However, we **did not exploit** our **technique** (4.1.3) with a budget that considers **the number of features**. Applying such a method on the number of features would certainly be beneficial to train any ML algorithm even for those that struggle with the curse of dimensionality. Indeed, all the algorithms would be capable of being

trained at the beginning, and even if some are not trained anymore (due to the increase of the features along the iterations), depending to the evolutionary algorithm used, they might be kept until the end of the optimization. Without such a technique, algorithms with high complexity on the number of variables would have been discarded from the beginning. Like the number of samples, this might accelerate the optimization process for large datasets in terms of features. Time series classification algorithms are essentially costly due to the number of features (number of steps), it might be a great first case of study. Apart from playing with the features, different aspects could be investigated:

- The SH parameters (4.1) that we empirically defined. Tuning these parameters might contribute to an even better acceleration on the optimization process.

- MM-SH showed great results on classification problems, but has not been tested on regression problems, nor on time series classification problems and or unsupervised problems.

- The technique might help to accelerate the optimization process of multi-objective problems which tend to make the optimization slower as mentioned in the second perspective.

- The introduced method reduces the number of candidates along the iteration, which logically reduces the memory footprint of the optimization. However we did not monitor the memory to prove it.

Finally, we propose a set of mixed **minor perspectives**:

- AutoTSC (5.3.3) needs few refurnishment (adding the synthetic features) to compete with the two time series classification datasets that gave better results with TPOT, an AutoML that have a search space of classical machine learning algorithms.

- Interpretability Engine (1.3.2) is a great first proof of concept to interpret the deployed model, but it still lacks of methods to explain the individual predictions. Including SHAP [77] or Anchors [117] would complete the tool.

- We noticed that new preprocessing methods have been developed [23]. Including them in the search space might improve the overall optimization process for certain datasets.

To conclude this thesis, machine learning is an exciting playground that helps humans solve complex problems. However, some challenges still persist in this domain. The fact that running an AutoML is costly, and the lack of integrated tools that make any model interpretable. We hope that this manuscript will be a step forward on both challenges.

Appendix A

Tables

Table A.1: Datasets used by the AutoML Benchmark [43], ordered by the number of instances

Dataset	# Inst.	# Attr.	# Class.	Majority class (%)
Australian	690	15	2	55.51
blood-transfusion	748	5	2	76.2
vehicle	846	19	4	25.77
credit-g	1000	21	2	70
cnae-9	1080	857	2	11.11
car	1728	7	4	70.02
mfeat-factors	2000	217	10	10
kc1	2109	22	2	84.54
segment	2310	20	7	14.29
jasmine	2984	145	2	50
kr-vs-kp	3196	37	2	52.22
sylvine	5124	21	2	50
phoneme	5404	6	2	70.65
christine	5418	1637	2	50
fabert	8237	801	7	23.39
dilbert	10000	2001	5	20.49
Robert	10000	7201	10	10.43
guillermo	20000	4297	2	59.99
riccardo	20000	4297	2	75
Amazon_employee_access	32769	10	2	94.21
nomao	34465	119	2	71.44
jungle_chess_2pcs_raw_endgame_complete	44819	7	3	53.64
bank-marketing	45211	17	2	88.48
adult	48842	15	2	76.07
KDDCup09_appetency	50000	231	2	98.22
Shuttle	58000	10	7	78.6
Volkert	58310	181	10	21.96
Helena	65196	28	100	6.14
connect-4	67557	43	3	x 65.83
Fashion-MNIST	70000	785	10	10
APSFailure	76000	171	2	98.19
Jannis	83733	55	4	46.01
numerai28.6	96320	22	2	50.52
higgs	98050	29	2	52.86
MiniBooNE	130064	51	2	71.94
Dionis	416188	61	355	59
Albert	425240	79	7	50
Airlines	539383	8	2	55.46
Covertype	581012	55	2	51.24

denote the cardinality.

Table A.2: Classifiers in ML search space of TPOT and Mary-Morstan

Algorithm	**Hyperparameters**
GaussianNB	$\emptyset$
BernoulliNB	alpha $\in \{0.001, 0.01, 0.1, 1.0, 10.0, 100.0\}$
MultinomialNB	alpha $\in \{0.001, 0.01, 0.1, 1.0, 10.0, 100.0\}$,
	fit_prior $\in$ {true, false}
DecisionTreeClassifier	criterion $\in$ {gini, entropy}
	max_depth $\in [1..11]$
	min_samples_split $\in [2..21]$
	min_samples_leaf $\in [1..21]$
ExtraTreesClassifier	criterion $\in$ {gini, entropy}
	n_estimators = 100
	max_features $\in \{\frac{1}{20}n\}_{n=1}^{20}$
	min_samples_split $\in [2..21]$
	min_samples_leaf $\in [1..21]$
	bootstrap $\in$ {true, false}
RandomForestClassifier	criterion $\in$ {gini, entropy}
	n_estimators = 100
	max_features $\in \{\frac{1}{20}n\}_{n=1}^{20}$
	min_samples_split $\in [2..21]$
	min_samples_leaf $\in [1..21]$
	bootstrap $\in$ {true, false}
GradientBoostingClassifier	max_depth $\in [1..11]$
	n_estimators = 100
	max_features $\in \{\frac{1}{20}n\}_{n=1}^{20}$
	subsample $\in \{\frac{1}{20}n\}_{n=1}^{20}$
	min_samples_split $(MLsearchspace) \in [2..21]$
	min_samples_leaf $\in [1..21]$
	learning_rate $\in \{0.001, 0.01, 0.1, 0.5, 1.\}$
KNeighborsClassifier	n_neighbors $\in [1..100]$
	weights $\in$ {uniform, distance}
	p $\in \{1, 2\}$
LinearSVC	penalty $\in$ {l1, l2}
	loss $\in$ {hinge, squared_hinge}
	dual $\in$ {true, false}
	tol $\in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$
	C $\in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, .5, 1, 5, 10, 15, 20, 25\}$
LogisticRegression	penalty $\in$ {l1, l2}
	dual $\in$ {true, false}
	solver = lbfgs
	C $\in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, .5, 1, 5, 10, 15, 20, 25\}$

Table A.3: Preprocessing methods in ML search space of TPOT and Mary-Morstan

Algorithm	**Hyperparameters**
Binarizer	threshold $\in \{\frac{1}{20}n\}_{n=0}^{20}$
FastICA	tol $\in \{\frac{1}{20}n\}_{n=0}^{20}$
Normalizer	tol $\in$ {l1, l2, max}
StandardScaler	$\emptyset$
MaxAbsScaler	$\emptyset$
MinMaxScaler	$\emptyset$
Nystroem	kernel $\in$ {rbf, cosine, laplacian, polynomial, poly, linear, sigmoid}
	gamma $\in \{\frac{1}{20}n\}_{n=1}^{20}$
	n_component $\in$ [1..11]
PCA	svd_solver = randomized
	iterated_power $\in$ [1..11]
PolynomialFeatures	degree = 2
	include_bias = False
	interaction_only = False
RBFSampler	gamma $\in \{\frac{1}{20}n\}_{n=0}^{20}$
RobustScaler	$\emptyset$
ZeroCount	$\emptyset$
SelectFwe	alpha $\in \{\frac{1}{20}n\}_{n=0}^{20}$
SelectPercentile	percentile $\in$ [1..100]
VarianceThreshold	threshold $\in \{10^{-4}, 5.10^{-4}, 10^{-3}, 5.10^{-3}, 10^{-2}, 5.10^{-2}, .1, .2\}$
RFE	step $\in \{\frac{1}{20}n\}_{n=1}^{6}$
	estimator = ExtraTreesClassifier
	n_estimators = 100
	criterion $\in$ {gini, entropy}
	max_features $\in \{\frac{1}{20}n\}_{n=0}^{20}$
SelectFromModel	threshold $\in \{\frac{1}{20}n\}_{n=1}^{6}$
	estimator = ExtraTreesClassifier
	n_estimators = 100
	criterion $\in$ {gini, entropy}
	max_features $\in \{\frac{1}{20}n\}_{n=0}^{20}$
CombineDFs	$\emptyset$
CombineTwoPreviousesDFs	$\emptyset$

Table A.4: Shared settings of TPOT and Mary-Morstan-SH

Parameter	**Value**
Population size $\mathcal{P}$	100
Generations G	25* or 100
Per-individual mutation rate	90%
Per-individual crossover rate	10%
TPOT Pareto selection	100 individuals according to NSGA-II
Mutation	Point, insert, shrink 1/3 chance of each
Crossover	OnePoint
Candidate evaluation	5-fold cross-validation
Maximum evaluation time per candidate	5 minutes
Number of jobs	1

* only for large datasets.

Table A.5: ML-TSC search space used for AutoTSC and Random Search

Algorithm	**Hyperparameters**
ComposableTimeSeriesForestClassifier *(interval-based)*	n_estimators ∈ {50, 100, 200} criterion ∈ {gini, entropy} max_depth ∈ [1..11] min_samples_split ∈ [2..21] min_samples_leaf ∈ [1..21] bootstrap ∈ {true, false} oob_score ∈ {true, false}
ShapeletTransformClassifier *(shapelet-based)*	time_contract_in_mins ∈ {1, 2} n_estimators ∈ {10, 50, 100, 250, 500}
ROCKETClassifier *(kernel-based)*	num_kernels ∈ [100..10000] ensemble ∈ {true, false} ensemble_size ∈ [2..25]
BOSSEnsemble *(dictionary-based)*	threshold ∈ $\{\frac{4}{5} + \frac{1}{50}n\}_{n=0}^{10}$ max_ensemble_size ∈ {100, 250, 500} min_window ∈ [5..15]
IndividualBOSS *(dictionary-based)*	window_size ∈ [5..50] word_length ∈ [4..14] norm ∈ {true, false} alphabet_size ∈ [2..4] save_words ∈ {true, false}
ContractableBOSS *(dictionary-based)*	n_parameter_samples ∈ {250} max_ensemble_size ∈ {25, 50, 100} min_window ∈ [5..15]
TemporalDictionaryEnsemble *(dictionary-based)*	n_parameter_samples ∈ {125, 250, 500} max_ensemble_size ∈ {25, 50, 100} max_win_len_prop ∈ {1} min_window ∈ [5..15] randomly_selected_params ∈ {25, 50, 100} dim_threshold ∈ $\{\frac{3}{4} + \frac{1}{20}n\}_{n=0}^{5}$ max_dims ∈ {10, 20, 40}
IndividualTDE *(dictionary-based)*	window_size ∈ [5..50] word_length ∈ {4, 8, 16} norm ∈ {true, false} igb ∈ {true, false} alphabet_size ∈ {2, 3, 4} bigrams ∈ {true, false} dim_threshold ∈ $\{\frac{3}{4} + \frac{1}{20}n\}_{n=0}^{5}$ max_dims ∈ {10, 20, 40}
WEASEL *(dictionary-based)*	anova ∈ {true, false} bigrams ∈ {true, false} binning_strategy ∈ {equi-d., equi-w., inf-g.} window_inc ∈ [2..6] p_threshold ∈ $\{5.10^{-2}, 10^{-1}, 5.10^{-1}\}$
MUSE *(dictionary-based)*	anova ∈ {true, false} bigrams ∈ {true, false} window_inc ∈ [2..6] p_threshold ∈ $\{5.10^{-2}, 10^{-1}, 5.10^{-1}\}$ use_first_order_differences ∈ {true, false}
ProximityTree *(distance-based)*	n_stump_evaluations ∈ {3, 5, 10}
KNeighborsTimeSeriesClassifier *(distance-based)*	weights ∈ {uniform, distance} distance ∈ {dtw, ddtw, wdtw, wddtw, lcss, erp, msm} n_neighbors ∈ [1..4]
ShapeDTW *(distance-based)*	n_neighbors ∈ {1, 2, 3} subsequence_length ∈ {15, 30, 45} shape_descriptor_function ∈ {raw, derivative}
DecisionTreeClassifier *(interval-based)*	criterion ∈ {gini, entropy} max_depth ∈ [1..11] min_samples_split ∈ [2..21] min_samples_leaf ∈ [1..21]
TimeSeriesForestClassifier *(interval-based)*	min_interval ∈ [3..10] n_estimators ∈ {100, 200, 400}
RandomIntervalSpectralForest *(interval-based)*	n_estimators ∈ {100, 200, 400} min_interval ∈ {8, 16, 32} acf_lag ∈ {50, 100, 200} acf_min_values ∈ {2, 4, 8}

based on sktime v0.6.0

Table A.6: Time Series Classification datasets picked from the UCR Archive

Dataset	**#Inst.**	**#Seq.**	**#Class.**	**Majority class**
SmoothSubspace	150	15	3	33.33
ArrowHead	36	251	3	38.39
ECG200	100	96	2	66.50
Wine	57	234	2	51.35
Lightning7	70	319	7	26.57
DistalPhalanxOutlineAgeGroup	400	80	3	59.74
MiddlePhalanxOutlineAgeGroup	400	80	3	48.01
ProximalPhalanxOutlineAgeGroup	400	80	3	47.77
Herring	64	512	2	60.16
Ham	109	431	2	51.87
DistalPhalanxOutlineCorrect	600	80	2	61.53
ChlorineConcentration	467	166	3	53.56
FiftyWords	450	270	50	12.04
Yoga	300	426	2	53.64
ACSF1	100	1460	10	10.00
Earthquakes	322	512	2	79.83
InlineSkate	100	1882	7	18.00
ScreenType	375	720	3	33.33
SmallKitchenAppliances	375	720	3	33.33
ShapesAll	600	512	60	1.67
EOGVerticalSignal	362	1250	12	8.43
MixedShapesRegularTrain	500	1024	5	25.78
UWaveGestureLibraryAll	896	945	8	12.51
NonInvasiveFetalECGThorax2	1800	750	42	2.60

denote the cardinality. Large datasets are written in bold.

Table A.7: Settings of AutoTSC

Parameter	**Value**
Wall Time T	24 hours
Population size $\mathcal{P}$	5
Per-individual mutation rate	90%
Per-individual crossover rate	10%
Selection	SelectKBest
Mutations	Replace ReplaceNodeOnly Gaussian, UniformInteger OneRandom
Crossover	OnePointAverage
Distribution P	Rep.NodeOnly. 25% of T uniform* remaining T
Candidate evaluation	5-fold cross-validation
Max. evaluation time per candidate	5 minutes
Number of jobs	1

* distribution is made uniformly among all mutations

Appendix B

Figures

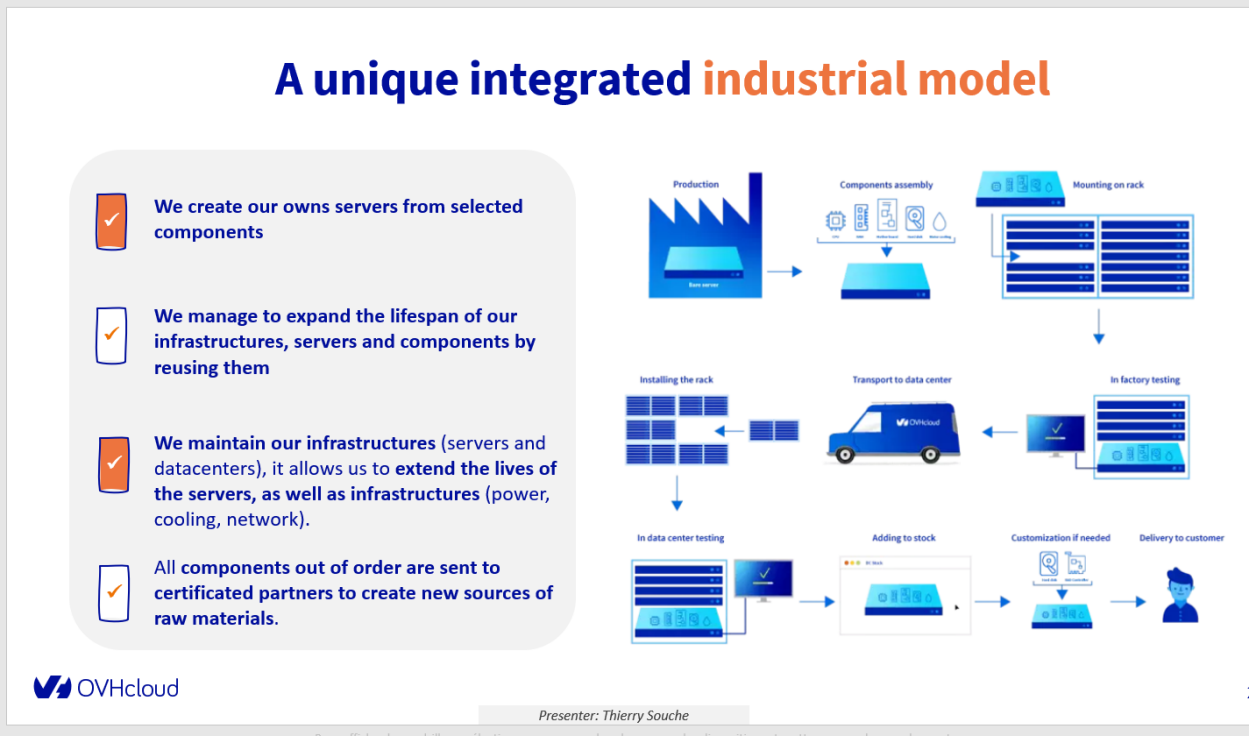
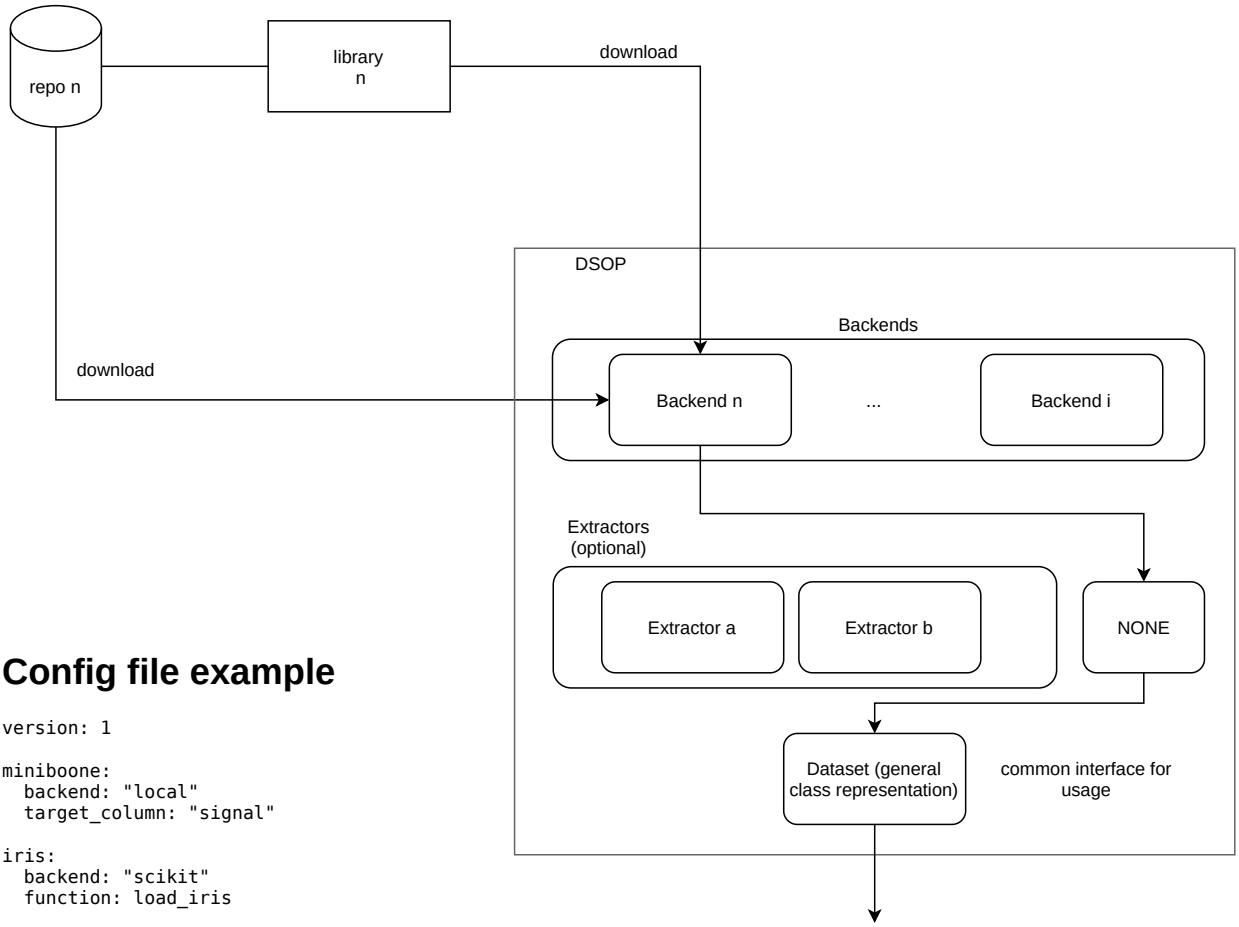


Figure B-1: OVHCloud industrial process.



Config file example

```
version: 1

miniboone:
  backend: "local"
  target_column: "signal"

iris:
  backend: "scikit"
  function: load_iris

covertype:
  backend: "scikit"
  function: fetch_covtype

spambase:
  backend: 'datahub'
  url: 'https://datahub.io/machine-learning/spambase/datapackage.json'

cloud:
  backend: 'openml'
  dataset_id: 890

adult:
  backend: 'uci'
  url_data: 'https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data'

O2:
  backend: 'local'
  extractor: 'autonlp_challenge'
  path: 'misc/dataset-autonlp/O2.data'

O1:
  backend: 'requests'
  extractor: 'autonlp_challenge'
  url: 'https://storage.bhs.cloud.ovh.net/v1/AUTH_1de31e43fad74b8cb021810be9eb69f1/autonlp-datasets/O1/O1.data'
```

Figure B-2: DSOP architecture with a configuration file example.

Classic Usage Example

```
# installation (multiple lines)
pip3 install lib1
pip3 install lib2
pip3 install requests
...

# configuration (multiple files + different instanciations)
lib1.instanciate('file1')
lib2.instanciate('file2')
...
# usage (interface differs, parameters differs)
dataset1 = lib1.download(dataset_id=1324)
dataset2 = lib2.get(dataset_name='minibOone')


print(dataset1.description.show_features())
print(dataset2._variables())
```

Improved Usage Example

```
# installation (one line)
pip3 install dsop

# configuration (one file)
acsf1:
  backend: lib1
  extractor: ...
minibOone:
  backend: lib2
  url: http://domain.tld/path/file.csv

# usage (common interface, backend agnostic)
dataset1 = dsop.get_dataset('config.yaml', 'acsf1')
dataset2 = dsop.get_dataset('config.yaml', 'minibOone')

print(dataset1.features)
print(dataset2.features)
```

Figure B-3: Classical usage of getting data versus DSOP usage.

```
interpretability-engine --token xxx --deployment-url https://xxxx.c1.gra.
    serving.ai.ovh.net/iris/ --samples-path iris.csv --features 0 1 2 3 --
    feature-names "sepal.length" "sepal.width" "petal.length" "petal.width"
     --label-names "setosa" "vergicolor" "virginica"
```

Listing B.1: Example of PDP for Iris dataset with Interpretability Engine usage through the CLI

```
# Import your local data to an Object Storage container
ovhai data upload myBucket@GRA my_dataset.zip

# Run a Jupyter notebook with PyTorch pre-installed and mount your data in
    a read-write folder/data with 2 GPUs
ovhai notebook run \
  --gpu 2
  --volume myBucket@GRA:/data:RW \
  ovhcom/ai-training-pytorch:1.6.0
```

Listing B.2: Example of AI training usage through the CLI

```
environment = "prod"
region = "BHS5" # default region to instance the node (can be specified
    per node)

network = {
  "private_subnet"      = "10.2.0.0/16" # VRack private subnet
  "public_network_name" = "Ext-Net"
  "vrack_network_name"  = "cluster-slurm-dhcp"
}

computes = {
  "slurm-compute3" : {
    flavor : "c2-7",
    image : "Ubuntu 20.04",
    partition : "debug",
    cpu : 2,
    memory : 6000,
    labels = ""
  },
  "slurm-compute4" : { # add a node
    flavor : "c2-120",
    image : "Ubuntu 20.04",
    partition : "prod",
    cpu : 30,
    memory : 120000,
    labels = ""
  },
}
```

Listing B.3: Terraform configuration file for Slurm Cluster Prod - add slurm-compute4

```
# Run Terraform (add / remove instances)
cd infrastructure && make apply ENV=prod

# Run Ansible (hosts file generated by Terraform)
make run VAULT_FILE=../vault.txt HOST_FILE=environments/prod/hosts
```

Listing B.4: Commands to add and configure Slurm node(s)

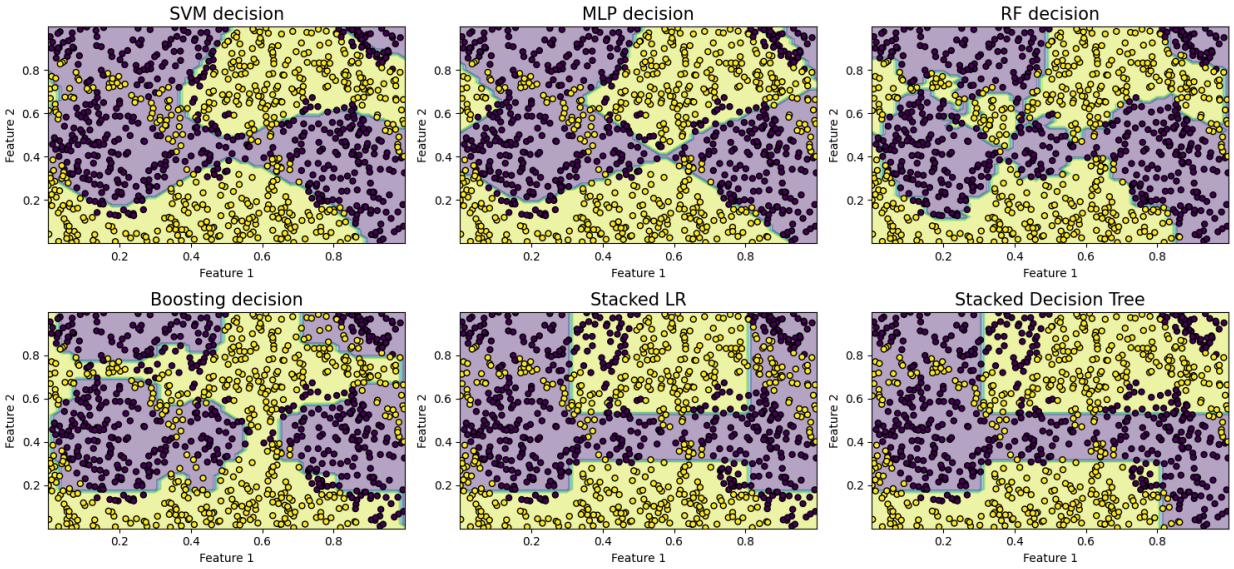


Figure B-4: A comparison of classifiers trained on P2 dataset. Highlight the difficulty of shaping a model on large (in terms of number of samples) and scattered dataset.

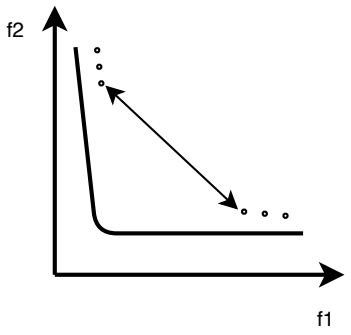


Figure B-5: Bi-objective pareto well-converged

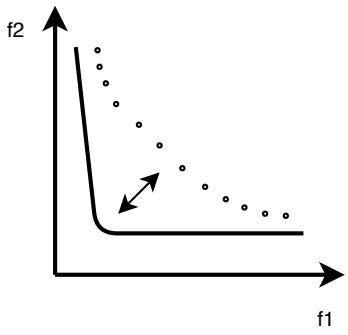


Figure B-6: Bi-objective pareto well-diversified

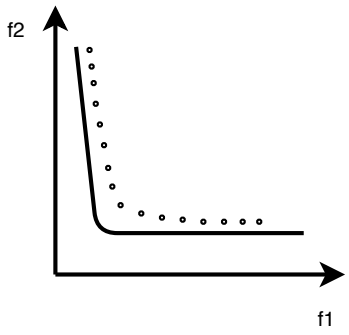


Figure B-7: Bi-objective pareto well-converged and well-diversified

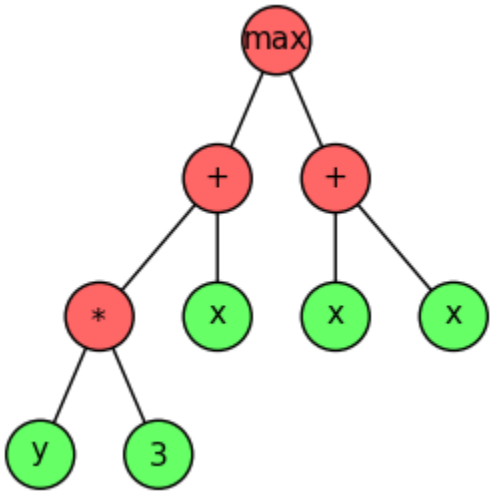
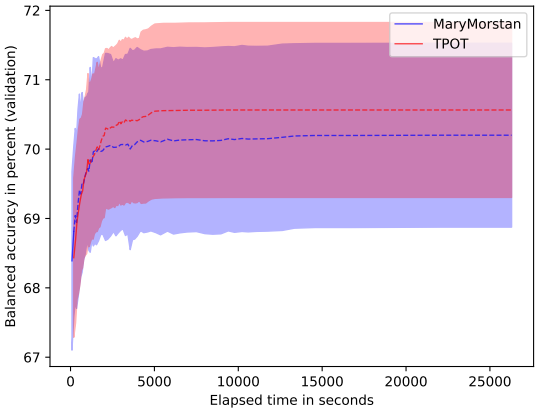


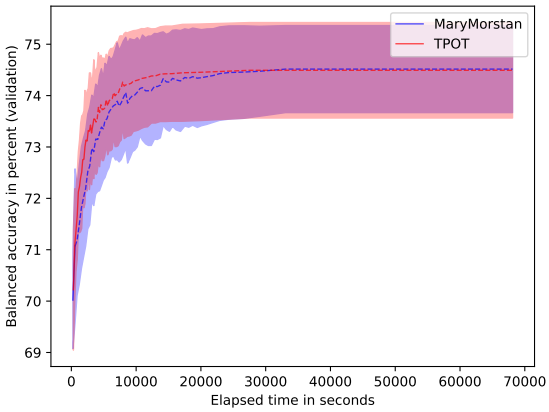
Figure B-8: Representation of max(x+3y, x+x) in Genetic Programming (GP).

```
NUM_TESTS = 10
MAX_EVAL_SECS = 2

def _pre_test(func):
    """Check if the wrapped function works with a pretest data set.
    Reruns the wrapped function until it generates a good pipeline, for a max of
    NUM_TESTS times.
    """
    ...

    @wraps(func)
    def check_pipeline(self, *args, **kwargs):
        bad_pipeline = True
        num_test = 0  # number of tests

        # a pool for workable pipeline
        while bad_pipeline and num_test < NUM_TESTS:
            ...
            try:
                ...
                pass_gen = False
                num_test_expr = 0
                # to ensure a pipeline can be generated or mutated.
                while not pass_gen and num_test_expr < int(NUM_TESTS/2):
                    try:
                        expr = func(self, *args, **kwargs)
                        pass_gen = True
                    except:
                        num_test_expr += 1
                        pass
                ...
                for expr_test in expr_tuple:
                    pipeline_code = generate_pipeline_code(
                        expr_to_tree(expr_test, self._pset),
                        self.operators
                    )
                    sklearn_pipeline = eval(pipeline_code, self.operators_context)
                    ...

                    bad_pipeline = False
            except BaseException as e:
                ...
                # Use the pbar output stream if it's active
                self._update_pbar(pbar_num=0, pbar_msg=message)
            finally:
                num_test += 1

        return expr

    return check_pipeline
```

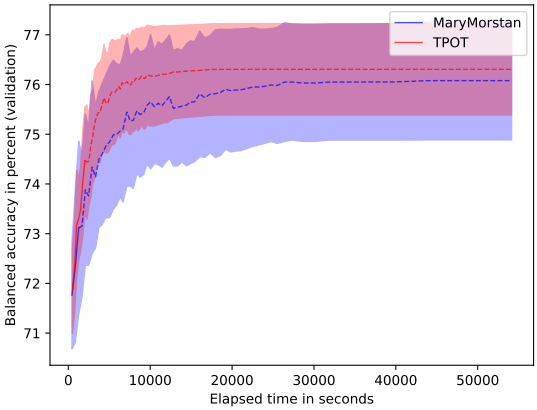
Listing B.5: Extract from the decorator used in TPOT v0.11.5 to check if a pipeline is valid



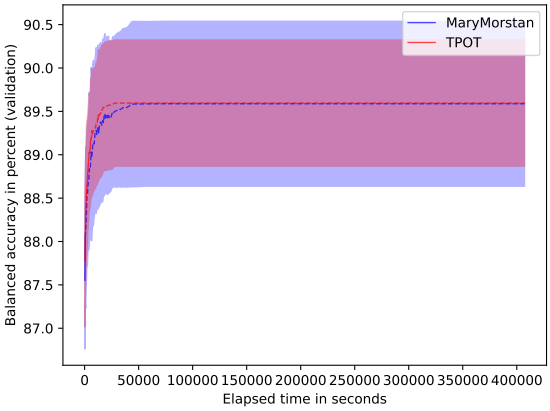
(a) blood-transfusion dataset.



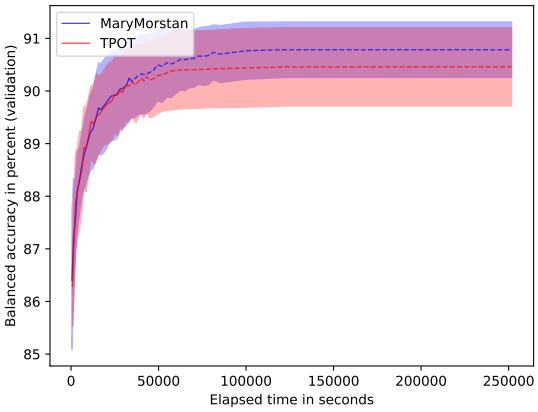
(b) credit-g dataset.



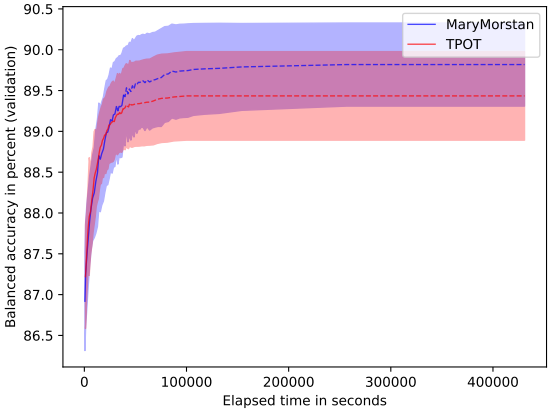
(c) kc1 dataset.



(d) Australian dataset.

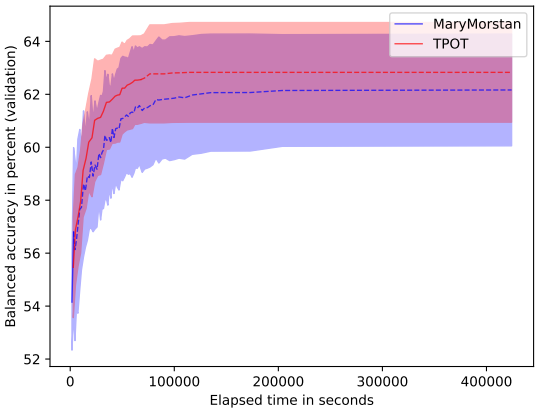


(e) vehicle dataset.

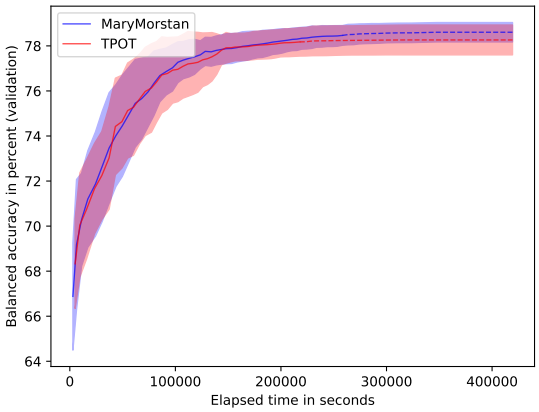


(f) phoneme dataset.

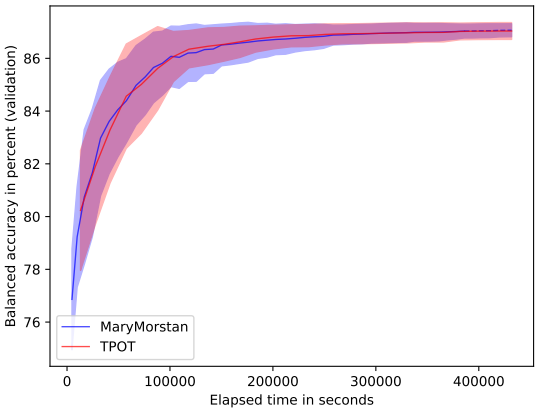
Figure B-9: Results of Mary-Morstan vs TPOT for each dataset. The line represents an average performance with a dotted when runs are missing (e.g. reach the last generation), and the frame represents the standard deviation.



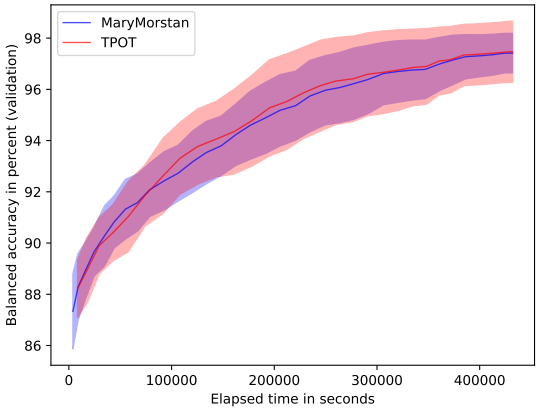
(g) jasmine dataset.



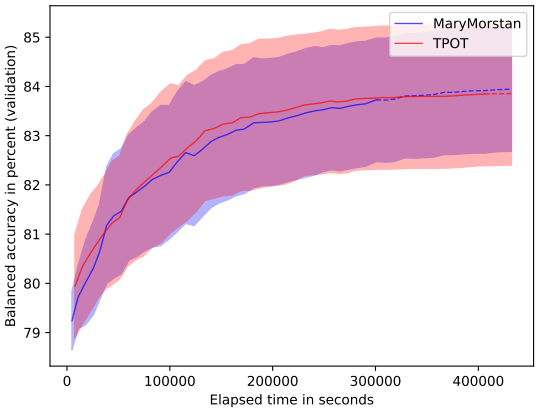
(h) Amazon_employee_access dataset.



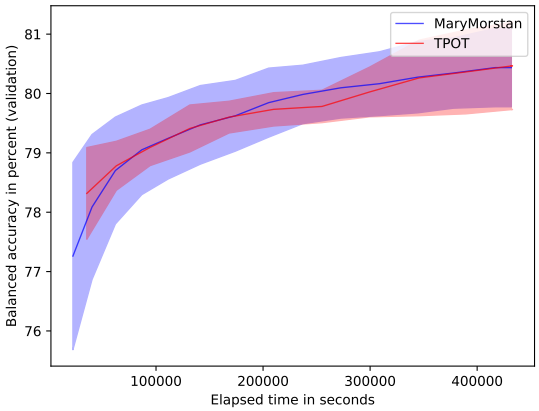
(i) bank-marketing dataset.



(j) jungle_chess_2pcs_raw_endgame_complete dataset.

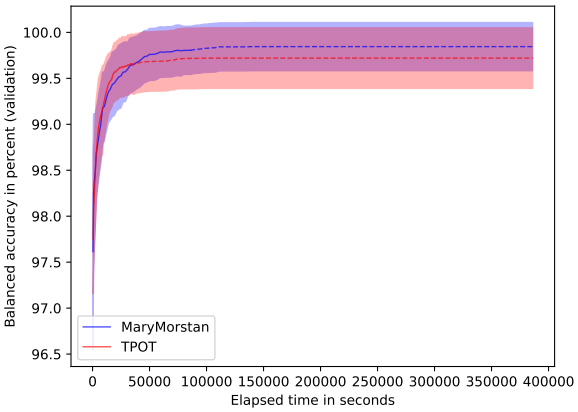


(k) adult dataset.

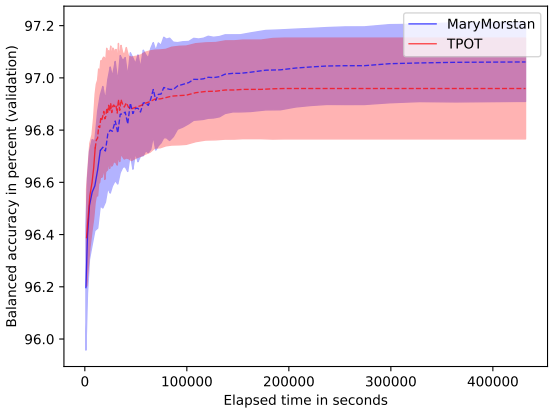


(l) connect-4 dataset.

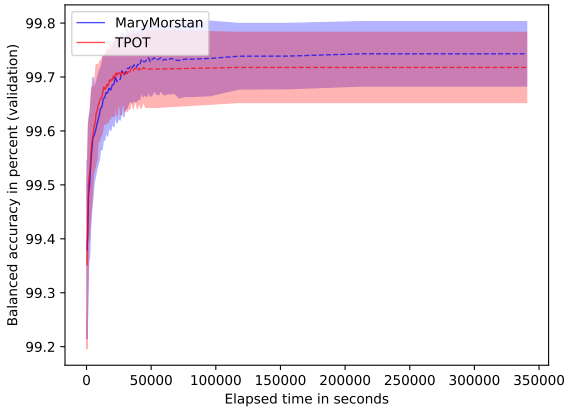
Figure B-9: Results of Mary-Morstan vs TPOT for each dataset. The line represents an average performance with a dotted when runs are missing (e.g. reach the last generation), and the frame represents the standard deviation.



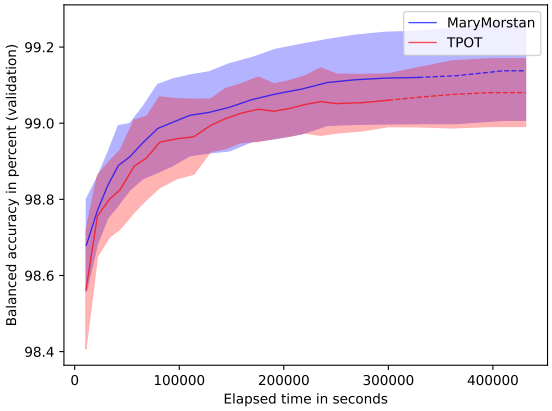
(m) car dataset.



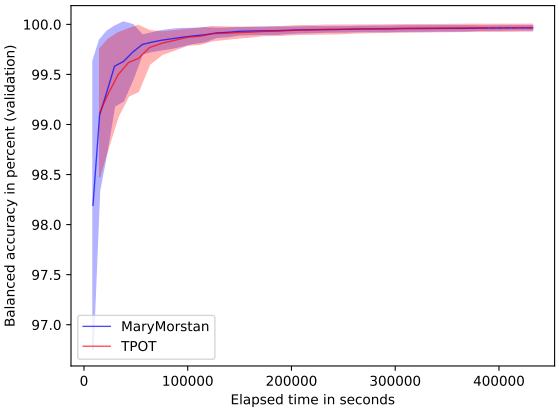
(n) segment dataset.



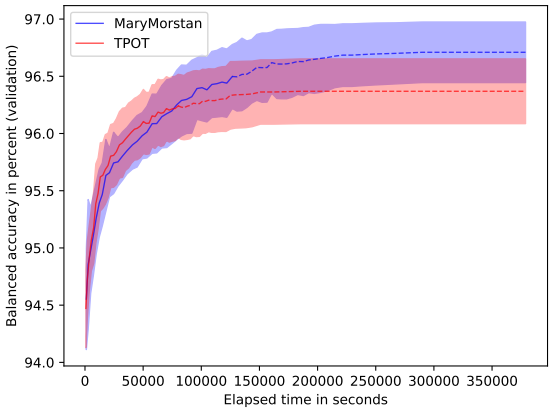
(o) kr-vs-kp dataset.



(p) mfeat-factors dataset.

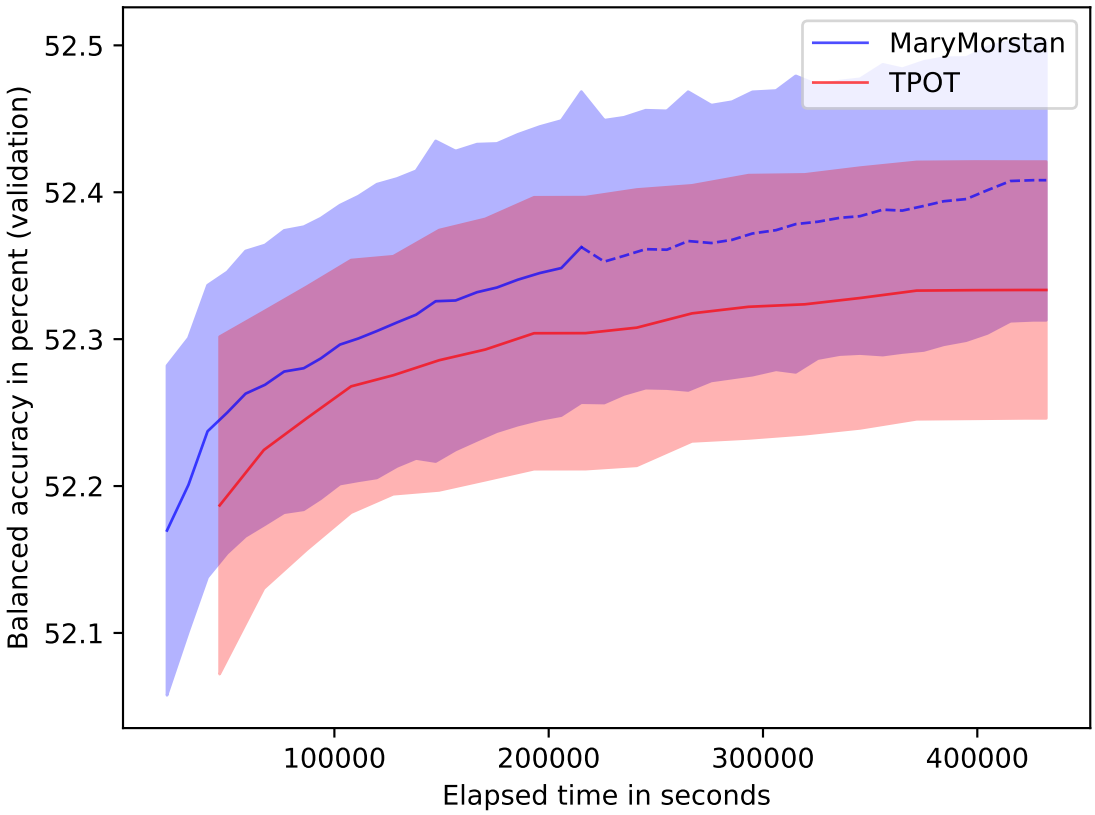


(q) Shuttle dataset.



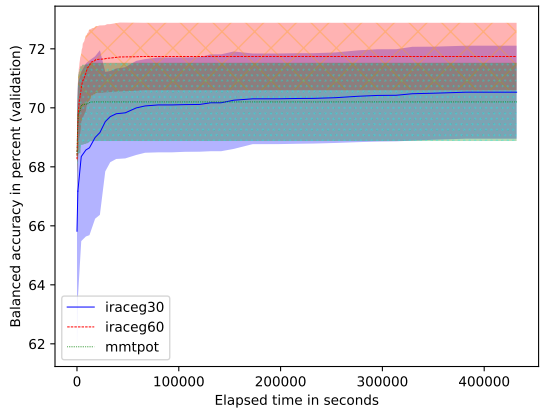
(r) Sylvine dataset.

Figure B-9: Results of Mary-Morstan vs TPOT for each dataset. The line represents an average performance with a dotted when runs are missing (e.g. reach the last generation), and the frame represents the standard deviation.

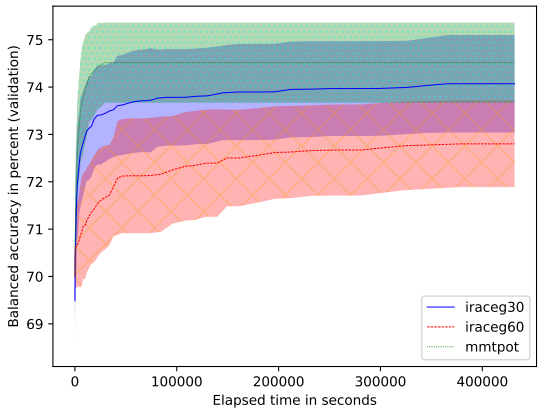


(s) numerai28.6 dataset.

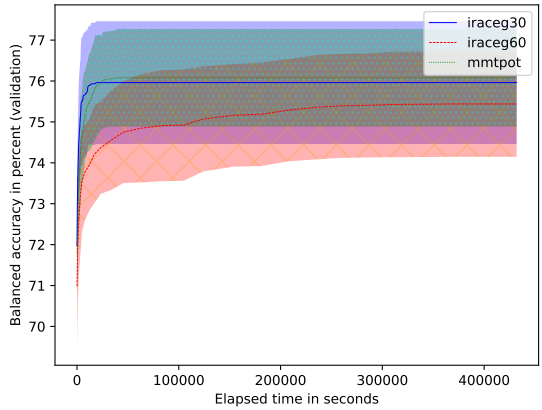
Figure B-9: Results of Mary-Morstan vs TPOT for each dataset. The line represents an average performance with a dotted when runs are missing (e.g. reach the last generation), and the frame represents the standard deviation.



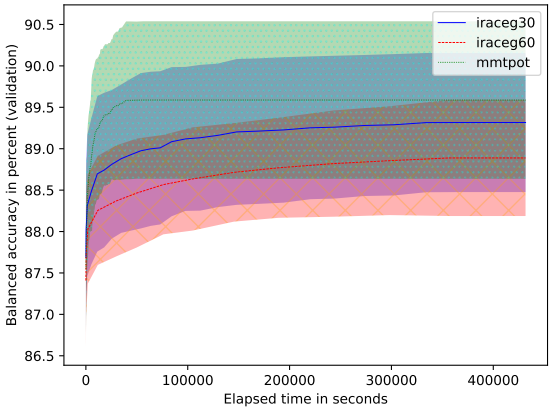
(a) blood-transfusion dataset.



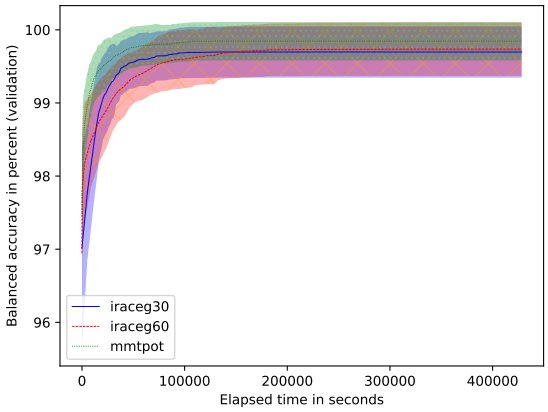
(b) credit-g dataset.



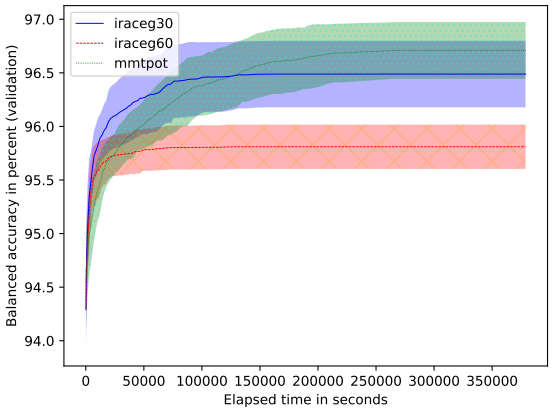
(c) kc1 dataset.



(d) Australian dataset.

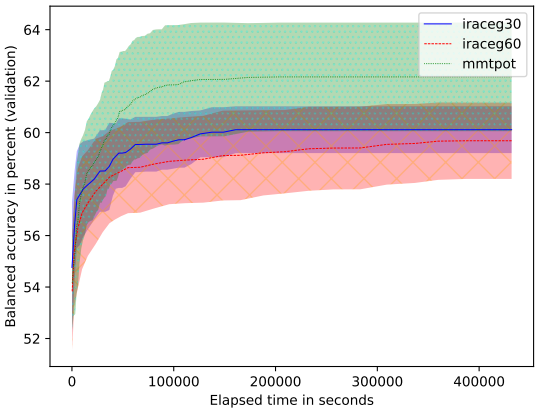


(e) car dataset.

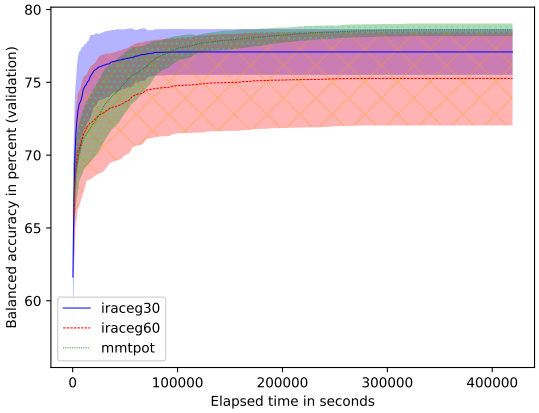


(f) Sylvine dataset.

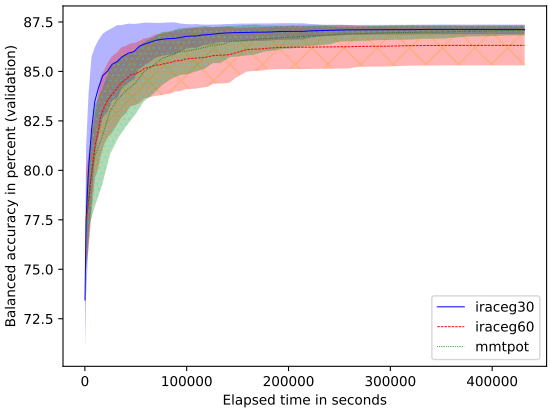
Figure B-10: Results of Mary-Morstan (TPOT) vs Mary-Morstan (I-Race with 30 generation) vs Mary-Morstan (I-Race with 60 generations) for each dataset. The line represents the average performance, and the frame represents the standard deviation. M-M(I-Race with generation 30) is represented in solid blue line, M-M (I-Race with generation 30) is represented in red dashed lines, and M-M (TPOT) is represented in green dotted lines.



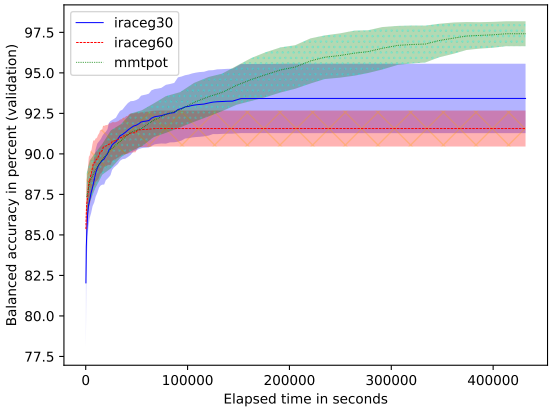
(g) jasmine dataset.



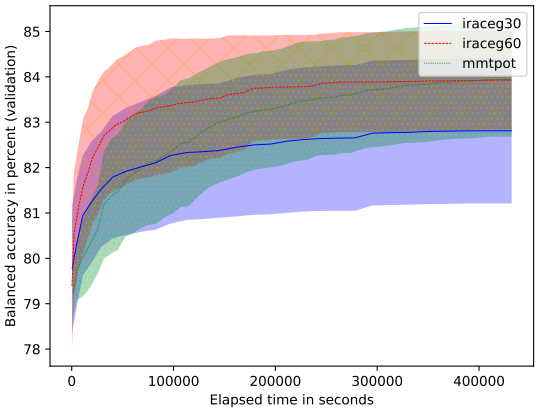
(h) Amazon_employee_access dataset.



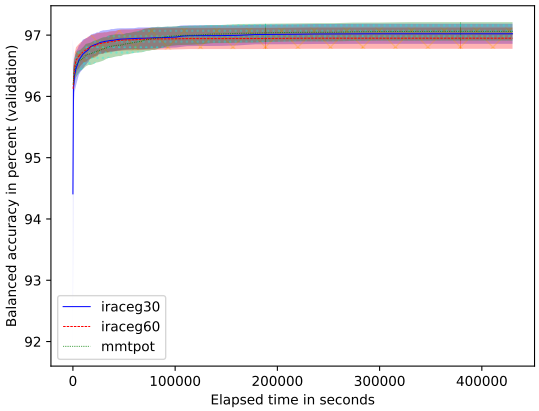
(i) bank-marketing dataset.



(j) jungle_chess_2pcs_raw_endgame_complete dataset.



(k) adult dataset.



(l) segment dataset.

Figure B-10: Results of Mary-Morstan (TPOT) vs Mary-Morstan (I-Race with 30 generation) vs Mary-Morstan (I-Race with 60 generations) for each dataset. The line represents the average performance, and the frame represents the standard deviation. M-M(I-Race with generation 30) is represented in solid blue line, M-M (I-Race with generation 30) is represented in red dashed lines, and M-M (TPOT) is represented in green dotted lines.

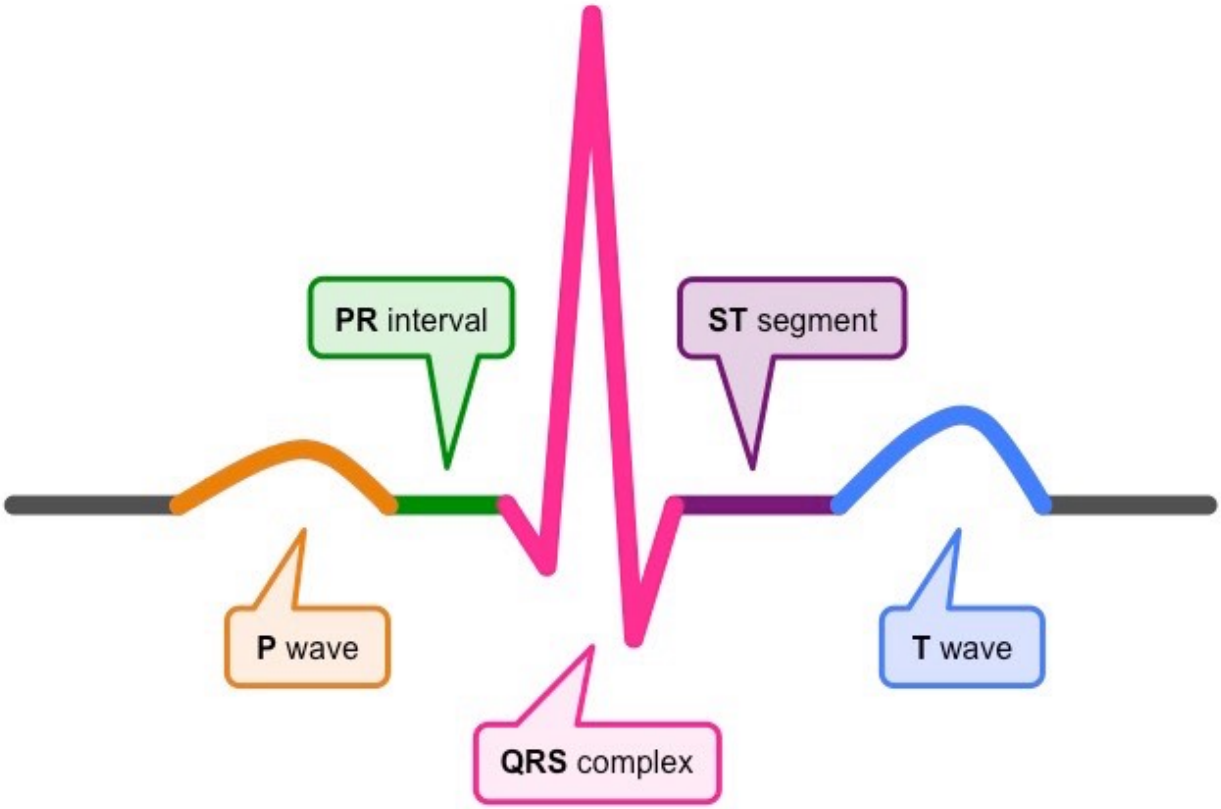


Figure B-11: Electrical Activity of the Heart

Algorithm 3 simple EA, see Algorithm 1 for completion

...
$\alpha \leftarrow \text{generate}(\mathcal{P})$
$\mathfrak{P}_p \leftarrow \{(\alpha_k, \text{evaluate}(\alpha_k, \mathcal{D}_t)), k = 1, ..., \mathcal{P}\}$
while ... **do**
  $\alpha \leftarrow \text{variation}(\alpha,\ P)$
  $\mathfrak{P}_o \leftarrow \{(\alpha_k, \text{evaluate}(\alpha_k, \mathcal{D}_t)), k = 1, ..., \mathcal{P}\}$
  $\mathfrak{P}_p \leftarrow \text{select}\ (\mathfrak{P}_o,\ \mathcal{P})$
end while
...

Algorithm 4 (μ, λ)-ES, see Algorithm 1 for completion

Input: ...; offspring size, λ; selection size; ...

...
while ... **do**
  ...
  $\mathfrak{P}_p \leftarrow \text{select}\ (\mathfrak{P}_o,\ \mu)$
end while
...

Algorithm 5 $(\mu + \lambda + \kappa)$-ES, see Algorithm 1 for completion

Input: ...; offspring size, λ; selection size, μ; random size κ; ...

...
while ... **do**
  ...
  $\mathfrak{P}_k \leftarrow \text{generate}_{\text{randomly}}(\kappa)$
  $\mathfrak{P}_p \leftarrow \text{select}\ (\mathfrak{P}_p \cup \mathfrak{P}_o \cup \mathfrak{P}_k,\ \mu)$
end while
...

Bibliography

[1] Wikipedia OVH, 2022.

[2] Martín Abadi and Ashish Agarwal. TensorFlow, Large-scale machine learning on heterogeneous systems, November 2015.

[3] Felix Abecassis and Louis Capps. SLURM: Seamless Integration With Unprivileged Containers, 2019.

[4] Eric Angel, Evripidis Bampis, and Laurent Gourvès. A Dynasearch Neighborhood for the Bicriteria Traveling Salesman Problem. In Xavier Gandibleux, Marc Sevaux, Kenneth Sörensen, and Vincent T'kindt, editors, *Metaheuristics for Multiobjective Optimisation*, pages 153–176, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[5] Julia Angwin and Jeff Larson. There's software used across the country to predict future criminals. And it's biased against blacks., 2016.

[6] Daniel W. Apley and Jingyu Zhu. Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models. *arXiv:1612.08468 [stat]*, August 2019. arXiv: 1612.08468.

[7] Monica Arul and Ahsan Kareem. Applications of shapelet transform to time series classification of earthquake, wind and wave data. *Engineering Structures*, 228:111564, 2021.

[8] Anthony Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn Keogh. The UEA multivariate time series classification archive, 2018. *arXiv:1811.00075 [cs, stat]*, October 2018. arXiv: 1811.00075.

[9] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr, and Luc Sarzyniec. Adding Virtualization Capabilities to the Grid'5000 Testbed. In Ivan I. Ivanov, Marten van Sinderen, Frank Leymann, and Tony Shan, editors, *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*, pages 3–20. Springer International Publishing, 2013.

[10] Wolfgang Banzhaf, Frank D. Francone, Robert E. Keller, and Peter Nordin. *Genetic Programming: An Introduction: on the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[11] Y. Bengio. Gradient-Based Optimization of Hyperparameters. *Neural Computation*, 12(8):1889–1900, August 2000.

[12] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for Hyper-parameter Optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 2546–2554, USA, 2011. Curran Associates Inc.

[13] James Bergstra and Yoshua Bengio. Random Search for Hyper-parameter Optimization. *J. Mach. Learn. Res.*, 13:281–305, February 2012.

[14] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies – A comprehensive introduction. *Natural Computing*, 1(1):3–52, March 2002.

[15] Leonardo C. T. Bezerra, Manuel López-Ibáñez, and Thomas Stützle. Automatic Design of Evolutionary Algorithms for Multi-Objective Combinatorial Optimization. In Thomas Bartz-Beielstein, Jürgen Branke, Bogdan Filipič, and Jim Smith, editors, *Parallel Problem Solving from Nature – PPSN XIII*, pages 508–517, Cham, 2014. Springer International Publishing.

[16] C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer New York, 2016.

[17] Cody Blakeney, Gentry Atkinson, Nathaniel Huish, Yan Yan, Vangelis Metris, and Ziliang Zong. Measure Twice, Cut Once: Quantifying Bias and Fairness in Deep Neural Networks. *arXiv:2110.04397 [cs]*, October 2021. arXiv: 2110.04397.

[18] 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, Pittsburgh, Pennsylvania, United States, 1992. ACM Press.

[19] Hamparsum Bozdogan. Model Selection and Akaike's Information Criterion (AIC): The General Theory and Its Analytical Extensions. *Psychometrika*, 52:345–370, 1987.

[20] Pavel Brazdil, Christophe Giraud-Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to Data Mining*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[21] Jason Brownlee. LSTMs for Human Activity Recognition Time Series Classification.

[22] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble Selection from Libraries of Models. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 18–, New York, NY, USA, 2004. ACM.

[23] Patricio Cerda and Gaël Varoquaux. Encoding high-cardinality string categorical variables. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020. arXiv: 1907.01860.

[24] Boyuan Chen, Harvey Wu, and Warren Mo. Autostacker: A Compositional Evolutionary Learning System. *GECCO 2018*, 2018.

[25] Alejandro Cholaquidis, Ricardo Fraimand, and Mariela Sued. Semi-supervised learning: When and why it works. *arXiv:1805.09180 [cs, stat]*, May 2018. arXiv: 1805.09180.

[26] Clarisse Dhaenens. *Metaheuristics for big data*. 2016. OCLC: 959031861.

[27] White Colin. Local Search is State of the Art for NAS Benchmarks. *ICML Workshop 2020*, 2020.

[28] Alex G. C. de Sá, Walter José G. S. Pinto, Luiz Otavio V. B. Oliveira, and Gisele L. Pappa. RECIPE: A Grammar-Based Framework for Automatically Evolving Classification Pipelines. In James McDermott, Mauro Castelli, Lukas Sekanina, Evert Haasdijk, and Pablo García-Sánchez, editors, *Genetic Programming*, volume 10196, pages 246–261. Springer International Publishing, Cham, 2017.

[29] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[30] Angus Dempster, François Petitjean, and Geoffrey I. Webb. ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels. *arXiv:1910.13051 [cs, stat]*, October 2019. arXiv: 1910.13051.

[31] Houtao Deng, G. Runger, E. Tuv, and V. Martyanov. A time series forest for classification and feature extraction. *Inf. Sci.*, 239:142–153, 2013.

[32] Pedro Domingos. A Few Useful Things to Know About Machine Learning. *Commun. ACM*, 55(10):78–87, October 2012.

[33] A Eiben and Jim Smith. *Introduction To Evolutionary Computing*, volume 45. 2003.

[34] Suilan Estevez-Velarde. Solving Heterogeneous AutoML Problems with AutoGOAL. *ICML Workshop 2020*, 2020.

[35] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. *arXiv:1807.01774 [cs, stat]*, July 2018. arXiv: 1807.01774.

[36] Hua Fang, Zhaoyang Zhang, Jin Wang, Mahmoud Daneshmand, Chonggang Wang, and Honggang Wang. A Survey of Big Data Research. *IEEE Network*, 29, 2015.

[37] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Practical Automated Machine Learning for the AutoML Challenge 2018. In *ICML 2018 AutoML Workshop*, July 2018.

[38] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and Robust Automated Machine Learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015.

[39] Evelyn Fix and J. L. Hodges. Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties. *International Statistical Review / Revue Internationale de Statistique*, 57(3):238, December 1989.

[40] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13:2171–2175, July 2012.

[41] Jerome Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29, 2000.

[42] John Cristian Borges Gamboa. Deep Learning for Time-Series Analysis. *arXiv:1701.01887 [cs]*, January 2017. arXiv: 1701.01887.

[43] Pieter Gijsbers. An Open Source AutoML Benchmark. *6th ICML Workshop on Automated Machine Learning (2019)*, 2019.

[44] Mael Guilleme. Agnostic local explanation for time series classification. *ICTAI 2019*.

[45] Laura Gustafson. *Bayesian Tuning and Bandits: An Extensible, Open Source Library for AutoML*. PhD thesis, MIT, June 2018.

[46] Isabelle Guyon. Analysis of the AutoML Challenge series 2015-2018. 2018.

[47] Isabelle Guyon, Kristin Bennett, Gavin Cawley, Hugo Jair Escalante, Sergio Escalera, Tin Kam Ho, Nuria Macia, Bisakha Ray, Mehreen Saeed, Alexander Statnikov, and

Evelyne Viegas. Design of the 2015 ChaLearn AutoML challenge. pages 1–8. IEEE, July 2015.

[48] Aaron Harris. *Why Training Machine Learning With Multitenant Data Produces Better Results*. 2018.

[49] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. Classification of Time Series by Shapelet Transformation. *Data Min. Knowl. Discov.*, 28(4):851–881, July 2014. Place: USA Publisher: Kluwer Academic Publishers.

[50] Tin Kam Ho. Random Decision Forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ICDAR '95, page 278, USA, 1995. IEEE Computer Society.

[51] F. Hutter. *Automated Configuration of Algorithmsfor Solving Hard Computational Problems*. PhD thesis, 2004.

[52] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In *Proceedings of the conference on Learning and Intelligent OptimizatioN (LION 5)*, pages 507–523, January 2011.

[53] Frank Hutter. *Automatic Machine Learning:Methods, Systems, Challenges*. Springer, 2018.

[54] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Parallel Algorithm Configuration. In Youssef Hamadi and Marc Schoenauer, editors, *Learning and Intelligent Optimization*, pages 55–70, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[55] H. Jain and K. Deb. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach. *IEEE Transactions on Evolutionary Computation*, 18(4):602–622, 2014.

[56] Kevin Jamieson and Ameet Talwalkar. Non-stochastic Best Arm Identification and Hyperparameter Optimization. *arXiv:1502.07943 [cs, stat]*, February 2015. arXiv: 1502.07943.

[57] José M. Jerez, Ignacio Molina, Pedro J. García-Laencina, Emilio Alba, Nuria Ribelles, Miguel Martín, and Leonardo Franco. Missing data imputation using statistical and machine learning methods in a real breast cancer problem. *Artificial Intelligence in Medicine*, 50(2):105–115, 2010.

[58] Haifeng Jin, Qingquan Song, and Xia Hu. Efficient Neural Architecture Search with Network Morphism. *arXiv:1806.10282 [cs, stat]*, June 2018. arXiv: 1806.10282.

[59] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-Keras: An Efficient Neural Architecture Search System. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956. ACM, 2019.

[60] Yaochu Jin, editor. *Multi-Objective Machine Learning*, volume 16 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[61] Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient Global Optimization of Expensive Black-Box Functions. *J. of Global Optimization*, 13(4):455–492, December 1998.

[62] Pratik Kanani and Mamta Padole. ECG Heartbeat Arrhythmia Classification Using Time-Series Augmented Signals and Deep Learning Approach. *Procedia Computer Science*, 171:524–531, 2020.

[63] Zohar Karnin, Tomer Koren, and Oren Somekh. Almost Optimal Exploration in Multi-armed Bandits. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1238–III–1246, Atlanta, GA, USA, 2013. JMLR.org.

[64] Ching-Yun Ko, Zhaoyang Lyu, Tsui-Wei Weng, Luca Daniel, Ngai Wong, and Dahua Lin. POPQORN: Quantifying Robustness of Recurrent Neural Networks. *arXiv:1905.07387 [cs, stat]*, May 2019. arXiv: 1905.07387.

[65] Ron Kohavi and others. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

[66] Brent Komer, James Bergstra, and Chris Eliasmith. Hyperopt-Sklearn: Automatic Hyperparameter Configuration for Scikit-Learn. In Stéfan van der Walt and James Bergstra, editors, *Proceedings of the 13th Python in Science Conference*, pages 33 – 39, 2014.

[67] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 521:436–44, May 2015.

[68] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient BackProp. In Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. Series Title: Lecture Notes in Computer Science.

[69] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *arXiv:1603.06560 [cs, stat]*, March 2016. arXiv: 1603.06560.

[70] Jason Liang, Elliot Meyerson, Babak Hodjat, Dan Fink, Karl Mutch, and Risto Miikkulainen. Evolutionary Neural AutoML for Deep Learning. *arXiv:1902.06827 [cs]*, February 2019. arXiv: 1902.06827.

[71] Anji Liu, Yitao Liang, Ji Liu, Guy Van den Broeck, and Jianshu Chen. On Effective Parallelization of Monte Carlo Tree Search. *arXiv:2006.08785 [cs, stat]*, October 2020. arXiv: 2006.08785.

[72] Sijia Liu, Parikshit Ram, Deepak Vijaykeerthy, Djallel Bouneffouf, Gregory Bramble, Horst Samulowitz, Dakuo Wang, Andrew Conn, and Alexander Gray. An ADMM Based Framework for AutoML Pipeline Configuration. *arXiv:1905.00424 [cs, stat]*, December 2019. arXiv: 1905.00424.

[73] Zhengying Liu and Isabelle Guyon. How far are we from true AutoML: reflection from winning solutions and results of AutoDL challenge. *ICML Workshop 2020*, 2020.

[74] Zhengying Liu, Adrien Pavao, Zhen Xu, Sergio Escalera, Fabio Ferreira, Isabelle Guyon, Sirui Hong, Frank Hutter, Rongrong Ji, Julio C. S. Jacques Junior, Ge Li,

Marius Lindauer, Zhipeng Luo, Meysam Madadi, Thomas Nierhoff, Kangning Niu, Chunguang Pan, Danny Stoll, Sebastien Treguer, Jin Wang, Peng Wang, Chenglin Wu, Youcheng Xiong, Arbe r Zela, and Yang Zhang. Winning solutions and post-challenge analyses of the ChaLearn AutoDL challenge 2019. *arXiv:2201.03801 [cs]*, January 2022. arXiv: 2201.03801.

[75] Markus Löning, Anthony Bagnall, Sajaysurya Ganesh, Viktor Kazakov, Jason Lines, and Franz J. Király. sktime: A Unified Interface for Machine Learning with Time Series. *arXiv:1909.07872 [cs, stat]*, September 2019. arXiv: 1909.07872.

[76] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[77] Scott M. Lundberg, Gabriel G. Erion, and Su-In Lee. Consistent Individualized Feature Attribution for Tree Ensembles. *arXiv:1802.03888 [cs, stat]*, February 2018. arXiv: 1802.03888.

[78] Scott M Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[79] Bernard Marr. *The 4th Industrial Revolution: How Mining Companies Are Using AI, Machine Learning And Robots*. 2018.

[80] Wali Mashwani, Abdel Salhi, Muhammad Asif Jan, Rashida Khanum, and Muhammad Sulaiman. IMPACT ANALYSIS OF CROSSOVERS IN MULTIOBJECTIVE EVOLUTIONARY ALGORITHM. *SCIENCE INTERNATIONAL*, Publications International Lahore, Pakistan:ISSN 1013–5316, December 2015.

[81] Ujjwal Maulik, Sanghamitra Bandyopadhyay, and Anirban Mukhopadhyay. *Multiobjective Genetic Algorithms for Clustering*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[82] Guillaume M.J-B. Parallel Monte-Carlo Tree Search. 2008.

[83] Felix Mohr, Marcel Wever, and Eyke Hüllermeier. ML-Plan: Automated machine learning via hierarchical planning. *Machine Learning*, 107(8):1495–1515, September 2018.

[84] Christoph Molnar. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. 2019.

[85] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. Quantifying Interpretability of Arbitrary Machine Learning Models Through Functional Decomposition. *arXiv:1904.03867 [cs, stat]*, April 2019. arXiv: 1904.03867.

[86] Christoph Molnar, Gunnar König, Julia Herbinger, Timo Freiesleben, Susanne Dandl, Christian A. Scholbeck, Giuseppe Casalicchio, Moritz Grosse-Wentrup, and Bernd Bischl. Pitfalls to Avoid when Interpreting Machine Learning Models. *arXiv:2007.04131 [cs, stat]*, July 2020. arXiv: 2007.04131.

[87] Julia Moosbauer, Julia Herbinger, Giuseppe Casalicchio, Marius Lindauer, and Bernd Bischl. Explaining Hyperparameter Optimization via Partial Dependence Plots. *arXiv:2111.04820 [cs, stat]*, November 2021. arXiv: 2111.04820.

[88] James Morrill, Adeline Fermanian, Patrick Kidger, and Terry Lyons. A Generalised Signature Method for Multivariate Time Series Feature Extraction. *arXiv:2006.00873 [cs, stat]*, February 2021. arXiv: 2006.00873.

[89] Olivier Nicol. *Data-driven evaluation of contextual bandit algorithms and applications to dynamic recommendation*. PhD Thesis, 2014.

[90] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. *Proceedings of GECCO 2016*, March 2016. arXiv: 1603.06212.

[91] Randal S. Olson, Ryan J. Urbanowicz, Peter C. Andrews, Nicole A. Lavender, La Creis Kidd, and Jason H. Moore. Automating biomedical data science through tree-based pipeline optimization. *arXiv:1601.07925 [cs]*, January 2016. arXiv: 1601.07925.

[92] L. Parmentier, O. Nicol, L. Jourdan, and M. Kessaci. TPOT-SH: A Faster Optimization Algorithm to Solve the AutoML Problem on Large Datasets. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 471–478, 2019.

[93] L. Parmentier, O. Nicol, L. Jourdan, and M. Kessaci. AutoTSC: Optimization Algorithm to Automatically Solve the Time Series Classification Problem. In *2021 IEEE 33st International Conference on Tools with Artificial Intelligence (ICTAI)*, 2021.

[94] Karl Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, November 1901.

[95] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[96] D. Pereira, Anabela Afonso, and Fátima Medeiros. Overview of Friedman's Test and Post-hoc Analysis. *Communications in Statistics - Simulation and Computation*, 44:2636–2653, 2015.

[97] Luis Perez and Jason Wang. The Effectiveness of Data Augmentation in Image Classification using Deep Learning. *arXiv:1712.04621 [cs]*, December 2017. arXiv: 1712.04621.

[98] François Petitjean, Alain Ketterlin, and Pierre Gancarski. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition*, 44:678–, 2011.

[99] Florian Pfisterer, Stefan Coors, Janek Thomas, and Bernd Bischl. Multi-Objective Automatic Machine Learning with AutoxgboostMC. *arXiv:1908.10796 [cs, stat]*, August 2019. arXiv: 1908.10796.

[100] Kedar Potdar, Taher Pardawala, and Chinmay Pai. A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers. *International Journal of Computer Applications*, 175:7–9, October 2017.

[101] Herilalaina Rakotoarison, Marc Schoenauer, and Michèle Sebag. Automated Machine Learning with Monte-Carlo Tree Search (Extended Version). *arXiv:1906.00170 [cs, stat]*, June 2019. arXiv: 1906.00170.

[102] Esteban Real, Chen Liang, David R. So, and Quoc V. Le. AutoML-Zero: Evolving Machine Learning Algorithms From Scratch. *arXiv:2003.03384 [cs, stat]*, March 2020. arXiv: 2003.03384.

[103] Albert Reuther, Chansup Byun, William Arcand, David Bestor, Bill Bergeron, Matthew Hubbell, Michael Jones, Peter Michaleas, Andrew Prout, Antonio Rosa, and Jeremy Kepner. Scalable System Scheduling for HPC and Big Data. *Journal of Parallel and Distributed Computing*, 111:76–92, January 2018. arXiv: 1705.03102.

[104] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *arXiv:1602.04938 [cs, stat]*, February 2016. arXiv: 1602.04938.

[105] Chris Russell, Matt J Kusner, Joshua Loftus, and Ricardo Silva. When Worlds Collide: Integrating Different Counterfactual Assumptions in Fairness. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6414–6423. Curran Associates, Inc., 2017.

[106] Sujay S Kumar, Tenzin Wangyal, Varun Saboo, and Ramamoorthy Srinath. Time Series Neural Networks for Real Time Sign Language Translation. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 243–248, 2018.

[107] 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.

[108] Patrick Schäfer. The BOSS is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29, 2015.

[109] Robert E. Schapire and Yoram Singer. Improved Boosting Algorithms Using Confidence-Rated Predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, COLT' 98, pages 80–91, New York, NY, USA, 1998. Association for Computing Machinery. event-place: Madison, Wisconsin, USA.

[110] Klaus Schwab. *The Fourth Industrial Revolution*. 2017. OCLC: 1004975093.

[111] Valerie Sessions and Marco Valtorta. The Effects of Data Quality on Machine Learning Algorithms. 2006.

[112] David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman and Hall/CRC, 5 edition, June 2020.

[113] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[114] T. Swearingen, W. Drevo, B. Cyphers, A. Cuesta-Infante, A. Ross, and K. Veeramachaneni. ATM: A distributed, collaborative, scalable system for automated machine learning. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 151–162, December 2017.

[115] Romain Tavenard, Johann Faouzi, Gilles Vandewiele, Felix Divo, Guillaume Androz, Chester Holtz, Marie Payne, Roman Yurchak, Marc Rußwurm, Kushal Kolar, and Eli Woods. Tslearn, A Machine Learning Toolkit for Time Series Data. *Journal of Machine Learning Research*, 21(118):1–6, 2020.

[116] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In *Proc. of KDD-2013*, pages 847–855, 2013.

[117] Marco Tulio and Carlos Guestrin. Anchors: High-Precision Model-Agnostic Explanations. *AAAI*, 2018.

[118] Eric Valdez-Valenzuela, Angel Kuri-Morales, and Helena Gomez-Adorno. Measuring the Effect of Categorical Encoders in Machine Learning Tasks Using Synthetic Data. In Ildar Batyrshin, Alexander Gelbukh, and Grigori Sidorov, editors, *Advances in Computational Intelligence*, volume 13067, pages 92–107. Springer International Publishing, Cham, 2021. Series Title: Lecture Notes in Computer Science.

[119] Jason Van Hulse. *Data Quality in Data Mining and Machine Learning.* PhD Thesis, Florida Atlantic University, Boca Raton, FL, USA, 2007.

[120] Michel Veyleysen. *Machine learning of high-dimensional data: Local artificial neural networks and the curse of dimensionality.* PhD thesis, Université cahtolique de Louvain, 2000.

[121] Marcel Wever, Felix Mohr, and Eyke Hüllermeier. ML-Plan for Unlimited-Length Machine Learning Pipelines. 2018.

[122] David H. Wolpert. Stacked Generalization. *Neural Networks*, 5:241–259, 1992.

[123] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, January 2008.

[124] Sanjay Yadav and Sanyam Shukla. Analysis of k-Fold Cross-Validation over Hold-Out Validation on Colossal Datasets for Quality Classification. pages 78–83. IEEE, February 2016.

[125] Lexiang Ye and Eamonn Keogh. Time Series Shapelets: A New Primitive for Data Mining. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 947–956, New York, NY, USA, 2009. Association for Computing Machinery. event-place: Paris, France.

[126] Dani Yogatama and Gideon Mann. Efficient Transfer Learning Method for Automatic Hyperparameter Tuning. In Samuel Kaski and Jukka Corander, editors, *Proceedings*

of the Seventeenth International Conference on Artificial Intelligence and Statistics, volume 33 of *Proceedings of Machine Learning Research*, pages 1077–1085, Reykjavik, Iceland, April 2014. PMLR.

[127] Andy B. Yoo, Morris A. Jette, and Mark Grondona. SLURM: Simple Linux Utility for Resource Management. In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 44–60, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[128] Jiaping Zhao and Laurent Itti. shapeDTW: Shape Dynamic Time Warping. *Pattern Recognition*, 74:171–184, 2018.

[129] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization*, volume 3242, 2001.

[130] Indré Žliobaitė. Measuring discrimination in algorithmic decision making. *Data Mining and Knowledge Discovery*, 31(4):1060–1089, July 2017.