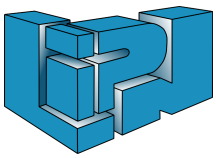
Unsupervised Learning of Data Representations and Cluster Structures : Applications to Large-scale Health Monitoring of Turbofan Aircraft Engines

Florent Forest



École Doctorale Galilée (ED 146)

Laboratoire d'Informatique de Paris Nord (LIPN — UMR CNRS 7030)

Équipe A3 : Apprentissage Artificiel & Applications

Thèse de doctorat de l'Université Sorbonne Paris Nord

présentée par

Florent Forest

en vue de l'obtention du titre de

Docteur en Informatique

Unsupervised Learning of Data Representations and Cluster Structures: Applications to Large-scale Health Monitoring of Turbofan Aircraft Engines

Apprentissage non supervisé de représentations de données et structures de partitionnement : applications à la surveillance à grande échelle de turbofans

Thèse soutenue le 22 mars 2021

Rapporteurs	M. Christophe Biernacki — Professeur, INRIA/Université de Lille 1
	M. Michel Verleysen — Professeur, UC Louvain
Examinateurs	M. Étienne Côme — CR, IFSTTAR/Université Gustave Eiffel
	M. Éric Gaussier — Professeur, Université Grenoble Alpes
	M. Christophe Cérin — Professeur, Université Sorbonne Paris Nord
Invité	M. Jérôme Lacaille — Expert émérite, Safran Aircraft Engines
Directeurs	M. Mustapha Lebbah — MCF HDR, Université Sorbonne Paris Nord
	Mme. Hanane Azzag — MCF HDR, Université Sorbonne Paris Nord

Florent Forest

Unsupervised Learning of Data Representations and Cluster Structures: Applications to Large-scale Health Monitoring of Turbofan Aircraft Engines

Apprentissage non supervisé de représentations de données et structures de partitionnement : applications à la surveillance à grande échelle de turbofans

Thèse de doctorat de l'Université Sorbonne Paris Nord, 22 mars 2021
Directeurs : M. Mustapha Lebbah, Mme. Hanane Azzag
Rapporteurs : M. Christophe Biernacki, M. Michel Verleysen
Examinateurs : M. Étienne Côme, M. Éric Gaussier, M. Christophe Cérin

Université Sorbonne Paris Nord
Équipe A3 : Apprentissage Artificiel & Applications
Laboratoire d'Informatique de Paris Nord (LIPN — UMR CNRS 7030)
École Doctorale Galilée (ED 146)
99 avenue Jean-Baptiste Clément
93430 Villetaneuse

Manuscript compiled on March 25, 2021.

Abstract

This thesis is interested in unsupervised statistical learning methods and their applications to health monitoring of aircraft engines at an industrial scale. Our first objective is to make health monitoring methodologies scale to massive data sets and allow engineering team to flexibly deploy various use cases. Besides the engineering aspects, we also try to address two fundamental theoretical challenges in unsupervised learning. First, the links between cluster structure and representation. And second, the very definition of structure, arising from the problem of model selection in clustering.

Modern aircraft engines generate growing amounts of data during manufacturing, tests and flights, that can be leveraged for health monitoring and predictive maintenance, in order to improve safety, availability and reduce costs. In this work, we use sensor measurements collected on board of civil short and mid-range aircraft. These data sets are temporal and highly multidimensional due to the large number of sensors and sampling frequencies. Hence, applications need to scale to the large volumes of data, driven by the growing number of daily operating engines.

Among all unsupervised learning approaches, clustering and self-organizing maps (SOM) provide useful insights on the distribution of complex and high-dimensional unlabeled data sets. To describe the internal state of an engine, expert indicators or features need to be extracted from raw data, before applying clustering algorithms. Our first contribution is to scale these methodologies using *Big Data* tools and distributed computing, in order to process entire fleets. We propose a generic and scalable pipeline enabling engineers to analyze flight data on a cluster. In addition, we present an new application to monitoring of vibration signatures.

Another option is to automatically extract relevant features with deep neural networks, known as deep learning, which had a great impact in many areas of machine learning. Recently, its ability to improve clustering has been investigated. A second contribution of this thesis is a Deep Embedded SOM, a neural network-based model performing joint representation learning with an autoencoder and self-organization of the cluster prototypes.

The third contribution of this thesis concerns model selection, which is perhaps the most difficult problem in clustering. We adopt the framework of cluster stability analysis, and propose a novel concept of within-cluster stability, leading to a criterion called Stadion (stability difference criterion) able to effectively select the number of clusters in a data set. We also apply Stadion to time series clustering validation by leveraging invariant transformations of the data.

Keywords: unsupervised learning; clustering; self-organizing maps; deep learning; time series; cluster stability analysis; big data processing; aircraft engines; health monitoring of industrial assets.

Résumé

Cette thèse porte sur des méthodes d'apprentissage statistique non supervisées et leurs applications à la surveillance de santé (*health monitoring*) des moteurs d'avion à une échelle industrielle. Notre premier objectif est de faire passer les méthodologies de health monitoring à l'échelle de jeux de données massifs et de permettre aux ingénieurs de déployer de manière agile divers cas d'utilisation. Outre les aspects d'ingénierie, nous aborderons également deux défis théoriques fondamentaux en apprentissage non supervisé. Premièrement, les liens entre structure de partitionnement et représentation. Et deuxièmement, la définition même de la structure, découlant du problème de sélection du modèle en partitionnement de données (*clustering*).

Aujourd'hui, les moteurs d'avion génèrent des quantités croissantes de données au cours de leur fabrication, des essais et des vols, pouvant être exploitées pour la surveillance et la maintenance prédictive, afin d'améliorer la sécurité, la disponibilité et de réduire les coûts. Dans ce travail, nous utiliserons des mesures de capteurs embarqués à bord d'avions civils court et moyen courrier. Ces jeux de données sont temporels et hautement multidimensionnels en raison du nombre de capteurs et leur fréquences d'échantillonnage. Par conséquent, les applications doivent s'adapter aux grands volumes de données qui ne cessent de croître avec la hausse du trafic aérien.

Parmi les approches d'apprentissage non supervisées, le clustering et les cartes auto-organisées (SOM) fournissent des informations utiles sur la distribution de jeux de données non étiquetés complexes et en grande dimension. Pour décrire

l'état interne d'un moteur, des indicateurs experts doivent être extraits des données brutes, avant d'appliquer des algorithmes de clustering. Notre première contribution est de faire passer à l'échelle ces méthodologies via les outils du *Big Data* et le calcul distribué, afin de traiter des flottes entières. Nous proposons une chaîne de traitement générique permettant aux ingénieurs d'analyser les données de vol stockées sur un cluster. En outre, nous présentons une application à la surveillance de signatures vibratoires.

Une autre option, appelée apprentissage profond, consiste à extraire automatiquement des caractéristiques pertinentes à l'aide de réseaux de neurones profonds. Cette approcha a bouleversé de nombreux domaines de l'apprentissage automatique ces dernières années. Récemment, sa capacité à améliorer le clustering a été étudiée. Une deuxième contribution de cette thèse est un modèle SOM profond (Deep Embedded SOM), basé sur des réseaux neurones combinant l'apprentissage de représentations via un auto-encodeur et l'auto-organisation des prototypes.

La troisième contribution de cette thèse concerne la sélection de modèle, l'un des problèmes les plus ardus en clustering. Nous adoptons le cadre de l'analyse de stabilité, et proposons un nouveau concept de stabilité intra-cluster, conduisant à un critère appelé Stadion (critère de différence de stabilité) capable de sélectionner efficacement le nombre de clusters dans un jeu de données. Nous appliquerons également Stadion à la validation du clustering de séries temporelles en tirant parti des transformations invariantes des données.

Mots-clés : apprentissage non supervisé ; partitionnement ; cartes auto-organisatrices ; apprentissage profond ; séries temporelles ; analyse de stabilité ; traitement de données massives ; moteurs d'avion ; surveillance de santé de systèmes industriels.

Remerciements

Mes premiers remerciements chaleureux vont à mes directeurs de thèse, Mustapha Lebbah et Hanane Azzag, ainsi qu'à Jérôme Lacaille, pour leur encadrement tout au long de ces trois années. Mustapha et Hanane, vos conseils, votre optimisme, votre disponibilité à toute heure, ont facilité la traversée de cette aventure. Jérôme, ton encadrement scientifique au sein de Safran Aircraft Engines était indispensable, et ton souhait de valoriser et faire rayonner nos travaux de recherche dans l'entreprise a constitué un moteur essentiel.

Christophe Biernacki et Michel Verleysen m'ont fait l'honneur d'accepter de rapporter ma thèse. Je ne saurais assez vous remercier pour vos retours plus qu'encourageants. Un grand merci également à mes examinateurs, Éric Gaussier, Étienne Côme et Christophe Cérin, pour les échanges enrichissants. Étienne, mes applications industrielles se sont largement basées sur tes travaux. Malheureusement, le jury a dû se tenir partiellement à distance cette année, mais j'espère sincèrement avoir l'occasion de nous rencontrer lors d'événements scientifiques dans l'avenir.

Je remercie l'entreprise Safran Aircraft Engines pour avoir rendu possible ce projet. Faire partie de Safran m'a permis de travailler sur des cas concrets et passionnants, et de me développer autant techniquement qu'humainement. J'ai eu la chance d'être intégré au Datalab, une équipe dynamique et pleine d'entrain, dans une ambiance toujours très conviviale. Ce fut également un réel plaisir de collaborer avec les ingénieurs des différents bureaux d'études et nos voisins du PHM.

Merci à tous mes collègues et amis, thésards ou non, au LIPN, en CIFRE à Safran ou ailleurs, pour les moments de convivialité, d'entraide et de soutien, sources de motivation, ainsi que de belles collaborations qui continueront à se développer. En particulier, merci à la team C4E (Clustering4Ever) qui verra toujours les choses en grand. Mes pensées vont évidemment à ma famille, sans qui je ne serais pas là, et qui m'a entre autres encouragé à emprunter le chemin ardu de la thèse; et bien sûr, à Marie-Cécile, présente à mes côtés jour après jour.

Je souhaiterais enfin mentionner les divers outils logiciels libres et/ou gratuits et autres sites web donnant un accès libre à la connaissance, qui sont d'une grande utilité pour tous les doctorants.

Cette thèse a été soutenue par l'ANRT via le contrat CIFRE n° 2017/1279.

Contents

Introduction

> *The purpose of science is to find meaningful simplicity in the midst of disorderly complexity.*
>
> **— Herbert Simon**
> (Models of my Life, 1991)

The introduction of this thesis manuscript starts by introducing the different stakeholders, the context and motivations behind this PhD project. The university laboratory and company are presented, along with their structure, entities and main products. Then, we briefly introduce turbofan aircraft engines and the principles of health monitoring, as well the main theoretical and technological challenges arising when scaling up the size and complexity of data sets. We will justify the adoption of the unsupervised learning setting, to extract insights from high-dimensional data sets. The second section provides an overview of the structure of the rest of the manuscript. Finally, the third section lists the contributions of this thesis, in terms of research publications (accepted and submitted papers), patents, and open-source software contributions.

Context and motivations

The work presented in this thesis is the result of a collaboration between the computer science lab LIPN (Laboratoire d'Informatique de Paris Nord)[1], at Université Sorbonne Paris Nord[2], and the company Safran Aircraft Engines[3] (SAFRAN group).

University Sorbonne Paris Nord (ex University Paris 13) is one of the thirteen universities that were created after the reorganization of the old Sorbonne after 1968. The department of Computer Science of the University Sorbonne Paris Nord (LIPN) has been created in 1985. It is affiliated both with the university and with the CNRS. Its members conduct research in several areas: combinatorics, combinatorial

[1] https://lipn.univ-paris13.fr/
[2] https://www.univ-paris13.fr/
[3] https://www.safran-aircraft-engines.com/

Fig. 0.1.: CFM56-7B (left) and LEAP (right) engines. [Source]

optimization, algorithmics, logics, software engineering, natural languages, and machine learning. The department is structured into the following five teams:

- Team A3: Machine learning.
- Team AOC: Combinatorial optimization.
- Team CALIN: Combinatorics.
- Team LoVe: Logic and verification.
- Team RCLN: Natural languages.

Safran Aircraft Engines (Safran A.E.), a company of the SAFRAN technology group, designs, develops, produces and markets, alone or in partnership, engines for civil and military aircraft, space launchers and satellites. Safran Aircraft Engines also provides airlines, the army, aircraft operators and leasing firms with a complete range of services for their aircraft engines. It covers the complete engine life cycle, from service entry to dismantling. To keep the pace of competition and technological progress, Safran Aircraft Engines must stay a competitive engine manufacturer by providing new innovative engines that are more energy-efficient, lightweight, producing less acoustic noise, emissions, etc.

In this PhD, we are interested in aircraft engines equipping short to mid-range civil aircraft. These products belong to the CFM company[4], a joint venture between the French Safran A.E. and the American General Electric (GE). Figure 0.1 shows a CFM56-7B engine (the most sold engine in the world), and the current generation LEAP. In this work, we will analyze data from LEAP engines only. Table 0.1 contains the fleet statistics for the LEAP engine, as of March 31, 2020. More than 1000 aircraft and 2400 engines are in operation, for a total of over 3.5 million cycles. A *cycle* is the technical term used instead of flight to designate a full engine start and stop.

[4] `https://www.cfmaeroengines.com/`

Tab. 0.1.: LEAP engines fleet statistics (as of March 31, 2020). [Source]

	Aircraft	Engines	Hours	Cycles
LEAP-1A (A320neo)	666	1 488	6 158 695	2 967 414
LEAP-1B (737 MAX)	389	923	1 688 692	607 340
Total LEAP	1 055	2 411	7 847 387	3 574 754
Total CFM56 (for reference)	14 614	33 560	1 072 962 020	578 290 661

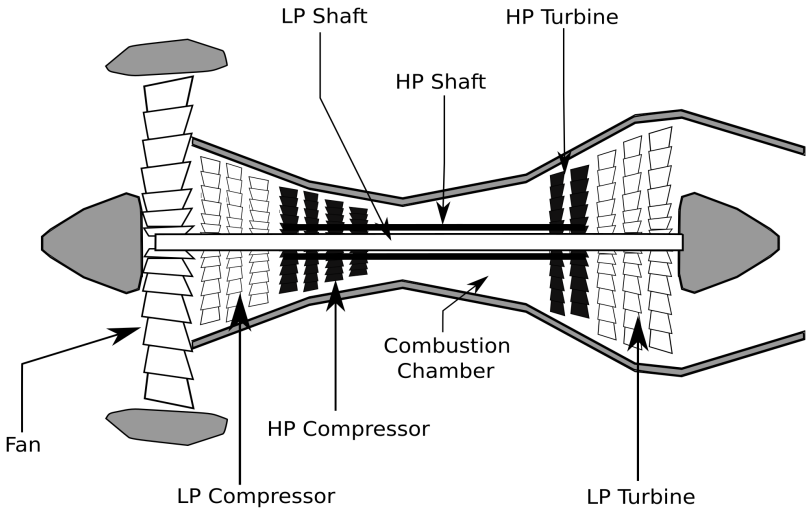


Fig. 0.2.: Simplified diagram of a turbofan engine with fan, low-pressure and high-pressure compressors and turbines attached to their respective shafts.

Before going further into this introduction, a brief introduction must be provided on how a turbofan engine produces thrust. First, incoming air is captured by the engine inlet. Part of this air passes through the fan and continues on into the core compressor and then the burner, where it is mixed with fuel and combustion occurs. The hot exhaust passes through the core and fan turbines and then out the nozzle. The rest of the incoming air passes through the fan and goes around the engine, called *bypassing*. The air that goes through the fan has a velocity that is slightly increased from free stream. So a turbofan gets some of its thrust from the core and some of its thrust from the fan. The ratio of the air that bypasses to the air that goes through the core is called the bypass ratio.

Although the basic principle of a jet engine is simple (Newton's third law), the entire system is of utmost complexity. An engine is made of several subsystems and a large number of moving and non-moving parts, as can be seen on the 3D cross-section of a CFM65-7B engine on Figure 0.1 (left). It is useful to have a simplified view of the main engine components (see Figure 0.2). A turbofan engine is composed of

two main shafts, the low pressure (LP) shaft and high pressure (HP) shaft (as well as a radial drive shaft (RDS), not represented here). The LP shaft is powered by the LP turbine and drives the fan (engine inlet) and LP compressor. The HP shaft is powered by the HP turbine (following the combustion chamber) and drives the HP compressor. Lastly, the RDS is linked to the HP shaft and provides power to the accessory gearbox.

Extremely high reliability is needed to reach the required level of safety for civil or military flights. In general, the probability of failure must be smaller than 10^{-9} per hour of flight, and aircraft are made to last typically up to 30 years or 100k cycles. This is achieved by the manufacturer through robust design and manufacturing, strict norms and certifications, extensive testing, and regular visual inspections and maintenance during their operation in order to replace parts. In addition, engines and aircraft are equipped with sensors collecting data during tests and flights. Various methods have been devised to analyze these data. During production, the standard methodology is Statistical Process Control (SPC). SPC consists in monitoring the deviation between parameters and their target value, and displays visual alerts to the operators if the deviation becomes too important.

More advanced techniques are developed in the general framework of *Prognostics and Health Monitoring* (PHM). The general aim is to improve availability and operation of engines [Blanchard et al., 2009, Bastard et al., 2016]. It consists in monitoring the state of an engine or a fleet of engines by using operational data and past events. The first objective is to avoid abnormal events as in-flight shutdowns, aborted take-offs and delays and cancellation. The second objective is optimizing maintenance operations to improve safety while reducing costs for manufacturers and airline companies. It is at the core of a *predictive maintenance* strategy (also called condition-based maintenance), which consists in adapting the maintenance plan to the actual state of each individual engine, unlike traditional time-based preventive maintenance, the state of each engine being the result of its actual use during its lifetime. This allows a more efficient scheduling of preventive and corrective actions (e.g. shop visits): time between actions can be increased if no maintenance is necessary (thus reducing costs), and actions can be taken earlier thanks to enhanced predictability of events (thus improving safety). Concretely, engine health monitoring (EHM) combines historical data and physical models to raise alerts, build models that evaluate wear of parts and their residual useful life, probability of failure, etc. These models can be based on thresholds, statistical models incorporating physical knowledge, or machine learning, i.e. statistical models whose parameters are learned from data.

Safran A.E. monitors the condition of its engines either in a test cell environment or by analyzing, post-flight, data collected during the flight. Data are collected using multiple sensors placed on the test cell, the aircraft or the engine itself. In this work, we tackle flight data analysis, on the ground.

At Safran A.E., the PHM department is responsible for developing algorithms to monitor the state of engine sub-systems. In addition, several teams of domain experts are specialized in each aspect of the engine (e.g., performance, vibration dynamics, acoustics, etc.). Finally, the Datalab team provides skills and support in statistics, data analysis, data and software engineering across various projects in the company, always in cooperation with domain experts. The role of a data lab in a company is to play a transverse role for all data-related projects and to break silos between different business entities. During this PhD, I have been a member of the Datalab team, and collaborated with other departments.

Challenges and objectives

Aircraft operation generate growing amounts of data that can be leveraged for various applications, including health monitoring, predictive maintenance, and services to airline companies. Health monitoring of industrial assets is a set of techniques that aims at increasing machine availability and safety, while reducing maintenance costs. Knowledge of a machine's condition can be extracted from data. Today's highly instrumented aircraft produce huge amounts of data, as hundreds of sensors measurements are recorded during whole flights at a high frequency. Data sets collected at Safran A.E. from the LEAP engines are in the range of gigabytes per flight, with thousands of engines operated every day. As a consequence, these data require specific software tools to be stored and processed efficiently: we have entered a *Big Data* era. Thus, a crucial objective of this PhD is to make possible the scaling up of aircraft engine data analysis and health monitoring applications at Safran A.E.

Map-Reduce has emerged as a paradigm enabling to process huge amounts of potentially unstructured data while abstracting out to the application programmer the precise details of where the different parts of the data are stored. According to its inventors [Dean and Ghemawat, 2008], "this allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system". Today, the modern distributed computing frameworks that have bloomed from the original Map-Reduce are far more powerful and flexible,

such as the Apache Spark framework used in this PhD. Still, we found that their relatively steep learning curve prevented domain experts to easily deploy their methodologies at the scale of entire fleets. Solving this issue is the goal of the generic analytics pipeline introduced in Chapter 8. Then, various use cases can be realized in collaboration with domain experts. Then, insights need to be extracted from vast amounts of unlabeled data, which are in addition high-dimensional. To this end, unsupervised learning algorithm are used, to address clustering (partitioning into groups without supervision), dimensionality reduction, and visualization, among others. We focus on self-organizing maps (SOM) [Kohonen, 1982], which have the advantage of producing interpretable results.

In this thesis, besides the engineering aspects, we are also interested in the theoretical challenges and tried to address two very fundamental issues in unsupervised learning. First, the links between structure and representation. And second, the very definition of structure, arising from the problem of model selection in clustering.

Clustering in high-dimensional spaces is a difficult problem which is recurrent in many domains, for example in image or signal analysis. The difficulty is due to the fact that high-dimensional data usually live in different low-dimensional subspaces hidden in the original space. In other words, structure depends on the representation. We explore the combination of representation learning, i.e. automatically learning useful features from data using neural networks, and develop a deep variant of the SOM (DESOM). However, as complex as the algorithm may be, it is of no use if its solutions cannot be evaluated properly, in an objective way, in order to selects its parameters. This evaluation, called model selection, is a major challenge in clustering, as there is no ground truth to evaluate against. The standard methods have strong limitations. We try to improve on a set of methods called cluster stability analysis, and propose a new criterion along with thorough experiments showing its effectiveness.

In a way, we address scalability issues in several ways. On one hand, we overcome large numbers of observations N using either distributed data-parallel processing, or efficient (linear) learning rules such as stochastic gradient descent. On the other hand, we tackle large numbers of variables P by dimensionality reduction and representation learning.

Overview

This manuscript is organized into three parts:

I **Clustering, self-organization and representation learning** (chapters 1, 2 and 3).
II **Model selection in clustering** (chapters 4, 5 and 6).
III **Industrial applications and scalability** (chapters 7 and 8).

Chapter 1: Clustering and self-organization

The first chapter of this thesis provides an introduction to the field of unsupervised machine learning. It presents the main algorithms for data clustering, as well as dimensionality reduction and visualization based on a self-organization process, such as Self-Organizing Maps (SOM).

Chapter 2: Unsupervised representation learning for clustering

In the second chapter, we dive into unsupervised learning methods based on deep neural networks in order to learn semantic representations of complex, high-dimensional data sets. A major family of models is autoencoders. Then, we present a state-of-the-art of recent approaches combining representation learning and clustering, called deep clustering methods.

Chapter 3: Deep Embedded SOM (DESOM)

The third chapter presents the first main contribution of this thesis to the field of deep clustering. The Deep Embedded SOM (DESOM) is a neural network-based model performing joint representation learning and self-organization of the cluster prototypes. We present in details the model architecture, training procedure, and study its performance and the influence of hyperparameters on several benchmarks.

Chapter 4: Model selection in clustering

Model selection is perhaps the most difficult problem in clustering, and is the subject of chapter 4. It allows to evaluate the results of a clustering algorithm in order to select its "best" parameters, such as the number of clusters. We introduce its challenges, and a selection of external and internal validation indices. In addition, we present specific indices to evaluate SOM models, that we implemented as an open-source Python module.

Chapter 5: Selecting the number of clusters with a stability trade-off

The fifth chapter tackles model selection using cluster stability analysis. Stability methods are based on the principle that a good clustering solution should be stable to perturbations of the data distribution. These perturbations may arise by sampling

or injection of noise. Our contribution is a novel concept of within-cluster stability, leading to a criterion called Stadion (stability difference criterion). Through extensive experiments and benchmarks on 80 data sets, we show its ability to effectively select the number of clusters compared with existing methods in literature.

Chapter 6: Validation of time series clustering with an invariance-guided criterion

In chapter 6, we apply the Stadion criterion to the task of whole time series clustering validation. Time series clustering is challenging in several ways (e.g. high dimensions, correlation, invariances to transformations) and model selection is not well studied. We propose to guide the perturbation process by leveraging invariant transformations of the data, whenever these are know beforehand. We experiment with shifting, scaling and warping transformations, and evaluate well-known center-based time series clustering algorithms.

Chapter 7: Scaling to Big Data with distributed computing

Chapters 7 and 8 aim at solving the challenges of the ever-growing amounts of data generated by aircraft operation. We first define the term *Big Data* and its implications. Then, we provide a technical introduction to the software tools that we use to process large flight data sets on a cluster, namely the Hadoop eco-system and the Spark distributed computing framework. Finally, we show how clustering algorithms can be distributed in the Map-Reduce paradigm to process data in parallel on clusters of machines.

Chapter 8: Industrial applications

Lastly, chapter 8 concerns two contributions around industrial applications developed at Safran A.E. with the objective of scaling methodologies for engine fleet health monitoring. With this goal in mind, a generic pipeline for large-scale processing of aircraft engine data has been developed, based on Spark and deployed on the production cluster. More precisely, it allows domain engineers to massively extract flight features, apply the SOM algorithm, save the models and visualize them through a web application. Studies conducted on LEAP engine data are presented.

Contributions

Accepted papers

- **Forest, F.**, Lacaille, J., Lebbah, M., & Azzag, H. (2018). A Generic and Scalable Pipeline for Large-Scale Analytics of Continuous Aircraft Engine Data.
  IEEE International Conference on Big Data.
  `https://doi.org/10.1109/BigData.2018.8622297`
- **Forest, F.**, Lebbah, M., Azzag, H., & Lacaille, J. (2019). Deep Embedded SOM: Joint Representation Learning and Self-Organization.
  European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN).
  `https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2019-30.pdf`
- **Forest, F.**, Lebbah, M., Azzag, H., & Lacaille, J. (2019). Deep Architectures for Joint Clustering and Visualization with Self-Organizing Maps.
  Workshop on Learning Data Representations for Clustering (LDRC), Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD).
  `https://doi.org/10.1007/978-3-030-26142-9_10`
- **Forest, F.**, Lebbah, M., Azzag, H., & Lacaille, J. (2020). Carte SOM profonde : Apprentissage joint de représentations et auto-organisation.
  CAp: Conférence d'Apprentissage.
  `https://hal.archives-ouvertes.fr/hal-02859997`
- **Forest, F.**, Cochard, Q., Noyer, C., Cabut, A., Joncour, M., Lacaille, J., Lebbah, M. & Azzag, H. (2020). Large-scale Vibration Monitoring of Aircraft Engines from Operational Data using Self-organized Models.
  Annual Conference of the PHM Society.
  `https://phmpapers.org/index.php/phmconf/article/view/1131`
- **Forest, F.**, Mourer, A., Lebbah, M., Azzag, H., & Lacaille, J. (2020). An Invariance-guided Stability Criterion for Time Series Clustering Validation.
  International Conference on Pattern Recognition (ICPR).

Submitted papers

- **Forest, F.**, Lebbah, M., Azzag, H., & Lacaille, J. (2020). Deep Embedded Self-Organizing Map for Joint Representation Learning and Topology-Preserving Clustering.
  Submitted to Neurocomputing (journal).

- Mourer, A., **Forest, F.**, Lebbah, M., Azzag, H., & Lacaille, J. (2020). Selecting the Number of Clusters K with a Stability Trade-off: an Internal Validation Criterion.
  Submitted to AISTATS.
  `https://arxiv.org/abs/2006.08530`

Patents

- Lacaille, J., **Forest, F.**. Système d'environnement informatique pour la surveillance de moteurs d'aéronefs / Computer environment system for monitoring aircraft engines.
  Publication No.: FR3089501 (2020-06-12)
  `https://bases-brevets.inpi.fr/fr/document/FR3089501.html`
  `https://patents.google.com/patent/FR3089501`

Open-source software

- **Spark ML SOM**. Distributed self-organizing map implementation with a Spark ML API (Scala). `https://github.com/FlorentF9/sparkml-som`
- **DESOM**. Keras implementation of the Deep Embedded SOM model (Python). `https://github.com/FlorentF9/DESOM`
- **SOMperf**. A collection of performance metrics for self-organizing maps (Python). `https://github.com/FlorentF9/SOMperf`
- **DeepTemporalClustering**. Keras implementation of the Deep Temporal Clustering model (Python). `https://github.com/FlorentF9/DeepTemporalClustering`
- **skstab**. Clustering stability analysis in Python with a scikit-learn compatible API (Python). `https://github.com/FlorentF9/skstab`

Introduction

Cette introduction étendue en français reprend l'introduction précédente, avec un résumé de chaque chapitre. Dans un premier temps, nous présentons les différents acteurs, le contexte et enjeux autour de ce projet de thèse. Le laboratoire universitaire et l'entreprise, ainsi que leur structure, leurs entités et leurs principaux produits sont présentés. Ensuite, nous décrivons brièvement le fonctionnement des moteurs d'avion (turboréacteurs ou turbofans) et les principes de la surveillance de santé (health monitoring), ainsi que les principaux défis théoriques et technologiques qui se posent lors du passage à l'échelle en terme de taille et de complexité des jeux de données. Nous justifierons l'adoption du cadre d'apprentissage non supervisé, afin d'extraire des informations à partir de jeux de données en grande dimension. Ensuite, des sections seront consacrées à résumer chacun des chapitres de ce manuscrit. Les contributions de cette thèse, en termes de publications (articles acceptés et soumis), de brevets, et de contributions de logiciels libres ont déjà été énumérées à la fin du précédent chapitre.

Contexte et motivations

Le travail présenté dans ce mémoire de thèse est le fruit d'une collaboration entre le Laboratoire d'Informatique de Paris Nord (LIPN) de l'Université Sorbonne Paris Nord, et l'entreprise Safran Aircraft Engines (groupe SAFRAN).

L'Université Sorbonne Paris Nord (ex Paris 13) est l'une des universités qui ont succédé à la Sorbonne après l'éclatement de l'Université de Paris en treize universités autonomes en 1968. Le Laboratoire d'Informatique de Paris-Nord (LIPN), créé en 1985, est associé au CNRS depuis janvier 1992 et a le statut d'unité mixte de recherche (UMR 7030) depuis janvier 2001. La recherche effectuée au LIPN se développe autour d'axes forts s'appuyant sur les compétences de ses membres, en particulier en combinatoire, en optimisation combinatoire, en algorithmique, en logique, en génie logiciel, en langage naturel, en apprentissage. Le laboratoire est structuré en cinq équipes qui reflètent ces axes :

- A3 : Apprentissage Artificiel et Applications.

- AOC : Algorithmes et Optimisation Combinatoire.
- CALIN : Combinatoire, ALgorithmique et INteractions.
- LoVe : Logique et Vérification.
- RCLN : Représentation des Connaissances et Langage Naturel.

Safran Aircraft Engines (Safran A.E.), société du groupe SAFRAN, conçoit, développe, produit, et commercialise, seul ou en coopération, des moteurs pour avions civils et militaires, pour lanceurs spatiaux et pour satellites. Safran Aircraft Engine propose également aux compagnies aériennes, aux forces armées et aux opérateurs d'avions une gamme complète de services pour leurs moteurs aéronautiques, couvrant le cycle de vie du moteur de son entrée en service à son démantèlement. Face à la concurrence et à l'amélioration permanente des technologies, Safran Aircraft Engine doit rester un motoriste compétitif en proposant des nouveaux moteurs innovants : moins bruyants, plus économes en consommation de carburant, plus légers, etc.

Dans le cadre de cette thèse, nous nous intéressons aux moteurs d'avions équipant des avions civils courts et moyens courriers. Ces produits sont conçus par la société CFM, une joint-venture entre le français Safran A.E. et l'américain General Electric (GE). La Figure 0.1 montre un moteur CFM56-7B (le moteur le plus vendu au monde), et la génération actuelle, le LEAP. Dans ce travail, nous analyserons les données des moteurs LEAP uniquement. La Table 0.1 contient les statistiques de la flotte moteurs LEAP en opération au 31 mars 2020. Plus de 1000 avions et 2400 moteurs sont en service, pour un total de plus de 3,5 millions de cycles. Un *cycle* est le terme technique utilisé à la place du mot vol pour désigner un démarrage et un arrêt complet du moteur.

Avant de poursuivre cette introduction, il convient d'expliquer brièvement la manière dont un turboréacteur à double flux produit de la poussée. Tout d'abord, l'air est capté à l'entrée du moteur. Une partie de cet air, appelé *flux primaire*, passe par la soufflante (le *fan*) et continue dans le compresseur du coeur, puis dans la chambre de combustion, où il est mélangé au carburant et où la combustion se produit. Les gaz d'échappement chauds traversent le coeur et les turbines de la soufflante, puis sortent par la tuyère. Le reste de l'air entrant traverse le fan et contourne le moteur, c'est ce qu'on appelle le *flux secondaire*. Celui-ci voit sa vitesse légèrement augmentée. Ainsi, un turbofan tire une partie de sa poussée du coeur et une autre partie de la soufflante. Le rapport entre le flux secondaire et le flux primaire est appelé *taux de dilution*.

Bien que le principe de base d'un moteur à réaction soit simple (troisième loi de Newton), l'ensemble du système est d'une extrême complexité. Un moteur est constitué de plusieurs sous-systèmes et d'un grand nombre de pièces fixes et mobiles,

comme on peut le voir sur la coupe 3D d'un moteur CFM65-7B sur la Figure 0.1 (à gauche). Il est utile d'avoir une vue simplifiée des principaux composants du moteur (voir Figure 0.2). Un turbofan est composé de deux arbres principaux, l'arbre basse pression (LP) et l'arbre haute pression (HP) (ainsi qu'un arbre d'entraînement radial (RDS), non représenté ici). L'arbre basse pression est alimenté par la turbine basse pression et entraîne la soufflante (entrée du moteur) et le compresseur basse pression. L'arbre HP est entraîné par la turbine HP (qui succède à la chambre de combustion) et entraîne le compresseur HP. Enfin, le RDS est relié à l'arbre HP et fournit la puissance au boîtier d'accessoires.

Une fiabilité extrêmement élevée est nécessaire pour atteindre le niveau de sécurité requis pour les vols civils ou militaires. Généralement, le taux de défaillance doit être inférieur à 10^{-9} par heure de vol, et les moteurs sont conçus pour durer de l'ordre de 15 ans ou XXK cycles (avion : jusqu'à 30 ans ou 100k cycles). Le constructeur assure cette performance grâce à une conception et une fabrication robustes, des normes et des certifications strictes, des essais approfondis, ainsi que des inspections visuelles et un entretien régulier pendant leur fonctionnement afin de remplacer les pièces. En outre, les moteurs et les avions sont équipés de capteurs qui collectent des données durant les essais et en vol. Diverses méthodes ont été conçues pour analyser ces données. En cours de production, la méthodologie standard est le contrôle statistique des procédés (SPC). Le SPC consiste à surveiller l'écart entre les paramètres et leur valeur cible, et affiche des alertes visuelles aux opérateurs si l'écart devient trop important.

Des techniques plus avancées sont développées dans le cadre du *Prognostics and Health Monitoring* (PHM). L'objectif général est d'améliorer la disponibilité et l'exploitation des moteurs [Blanchard et al., 2009, Bastard et al., 2016]. Il consiste à surveiller l'état d'un moteur ou d'une flotte en utilisant les données opérationnelles et un historique des événements. Le premier objectif est d'éviter les événements anormaux tels que les arrêts en vol, les décollages interrompus, les retards et les annulations. Le second objectif est d'optimiser les opérations de maintenance pour améliorer la sécurité tout en réduisant les coûts pour les constructeurs et les compagnies aériennes. Il est au cœur d'une stratégie de *maintenance prédictive*, qui consiste à adapter le plan de maintenance à l'état réel de chaque moteur individuel, contrairement à la maintenance préventive traditionnelle basée sur le temps, l'état de chaque moteur étant le résultat de son utilisation réelle au cours de sa vie. Ceci permet une programmation plus efficace des actions préventives et correctives (par exemple les visites en atelier) : le temps entre les actions peut être augmenté si aucune maintenance n'est nécessaire (ce qui réduit les coûts), et les actions peuvent être entreprises plus tôt grâce à une meilleure prévisibilité des événements (ce qui

améliore la sécurité). Concrètement, la surveillance de la santé des moteurs (engine health monitoring, EHM) combine des données historiques et des modèles physiques pour déclencher des alertes, construire des modèles qui évaluent l'usure des pièces et leur durée de vie résiduelle, la probabilité de défaillance, etc. Ces modèles peuvent être basés sur des seuils, des modèles statistiques intégrant des connaissances physiques ou l'apprentissage machine, c'est-à-dire des modèles statistiques dont les paramètres sont appris à partir de données.

Safran A.E. surveille l'état de ses moteurs soit dans un environnement de banc d'essai, soit en analysant après un vol, les données recueillies à l'aide de multiples capteurs placés sur la cellule d'essai, l'avion ou le moteur lui-même. Dans ce travail, nous abordons l'analyse au sol des données de vol.

Chez Safran A.E., le département PHM est chargé de développer des algorithmes pour surveiller l'état des divers sous-systèmes du moteur. En outre, plusieurs équipes d'experts sont spécialisées dans chaque aspect du moteur (par exemple, les performances, la dynamique vibratoires, l'acoustique, etc.). Enfin, l'équipe du Datalab possède des compétences en matière de statistiques, d'analyse de données ou de génie logiciel et arrive en support pour de nombreux projets de l'entreprise, toujours en coopération avec les bureaux d'étude. L'intérêt d'un "data lab" (laboratoire de données) se justifie à partir du moment où une entreprise qui souhaite valoriser ses données grâce aux techniques avancées de data science se retrouve face à des problèmes de silotage de l'information dans ses différents départements. Au cours de cette thèse, j'ai été membre de l'équipe du Datalab et ai collaboré avec d'autres départements et bureaux d'études.

Défis et objectifs

L'exploitation des aéronefs génère des quantités croissantes de données qui peuvent être exploitées pour diverses applications, notamment la health monitoring, la maintenance prédictive et les services aux compagnies aériennes. Le health monitoring d'actifs industriels est un ensemble de techniques qui visent à accroître la disponibilité et la sécurité des machines, tout en réduisant les coûts de maintenance. La connaissance de l'état d'une machine peut être extraite des données. Les avions actuels sont hautement instrumentés et produisent d'énormes quantités de données : des centaines de mesures de capteurs sont enregistrées à une fréquence élevée durant des vols entiers. Les jeux de données recueillis à Safran A.E. issus des moteurs LEAP sont de l'ordre des gigaoctets par vol, avec des milliers de moteurs en

fonctionnement chaque jour. En conséquence, ces données nécessitent des outils logiciels spécifiques pour être stockées et traitées efficacement : nous sommes entrés dans l'ère du *Big Data*. Ainsi, un objectif crucial de cette thèse est de rendre possible le passage à l'échelle des applications d'analyse de données des moteurs et de health monitoring à Safran A.E.

Le paradigme Map-Reduce apparu ces dernière années permet de traiter d'énormes quantités de données potentiellement non structurées, tout en cachant au programmeur la complexité de la localité précise du stockage des différentes parties des données. Selon ses pionniers [Dean and Ghemawat, 2008], "il permet aux programmeurs sans aucune expérience en systèmes parallèles et distribués d'utiliser facilement les ressources d'un grand système distribué". Aujourd'hui, les frameworks logiciels modernes de calcul distribué qui ont émergé à partir du Map-Reduce originel sont plus puissants et flexibles, comme Apache Spark, utilisé au cours de cette thèse. Néanmoins, nous avons constaté que leur courbe d'apprentissage relativement raide empêchait les experts du domaine de déployer facilement leurs méthodologies à l'échelle de flottes entières. Résoudre ce problème est l'objectif du pipeline générique d'analyse présenté au Chapitre 8. Ensuite, divers cas d'utilisation peuvent être réalisés en collaboration avec les bureaux d'étude. Ensuite, il faut extraire des informations d'une grande quantité de données non étiquetées, qui plus est de grande dimension. À cette fin, des algorithmes d'apprentissage non supervisé sont utilisés pour le clustering (partitionnement en groupes sans supervision), la réduction de dimension et la visualisation, entre autres. Nous nous concentrons sur les cartes auto-organisées (self-organizing maps, SOM) [Kohonen, 1982], qui ont l'avantage de produire des résultats interprétables.

Dans cette thèse, outre les aspects d'ingénierie, nous avons également tenté d'aborder deux défis théoriques fondamentaux en apprentissage non supervisé. Premièrement, les liens entre structure et représentation. Et deuxièmement, la définition même de la structure, découlant du problème de la sélection du modèle dans en clustering.

Le clustering dans des espaces de grande dimension est un problème difficile, récurrent dans de nombreux domaines, par exemple l'analyse d'images ou de signaux. La difficulté est due au fait que les données vivent généralement dans différents sous-espaces de plus faible dimension cachés dans l'espace d'origine. En d'autres termes, la structure dépend de la représentation. Nous explorons la combinaison de l'apprentissage de représentations, c'est-à-dire l'apprentissage automatique de caractéristiques utiles à partir de données en utilisant des réseaux de neurones, et nous développons une variante profonde de l'algorithme SOM (DESOM). Cependant, aussi complexe que soit l'algorithme, il demeure inutile si ses solutions ne peuvent

être évaluées de manière objective afin de sélectionner ses paramètres, tels que le nombre de clusters à découvrir. Cette évaluation, appelée sélection de modèle, est un défi majeur en clustering, car il n'y a pas de vérité terrain par rapport à laquelle évaluer une performance. Les méthodes classiques ont de fortes limitations. Nous essayons d'améliorer un ensemble de méthodes appelé analyse de stabilité des clusters, et proposons un nouveau critère ainsi que des expériences approfondies montrant son efficacité.

D'une certaine manière, nous abordons les questions de passage à l'échelle de plusieurs manières. D'une part, nous surmontons les grands nombres d'observations N en utilisant soit un traitement distribué des données en parallèle, soit des règles d'apprentissage efficaces (linéaires) telles que la descente de gradient stochastique. D'autre part, nous surmontons de grands nombres de variables P par réduction de la dimension et apprentissage de représentations.

Plan de la thèse

Ce manuscrit s'articule en trois parties :

I **Clustering, auto-organisation et apprentissage de représentations** (chapitres 1, 2 et 3).
II **Sélection de modèle en clustering** (chapitres 4, 5 et 6).
III **Applications industrielles et passage à l'échelle** (chapitres 7 et 8).

Les sections suivantes contiennent un résumé approfondi en français de chacun des chapitres, rédigés en anglais, de cette thèse.

Chapitre 1 : Clustering et auto-organisation

Le premier chapitre de cette thèse (Chapitre 1) est une introduction au domaine de l'apprentissage automatique non supervisé. Il présente les principaux algorithmes de partitionnement de données (clustering), ainsi que les méthodes de réduction de dimension et de visualisation basés sur un processus d'auto-organisation, tel que les cartes auto-organisées (SOM).

Chapitre 2 : Apprentissage de représentations pour le clustering

Dans le second chapitre (Chapitre 2), nous nous plongeons dans les méthodes d'apprentissage non supervisé basées sur les réseaux neuronaux profonds afin d'apprendre des représentations sémantiques de jeux de données complexes et

de grande dimension. Une importante famille de modèles est celle des auto-encodeurs. Ensuite, nous proposons un état de l'art des récentes approches combinant l'apprentissage de représentation et le clustering, appelées méthodes de clustering profond (deep clustering).

Chapitre 3 : DESOM : Carte auto-organisée profonde

Le Chapitre 3 présente la première contribution principale de cette thèse dans le domaine du clustering profond. Le Deep Embedded SOM (DESOM) est un modèle basé sur un réseau de neurones permettant l'apprentissage conjoint de représentations et l'auto-organisation des prototypes de clusters. Nous présentons en détail l'architecture du modèle, la procédure d'apprentissage, et étudions ses performances ainsi que l'influence des hyperparamètres sur plusieurs jeux de données standards.

Chapitre 4 : Sélection de modèle en clustering

La sélection de modèle est peut-être le problème le plus ardu dans le domain du clustering, et fait l'objet du Chapitre 4. Il s'agit d'évaluer les résultats d'un algorithme de clustering afin de sélectionner ses "meilleurs" paramètres, tels que le nombre de clusters. Nous en présentons les défis, ainsi qu'une sélection d'indices de validation externes et internes. De plus, nous présentons des indices pour évaluer spécifiquement les modèles SOM, que nous avons implémentés au sein d'un module Python open-source.

Chapitre 5 : Sélection du nombre de clusters par analyse de stabilité

Le Chapitre 5 aborde la sélection des modèles à l'aide de l'analyse de stabilité des clusters. Les méthodes de stabilité sont basées sur le principe qu'un bon clustering se doit d'être stable sous l'effet de perturbations de la distribution des données. Ces perturbations peuvent survenir par l'échantillonnage ou par injection de bruit dans les données. Notre contribution est un nouveau concept de stabilité intra-groupe, conduisant à un critère appelé Stadion (stability difference criterion). Par le biais d'expériences et d'analyses approfondies sur 80 jeux de données, nous démontrons sa capacité à sélectionner le nombre de clusters de manière compétitive par rapport aux méthodes de la littérature.

Chapitre 6 : Validation du clustering de séries temporelles invariantes

Dans le Chapitre 6, nous appliquons le critère Stadion à la validation du clustering de séries temporelles entières. Le partitionnement de séries temporelles est un défi à plusieurs égards (par exemple, dimensions élevées, corrélation, invariances aux transformations) et la sélection de modèle est relativement peu explorée dans ce champ d'application. Nous proposons de guider le processus de perturbation en

tirant parti des invariances des données, lorsque celles-ci sont connues à l'avance. Nous expérimentons avec des transformations de décalage temporel, de mise à l'échelle et de déformation temporelle (warping), et nous évaluons des algorithmes classiques de clustering de séries temporelles basées sur les centroïdes.

Chapitre 7 : Passage à l'échelle du Big Data avec le calcul distribué

Les Chapitres 7 et 8 visent à résoudre les problèmes posés par les quantités toujours croissantes de données générées par l'exploitation des aéronefs. Nous commençons par définir le terme *Big Data* et ses implications. Ensuite, nous fournissons une introduction technique aux outils logiciels que nous utiliserons pour traiter de grands jeux de données de vols sur un cluster de machines, à savoir l'écosystème Hadoop et le framework de calcul distribué Spark. Enfin, nous montrons comment les algorithmes de clustering peuvent être distribués dans le paradigme Map-Reduce pour traiter des données en parallèle sur des clusters de machines.

Chapitre 8 : Applications industrielles

Enfin, le Chapitre 8 concerne deux contributions autour d'applications industrielles développées chez Safran A.E. ayant pour objectif de passer à l'échelle les méthodologies de health monitoring des flottes de moteurs d'avion. Dans ce but, un pipeline générique pour le traitement à grande échelle des données moteur a été développé, basé sur Spark et déployé sur le cluster de production. Plus précisément, il permet aux ingénieurs du domaine d'extraire massivement des indicateurs de vols, d'appliquer l'algorithme SOM, de sauvegarder les modèles et de les visualiser par le biais d'une application web. Les études menées sur les données du moteur LEAP sont présentées.

Notations

X, Y, Z	Random variables
P	Dimension of data space
$\mathcal{X}$	Data space (generally $\mathcal{X} \subset \mathbb{R}^P$)
L	Dimension of latent space
$\mathcal{Z}$	Latent feature space (generally $\mathcal{Z} \subset \mathbb{R}^L$)
N	Number of training samples
$\mathbf{x}_i = (\mathbf{x}_i^1, \ldots, \mathbf{x}_i^P), 1 \leq i \leq N$	Data sample ($\mathbf{x}_i \in \mathcal{X}$)
$\mathbf{z}_i = (\mathbf{z}_i^1, \ldots, \mathbf{z}_i^L), 1 \leq i \leq N$	Latent data sample ($\mathbf{z}_i \in \mathcal{Z}$)
$\mathcal{P}$	Data-generating distribution over $\mathcal{X}$
$\mathcal{A}$	Learning algorithm
K	Number of clusters
$K^\star$	True number of clusters in a data set
$\mathbf{m}_k, 1 \leq k \leq K$	Cluster center/prototype vector
T	Temperature parameter in SOM algorithm
$\mathcal{K}$	SOM neighborhood kernel function
$\mathcal{L}$	Loss function
l_r	Learning rate
$\mathcal{C}_K$	Clustering partition into K clusters
D	Number of perturbed samples in stability estimation
$s(\cdot, \cdot)$	Similarity measure between two clusterings
ε-AP	Additive perturbation with noise level ε
$\mathbf{\Omega}$	Set of parameters for within-cluster stability estimation
$\mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$	Normal distribution with mean $\boldsymbol{\mu}$ and covariance $\mathbf{\Sigma}$
$\mathbf{I}$	Identity matrix
$\|\| \cdot \|\|_2$	ℓ_2/Euclidean norm
$d(\cdot, \cdot)$	Distance or dissimilarity in data space
$\delta(\cdot, \cdot)$	ℓ_1/Manhattan distance between units on a SOM
$\| \cdot \|$	Cardinality of a set
i.i.d.	independently and identically distributed
w.r.t.	with respect to
$:=$	Definition of a variable or function

A glossary of frequently used acronyms is available on page 276.

Part I

Clustering, self-organization and representation learning

1 Clustering and self-organization

> *If intelligence is a cake, the cherry on the cake is reinforcement learning, the icing on the cake is supervised learning, and the bulk of the cake is unsupervised learning.*
>
> **— Yann LeCun**

1.1 Statistical learning: an overview

Statistical learning, or *machine learning* [Hastie et al., 2008, Shalev-Shwartz and Ben-David, 2013], is a scientific area at the intersection of applied mathematics, statistics and computer science. It is a subset of the larger field of artificial intelligence. Its general goal is the ability to solve a task by learning from a set of data samples, called a *training data set*. Differently from traditional computer programming, machine learning automatically extracts meaningful patterns in data and achieves a task without being explicitly programmed. Therefore, this task is also often called *pattern recognition*. Today, it is already applied in a wide variety of domains ranging from computer vision (images and video), natural language processing (text), audio and speech processing, signal processing, robotics, healthcare, biology, manufacturing, astronomy, economics, advertising, art, etc. Statistical learning has gained traction in the current context of *Big Data*, that is a context of ever-increasing data volumes, improved access to data and facilitated data collection. This buzzword will be demystified in Chapter 7.

The next paragraphs provide an overview of the two main learning paradigms, *supervised* and *unsupervised* learning. Other types of learning systems, such as reinforcement learning, are outside the scope of this thesis.

1.1.1 Supervised learning

Supervised learning aims at predicting an outcome variable $Y \in \mathcal{Y}$, given a set of descriptive variables $X \in \mathcal{X}$ (generally $\mathcal{X} \subset \mathbb{R}^P$), also called *features*, that are assumed to have influence on the outcome Y. Supervised learning tasks are divided into two types:

- *Regression*, when Y is a continuous variable, typically $\mathcal{Y} \subset \mathbb{R}$.
- *Classification*, when Y is a discrete or categorical variable, for example $\mathcal{Y} = \{0, 1\}$ for binary classification.

In other words, we seek a function $f : \mathcal{X} \longrightarrow \mathcal{Y}$ describing the relationship between X and Y, called a *hypothesis* or *predictor*. However, X and Y are subject to uncertainty because this unknown relationship may depend on variables other than X and be subject to random noise. Thus, we adopt a probabilistic setting where (X, Y) are random variables from a joint distribution $\mathcal{P}$. The quality of a hypothesis is measured by a so-called *loss function* or *objective function* $\mathcal{L} : (\mathcal{X} \times \mathcal{Y}) \times f \longrightarrow \mathbb{R}^+$, such that $\mathcal{L}((X, Y), f)$ is small if and only if $f(X)$ is a good prediction of Y. The goal is to find a hypothesis $f^\star$ that minimizes the *risk* or *generalization error*, defined as the expected loss incurred when using f to predict Y:

$$\begin{aligned} f^\star &:= \underset{f \in \mathcal{F}(\mathcal{X}, \mathcal{Y})}{\operatorname{argmin}} \; R(f) \\ \text{where} \quad R(f) &:= \mathbb{E}_{(X,Y)\sim\mathcal{P}} \left[\mathcal{L}((X, Y), f)\right] \end{aligned} \tag{1.1}$$

where $\mathcal{F}(\mathcal{X}, \mathcal{Y})$ is the set of all possible hypotheses. Commonly used loss functions are the square loss $\mathcal{L}((X, Y), f) = \frac{1}{2}\left(f(X) - Y\right)^2$ for regression, and the 0-1 loss for classification $\mathcal{L}((X, Y), f) = \mathbb{1}_{f(X)\neq Y}$. The optimal predictor minimizing the risk is called the *Bayes predictor*, and in some cases it has an analytical form, for example for the square loss:

$$\begin{aligned} R(f) &= \mathbb{E}_{(X,Y)}\left[\left(f(X) - Y\right)^2\right] \\ &= \mathbb{E}_X \mathbb{E}_{Y|X}\left[\left(f(X) - Y\right)^2 |X\right] \\ &= \mathbb{E}_X \mathbb{E}_{Y|X}\left[f(X)^2 - 2f(X)\mathbb{E}\left[Y|X\right] + \mathbb{E}[Y^2|X]\right] \end{aligned}$$

which is minimized pointwise by $f^\star(x) = \mathbb{E}_{\mathcal{P}}\left[Y|X = x\right]$. Hence, supervised learning expresses as the task of estimating the conditional probability distribution of Y given X, $P(Y|X)$, through a *location* estimate. For MSE, this location is the mean, but if we use a ℓ_1 loss $\mathcal{L}((X, Y), f) = |f(X) - Y|$, the estimate will be the median.

However, in practice, we do not have access to the underlying distribution $\mathcal{P}$, and the optimal predictor needs to be approximated using a set of N training data samples, denoted $S_N = \{(x_1, y_1), \ldots, (x_N, y_N)\}$. A *learning algorithm*, or *learning rule*, is the process of choosing a sensible hypothesis given a training set. It is a function $\mathcal{A} : \bigcup_{n \geq 1} (\mathcal{X} \times \mathcal{Y})^n \longrightarrow \mathcal{F}(\mathcal{X}, \mathcal{Y})$. The most common learning rule is to select the function that minimizes the average loss on the observed data. This rule is called *Empirical Risk Minimization* (ERM):

$$\hat{f} := \underset{f \in \mathcal{F}(\mathcal{X}, \mathcal{Y})}{\operatorname{argmin}} \; R_N(f)$$
$$\text{where} \;\; R_N(f) := \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}((x_i, y_i), f) \tag{1.2}$$

The empirical risk is also commonly called *training error*. The goal of a learning system is to generalize well to unseen examples (i.e. achieve low risk). A low training error is not a guarantee for low generalization error. Very often, a predictor performs well on the training samples but fails to generalize (it is easy to build a predictor that simply memorizes the training set), a phenomenon known as *overfitting*. A solution is to limit the class of hypotheses to a set $\mathcal{H}$, using prior knowledge (known as *inductive bias*). This leads to the constrained formulation of ERM:

$$\hat{f}_{\mathcal{H}} := \underset{f \in \mathcal{H}}{\operatorname{argmin}} \; R_N(f) \tag{1.3}$$

where $\mathcal{H}$ is the *hypothesis space*, a subset of $\mathcal{F}(\mathcal{X}, \mathcal{Y})$, or the penalized formulation

$$\hat{f}_{\lambda} := \underset{f \in \mathcal{F}(\mathcal{X}, \mathcal{Y})}{\operatorname{argmin}} \; R_N(f) + \lambda \theta(f) \tag{1.4}$$

where θ penalizes some functions that are not desirable (generally the most complex or least regular ones). Let us decompose the risk of $\hat{f}$ into two parts:

$$R(\hat{f}_{\mathcal{H}}) = \underbrace{\min_{f \in \mathcal{H}} R(f)}_{\epsilon_{\text{app}}} + \underbrace{R(\hat{f}_{\mathcal{H}}) - \min_{f \in \mathcal{H}} R(f)}_{\epsilon_{\text{est}}}, \tag{1.5}$$

where we define:

- ϵ_{app}, the *approximation error*: the minimum risk achievable in the hypothesis class $\mathcal{H}$.
- ϵ_{est}, the *estimation error*: the difference between the actual empirical risk and the approximation error. This quantity is always positive as training error is only an estimate of the true risk. It becomes large in case of overfitting.

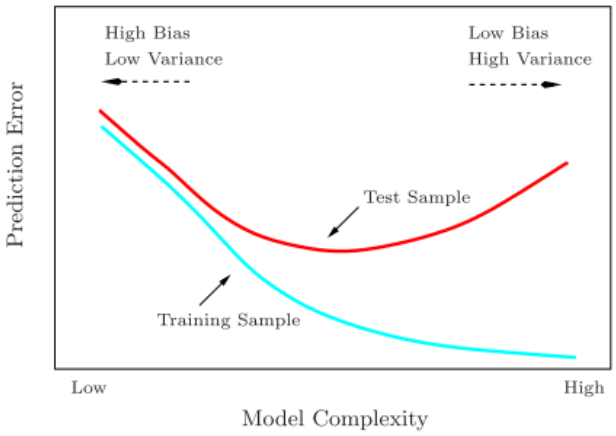


Fig. 1.1.: Bias-complexity trade-off. [Hastie et al., 2008]

In order to minimize the total risk, both errors need to be minimized. On the one hand, the hypothesis class $\mathcal{H}$ needs to be rich enough to contain good hypotheses that achieve a low approximation error. The expressiveness of this hypothesis class is also called the model's *complexity*. But at the same time, a larger $\mathcal{H}$ increases the chances of overfitting and thus, increases the estimation error. On the other hand, a hypothesis class too small reduces the estimation error but might produce a higher approximation error (conversely, this phenomenon is called *underfitting*). This is a central trade-off in learning theory, called the *bias-complexity trade-off* (see Figure 1.1).

Other learning rules are *Structural Risk Minimization* (SRM) and *Minimum Description Length* (MDL), which assign different weights to sets of hypotheses. Finally, *local averaging* or *learning by memorization* is used in k-nearest neighbors (k-NN), the simplest learning algorithm, but it suffers from high memory cost, requires to scan the entire training set during inference and turns out ineffective in high dimensions.

Well-known supervised algorithms include least squares and logistic regression, support vector machines (SVM), classification and regression decision trees, and neural networks.

1.1.2 Maximum likelihood estimation

Let us consider a distribution $\mathcal{P}_\theta$ with parameter θ. Assuming that the data is distributed according to $\mathcal{P}_\theta$, the *likelihood* of an observation $X = x$ is $\mathcal{P}_\theta(x)$ (where this notation encapsulates both the probability of a discrete random variable and

the probability density in the continuous case). Then, we can define the negative log-likelihood loss, referred to as the *log-loss*:

$$\mathcal{L}(x;\ \theta) := -\log \mathcal{P}_\theta(x) \tag{1.6}$$

Assuming a training set of independent and identically distributed (i.i.d.) samples $S_N = \{x_1, \ldots, x_N\}$, we define the *maximum likelihood estimator* (MLE) of θ:

$$\begin{aligned}\theta_{\text{MLE}} := \operatorname*{argmax}_\theta \log \prod_{i=1}^{N} \mathcal{P}_\theta(x_i) &= \operatorname*{argmax}_\theta \sum_{i=1}^{N} \log \mathcal{P}_\theta(x_i) \\ &= \operatorname*{argmin}_\theta \sum_{i=1}^{N} \mathcal{L}(\theta, x_i)\end{aligned} \tag{1.7}$$

The last formulation of the MLE is identical to the previously introduced ERM rule, using the log-loss.

1.1.3 The curse of dimensionality

This famous term was first coined by Richard Bellman [Bellman, 1961] about the exhaustive enumeration of product spaces. The curse of dimensionality also refers to the fact that in the absence of simplifying assumptions, the number of data samples required to estimate a function of several variables to a given accuracy (i.e. to get a reasonably low-variance estimate) on a given domain grows exponentially with the number of dimensions. This phenomenon has many unexpected manifestations, such as the concentration of norms and distances, the hypervolumes of cubes or spheres, tail distributions of Gaussians, etc. Consider uniformly distributed samples inside a hypercube with unit length in dimension P. In order for a neighborhood around some target point to cover a fraction r of the data samples, its expected edge length (i.e. range of each variable) is $r^{1/p}$. To cover a ratio $r = 10\%$ of the volume, this range reaches 46% with $P = 3$, 80% with $P = 10$ and 98% when $P = 100$ (see Figure 1.2 for an illustration). In other words, almost the entire volume of a cube is concentrated near its surface (the same computation can be done with a hypersphere). One of the consequences is that local averaging methods, such as nearest neighbors, are no longer usable, because the neighborhood must cover almost the entire space to average enough samples and obtain a low-variance estimate. This fact is also often called the "empty space phenomenon". Because the amount of available data is generally restricted to a limit number of observations, a crucial consequence for statistical learning in high-dimensional spaces is the need for either dimensionality reduction or strong prior knowledge, i.e. inductive bias.

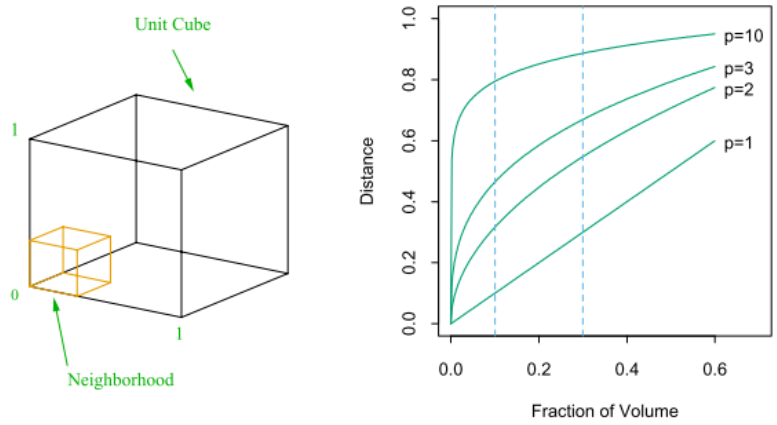


Fig. 1.2.: Curse of dimensionality illustrated by the volume covered by a neighborhood inside a hypercube. As the dimension increases, the neighborhood must cover the entire space. [Hastie et al., 2008]

1.1.4 Unsupervised learning

In many cases, labeled data are not readily available, because labeling the training data is expensive, tedious, and intractable at a large scale. In addition, one is often not interested in predicting a particular target variable, but rather in *understanding* the data. Unsupervised learning provides a way to understand the hidden structure of complex data sets. As opposed to supervised learning, the goal is not to predict a target variable, and no labels are provided. It aims at inferring properties of the full distribution $\mathcal{P}(X)$, which is more difficult than $\mathcal{P}(Y|X)$ in supervised learning, because the dimension of X is usually much higher than Y, and we are interested in properties richer than a location estimate. We will briefly introduce some common techniques and unsupervised learning tasks, before diving more specifically into clustering.

Some methods, called *semi-supervised*, are at the boundary of supervised and unsupervised learning, as they leverage unlabeled data in presence of very few labeled instances. This paradigm is very effective but has not be studied during this thesis.

Clustering

Clustering, also called *cluster analysis*, is a widely used unsupervised learning technique which aims at discovering meaningful groups in unlabeled data [Hastie et al., 2008, Jain, 2010, Aggarwal and Reddy, 2013, Shalev-Shwartz and Ben-David, 2013]. In literature, it has been defined as "grouping or segmenting a collection of objects into subsets or clusters, such that those within each cluster are more closely related

to one another than objects assigned to different clusters" [Hastie et al., 2008]. Before the next section, which is dedicated to clustering, the main algorithmic tool used throughout this thesis, we will first introduce other unsupervised learning tasks.

Association rules

Association rules [Agrawal et al., 1995, Hastie et al., 2008] is an early technique first applied to customer databases, i.e. a data set consisting in a typically binary matrix of size $N \times P$ where N is the number of customers and P the number of products, each element indicating whether a customer has purchased a product. Its goal is to construct simple rules that describe regions of high density, in particular *prototype* values $X = (X^1, \ldots, X^P)$ that appear frequently in the database. It tries to describe $\mathcal{P}(X)$ by its modes, a task also called "mode finding".

Dimensionality reduction and representation learning

Dimensionality reduction (DR) consists in reducing the dimension, i.e. the number of variables, of a data set in order to improve a downstream task. Reducing the dimension has several benefits: first, it is a form of compression and reduces the computational burden. Second, it improves the behavior of learning algorithms, by finding useful variables and escaping the "curse of dimensionality". Finally, reducing the dimension to only two or three enables direct visualization and interpretation. DR techniques can be divided into techniques for *feature extraction* and *feature selection*. Feature selection techniques find an appropriate subset of the original variables to represent the data (see for example [Guyon and Elisseeff, 2003] for an introduction). In contrast, feature extraction builds new variables carrying a large part of the global information. The term *representation learning*, or *feature learning*, coins the automatic extraction of meaningful features from high-dimensional data, often by using non-linear transformations. In particular, it corresponds to DR when the number of extracted features is lower than the original data dimension.

Dimensionality reduction is a problem of approximating data in high-dimensional vector spaces [Gorban and Zinovyev, 2008]. There are varying degrees of "coarseness" of these approximations. The most trivial is to collapse the entire data into a single representative point, e.g. its mean. Then, data can be approximated more finely using projections on "lines and planes" (and generally hyperplanes). These methods are called linear dimensionality reduction, including: Principal Component

Analysis (PCA) [Pearson, 1901, Shlens, 2014] (principal components provide a sequence of best linear approximations of a data set), non-negative matrix factorization (NMF), random projections, compressed sensing, multi-dimensional scaling (MDS) [Kruskal, 1964], among others. Nevertheless, these methods suffer from the limitation of linear models. Non-linear DR [Lee and Verleysen, 2007] is also called *manifold learning* as it approximates data by a lower-dimensional manifold. Such methods include Locally Linear Embedding (LLE), Isomap, Spectral Embedding, Laplacian eigenmap, Kernel PCA, Sammon's mapping, Stochastic Neighbor Embedding (SNE) [Hinton and Roweis, 2002], the famous t-distributed SNE (t-SNE) [Van Der Maaten and Hinton, 2008] and its variants [Van Der Maaten, 2009, Lee et al., 2015], and Uniform Manifold Approximation (UMAP) [McInnes et al., 2018]. It also includes topology-preserving maps such as Self-Organizing Maps (SOM) [Kohonen, 1982, Kohonen, 1990, Martinetz and Schulten, 1994], that are a main topic of this thesis and will be introduced in details later.

The idea in SNE/t-SNE is to learn a low-dimensional embedding that preserves global and local structures of the data. Let us note $\mathbf{x}_i$ the inputs, and $\mathbf{z}_i$ the embeddings. The first step is to transform Euclidean distances into similarities in the input and output spaces, interpreted as (symmetrized) conditional probabilities, producing two distributions p_{ij} and q_{ij}. Then, the embeddings are optimized by minimizing the Kullback-Leibler (KL) divergence between the two distributions using gradient descent. In SNE, Gaussian distributions are used:

$$p_{ij} := \frac{p_{i|j} + p_{j|i}}{2N} \text{ where } p_{i|j} := \frac{\exp\left(-||\mathbf{x}_i - \mathbf{x}_j||_2^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-||\mathbf{x}_i - \mathbf{x}_k||_2^2 / 2\sigma_i^2\right)}$$
$$q_{ij} := \frac{\exp\left(-||\mathbf{z}_i - \mathbf{z}_j||_2^2\right)}{\sum_{k \neq l} \exp\left(-||\mathbf{z}_k - \mathbf{z}_l||_2^2\right)}$$

where the variances σ_i^2 are hyperparameters, selected by the user by tuning the perplexity. The variance in embedding space is fixed to $1/\sqrt{2}$. The loss function is the KL-divergence between p and q:

$$\mathcal{L}_{\text{SNE}} := D_{\text{KL}}(p||q) = \sum_{i=1}^{N} \sum_{j=1}^{N} p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

In t-SNE, a Student's t-distribution is used instead in the embedding space. Thanks to heavier tails, it solves the so-called *crowding problem* with moderate similarities between input points. It expresses as

$$q_{ij} := \frac{(1 + ||\mathbf{z}_i - \mathbf{z}_j)||^2)^{-1}}{\sum_{k \neq l} (1 + ||\mathbf{z}_k - \mathbf{z}_l)||^2)^{-1}}.$$

Quite differently, UMAP is based on algebraic topology, Riemannian geometry and fuzzy sets. In broad terms, it starts by building a weighted k-NN graph. Then, it lays out this graph in a low-dimensional space using a force-directed layout algorithm (attractive and repulsive forces are applied iteratively at each edge or vertex).

Another important tool for non-linear DR is neural networks. Learning effective representations using multi-layer neural networks is the goal of deep learning [Bengio, 2009, Lecun et al., 2015], whether it is supervised or unsupervised. Neural architectures for unsupervised representation learning, in particular autoencoders, are the focus of Section 2.1 of the next Chapter 2. Finally, an emerging field is topological data analysis (TDA) [Carlsson, 2009], combining techniques from algebraic topology and data mining.

1.2 Cluster analysis

Cluster analysis [Hartigan, 1975, Diday and Simon, 1976, Hastie et al., 2008, Jain, 2010, Aggarwal and Reddy, 2013, Shalev-Shwartz and Ben-David, 2013] is one of the most common tasks in unsupervised learning. It consists in finding meaningful groups (also called *clusters*) of individuals in an unlabeled data set. For exploratory data analysis, this allows to gain insight into the structure of a data set, and can also be used for classification and vector quantization, where data points are encoded using their cluster index. An object can be described by a set of features, or by its relation to other objects, such as a distance or affinity matrix. We will mainly adopt the first point of view.

1.2.1 How to define clustering?

Clustering is the task of grouping a set of objects in such a way that members of the same cluster are more similar to one another than to members of other clusters. Citing [Jain, 2010], "the goal of data clustering is to discover the *natural* grouping of a set of patterns points, or objects". The notion of *natural grouping* is ambiguous; therefore the author continues by saying that "the *representation* of the data is closely tied with the purpose of the grouping. The representation must go hand in hand with the end goal of the user". In more technical terms, it is defined in [Hastie et al., 2008] as "grouping or segmenting a collection of objects into subsets or clusters, such that those within each cluster are more closely related to one another than objects assigned to different clusters", and similarly in [Ben-David, 2018] as the

"partitioning of data into groups (a.k.a. clusters) so that similar (or close w.r.t. the underlying distance function) elements share the same cluster and the members of each cluster are all similar (or, equivalently, dissimilar elements are separated into different clusters)". However, this goal is contradictory because of the non-transitivity of the notion of similarity: if A is similar to B, and B is similar to C, A is not necessarily similar to C. Since clustering is an ill-posed problem, it cannot be properly solved using this definition, and clustering algorithms often optimize only one of its aspects. As a consequence, there is a huge variety of clustering algorithms, each one optimizing different criteria and producing different results on a same data set. Unlike in supervised learning (where a loss function is clearly defined), there is no *ground truth* to evaluate results. The lack of a definition of an optimal clustering makes it a fundamentally ill-defined task. The choice of the algorithm depends on properties of the data and computational considerations, but more importantly it depends on the targeted application, for instance the number, size and geometry of the clusters we want to find, as well as the robustness of the solution, among others. In other words, it depends on *what we want to do next* with the results. As stated in [von Luxburg et al., 2012], "the difficulty with unsupervised clustering is that there are a huge number of possibilities regarding what will be done with it and (as yet) no abstraction akin to a loss function which distills the end-user intent".

Although supervised learning is theoretically grounded by, for example, generalization bounds, clustering still lacks such a solid theory. Convergence and properties and generalization bounds for clustering are discussed in [Von Luxburg and Ben-David, 2005]. In particular, *stability bounds* have been derived for particular algorithms. Clustering stability is the subject of Chapter 5. Several works tried to build a set of axioms that should be satisfied by a *good* clustering algorithm [Kleinberg, 2003, Ackerman and Ben-David, 2009]. However, it has been shown that there exists no clustering algorithm that satisfies three intuitive properties (scale invariance, richness and consistency), leading to a famous "impossibility theorem" [Kleinberg, 2003]. Other axiomatic attempts have produced less negative outcomes. See [von Luxburg et al., 2012] or [Ben-David, 2018] for more details on the current deficiencies in clustering research.

For model selection, e.g. choosing the number of clusters (the main subject of Chapters 4 and 5), a large number of criteria exist to measure the quality of a clustering, and which criteria one should use also depends on the application.

1.2.2 Formal definition

Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ be a data set consisting in N independent and identically distributed (i.i.d.) samples, drawn from a data-generating distribution $\mathcal{P}$ on an underlying space $\mathcal{X}$. This space is equipped with a distance function $d : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}^+$, which is symmetric and satisfies $d(x, x) = 0 \ \forall x \in \mathcal{X}$ but not necessarily the triangle inequality (or alternatively, a *similarity function* $s : \mathcal{X} \times \mathcal{X} \longrightarrow [0, 1]$). A clustering algorithm $\mathcal{A}$ takes as input the data set $\mathbf{X}$ and outputs a partition $\mathcal{C}_K = \{C_1, \ldots, C_K\}$ of $\mathbf{X}$ into $K \geq 1$ disjoint sets. Thus, a clustering can be represented by a function $\mathbf{X} \to \{1, \ldots, K\}$ assigning a label to every point of the input data set. Some algorithms can be extended to construct a partition of the entire underlying space. This partition is represented by an *extension operator* function $\mathcal{X} \to \{1, \ldots, K\}$ (e.g. for center-based algorithms, we compute the distance to the nearest center).

We will see that some clusterings are not defined in terms of partitions but are natively represented as hierarchies, i.e. a sequence of nested partitions called a *dendrogram*. A partition is obtained by cutting the dendrogram at a level determined by some criterion such as the number of clusters. We also distinguish *hard* clustering , where clusters are non-overlapping, and *soft* or *fuzzy* clustering where a sample is assigned to every cluster with some weight or probability.

As distance of similarities are crucial to every clustering algorithm, literature reviews on cluster analysis typically include a lengthy list of definitions for distances between objects (real-valued, binary or categorical vectors, matrices, functions, curves, strings, continuous or discrete sequences, graphs, etc.). It will not be the case here, as most of our applications will use Euclidean distance

$$d_{\text{EUC}}(\mathbf{x}, \mathbf{y}) := ||\mathbf{x} - \mathbf{y}||_2 = \sqrt{\sum_{j=1}^{P} (\mathbf{x}^j - \mathbf{y}^j)^2} \tag{1.8}$$

excepted for an application to time series, to which a special chapter is dedicated (Chapter 6). Hence, we refer for instance to [Deza and Deza, 2009] for an extensive encyclopedia of distances.

1.2.3 Center-based methods

Center-based or *centroid*-based clustering algorithms are widely used class of clustering algorithms, part of so-called *prototype methods*. Prototype methods aim at

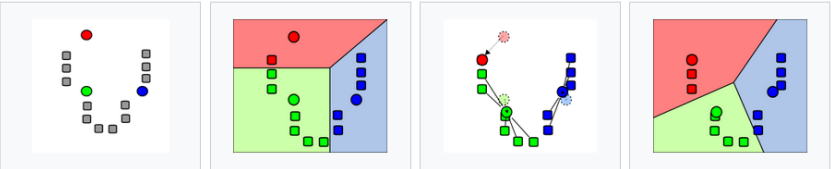


Fig. 1.3.: Illustration of one iteration step of the K-means algorithm.

representing a data set by a set of K points $\{\mathbf{m}_k\}_{1\leq k\leq K}$ in feature space called *prototypes*, which are typically not part of the training set. In center-based clustering, each cluster is summarized by such a center or prototype, and a sample is assigned to the cluster corresponding to the closest center w.r.t. the underlying distance.

K-means [Lloyd, 1982, MacQueen, 1967, Hartigan and Wong, 1979] is without a doubt the most famous and used center-based clustering algorithm. As its name suggests, it takes a parameter K corresponding to the number of clusters to find. The goal is to minimize following objective, called within-cluster sum of squared errors (WCSSE):

$$\min_{\mathcal{C}_K,\{\mathbf{m}_k\}_1^K} \sum_{k=1}^{K} |C_k| \sum_{\mathbf{x}\in C_k} ||\mathbf{x}-\mathbf{m}_k||_2^2 \tag{1.9}$$

This problem is proven to be actually NP-hard, and the terms K-means is mostly used to designate iterative algorithms finding local minima of this objective in polynomial time (Lloyd's algorithm [Lloyd, 1982]). It starts by initializing K centers (using random training samples or more advanced procedures to find a good initial solution for faster convergence, such as K-means++ [Arthur and Vassilvitskii, 2007]), and alternates between these two steps until convergence:

1. **Assignment**: each training sample is assigned to the closest center.

$$C_k \leftarrow \{\mathbf{x} \mid \operatorname*{argmin}_{k'=1,\ldots,K} ||\mathbf{x}-\mathbf{m}_{k'}||_2^2 = k\} \tag{1.10}$$

2. **Minimization**: each center is updated to become the means of the created clusters.

$$\mathbf{m}_k \leftarrow \frac{1}{|C_k|}\sum_{\mathbf{x}\in C_k} \mathbf{x} \tag{1.11}$$

The runtime complexity is $\mathcal{O}(KNI)$ where I is the number of iterations. See Figure 1.3 for an illustration. The standard K-means assumes squared Euclidean distance and real-valued, continuous features, because of the mean computation in the minimization step. It can be generalized to other types of variables by replacing this step. K-modes and K-prototypes are adapted for binary and mixed-type variables. When using arbitrary distances d, we can restrict the center to be

part of the training samples and search explicitly for the median element $\mathbf{m}_k = \underset{\mathbf{c}\in\mathbf{X}}{\operatorname{argmin}} \sum_{\mathbf{x}\in C_k} d(\mathbf{x}, \mathbf{c})$. This algorithm is called K-medoids or K-medians, and its complexity is quadratic due to the median search ($\mathcal{O}(KN^2I)$).

Fuzzy C-means (FCM) [Dunn, 1973, Bezdek et al., 1988] is a *soft* version of K-means, where each data point can be assigned to more than one cluster by introducing a membership function, taking values between 0 and 1. Limitations of K-means and its variants is the impossibility to separate non-convex clusters, highly non-Gaussian clusters, and to cope high-dimensional data with redundant or noisy features. In addition, the number of clusters K must be specified. Methods for selecting this parameter are the focus of Chapter 4 and Chapter 5.

1.2.4 Hierarchical methods

In contrast to K-means, hierarchical clustering methods do not need to specify a number of clusters, as they rather construct a binary tree of nested partitions, where the root is the entire data set and the leaves are singleton clusters for each sample. Each node corresponds to a cluster. The obtained sets of partitions are called *dendrogram* (from the Greek *dendron* = tree, *gram* = drawing), see the illustration on Figure 1.4. Suited for data sets that exhibit multiscale structure. Hierarchical methods are divided into two types:

- **Agglomerative** (bottom-up): starting from individual points, the pair of clusters with the lowest between-cluster dissimilarity is recursively merged.
- **Divisive** (top-down): starting from a single cluster, clusters are recursively split to produce two new clusters with the highest between-cluster dissimilarity.

To merge two clusters C, C' in agglomerative clustering, several between-cluster dissimilarities $d(C, C')$ are possible, called *linkage* criteria. The five most used ones are *single linkage*, *complete linkage*, *average linkage*, *centroid linkage*, and *Ward linkage* [Ward, 1963]. The corresponding distances are defined in Table 4.2 in Chapter 4. Hierarchical methods operate on distance matrices, thus they can be used with any distance or dissimilarity measure d, and with data sets described only through pairwise relations. The downside is that for data sets represented by attributes, it requires $N(N-1)/2$ distance computations.

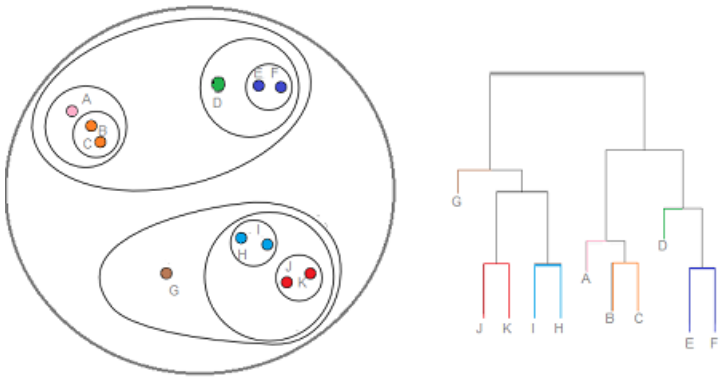


Fig. 1.4.: Illustration of a dendrogram produced by hierarchical clustering (right) and the corresponding cluster structure (left).

1.2.5 Spectral clustering

Spectral clustering [Von Luxburg, 2007] is a family of techniques based on the eigen-decomposition of the *graph Laplacian matrix* of the data. This matrix is constructed from a weighted or unweighted similarity graph, using for instance the fully connected affinity graph using a Gaussian kernel, an ε-neighborhood graph or a k-nearest neighbors graph. Spectral clustering proceeds by embedding data into the eigenspace of the graph Laplacian matrix, derived from the pairwise similarities between data points, and applying K-means on this representation to obtain the clusters. Compared to K-means, it is more powerful and able to discover non-convex clusters. However, the computation of the eigen-decomposition is costly for large data sets, and the spectral embedding cannot be generalized to points outside the training set in a straightforward way (*out-of-sample extension*).

1.2.6 Density-based methods

Density-based methods define clusters in terms of regions of high density, separated by regions of low density. The main advantage is the ability to discover clusters with arbitrary shapes. Its main representatives are DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [Ester et al., 1996] and OPTICS (Ordering Points To Identify the Clustering Structure) [Ankerst et al., 1999]. DBSCAN estimates the density by counting the number of points in a fixed-radius ρ neighborhood and connects two points if they lie within this radius. A point is called a core points if its ρ-neighborhood contains at least a minimum number of points (which is the second hyperparameter of DBSCAN, after ρ). Then, points are part of a cluster if they are in the ρ-neighborhood of a core point. Other points are considered as

noise. A main drawback is that DBSCAN cannot handle different densities in a data set (its parameters are global). OPTICS aims at solving this limitation. A general disadvantage is that these methods are limited to continuous Euclidean space.

1.2.7 Subspace clustering

The goal of subspace clustering [Parsons et al., 2004, Deng et al., 2016, Attaoui et al., 2020] is handling high-dimensional data by finding clusters within subspaces instead of the entire data space. Indeed in many real-world tasks, clusters may exist in different subspaces of the high-dimensional feature space. Subspace clustering can be classified into *hard* subspace clustering (HSC) and *soft* subspace clustering (SSC): the former determines the exact sub-spaces where the clusters are found, while the latter assigns weights to each feature. HSC methods can be further divided into *bottom-up* search methods (e.g. CLIQUE [Agrawal et al., 1998], MAFIA [Goil et al., 1999], ENCLUS [Cheng et al., 1999], SUBCLU [Kailing and Kriegel, 2004]...) that start with 1-dimensional clusters iteratively combined into higher-dimensional subspaces, and *top-down* methods (e.g. PROCLUS [Sembiring et al., 2010], ORCLUS...). SSC is strongly related to feature weighting clustering, and some representative methods are C-K-means [Modha and Spangler, 2002], W-K-means [Huang et al., 2005], Fuzzy Weighted K-means (FWKM) [Jing et al., 2005], or Fuzzy Subspace Clustering (FSC) [Gan et al., 2006], among many others. Finally, projected clustering is a term designating a similar set of techniques, more specifically combining a specific distance function and a standard clustering algorithm (e.g. DBSCAN or K-medoids).

1.2.8 Model-based clustering

Model-based clustering [Bock, 1996, McLachlan and Peel, 2000] tries to fit the data using probabilistic models. A popular approach is mixture modeling, using mixtures of parametric base probability density functions to represent clusters. It relies on the assumption that clustered data are drawn from one of several components, often represented by unobserved *latent* variables. For instance, Gaussian mixture models (GMM) is the probabilistic counterpart of K-means, where the data distribution is modeled using K Gaussian component distributions. Clustering then becomes an estimation task, often based on MLE and Expectation-Maximization (EM) procedures [Dempster et al., 1977]. Note that the K-means procedure is closely related to the

EM algorithm for estimating a particular Gaussian mixture model. Besides mixture modeling, topic models are another family of approaches.

1.2.9 Other clustering methods

Countless other algorithms exist and are omitted in this thesis: mean-shift, affinity propagation [Frey and Dueck, 2007], non-parametric methods such as support vector clustering, feature selection methods, grid-based clustering, swarm-based methods, etc. An important field, graph clustering, is also beyond our scope.

1.3 Self-organizing algorithms

This section introduces some of the main *self-organizing* clustering algorithms: Kohonen's Self-Organizing Map (SOM) [Kohonen, 1982, Kohonen, 1990, Cottrell et al., 2018], its probabilistic counterparts, as well as variants and extensions. This family of algorithms aims at clustering and visualizing high-dimensional data while preserving neighborhood properties, using a process called *self-organization*. They are also called *topographic map* or *topology-preserving* algorithms, because they preserve topographic properties between the input and output space, and their output often takes the form of a two-dimensional map. Formally, a topology-preserving algorithm is a transformation $\mathbb{R}^P \longrightarrow \mathbb{R}^L$, that preserves similarities, or similarity orderings, of the points in the input space when they are mapped into the output space [Martinetz and Schulten, 1994]. Self-organized models have been used for almost 40 years across various application domains such as biology, geology, healthcare, industry [Côme et al., 2011, Faure et al., 2017, Forest et al., 2020a] and humanities [Massoni et al., 2009]. The tasks they try to solve range from visualization, quantization to indexing and interactive image retrieval [Laaksonen et al., 2001], etc.

1.3.1 Kohonen's Self-Organizing Map

The Self-Organizing Map (SOM, sometimes written SOFM for Self-Organizing Feature Map), introduced by Finnish professor Teuvo Kohonen [Kohonen, 1982, Kohonen, 1990, Cottrell et al., 2018] is an unsupervised learning algorithm used for simultaneous clustering and visualization of high-dimensional data sets. The learning algorithm is a self-organization process, biologically inspired by the cortex

brain cells. This kind of learning is called *competitive learning*, opposed to *error-correction* learning used in feedforward neural networks. A *map* is a neural network of interconnected nodes, also called *cells*, *neurons* or *units*, organized as a grid (an undirected graph). The grid topology is most often two-dimensional and rectangular for visualization purpose but can have any dimension or topology. To each cell is associated a *prototype vector* belonging to the high-dimensional input space where the data lives. During an iterative learning process, the prototypes are updated to fit the training set; when a prototype is updated, the prototypes associated to neighboring cells are also updated using a certain weight. The weights decrease with the distance between cells on the grid. As a result, cells that are close on the map are associated to prototype vectors that are close in the input space. This allows the map to preserve the topology of the space. After convergence, the resulting map allows to efficiently visualize the high-dimensional input space on a low-dimensional map (e.g., the variations of different features or the distribution of data on the map). Due to its simplicity and interpretable results, SOM is a popular clustering and visualization tool.

The set of input data samples is denoted $\mathbf{X} = \{\mathbf{x}_i\}_{1 \leq i \leq N}, \mathbf{x}_i \in \mathbb{R}^P$. A SOM is composed of K units, associated to the set of prototype vectors $\{\mathbf{m}_k\}_{1 \leq k \leq K}$. In the standard SOM, the prototype vectors lie in the same space as the input data, i.e. $\mathbb{R}^P$. A data point is projected on the map by finding its closest prototype vector according to euclidean distance. The corresponding map unit is called the *best-matching unit* (BMU). We introduce the notation b_i for the BMU of $\mathbf{x}_i$:

$$b_i := \underset{k}{\operatorname{argmin}} ||\mathbf{x}_i - \mathbf{m}_k||_2^2 \tag{1.12}$$

The grid topology allows to define an inter-node distance $\delta(k, l)$, which is the topographic distance between units k and l on the map, here the Manhattan distance (the length of the shortest path on the map between the two units). We then define the neighborhood function of the SOM and a temperature parameter T, controlling the radius of the neighborhood around a unit. The function $\mathcal{K}^T$ is a function taking values between 0 and 1, with $\mathcal{K}^T(0) = 1$ and $\lim_{d \to \pm\infty} \mathcal{K}^T(d) = 0$. Common choices are a Gaussian or a rectangular window function centered around zero. This function is used to weight the updates of the prototype vectors: the highest weight of 1 is given to the central node, and the neighboring nodes receive a weight decreasing with their distance to the central node. Nodes far away will not be updated. The temperature parameter T defines the *size* of the neighborhood, i.e. the influence of a node on its neighbors. During learning, this parameter is decreased as in simulated annealing, so that all prototypes are moved on the first iterations, and only one vector at a time towards the end of the training. Concretely, when the temperature

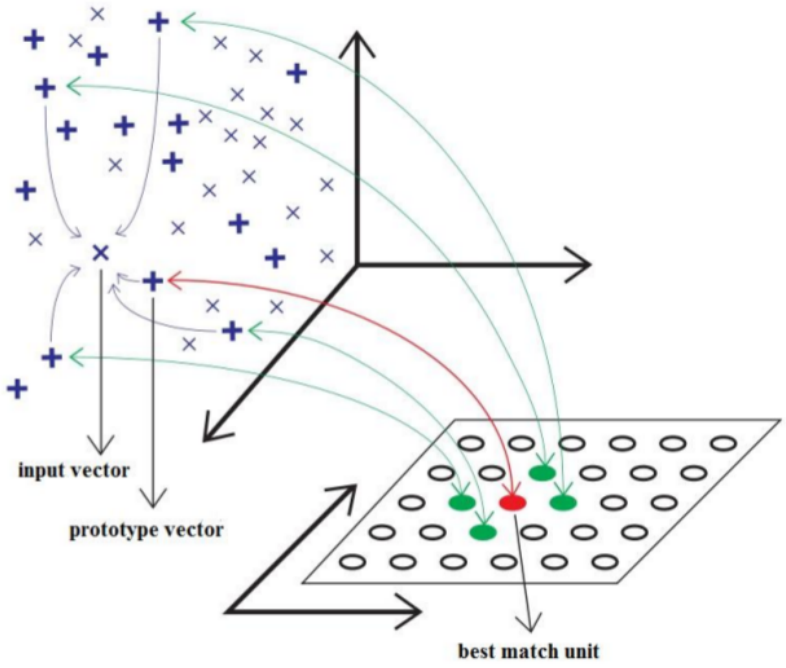


Fig. 1.5.: Illustration of the SOM quantization principle.

approaches zero, the neighborhood function becomes a function that equals 1 at 0 and 0 everywhere else. See Figure 1.5 for an illustration of the SOM projection from input space onto the map, and Figure 1.6 for examples of neighborhood functions. In this work, we will use a Gaussian neighborhood function, expressed as follows:

$$\mathcal{K}^T(d) := e^{-\frac{d^2}{T^2}}$$

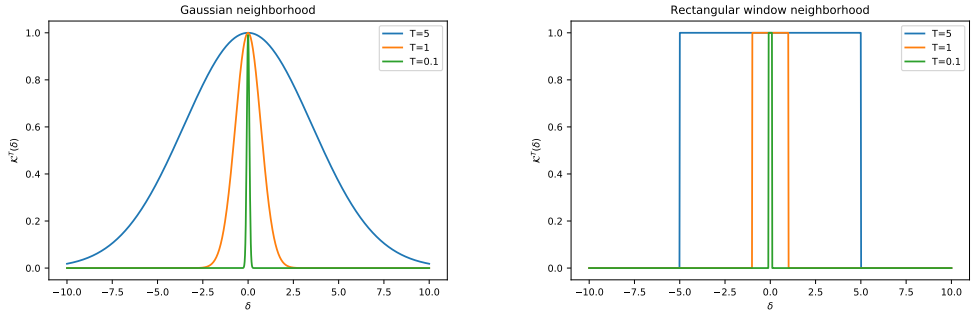


Fig. 1.6.: Gaussian and rectangular neighborhood functions for different temperatures.

The temperature T is decreased at each training iteration using a *decay function*. A common choice is exponential decay, starting from an initial temperature T_{max} towards a final temperature T_{min}, i.e. at iteration t:

$$T(t) := T_{max} \left(\frac{T_{min}}{T_{max}} \right)^{t/\text{iterations}}$$

The learning algorithm is a two-step iterative process consisting in:

1. An **assignment** phase where data points are assigned to a prototype.
2. A **minimization** phase where the prototypes are updated by moving them towards the data points.

The original SOM learning algorithm, also called *stochastic* or *Kohonen algorithm*, takes each training sample $\mathbf{x}_i$ individually and updates every prototype vector by moving them closer to the point $\mathbf{x}_i$. The updates are weighted by the neighborhood around the best-matching unit, so that neighboring units receive a large update and very distant units are not updated at all. This expresses as the following update rule:

$$\mathbf{m}_k \leftarrow \mathbf{m}_k + \alpha \mathcal{K}^T \left(\delta(b_i, k) \right) \left(\mathbf{x}_i - \mathbf{m}_k \right) \tag{1.13}$$

where α is a learning rate that is decreased during training. The stochastic algorithm is detailed in algorithm 1.1. A disadvantage of this algorithm is that it converges

Algorithm 1.1: Stochastic SOM algorithm.

Input: training set $\mathbf{X}$; SOM map size; temperatures T_{max}, T_{min}; $iterations$
Output: SOM code vectors $\{\mathbf{m}_k\}$
Initialize SOM parameters $\{\mathbf{m}_k\}$;
for $t = 1, \ldots, iterations$ **do**
    $T \leftarrow T_{max} \left(\frac{T_{min}}{T_{max}} \right)^{t/iterations}$;
    Load next training sample $\mathbf{x}_i$;
    Compute BMU b_i ;
    for $k = 1, \ldots, K$ **do**
        Update prototype $\mathbf{m}_k$ (by Equation 1.13) ;

slowly, is sequential and cannot be parallelized. Therefore, another algorithm was introduced: the batch SOM algorithm. It consists in minimizing following cost function, called *distortion*:

$$\mathcal{L}_{\text{SOM}}(\{\mathbf{m}_k\}, \mathbf{X}, b, T) := \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \mathcal{K}^T \left(\delta(b_i, k) \right) ||\mathbf{x}_i - \mathbf{m}_k||_2^2 \tag{1.14}$$

Distortion is not directly differentiable because of the BMU assignments b. However, it can be empirically minimized using a dynamic clusters method [Diday and Simon, 1976] (similar to K-means) which alternates between two steps:

1. Assignment of best-matching units using Equation 1.12
2. Minimization of distortion by fixing assignments, using following update rule:

$$\mathbf{m}_k \leftarrow \frac{\sum_{l=1}^{K} \mathcal{K}^T\left(\delta(k,l)\right) \sum_{i=1}^{N} \mathbb{1}_{[b_i=l]} \mathbf{x}_i}{\sum_{l=1}^{K} \mathcal{K}^T\left(\delta(k,l)\right) \sum_{i=1}^{N} \mathbb{1}_{[b_i=l]}} \tag{1.15}$$

The batch algorithm pseudo-code is detailed in algorithm 1.2. The two-step training

Algorithm 1.2: Batch SOM algorithm.

Input: training set $\mathbf{X}$; SOM map size; temperatures T_{max}, T_{min}; $iterations$
Output: SOM code vectors $\{\mathbf{m}_k\}$
Initialize SOM parameters $\{\mathbf{m}_k\}$;
for $t = 1, \ldots, iterations$ **do**
    $T \leftarrow T_{max} \left(\frac{T_{min}}{T_{max}}\right)^{t/iterations}$;
    Compute all BMUs $\{b_i\}_{i=1\ldots N}$;
    for $k = 1, \ldots, K$ **do**
        Update prototype $\mathbf{m}_k$ (by Equation 1.15) ;

procedure alternates between minimizing the distortion loss with respect to the BMU assignments (leaving the prototypes fixed) and minimizing the loss with respect to the prototype vectors $\{\mathbf{m}_k\}$ (with the assignments fixed). As we can see, the training procedure is the same as for the K-means algorithm, where we alternately update the cluster assignments and the centroids. Actually, SOM can be viewed as a *constrained* K-means in which the prototypes are encouraged to lie in a two-dimensional manifold, because of the constraint introduced by the neighborhood. When the temperature parameter approaches zero at the end of training, SOM becomes identical to K-means; this can be seen by taking the limit of the loss function 1.14 at 0:

$$\lim_{T \to 0} \sum_{i} \sum_{k=1}^{K} \mathcal{K}^T\left(\delta(b_i), k\right)\right) ||\mathbf{x}_i - \mathbf{m}_k||_2^2 = \sum_{k=1}^{K} \sum_{i, b_i = k} ||\mathbf{x}_i - \mathbf{m}_k||_2^2 \tag{1.16}$$

The distortion becomes identical to the K-means loss (see Equation 1.9).

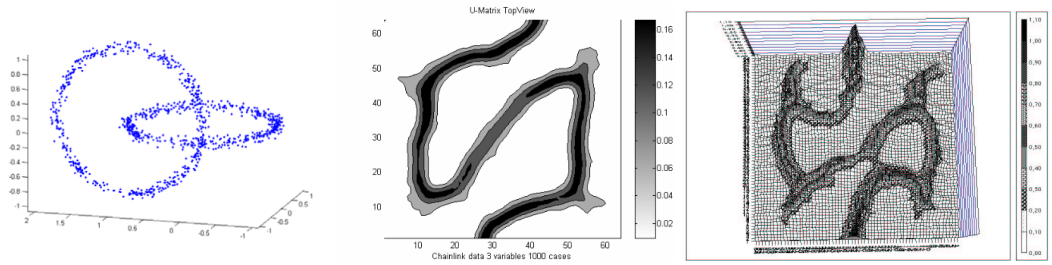


Fig. 1.7.: U-matrix visualizations of the chainlink data set. [Ultsch, 1999, Ultsch, 2003b]

1.3.2 Data visualization using SOM

The main purpose of the SOM is to visualize large high-dimensional data sets by visualizing the prototypes and other properties of the resulting map. There exist multiple visualization techniques, and this section will present some of the most used representations. An overview of several SOM-based visualization techniques is presented in [Vesanto, 1999]. The map cells can represent various quantities using color and shape variation. The most common visualization is the representation of the value of one feature (one dimension) of the map prototypes. This allows to visualize the variations of one feature across the input space, and identify different areas. It is also common to represent the distribution of the data on the map by representing the number of points assigned to each map unit (unit counts), by varying a marker size or the cell color. The curvilinear representation [Demartines and Blayo, 1992] allows to represent the unfolding of the grid on a 2D plane. [Cottrell and De Bodt, 1996] proposes a visualization combining distances to neighboring units and unit counts. SOM training can also be visualized using adaptive coordinates and cluster connection [Merkl and Rauber, 1997]. In [Kaski et al., 1998], a visualization of variable contribution to the cluster structure is proposed. The double SOM [Su and Chang, 2001, Ressom et al., 2003] and ViSOM allow to visualize the data set as a 2D scatter plot, learned during training. Other visualization techniques are based on distance or density matrices. The U-matrix [Ultsch, 1999] represents the average distance between the prototypes of each map unit and their neighboring units. Formally, for a given SOM unit k and its set of direct neighbors $\mathrm{NN}(k)$:

$$U(k) := \sum_{k' \in \mathrm{NN}(k)} ||\mathbf{m}_k - \mathbf{m}_{k'}||_2^2.$$

An example U-matrix is shown on Figure 1.7. It displays the "landscape" of the distance relationships in the input data space, where the "mountains" correspond to cluster boundaries, and "valleys" to cluster centers. Another matrix representation is the P-matrix [Ultsch, 2003a], which is based on density estimation (specifically,

Pareto Density Estimation). The U*-matrix [Ultsch, 2003b] combines local distance information from the U-matrix and local density information from the P-matrix, using following scaling:

$$U*(k) := U(k)\left(1 + \frac{P(k) - \bar{P}}{\bar{P} - \max(P)}\right).$$

It is able to better exhibit the cluster structures than the other matrices. More recently, the CONNvis [Taşdemir and Merényi, 2009] visualizes the data distribution using a connectivity matrix based on a weighted Delaunay triangulation. Finally, the prototype vectors can also be directly visualized after projection into 2 or 3 dimensions using classical DR techniques such as PCA or t-SNE.

1.3.3 Clustering of the SOM

The SOM clusters the data into a number of classes equal to the number of neurons in the map. This number is usually too large and we want to obtain a sensible number of classes for interpretation. Moreover, because of neighborhood relations, neighboring map units reflect the properties of the same clusters. To remove this overlap, a second clustering step is often used to group the SOM clusters into a reduced number of "super-clusters". This process can be achieved by three ways:

- Manually or semi-supervised: an expert may directly look at the prototypes and associate labels to them. This semi-supervised method then allows to classify any new sample by assigning it the class of its best matching unit, without having to label a large data set but only the prototypes. In the case when the data set is labeled or even partially labeled, classification can be performed by using the available labels after the SOM has been trained. For instance, the majority vote consists in assigning to each prototype the class that has the most training samples in its cluster.
- Visually: regions on the map can be detected by looking at visualization such as the U/P/U*-matrices.
- Automated clustering of the SOM: an second clustering algorithm detects the super-clusters automatically. Related works are presented in this paragraph.

To this end, hierarchical (SOM+HC) or K-means (SOM+KM) clustering of the prototypes are commonly used [Ambroise et al., 2000]. In particular, [Vesanto and Alhoniemi, 2000] showed that SOM is an efficient pre-processing step for clustering large data sets: by clustering the prototypes found by SOM instead of directly clustering the data, the computational load can be decreased considerably. The authors

investigated agglomerative clustering (single, average and complete linkage) and a partitive method (K-means). To select the number of clusters, the Davies-Bouldin index is used (see Chapter 4 for its definition). The SOM neighborhood relation can be used to constrain the possible merges in the construction of the dendrogram in agglomerative clustering. For example in [Murtagh, 1995], a contiguity constraint is added to merge only clusters corresponding to neighboring units on the map. Another solution is to exclude so-called *interpolating units* (i.e. units only connecting different regions of the manifold but corresponding to a very low data density) from the subsequent analysis. [Petersohn, 1998] proposes a two-step approach where the SOM is clustered using a second SOM, with a number of units equal to the number of target classes (this number is selected with a quality criterion based on inner-class homogeneity and heterogeneity between classes).

Many approaches to cluster the SOM are based on distances between prototypes [Vellido et al., 1999], improved in [Vesanto and Sulkava, 2002] with a hierarchical technique and pruning, and on the U/P/U*-matrices introduced previously. [Costa and Netto, 1999] uses a postprocessing of the U-matrix inspired by image segmentation techniques. In [Ultsch, 2003b], Ward clustering shows good results on the U*-matrix, compared with the U/P-matrices. The U*C algorithm [Ultsch, 2005] is based on the three matrices and finds automatically the number of super-clusters. It consists in an *immersion* step (gradient descent or ascent on the U/P-matrices to find distance/density modes) and an assignment step using *watershed* transformations on the U*-matrix. The Simultaneous Two-Level SOM clustering (S2L-SOM) [Cabanes and Bennani, 2007] simultaneously learns the SOM and neighborhood connections that automatically determine the set of final clusters. It obtains better results that SOM+HC or SOM+KM on several benchmark data sets. A variant, Density-based Simultaneous Two-Level SOM (DS2L-SOM) [Cabanes and Bennani, 2010], adds a refinement step based on density modes to determine a threshold matrix. [Azzag and Lebbah, 2008] propose a new similarity measure, consisting in weighting the Ward criterion with the topographic neighborhood on the map, to use with hierarchical clustering as well as the AntTree (artificial ants) algorithm [Azzag et al., 2003]. CONNvis [Taşdemir and Merényi, 2009] facilitates the extraction of clusters by an automatic thresholding method. Hierarchical clustering of the CONN matrix is proposed in [Taşdemir et al., 2011]. Finally, spectral clustering has also been used [Taşdemir, 2011].

1.3.4 Supervised SOM

Although it is mostly used in an unsupervised way, supervised methods to train the SOM have also been devised (i.e. for supervised classification), by taking advantage of a labeled data set. [Midenet and Grumbach, 1990, Idan and Chevalier, 1991] introduce a supervised learning method where the input vector is composed of the original features and the class information (e.g. a C-dimensional vector for classification with C classes), concatenated together. This enables the learning algorithm to learn prototypes that contain the class information, and a topology that is aware of this information. For prediction, the class features of the input vector are either replaced by an average value (calculated on the training set) or ignored when searching for the closest prototype; the predicted class is then derived from the class features of the best matching unit. Supervised SOM thus provides an alternative to backpropagation neural networks. The authors apply this technique on the MNIST handwritten digits recognition task.

1.3.5 Anomaly detection using SOM

The distance of unseen samples to the map can be used as an anomaly score [Harris, 1993]. Confidence intervals can be built globally or locally for each SOM cluster, as in [Bellas et al., 2014].

1.3.6 Extensions and probabilistic topographic maps

There are countless extensions and variants of the SOM and different topology-preserving mappings. They cannot be listed exhaustively here. In this paragraph, we will mention extensions to different variable types, several variants of the SOM and other self-organizing algorithms, and introduce probabilistic topographic maps that constitute an important family of algorithms.

Natively, the SOM is only adapted to continuous data with the Euclidean distance. Extensions to binary data have been developed, using for instance the Hamming distance [Lebbah et al., 1999], and for mixed data [Lebbah and Chazottes, 2005].

PCASOM [López-Rubio et al., 2004] is a SOM performing PCA (each neuron is associated with a local orthonormal basis), with the advantage of handling multimodal

distributions. It follows Kohonen's ASSOM (adaptive subspace SOM) algorithm [Kohonen, 1995]. Elastic graphs [Gorban and Zinovyev, 2008], in particular the elastic map, is another approach to learn data manifolds based on an energy function.

SOM uses a fixed, predefined network architecture (often a rectangular grid), whereas the choice of the map topology should depend on the input space for best results. However, it is not straightforward to find the best topology (number and arrangement of the map cells). [Dittenbach et al., 2000] proposed an architecture called the Growing Hierarchical SOM (GH-SOM) that builds a hierarchy of layers composed of incrementally growing SOMs. At each step, a SOM is trained in a traditional way. The map unit with the largest quantization error (deviation between its prototype vector and its assigned input data vectors) is called the *error unit*. A row or column of new cells is inserted between the error unit and its most dissimilar neighbor, making the SOM *grow* to represent more detail for this unit. Other approaches are Growing SOM (GSOM) and Dynamic SOM [Alahakoon et al., 2000]. The Neural Gas (NG) or Growing Neural Gas (GNG) model [Martinetz and Schulten, 1991, Fritzke, 1994, Fritzke, 1995] is an unsupervised learning model similar to SOM, the main differences being that NG learns not only the prototypes but also the connections between units (whereas SOM has a fixed topology) and that it uses the ranking order and not distance to update the prototypes. Because it organizes the prototypes in space and also creates or destroys unused connections between map units, the NG is called a *growing cell structure*. The model is thus able to automatically find a network structure.

Probabilistic topographic maps have the clear advantage of producing a probability density, allowing to handle missing values or even building mixtures of SOMs in a principled way. In addition, by changing the distribution is may handle any type of data (continuous, categorical or mixed). Variants of SOM based on mixture models and trained with EM algorithms are proposed in [Heskes, 2001, Verbeek et al., 2005]. The PrSOM (Probabilistic SOM), introduced by [Anouar et al., 1998], is a probabilistic topographic map model inspired by SOM and RBF (radial basis function) networks. It can be regarded as a mixture of *local* mixtures of Gaussians. These models have been applied and extended to binary data, e.g. BeSOM with Bernoulli distribution [Lebbah et al., 2007], and sequences of non-i.i.d. data, e.g. PrSOMS in [Jaziri et al., 2011, Lebbah et al., 2015].

The Generative Topographic Mapping (GTM) [Svensen et al., 1997, Bishop et al., 1998] is a probabilistic topographic map model for simultaneous clustering and visualization of high-dimensional data sets, but unlike Kohonen's model which uses a self-organization process, it performs a probability density estimation using an

EM procedure. GTM uses a probability density model where we assume that the data distribution, living in a P-dimensional space, is generated by variables from a L-dimensional latent space with $L < P$, using the non-linear mapping

$$\mathbf{y}(\mathbf{u}; \mathbf{W}) := \mathbf{W}\phi(\mathbf{u})$$

where $\phi = \{\phi_j\}, j = 1 \dots M$ is a set of M non-orthogonal basis functions and $\mathbf{W}$ is a $P \times M$ matrix of weight parameters. We know that this form can approximate any continuous mapping, given sufficiently many basis functions (universal approximation theorem for neural networks with one hidden layer [Csáji, 2001]); this number can grow exponentially with the dimension of $\mathbf{u}$ but as in most cases $L = 2$, it is not a limitation. In the standard model, the basis functions are radially symmetric Gaussians. Due to the smoothness of this transformation, points from the latent space will be mapped onto a L-dimensional manifold in data space. However, the data do not lie exactly on this manifold; thus, we define a posterior probability density decreasing with the distance to the manifold. Concretely, the posterior probability density consists in spherical Gaussian distributions centered on the transformed point, with a common inverse variance β:

$$p(\mathbf{x}|\mathbf{u}, \mathbf{W}, \beta) := \left(\frac{\beta}{2\pi}\right)^{P/2} \exp\left(-\frac{\beta}{2}||\mathbf{y}(\mathbf{u}; \mathbf{W}) - \mathbf{x}||^2\right).$$

The Gaussians can be regarded as noise distributions taking into account variance away from the manifold. It is also possible to introduce a manifold-aligned noise model, to take into account variance along the manifold directions, which can be quite different depending on the data distribution [Bishop et al., 1998]. This variant uses a more complex covariance matrix parameter instead of the isotropic inverse variance parameter. In order to obtain a latent variable model similar in spirit to the SOM, we introduce a regular grid of points $\{\mathbf{u}_i\}, i = 1 \dots K$ in latent space and define a prior distribution consisting in a superposition of delta distributions located at the nodes of the grid:

$$p(\mathbf{u}) := \frac{1}{K} \sum_{k=1}^{K} \delta(\mathbf{u} - \mathbf{u}_k).$$

The mappings of the grid points define a set of prototype vectors $\mathbf{m}_k$ in data space:

$$\mathbf{m}_k := \mathbf{y}(\mathbf{u}_i; \mathbf{W}).$$

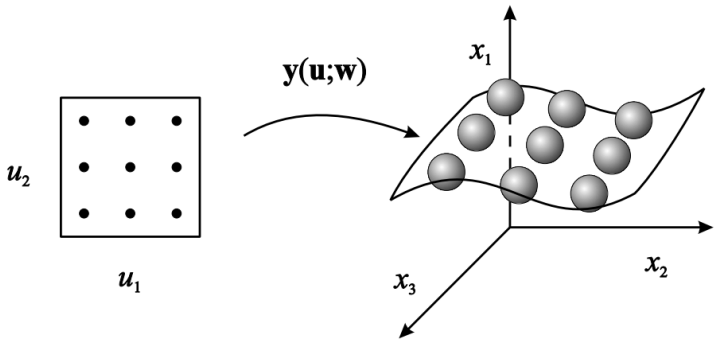


Fig. 1.8.: Illustration of the GTM mapping from latent to original space.

Finally, the probability density of the model is obtained by integrating over the latent variable:

$$p(\mathbf{x}|\mathbf{W},\beta) = \int p(\mathbf{x}|\mathbf{u},\mathbf{W},\beta)p(\mathbf{u})d\mathbf{u} = \sum_{k=1}^{K} \frac{1}{K}\left(\frac{\beta}{2\pi}\right)^{P/2} \exp\left(-\frac{\beta}{2}||\mathbf{m}_k - \mathbf{x}||^2\right).$$

As a consequence, the GTM can be regarded as a constrained mixture of Gaussians where the centers are constrained to lie within a L-dimensional manifold of data space, as illustrated in Figure 1.8. As in SOM, the resulting grid can be visualized or used for clustering, classification, anomaly detection or any other task. But unlike SOM, the topographic properties of the GTM are not due to neighborhood constraints on the grid nodes, but are a consequence of the smoothness of the mapping between latent space and feature space.

The GTM presented here is for continuous data, but can be naturally extended to discrete data. For binary data, we can introduce a Bernoulli distribution for each binary component, where the means are given by the same kind of mapping but adding a sigmoid function to obtain a probability. For categorical data with more than two classes, a multinomial distribution can be used instead, with a softmax activation. At least, if the data is a combination of continuous and discrete variables, the conditional distribution can be written as a product of Gaussian and multinomial distributions (assuming conditional independence of the observed variables given the latent), as explained in [Bishop et al., 1998].

Principled extension to discrete and mixed data is just one of the advantages provided by probabilistic maps compared to the SOM; other extensions include extensions for handling missing data, adaptive regularization through Bayesian inference [Bishop et al., 1998], hierarchical structures [Tino and Nabney, 2002], outlier detection [Bullen et al., 2003], and others. Because GTM can be expressed only in terms of dot products, it is also possible to use a kernel version of GTM [Olier et al., 2010].

1.3.7 Software implementations

Open-source software for SOM are available in various languages: the `SOM toolbox`[1] in Matlab [Alhoniemi et al., 1999], `SOMbrero`[2] in R [Boelaert et al., 2014], `SOMpy`[3] [Moosavi et al., 2014] or `minisom`[4] in Python, and in Scala with distributed versions using Spark[5] [Sarazin et al., 2014a, LIPN, 2018]. During this thesis, a new distributed implementation of batch SOM in Scala and Spark was implemented, `Spark ML SOM`[6] [Forest, 2019]. Unlike other software packages, it adopts a Spark ML and DataFrame API, making it more practical to use and integrate in modern Spark projects.

1.4 Conclusion

This chapter provided an introduction to statistical learning, with a focus on unsupervised learning, which is the learning setting adopted in this thesis to handle unlabeled data sets. We defined cluster analysis and presented the main algorithms, in particular center-based, hierarchical, spectral, density-based, subspace and model-based methods. Then, we tackled in depth a sub-family of clustering algorithms, namely self-organizing algorithms. Its main representative is Kohonen's Self-Organizing Map (SOM). In all cases, we supposed that observations are described by a set of features with an underlying cluster structure that can be discovered by the algorithms. This is not the case for complex, high-dimensional data, where a relevant representation needs to be extracted first. We have quickly introduced dimensionality reduction and manifold learning methods, but intentionally left neural network approaches for the coming chapter. The next chapter is dedicated to unsupervised neural networks to learn useful high-level representations and improve clustering.

[1] `https://github.com/ilarinieminen/SOM-Toolbox`
[2] `https://cran.r-project.org/package=SOMbrero`
[3] `https://github.com/sevamoo/SOMPY`
[4] `https://github.com/JustGlowing/minisom`
[5] `https://github.com/Clustering4Ever/Clustering4Ever`
[6] `https://github.com/FlorentF9/sparkmlsom`

2 Unsupervised representation learning for clustering

> *The key to artificial intelligence has always been the representation.*
>
> **— Jeff Hawkins**

A recent family of algorithms able to effectively extract high-level representations from complex, high-dimensional data are deep neural networks [LeCun et al., 1998, Bengio, 2009, Bengio et al., 2013, Lecun et al., 2015]. Their ability to learn useful hierarchical representations has been demonstrated extensively in the context of supervised learning, in particular in the fields of computer vision, natural language processing, speech recognition, among others. The type of data where these methods are especially effective are raw signals with a large number of dimensions with local correlation structures, such as spatial (images), sequential (text, audio, speech, time series), spatio-temporal (videos), or functional in general. A major motivation is that data show lots of variations but are actually living in a (very) lower-dimensional manifold [Alain and Bengio, 2014]. According to [Bengio, 2012], "deep learning algorithms seek to exploit the unknown structure in the input distribution in order to discover good representations, often at multiple levels, with higher-level learned features defined in terms of lower-level features. The objective is to make these higher-level representations more abstract, with their individual features more invariant to most of the variations that are typically present in the training distribution, while collectively preserving as much as possible of the information in the input. Ideally, we would like these representations to disentangle the unknown factors of variation that underlie the training distribution". This idea of a "good representation" and *disentanglement* will be explicited later.

While a lot of effort has been concentrated on supervised learning, where the network is trained to learn representations that are adapted for a specific supervised task (e.g., the layers of a convolutional network for image classification learn representations that help the top layer to classify the data, by mapping the data to a feature space where the target classes are easily separable), unsupervised deep learning methods have also been developed. The general goal of unsupervised representation

learning is to learn features from unlabeled data [Bengio, 2012, Bengio et al., 2013]. In general, the main motivation of unsupervised representation learning is to capture most information contained in the data while reducing the dimension or learning more robust representations, optimizing some unsupervised criterion. These representations are then used to solve downstream tasks, such as classification, regression, or clustering, leading to an infinite number of applications. The latter is the focus of this chapter.

The following section will introduce neural network architectures for unsupervised representation learning. One of the most famous is the *autoencoder* (AE), and we will present several of its variants. Then, we will shortly introduce *adversarial* methods. Finally, *self-supervised learning*, or *predictive learning*, designates unsupervised learning methods that learn representations by predicting a part of the input, given the rest of the input data. It recently raised interest in the fields of computer vision and natural language processing. It has not been used during this PhD but we think it is an essential part of modern unsupervised representation learning.

In either approaches, the goal is to improve some downstream application-oriented task. Thus, the representations need to be transferable and generalizable. One of the most desirable property is disentanglement [Locatello et al., 2020], i.e. to separate independent latent factors of variation. Other desiderata for a "good representation" are sparsity or smoothness. Another property is that objects that are semantically close, should be close in the latent feature space. This property is strongly connected to clustering but also *metric learning* (this link will be discussed later), and *multitask learning* (not further tackled, the reader can refer to [Caruana, 1997]). More specifically, we will investigate *deep clustering* approaches, where the latter property is enforced by some form of regularization based on a clustering loss, or a specific training strategy. In layman's words, latent embeddings corresponding to semantically similar elements are being "pushed together" to facilitate clustering in latent space. The actual clustering may happen as part of the process, or be performed as a subsequent step. Lastly, we consider deep self-organized models, a combination of topology-preserving and deep clustering approaches, and introduce related work necessary for the contribution of Chapter 3.

2.1 Unsupervised learning of representations

2.1.1 Autoencoders

Autoencoders, also sometimes called *autoassociators*, are neural networks trained to learn to reconstruct their inputs, in order to extract useful intermediate representations in an unsupervised way while minimizing information loss during this process [Hinton and Salakhutdinov, 2006, Bengio et al., 2007]. These representations or features can then be used to improve downstream tasks such as clustering or supervised learning, that benefit from dimensionality reduction and higher-level features. In other words, it is trying to learn an approximation to the identity function: this may seem trivial, but by placing various constraints on the network's architecture and activations, it will extract useful representations. In this section, we will introduce different AE variants and how to train them. The idea of autoencoders has been used for several decades, and can be seen as a non-linear equivalent to PCA. Unlike the latter, they are able to learn complex transformations of the input data. In addition, they can be trained using SGD, which has a linear complexity with the number of samples, unlike most non-linear DR techniques.

Autoencoders learn to map the input space $\mathcal{X}$ to a latent feature space $\mathcal{Z}$, and then reconstruct the original data, trying to match the input as closely as possible. The part of the network mapping the input to the latent internal representation is called *encoder* and the part of the network responsible for reconstructing the input is called the *decoder*. In a general setting, X and Z are respectively two random variables, and the encoder is a stochastic, parametric mapping $q_{\phi}(Z|X)$ from X to Z, represented by a neural network with parameters ϕ.

A MDL training principle is derived in [Hinton and Zemel, 1993]. Here, we will derive autoencoders through a slightly different approach called *infomax principle*. Under this principle, a "good" representation maximizes the *mutual information* between X and Z:

$$\begin{aligned}\mathbb{I}(X;Z) &= \mathbb{H}(X) - \mathbb{H}(X|Z) \\ &= C(X) + \mathbb{E}_{(X,Z)\sim q_{\phi}}\left[\log q_{\phi}(X|Z)\right]\end{aligned}$$

As $C(X)$ does not depend on Z, we only need to maximize the second term. We can see that the expectation involves $X|Z$ and not $Z|X$. By introducing the decoder

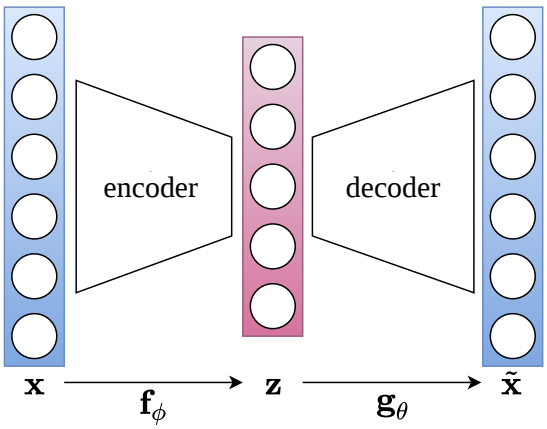


Fig. 2.1.: Deterministic autoencoder architecture.

$p_{\boldsymbol{\theta}}(X|Z)$, and using the fact that the Kullback-Leibler (KL) divergence $D_{\mathrm{KL}}(q||p)$ is always positive, we maximize the following lower bound:

$$\max_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathbb{E}_{(X,Z)\sim q_{\boldsymbol{\phi}}} \left[\log p_{\boldsymbol{\theta}}(X|Z)\right]. \tag{2.1}$$

Deterministic autoencoders

In a deterministic AE, the encoder can be represented by a function $\mathbf{f}_\phi : \mathcal{X} \longrightarrow \mathcal{Z}$, where ϕ are the parameters of a neural network. Similarly, the decoder is a function $\mathbf{g}_{\boldsymbol{\theta}} : \mathcal{Z} \longrightarrow \mathcal{X}$. An AE architecture is represented on Figure 2.1. In this case, the previous mappings become deterministic: $Z = \mathbf{f}_{\boldsymbol{\phi}}(X)$ (i.e. $q_{\boldsymbol{\phi}}(Z|X) = \delta(Z - \mathbf{f}_{\boldsymbol{\phi}}(X))$) and $\tilde{X} = \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{f}_{\boldsymbol{\phi}}(X))$. The objective in Eq (2.1) becomes

$$\max_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathbb{E}_{X\sim q_{\boldsymbol{\phi}}} \left[\log p_{\boldsymbol{\theta}}(X|Z = \mathbf{f}_{\boldsymbol{\phi}}(X))\right],$$

and using an empirical mean estimate over a training set $\{x_1, \ldots, x_N\}$, we obtain

$$\max_{\boldsymbol{\phi},\boldsymbol{\theta}} \sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}|\mathbf{z}^{(i)} = \mathbf{f}_{\boldsymbol{\phi}}(\mathbf{x}^{(i)}))$$

which is equivalent to

$$\max_{\boldsymbol{\phi},\boldsymbol{\theta}} \sum_{i=1}^{N} \log p(\mathbf{x}^{(i)}|\tilde{\mathbf{x}}^{(i)} = \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{f}_{\boldsymbol{\phi}}(\mathbf{x}^{(i)}))).$$

This last expression can be turned into a minimization of the negative sum of individual log-losses $\mathcal{L}(\mathbf{x}, \tilde{\mathbf{x}}) := -\log p(\mathbf{x}|\tilde{\mathbf{x}})$.

In standard autoencoders, the reconstruction $\tilde{\mathbf{x}}$ is meant to be the *mean* of a distribution that may have generated $\mathbf{x}$. In case of continuous variables ($\mathbf{x} \in \mathbb{R}^P$), the most common choice is a Gaussian distribution: $X|\tilde{X} = \tilde{\mathbf{x}} \sim \mathcal{N}(\tilde{\mathbf{x}}, \sigma^2\mathbf{I})$. This leads to the mean squared error (MSE) loss:

$$\mathcal{L}_{\text{MSE}}(\mathbf{x}, \tilde{\mathbf{x}}) := ||\mathbf{x} - \tilde{\mathbf{x}}||_2^2.$$

For binary variables or variables that can be expressed as a probability ($\mathbf{x} \in \{0,1\}^P$ or $\mathbf{x} \in [0,1]^d$), the common choice is a Bernoulli distribution: $X|\tilde{X} = \tilde{\mathbf{x}} \sim \mathcal{B}(\tilde{\mathbf{x}})$. This leads to the *binary cross-entropy loss*:

$$\mathcal{L}_{\text{BCE}}(\mathbf{x}, \tilde{\mathbf{x}}) := -\sum_{j=1}^{P}[\mathbf{x}^j \log \tilde{\mathbf{x}}^j + (1 - \mathbf{x}^j) \log(1 - \tilde{\mathbf{x}}^j)]. \tag{2.2}$$

Similarly, the categorical cross-entropy loss handles categorical variables. The previous loss functions are called *reconstruction error* as they measure the discrepancy between the input $\mathbf{x}$ and its reconstruction $\tilde{\mathbf{x}}$.

Links with Principal Component Analysis

The ability of artificial neurons to learn principal components has been established a long time ago. Consider a simple artificial neuron with input $\mathbf{x} \in \mathbb{R}^P$, weights $\mathbf{w} \in \mathbb{R}^P$, with a linear output activation equal to $y = \sum_{j=1}^{P} \mathbf{w}^j \mathbf{x}^j = \mathbf{w}^T\mathbf{x}$. Hebb's learning rule [Hebb, 1949] states that weights increase if input and output are correlated, i.e.

$$\Delta\mathbf{w} = \alpha\mathbf{x}y = \alpha\mathbf{x}\mathbf{x}^T\mathbf{w},$$

where α is a learning rate. See for example the *perceptron* [Rosenblatt, 1958] for an illustration of this learning rule. A problem with Hebb's rule is that weights can "explode". A solution is to add a *forgetting term*, as in Oja's learning rule [Oja, 1982]:

$$\Delta\mathbf{w} = \alpha(\mathbf{x}y - y^2\mathbf{w}) = \alpha(\mathbf{x}\mathbf{x}^T\mathbf{w} - \mathbf{w}^T\mathbf{x}\mathbf{x}^T\mathbf{w}\mathbf{w}).$$

We can show that this rule leads to learn *principal components* [Becker, 1991, Oja, 1992]. After convergence, we have

$$\begin{aligned}
\mathbb{E}[\Delta\mathbf{w}] = 0 &\iff \alpha(\mathbb{E}\left[\mathbf{x}\mathbf{x}^T\right]\mathbf{w} - \mathbf{w}^T\mathbb{E}\left[\mathbf{x}\mathbf{x}^T\right]\mathbf{w}\mathbf{w}) = 0 \\
&\iff \alpha(\mathbf{C}\mathbf{w} - \mathbf{w}^T\mathbf{C}\mathbf{w}\mathbf{w}) = 0 \\
&\iff \mathbf{C}\mathbf{w} = \mathbf{w}^T\mathbf{C}\mathbf{w}\mathbf{w} = \lambda\mathbf{w}
\end{aligned}$$

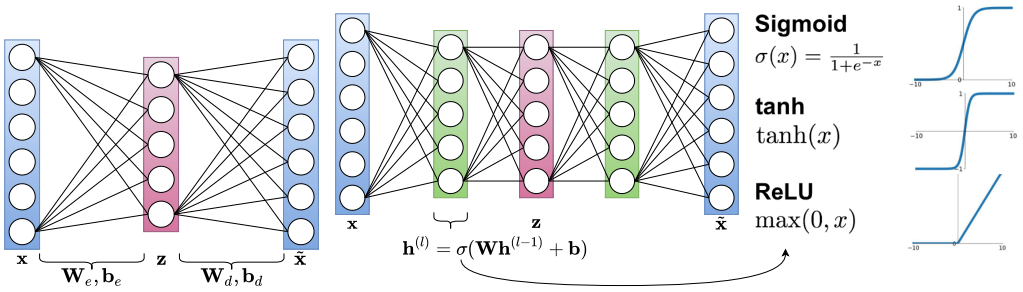


Fig. 2.2.: Linear autoencoder (left) and deep autoencoder with activation functions (right).

where $\mathbf{C} = \mathbb{E}\left[\mathbf{x}\mathbf{x}^T\right]$ is the covariance matrix. Thus, $\mathbf{w}$ is an eigenvector of the covariance matrix, i.e. a principal component.

Actually, a linear AE (where encoder and decoder are simply matrix products without any non-linear activation function, shown on the left side of Figure 2.2) trained with MSE reconstruction error is equivalent to PCA, without the orthogonality constraint. The reconstruction is simply:

$$\tilde{\mathbf{x}} = \mathbf{W}_d\left(\mathbf{W}_e\mathbf{x} + \mathbf{b}_e\right) + \mathbf{b}_d.$$

Ignoring the biases, the MSE loss becomes

$$\mathcal{L}_{\text{MSE}} = \sum_{i=1}^{N} ||\mathbf{x}_i - \tilde{\mathbf{x}}_i||_2^2 = \sum_{i=1}^{N} ||\mathbf{x}_i - \mathbf{W}_d\mathbf{W}_e\mathbf{x}_i||_2^2 = ||\mathbf{X} - \mathbf{X}\mathbf{W}_d\mathbf{W}_e||_F^2.$$

The links between AEs and SVD or PCA are further studied in [Bourlard and Kamp, 1988] and [Plaut, 2018].

Regularized autoencoders

For now, nothing prevents the encoder to simply learn the identity function, although the goal is to learn *good representations* of our data. For this reason, AEs are always trained with some type of regularization to reduce the size of the hypothesis space [Alain and Bengio, 2014].

The most basic form of regularization is to use a intermediate feature space with a dimension lower than the input, forcing the model to learn an efficient code with fewer parameters. An AE where the internal layer has a smaller number of units than the input is called an *under-complete* AE and maps the input space to a lower-dimensional feature space, thus achieving (non-linear) dimensionality reduction.

When the internal layer has more units than the input layer, an AE is called *over-complete*. Over-complete AEs can potentially learn the identity function. To prevent such a trivial solution, they must be trained not only to optimize a reconstruction error, but with additional constraints on the *structure* of the latent space.

An example of structural constraint is *sparsity*: *sparse autoencoders* [Ng, 2011, Makhzani and Frey, 2013, Arpit et al., 2016] also achieve data compression by using a potentially larger number of units in the hidden representation layer, but where only a small number units are active at the same time. Links between sparse representations and the functioning of the visual cortex V1 is discussed in [Olshausen and Fieldt, 1997]. Sparsity can be achieved by using, for instance, KL-divergence [Ng, 2011] or ℓ_1 [Arpit et al., 2016] penalties. Penalizing the weights amplitude is called weight decay and is a common form of regularization (ℓ_2 weight decay is also possible). Quite differently, *contractive autoencoders* [Rifai et al., 2011] use the squared Frobenius norm of the Jacobian matrix of the encoder as a penalty. In case of a linear AE, it becomes equivalent to ℓ_2 weight decay.

Denoising autoencoders (DAEs) [Vincent et al., 2010] learn to reconstruct a corrupted version of the input: if $\mathbf{x}$ is the input vector, denoising AEs try to minimize the loss $\mathcal{L}(\mathbf{x}, \tilde{\mathbf{x}}')$ where $\mathbf{x}'$ is the result of a stochastic corruption process $\mathbf{x} \longrightarrow \mathbf{x}'$ such as additive noise or random drop-out. By learning to remove the noise and recover the original data by removing the noise (hence the *denoising* term), the network actually learns robust representations and extracts useful features of the input distribution. Concretely, a DAE is a two-layer network defined as

$$\mathbf{x}' = \text{corrupt}(\mathbf{x}) \qquad \mathbf{h}' = \text{corrupt}(\mathbf{h})$$
$$\mathbf{h} = a_1(\mathbf{W}_1\mathbf{x}' + \mathbf{b}_1) \qquad \mathbf{y} = a_2(\mathbf{W}_2\mathbf{h}' + \mathbf{b}_2)$$

where a_1, a_2 are the activation functions, $\mathbf{W}_1$, $\mathbf{b}_1$ and $\mathbf{W}_2$, $\mathbf{b}_2$ are the weight and bias of the encoder and decoder layers, and corrupt is the corruption process.

For a long time, supervised deep neural networks were often pretrained using so-called *unsupervised greedy layer-wise pretraining* using *stacked denoising autoencoders* (SDAE) [Hinton and Salakhutdinov, 2006, Bengio et al., 2007, Vincent et al., 2010, Erhan et al., 2010], in order to initialize each layer with a good solution. It consists in stacking several DAEs by using the output of each AE as the input of the next AE. The SDAE is first pretrained using greedy layer-wise training. Then, all encoder layers followed by all decoder layers are concatenated in reverse layer-wise training order, forming the deep SDAE. Then, the SDAE is finetuned on reconstruction error. Finally, the stacked encoder layers form the encoder part of the deep AE, and the stacked decoder layers form the decoder part. However, the necessity of this

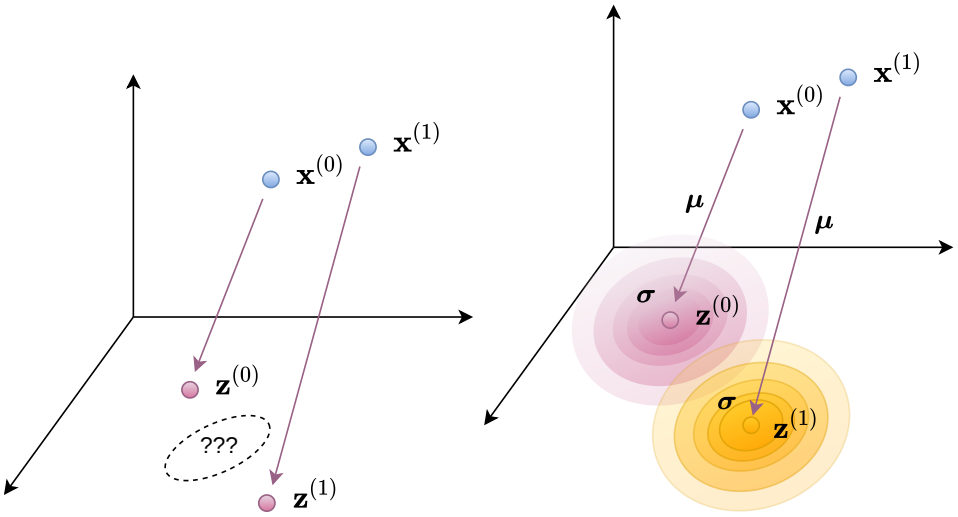


Fig. 2.3.: Visual illustration of the latent space structure for a vanilla autoencoder (left) and a variational autoencoder with Gaussian prior (right).

pretraining is being doubted nowadays [Zhou et al., 2015]. Thanks to improvements in network architecture (e.g. ReLU activations) and optimization techniques (e.g. SGD with momentum [Sutskever et al., 2013, Kingma and Ba, 2015]), even very deep networks can be trained from scratch with random initialization.

Variational autoencoders

The standard under-complete AE has still limitations. First, the latent space has no structure and may not be continuous (i.e. two samples that are close in input space are not necessarily close in latent space, and vice-versa). More importantly, it may *overfit*, meaning that it would not generalize well to unseen data samples. In theory, even a single continuous latent variable can memorize the entire training set by using one real number per sample. Thirdly, we cannot *explore* nor *sample* points from the latent space. This motivated the use of *variational autoencoders* (VAE) [Kingma and Welling, 2014, Rezende et al., 2014, Kingma and Welling, 2019], which are deep latent-variable probabilistic models. The VAE was originally inspired from the Helmholtz Machine (Dayan et al., 1995), however its wake-sleep training algorithm was inefficient. Instead of directly producing a latent code $\mathbf{z}$, the VAE outputs a probabilistic output, for instance a mean $\boldsymbol{\mu}$ and a standard deviation $\boldsymbol{\sigma}$. See Figure 2.3 for a visual illustration.

In contrast to the deterministic AE, the decoder is now a *generative model* $p_{\boldsymbol{\theta}}(X, Z)$, where the latent code Z is sampled from a *prior distribution* $p_{\boldsymbol{\theta}}(Z)$, and data samples are generated with $p_{\boldsymbol{\theta}}(X|Z)$, called the *likelihood*. The probabilistic encoder $q_{\boldsymbol{\phi}}(Z|X)$

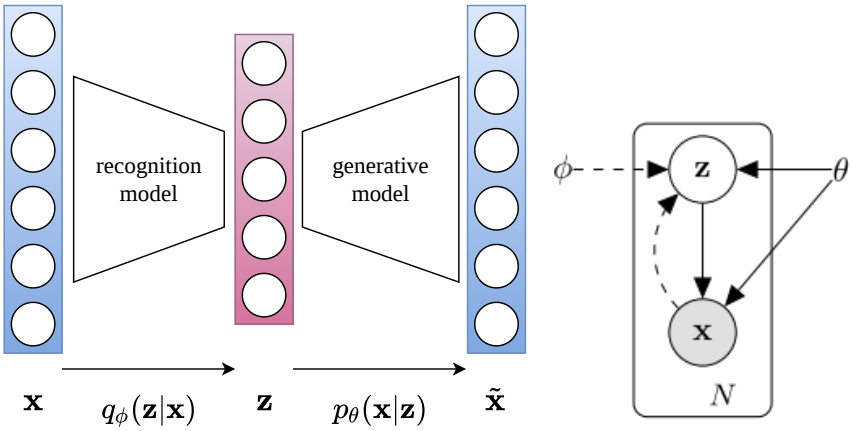


Fig. 2.4.: Variational autoencoder architecture (left) and graphical representation of the generative model (right).

is called *inference* or *recognition model*. Its role is to approximate the *posterior* $p_{\boldsymbol{\theta}}(Z|X)$, which is intractable. This process is called *variational inference*. The model is represented on Figure 2.4.

The objective is to estimate the parameters $\boldsymbol{\theta}$ and the variational parameters $\boldsymbol{\phi}$ using maximum likelihood estimation. The log-likelihood of an input $\mathbf{x}$ decomposes as follows (where $p_{\boldsymbol{\theta}}(\mathbf{x})$ is a shortcut notation for $p_{\boldsymbol{\theta}}(X = \mathbf{x})$, and similarly for every random variable):

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right] \\
&= \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right]}_{\mathcal{L}_{\text{ELBO}}(\mathbf{x};\,\boldsymbol{\theta},\boldsymbol{\phi})} + \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right]}_{D_{\text{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))} \\
&\geq \mathcal{L}_{\text{ELBO}}(\mathbf{x};\,\boldsymbol{\theta},\boldsymbol{\phi})
\end{aligned}$$

The right-hand side term is the KL divergence between $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, corresponding to the error made by approximating the true posterior. As it is always positive, we obtain a lower bound on the marginal log-likelihood called the *evidence lower bound (ELBO)*. Note that the KL divergence also determines the tightness of this bound. The VAE tries to maximize this lower bound, yielding the objective

$$\max_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathcal{L}_{\text{ELBO}}(\mathbf{x};\,\boldsymbol{\theta},\boldsymbol{\phi})$$

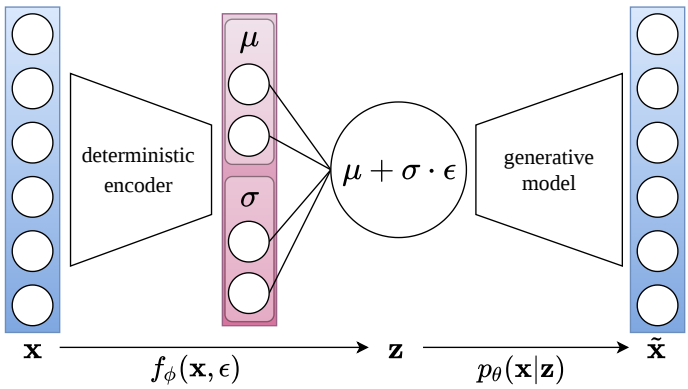


Fig. 2.5.: Variational autoencoder with reparameterization trick for a Gaussian prior.

that can be rewritten as follows (decomposing $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) = p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{z})$):

$$\mathcal{L}_{\text{ELBO}}(\mathbf{x};\ \boldsymbol{\theta}, \boldsymbol{\phi}) := \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] - D_{\text{KL}}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z})\right). \tag{2.3}$$

The left-hand side term is a reconstruction error, as in the standard AE (see Eq 2.1). The second term, a KL divergence between $q_\phi(Z|X)$ and $p_{\boldsymbol{\theta}}(Z)$, acts as a *regularizer* pushing the encoder distribution closer to the prior distribution. The typical choice for the prior is a Gaussian with zero mean and unit variance. In the rest of this section, we will derive only the fully Gaussian case where the posterior is also Gaussian, i.e.:

- $p_{\boldsymbol{\theta}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$
- $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I})$

Note that a reason to choose $p(\mathbf{z}) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ as the prior is to obtain conditionally independent latent variables from the recognition model, corresponding to disentangled factors of variation in the data.

The *reparameterization trick* is a technique allowing to sample while keeping operations differentiable to allow end-to-end optimization by backpropagation. Citing [Kingma and Welling, 2019], "its learning algorithm is a mix of classical (amortized, variational) expectation maximization but through the reparameterization trick ends up backpropagating through the many layers of the deep neural networks embedded inside of it.". The basic idea is to replace the stochastic $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ with a deterministic function $f_{\boldsymbol{\phi}}(\mathbf{x}, \epsilon)$, where the only stochastic part is introduced by and independent variable ϵ. In the Gaussian case, we use the reparameterization $\mathbf{z} = f_{\boldsymbol{\phi}}(\mathbf{x}, \epsilon) = \boldsymbol{\mu} + \boldsymbol{\sigma} \cdot \boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. See Figure 2.5 for an illustration.

Stochastic Gradient Variational Bayes (SGVB) [Kingma and Welling, 2014] is an efficient estimation method in case of intractable marginal likelihood/posterior

and large data sets. It replaces the expectation with a one-sample Monte-Carlo estimation, combined with SGD optimization. Finally, the ELBO for a given input $\mathbf{x}$ becomes

$$\mathcal{L}_{\text{ELBO}}(\mathbf{x};\ \boldsymbol{\theta}, \boldsymbol{\phi}) = \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) + \frac{1}{2}\sum_j \left(1 + \log(\boldsymbol{\sigma}^j)^2 - (\boldsymbol{\mu}^j)^2 - (\boldsymbol{\sigma}^j)^2\right).$$

The reconstruction term, as previously, in practice turns into the MSE for Gaussian outputs.

The relative weight between the two terms of the VAE loss, reconstruction and KL-divergence, is extremely important and rarely highlighted in basic tutorials. If the reconstruction's weight is too high, reconstruction quality will be good but the latent space structure will be weak. Conversely, a KL weight too high will produce good latent space structure but bad reconstruction quality (which is easier to spot). [Bowman et al., 2016] uses a weight annealing of the KL term during training. In the β-VAE [Higgins et al., 2017], a tunable β hyperparameter weights between both terms. [Sønderby et al., 2016] observed that "batch normalization and deterministic warm-up (gradually turning on the KL-term) are crucial for training variational models with many stochastic layers."

The VAE introduced here has a single level of latent variable $\mathbf{z}$, but more complex models with multiple stochastic layers and latent variables $\mathbf{z}_1, \ldots, \mathbf{z}_L$ are possible, see for instance the Ladder VAE [Sønderby et al., 2016]. Other methods using multiple stochastic variables are Importance Weighted Autoencoders, Normalizing Flows, Inverse Autoregressive Flows, Variational Gaussian Processes, or Auxiliary Deep Generative Models, but are out of this scope. Finally, recent works propose to improve the latent space structure and disentanglement (see for instance InfoVAE [Zhao et al., 2017], β-TCVAE [Chen et al., 2018], FactorVAE [Kim and Mnih, 2018], π-VAE [Mishra et al., 2020], etc.).

2.1.2 Adversarial methods

Adversarial learning methods have become increasingly popular, although such methods were not used during this PhD, we will mention them briefly. We speak of adversarial training when the optimization involves two components with competing objectives, and the goal is to reach an equilibrium similar to a Nash equilibrium in game theory. The most popular example is generative adversarial networks (GAN) [Goodfellow et al., 2014], consisting in a generative model, the *generator*, and a classifier, the *discriminator*. The training combines two objectives: on one hand,

the discriminator must detect if the output of the generator is real (i.e. part of the training set) or fake (i.e. artificially generated by the generator); on the other hand the generator must generate realistic samples to deceive the discriminator. This paradigm has lead to impressive results in image generation. In a standard GAN, the input of the generator is random noise; extensions are able to better control its output, e.g. conditional GAN [Mirza and Osindero, 2014] and InfoGAN [Chen et al., 2016]. Adversarial autoencoders (AAE) [Makhzani et al., 2014] are similar to VAE but use adversarial training to match the posterior to the prior distribution. We will see a few clustering methods based on this paradigm.

Both the VAE and GAN are part of the family of generative models: the VAE decoder or the generator can be used to generate new data samples. In deterministic AEs, it is possible (but not in a principled way) to adding noise, interpolate or extrapolate in latent space [Devries and Taylor, 2017].

2.1.3 Self-supervised or predictive learning

In self-supervised learning, high-level representations are learned by predicting a hidden part of the input, given the rest of the input data, called the *context*. Therefore, it most often consists in generic *pretext tasks* or *auxiliary tasks* (as opposed to the application-oriented downstream tasks) of *context prediction*. It takes its roots in natural language processing, where context prediction has allowed to learn *word embeddings*, i.e. dense, distributed vector representations of words or documents. The principle of word2vec [Mikolov et al., 2013] is to consider each word surrounded by a certain number of its preceding and following words (the context). Then, it predicts either the word given its context (CBOW variant) or the context given the target word (skip-gram variant). More recently, the same principle is being used in transformer-based embedding networks such as BERT, GPT, etc. Then, these embeddings facilitate downstream application tasks such as named entity recognition, sentiment analysis, chatbots, etc. In computer vision, it has been successfully applied to image classification and clustering, object detection, semantic segmentation, photo restoration, image super-resolution, etc. Examples of auxiliary tasks in computer vision include predicting the relative positioning of objects in an image [Doersch et al., 2015], classifying image rotations [Gidaris et al., 2018], colorizing images [Zhang et al., 2016], solving Jigsaw puzzles [Noroozi and Favaro, 2017] (see Figure 2.6), inferring the order of frames inside a video [Misra et al., 2016], among others. Some approaches are also related to metric learning (i.e. learning a similarity between objects) such as [Wang and Gupta, 2015] where authors use successive frames of videos as a supervisory signal. Finally, in a different

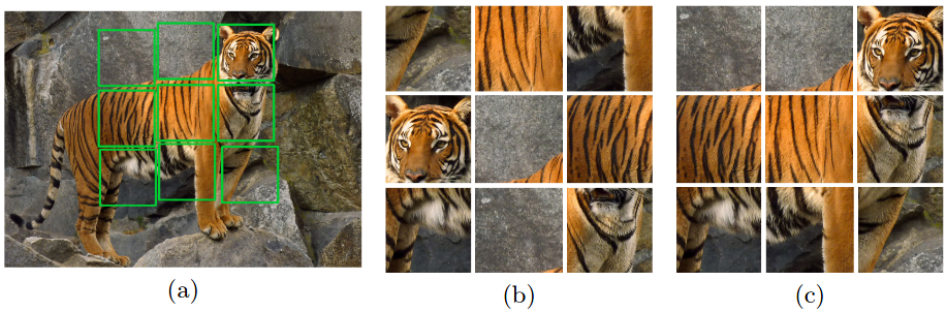


Fig. 2.6.: Solving jigsaw puzzles can be used as a pretext task to learn image representations. Patches are extracted from an image (a), shuffled (b), and the task is to retrieve their relative positions (c). [Noroozi and Favaro, 2017]

field, [Jawed et al., 2020] uses time series forecasting as a pretext task for time series classification with a CNN.

2.2 Learning representations for data clustering

The performance of clustering algorithms depends highly on the distribution of the data, the similarity measures and how well the distribution can be separated. In the previous chapter, we have seen various clustering algorithms, that can be broadly divided into three categories:

- Algorithms operating on the raw features.
- Those operating on a subset of raw features. This includes subspace clustering.
- Those operating on a transformation of the original features.

For complex data, it is difficult to select similarity measures, and distance metrics such as Euclidean become ineffective in high dimensions (curse of dimensionality). High-dimensional data usually live in different low-dimensional subspaces hidden in the original space, makes the first category ineffective. Whenever this subspace is actually a subset of the original variables, the second category of methods proves effective. However, often the original variables describing the data are not adapted for direct clustering, and structure is hidden in *transformations* of these variables. Transformations are required to map from the original space where the data reside to a different, more *clustering-friendly* (a term coined by [Yang et al., 2017a]), feature space. Often, we want this feature space to be *flat* and Euclidean, in order to use standard clustering algorithms based on Euclidean distance. Ideally, we want the clusters to be linearly separable in this space. For this reason, we are interested in the last of these three categories. For example, K-means or GMM with distance metrics

limited to the original data space are poorly adapted for clustering high-dimensional data with a lot of variations. Some challenges of complex and high-dimensional data clustering are exposed in the first paragraph. To solve them, an intuitive solution is to first reduce the dimension as a preprocessing step (while minimizing information loss) and then cluster the data in a low-dimensional space. This can be achieved by using dimensionality reduction techniques introduced in Paragraph 1.1.4 of previous chapter and Section 2.1. In this two-stage approach, one

1. Optimizes a pure information loss criterion between data points and their low-dimensional representations (this generally takes the form of a reconstruction loss between a data point and its reconstruction, e.g. mean squared error).
2. Optimizes a pure clustering criterion using some clustering algorithm (e.g. K-means quantization error).

For instance, autoencoders can be used as a preprocessing tool to improve the performance of standard clustering algorithms that struggle with complex and high-dimensional data: the performance of K-means on various data sets is greatly improved by first reducing the dimension with PCA or an AE (see Table 2.1). In [McConville et al., 2021], an additional UMAP step is added after the AE output, further improving the clustering. In contrast, approaches have been developed to treat clustering and dimensionality reduction as a joint task. Some of these techniques are presented in the second paragraph, where we will present "traditional" methods not based on deep representation learning. However, none of them can tackle complex non-linear transformations of the data. In the last paragraph of this section, we will see how deep neural networks can learn complex mappings from the data space to a latent space of higher-level features with a cluster structure. In particular, we will see that it is possible to achieve simultaneous representation learning and clustering. In other words, the similarities can be *learned*.

2.2.1 Joint dimensionality reduction and clustering

In this paragraph, we present methods that intrinsically handle high-dimensional, in particular noisy, correlated, redundant variables. Most methods are based on the ideas of feature selection, subspace clustering and parsimonious modeling. In an early approach, reduced K-means (RKM) [De Soete and Carroll, 1994], the solution is constrained to a lower-dimensional space. A related approach also using matrix factorization is presented by [Yang et al., 2017a]. In a different line of work, *discriminative clustering* with linear discriminant analysis (LDA) has been successfully used to jointly find a discriminative feature space improving clustering performance

of K-means on high-dimensional data [De La Torre and Kanade, 2006, Ding and Li, 2007, Ye, 2007]. See [Wang et al., 2019] for a recent work and review on this topic. Sparse subspace clustering [Elhamifar and Vidal, 2013] is adapted for data lying in a union of low-dimensional subspaces, and uses spectral clustering. [Patel et al., 2013] additionally embeds the data in a latent space. Joint feature selection and clustering using LASSO regularization [Tibshirani, 1996] (sparse or group-sparse [Yuan and Lin, 2006]) has been proposed. For example, sparse K-means [Witten and Tibshirani, 2010] and group-sparse K-means [Sun et al., 2012, Chavent et al., 2020] allows to simultaneously select the relevant variables for K-means clustering. An EM-based algorithm HDDC [Bouveyron et al., 2007] has been developed for Gaussian mixtures. Feature selection and subspace methods for the SOM algorithm were introduced for instance in the 2S-SOM method [Kaly et al., 2004] and in [Benabdeslem and Lebbah, 2007], and for the GNG in [Attaoui et al., 2020].

2.2.2 Links with metric learning

Learning representations for clustering is tightly connected to the field of *metric learning* (sometimes called *similarity learning*), where the goal is to find automatically an appropriate distance measure (or metric) between objects. Distance and similarity measures are omnipresent in machine learning and in particular unsupervised data exploration: nearest neighbors, clustering, dimensionality reduction, visualization, kernel methods, ranking, etc., and fully determine the results and success of these methods. Metric learning takes place either in a supervised learning setting (using target labels), a weakly supervised setting (using for example only *must-link* or *cannot-link* pairwise constraints), or in a semi-supervised setting (using a few labeled data or side information and a larger sample of unlabeled data). The learned metric may be linear or non-linear. A typical (linear) task is learning a Mahalanobis-like distance $d_M(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x}-\mathbf{y})^T M(\mathbf{x}-\mathbf{y})}$ for some matrix M. We will not directly use metric learning in this work and thus only refer to [Kulis, 2012, Bellet et al., 2015] for a review. Indeed, it requires a certain amount of supervision, and this thesis is focused on unsupervised approaches. Other related fields such as *kernel learning* or *multiple kernel learning* are out of the scope of this thesis.

Metric learning may also be achieved by learning neural representations, often called embeddings, using self-supervised approaches. In these methods, constraints are replaced by positive and negative samples and differentiable loss functions such as a *triplet loss* or a *ranking loss*. Examples are [Wang and Gupta, 2015] in computer vision, where successive frames of a video are used to guide a ranking loss, or

[Franceschi et al., 2019], where time series embeddings are learned by mining subsequences from inside and outside the input, and a triplet loss.

Domain knowledge such as labels or constraints can be used to help find similarities, an somewhat equivalently, new representations. However, as outlined in [Bengio et al., 2013], learning with generic priors as the ones introduced in the first section is also possible. The next section presents *deep clustering* methods, where these generic priors are augmented with a prior of cluster structure.

2.3 Deep clustering methods

Recently, many propositions have been made concerning *joint* representation learning and clustering. So-called *deep clustering* approaches treat clustering and representation learning as a joint task and focus on learning representations that are more *clustering-friendly* (a term coined by [Yang et al., 2017b]). Instead of learning a generic representation of the data (e.g. in the sense of a reconstruction criterion) and then applying a clustering algorithm, it has been proposed to learn a representation specifically for clustering tasks. The principle is to learn a latent feature space that preserves a specific prior knowledge, in this case, cluster structure.

Surveys of clustering with deep learning are presented in [Aljalbout et al., 2018] and [Min et al., 2018]. To our knowledge, the first works of this kind were proposed in 2014 by authors of [Song et al., 2014] and [Huang et al., 2014]. The first work [Song et al., 2014] proposes to jointly learn representations with an autoencoder and clusters using K-means in latent space. The AE reconstruction loss is regularized by the K-means loss, and the training procedure alternates between updating the AE network and the cluster centers using K-means. The second approach, Deep Embedding Network (DEN) [Huang et al., 2014], uses an autoencoder but does not rely on a standard clustering algorithm. It combines a locality preserving loss pushing neighboring points together in latent space, and a group sparsity constraint on the hidden units with the number of groups corresponding to the number of clusters. This idea of regularizing the latent space to *push* similar points together to enhance cluster structure, is at the core of most approaches until today.

Deep clustering methods can be classified into two main families. First, feature-based approaches, taking as input a $N \times P$ attribute matrix, and learning a non-linear mapping to a feature space where linear clustering methods are used (most often K-means or GMM). We thereafter present such approaches. Note that we keep the previously introduced notations, where $\mathbf{x}_i \in \mathcal{X}$ are training samples, $\mathbf{z}_i \in \mathcal{Z}$

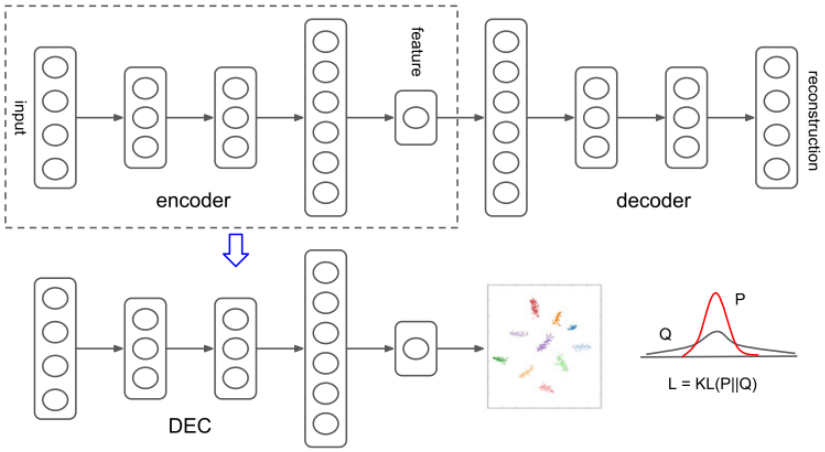


Fig. 2.7.: Deep Embedded Clustering (DEC) architecture. [Xie et al., 2016]

are the latent embeddings, and $\{\mathbf{m}_k\}_1^K \in \mathcal{Z}$ are the cluster prototypes lying in latent space, where K is the number of clusters. Another early approach, Deep Embedded Clustering (DEC) [Xie et al., 2016], starts by pretraining an autoencoder using a reconstruction loss in a SDAE fashion. Then, it jointly finetunes the encoder and learns soft cluster assignments by optimizing a Kullback-Leibler divergence that minimizes intra-cluster variance, by pushing latent points together to form clusters. More specifically, they define two distributions. First, q represents the soft membership probabilities with a differentiable t-distribution similarity kernel ([Van Der Maaten and Hinton, 2008]) between embeddings and cluster centers. Then, p is a *hardened* version of q, strengthening the predictions by squashing low-confidence predictions towards zero:

$$q_{ik} = \frac{(1 + ||\mathbf{z}_i - \mathbf{m}_k)||^2)^{-1}}{\sum_{k'}(1 + ||\mathbf{z}_i - \mathbf{m}_{k'})||^2)^{-1}}, \; p_{ik} = \frac{q_{ik}^2 / \sum_i q_{ik}}{\sum_{k'} \left(q_{ik'}^2 / \sum_i q_{ik'}\right)}.$$

The clusters are optimized using the so-called *soft hardening loss*, pushing q towards p by minimizing their KL-divergence:

$$\mathcal{L}_{\text{DEC}}(\mathbf{W}_\mathbf{e}, \{\mathbf{m}_k\}_1^K) = D_{\text{KL}}(p||q) = \sum_{i=1}^{N}\sum_{k=1}^{K} p_{ik} \log \frac{p_{ik}}{q_{ik}}.$$

The DEC architecture and training are summarized on Figure 2.7. However, the AE reconstruction is no longer optimized during the clustering step, hurting the latent space structure. Improved Deep Embedded Clustering (IDEC) [Guo et al., 2017a]

improves on this approach by optimizing a MSE reconstruction loss jointly with the KL-divergence, i.e.:

$$\begin{aligned}\mathcal{L}_{\text{IDEC}}(\mathbf{W_e}, \mathbf{W_d}, \{\mathbf{m}_k\}_1^K) &= \mathcal{L}_{\text{R}}(\mathbf{W_e}, \mathbf{W_d}) + \gamma \mathcal{L}_{\text{DEC}}(\mathbf{W_e}, \{\mathbf{m}_k\}_1^K, \chi) \\ &= \frac{1}{N}\sum_{i=1}^{N} ||\tilde{\mathbf{x}}_i - \mathbf{x}_i||_2^2 + \gamma D_{\text{KL}}(p||q)\end{aligned}$$

where γ is a hyperparameter trading off between the reconstruction and clustering terms. The model is trained end-to-end with stochastic gradient descent (SGD). The Deep Clustering Network (DCN) [Yang et al., 2017b] combines representation learning with K-means clustering using SGD in an alternating training procedure, to alternately update the AE weights, cluster assignments and centroid vectors, similarly to [Song et al., 2014]. They introduce the term *K-means-friendly space* to describe the regularizing effect of joint training that improves clustering performance. The loss function of DCN over the entire training set is:

$$\begin{aligned}\mathcal{L}_{\text{DCN}}(\mathbf{W_e}, \mathbf{W_d}, \{\mathbf{m}_k\}_1^K, \chi) &= \mathcal{L}_{\text{R}}(\mathbf{W_e}, \mathbf{W_d}) + \gamma \mathcal{L}_{\text{KM}}(\mathbf{W_e}, \{\mathbf{m}_k\}_1^K, \chi) \\ &= \frac{1}{N}\sum_{i=1}^{N} ||\tilde{\mathbf{x}}_i - \mathbf{x}_i||^2 + \gamma \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{z}_i - \mathbf{m}_{\chi(i)}||^2\end{aligned}$$

where $\chi(i) = \underset{k}{\text{argmin}}\ ||\mathbf{z}_i - \mathbf{m}_k||_2^2$ is a hard cluster assignment function. The alternate training procedure is necessary because the assignment function χ is non-differentiable. We will take an approach similar to DCN in the next chapter. Deep K-means (DKM) [Fard et al., 2018] overcomes the non-differentiability of hard cluster assignments by introducing a smoothed version of the K-means loss with simulated annealing. Deep Continuous Clustering (DCC) [Shah and Koltun, 2018] also tackles this issue by formulating a continuous clustering loss, optimized jointly with an autoencoder, but in addition, the method needs no prior knowledge of the number of clusters.

Deep Multi-Manifold Clustering (DMC) is tackled in [Chen et al., 2017] and proposes a loss function combining a reconstruction loss, a clustering loss as well as a locality-preserving loss to enforce that close inputs receive similar embeddings (smoothness). In [Dahal, 2018], encoded inputs are mapped to a different embedding space specifically for clustering. [Aytekin et al., 2018] demonstrated how ℓ_2 normalization of the latent AE representations could greatly improve clustering performance when applying K-means on the encoded samples. Without any joint training, they obtained superior performance on MNIST and USPS, compared with DEC and IDEC. They trained a dense AE end-to-end with reconstruction loss, but constrained the embeddings to have unit norm, arguing that it would make the representations

cluster better in Euclidean space, because Euclidean distance between vectors would depend only on their angle (cosine distance) and not their norms. Unlike most approaches using a standard clustering algorithm in latent space, [Tian et al., 2017] proposes to directly solve clustering using the alternating direction method of multipliers method (a flavor of the augmented Lagrangian method) in a deep learning framework. Joint learning of representations with a GMM is tackled in [Wang and Jiang, 2018], where the separability between Gaussian components is enhanced in the objective function.

Deep clustering methods have also been applied to image clustering in particular, leveraging convolutional neural network (CNN) architectures, see for instance DCEC [Guo et al., 2017b] (identical to IDEC but with a CNN) and DEPICT [Dizaji et al., 2017]). Clustering Convolutional Neural Network (CCNN) [Hsu et al., 2017] uses pseudo-labels extracted from k-NN and a *feature drift* compensation. [Kilinc and Uysal, 2018] learns K-means-friendly representations by using self-supervision through data augmentation (image rotations) and an Auto-Clustering Output Layer [Kilinc and Uysal, 2017] with Graph-based Activity Regularization (GAR).

Some other approaches adopt a K-parallel architecture, with K distinct autoencoders for each cluster [Zhang et al., 2017a, Chazan et al., 2019, Opochinsky et al., 2020]. This way, a different latent space is learned for each cluster, instead of K centroids lying in the same feature space. A data point is assigned to the network with the lowest reconstruction error.

Recent approaches perform latent space clustering using generative models such as VAE (VaDE [Jiang et al., 2017], GMVAE [Dilokthanakul et al., 2017]) or GAN (WaMiC [Harchaoui et al., 2019], ClusterGAN [Mukherjee et al., 2019], Clustered generator model [Zhu et al., 2019]) with a GMM prior in latent space, achieving state-of-the-art results. For instance in the Variational Deep Embedding (VaDE) [Jiang et al., 2017], a generative process is defined where a cluster is selected from a GMM, an embedding $\mathbf{z}$ sampled from this cluster and decoded into an observable $\mathbf{x}$. The model is trained using the VAE ELBO (Equation 2.3). Differently, IMSAT [Hu et al., 2017] learns discrete representations by maximizing information between inputs and cluster assignments, and uses a data augmentation technique called self-augmented training. Dual-AAE [Ge et al., 2019] and ADEC [Zhou and Zhou, 2019] are methods based on adversarial autoencoders. Finally, while most approaches rely on (soft) K-means or GMM as the clustering component, the framework has been extended to non-parametric maximum margin clustering [Chen, 2015], density-based clustering [Lin et al., 2018], and mean-shift clustering [Madaan and Maiti, 2019]. The second family of deep clustering methods are similarity-based

approaches, taking as input a $N \times N$ affinity matrix. Sometimes the affinity is learned. Although these methods are flexible in terms of similarity measure, they often require to build an affinity graph of the data set at each step, incurring a high computational cost. Joint Unsupervised LEarning (JULE) [Yang et al., 2016] is based on agglomerative clustering. Its training alternates between steps of agglomerative clustering (merging clusters), and a optimization step where network parameters are updated to minimize the clustering loss.

Deep learning methods for spectral clustering have also been proposed. In SpectralNet [Shaham et al., 2018], a neural network trained with SGD learns to map data to the eigenspace of their associated Laplacian matrix, and allows out-of-sample extension. The affinity function is learned by means of metric learning with a Siamese network. Other approaches are Deep Spectral Clustering (DSC) [Yang et al., 2019] and the recent Spectral clustering via ensemble deep autoencoder (SC-EDAE) [Affeldt et al., 2020]. In [Yang et al., 2019], a dual AE is used to learn noise-robust representations, a spectral embedding and also uses mutual information maximization. The Dissimilarity Mixture Autoencoder (DMAE) [Lara and González, 2020] is a hybrid approach, generalizing GMM to different dissimilarity functions. To conclude, recent advances in deep clustering include graph clustering [Tian et al., 2014, Bo and Wang, 2020] and robustness to adversarial attacks [Yang et al., 2020].

Table 2.1 summarizes the ability of clustering methods to recover the ground-truth classes, in terms of unsupervised clustering accuracy (defined in Section 4.2, Chapter 4), ranging from K-means to the latest deep clustering advances. The data sets are described later in Table 3.3. As a whole, deep clustering is already a huge research field (the related works are impossible to list exhaustively; this state-of-the-art is only an overview). However, it is still in its infancy and many issues are unsolved or handled empirically. For instance, to solve the issue arising from optimizing conflicting loss functions in deep clustering (representation VS cluster structure), such as feature drift, [Mrabah et al., 2020] introduced the Dynamic AE (DynAE) that gradually moves from optimizing the reconstruction to the clustering.

2.4 Deep self-organized models

This PhD is particularly interested in self-organized clustering models. Thus, we propose to extend the ideas of deep clustering to the SOM algorithm. The next chapter will introduce the Deep Embedded SOM (DESOM), one the main contributions of this thesis, but we first provide a review of the state of the art and similar

Tab. 2.1.: Unsupervised clustering accuracy (%) obtained by traditional and deep clustering methods on benchmark data sets (average, on test set when available).

Method	MNIST	Fashion-MNIST	USPS	Reuters-10k
K	10	10	10	4
K-means	53.3	54.9	66.0	58.9
GMM [Harchaoui et al., 2019]	53.7	-	-	54.7
Spectral (N-cuts)	66.0	50.8	64.9	-
AE + K-means	80.1	48.9	68.0	53.8
AE + GMM [Harchaoui et al., 2019]	82.6	-	-	70.1
GMVAE [Dilokthanakul et al., 2017]	82.3	-	-	-
DCN [Yang et al., 2017b]	83.0	-	-	80.0
DKM [Fard et al., 2018]	84.0	-	75.7	58.3
DEC [Guo et al., 2017a]	86.6	51.8	74.1	73.7
IDEC [Guo et al., 2017a]	88.1	52.9	76.1	75.6
VaDE [Jiang et al., 2017]	94.5	-	56.6	79.8
N2D [McConville et al., 2021]	94.8	67.2	95.8	-
ClusterGAN [Mukherjee et al., 2019]	95.0	63.0	-	-
JULE [Yang et al., 2016]	96.1	56.3	95.0	-
DEPICT [Dizaji et al., 2017]	96.3	39.2	89.9	-
WaMiC [Harchaoui et al., 2019]	97.3	-	-	79.8
Dual AE [Yang et al., 2019]	98.0	66.2	86.9	-
GAR [Kilinc and Uysal, 2018]	98.3	-	96.5	-
IMSAT [Hu et al., 2017]	98.4	-	71.0	

approaches. SOM has been initially used with Euclidean distance, but any similarity function may be used, as in the relational SOM [Olteanu et al., 2013]. This requires an adequate similarity measure. We have already mentioned feature selection and subspace-based self-organized models in a previous paragraph [Benabdeslem and Lebbah, 2007, Kaly et al., 2004, Attaoui et al., 2020]. Here, we will not consider these kinds of approaches, and use the Euclidean distance, but in the intermediate feature space of neural networks.

Although less attention has been given to self-organizing map models than pure clustering, this attention raised in 2018 and 2019, with several works on this subject. Early on, layered SOM architectures had been proposed. The DSOM (Deep Self-Organizing Map) [Liu et al., 2015] is a multilayer SOM-based architecture for image recognition, similar in spirit to CNNs, and trained via the supervised learning method described previously. Each layer is composed of a *self-organizing layer* and a *sampling layer*, and a last layer, composed of a single SOM, for classification. The self-organizing layers consist in self-organizing maps sliding over the input image, each map focusing on a different part of the input. The sampling layer is composed of the winning units of each map of the self-organizing layer, resulting in a rectangular map that can serve as an input for the next layer. Finally, the classification SOM at the top of the network takes the whole last sampling layer as input. Compared to the supervised SOM baseline from [Idan and Chevalier, 1991], DSOM performs 7% better on the MNIST data set, but remains well under the accuracy obtained with convolutional networks.

In [Elend and Kramer, 2019], authors have shown evidence that using higher-level features extracted by convolutional layers of pretrained networks improves SOM quality, as measured by several external label-oriented indices but also visual quality. However, we think these results are somewhat biased because the features are extracted from a network that was pretrained in a supervised manner on the same data set. Thus, the top layers were trained to learn a feature space where the classes are well-separated. Of course, this will help the subsequent SOM to separate the classes and improve the label-oriented results. Maybe the performance improvements would have been lower using features from a network that was pretrained on another generic data set, e.g. ImageNet. Therefore, although the idea is to use a generic pretrained network for feature extraction in order to have a completely unsupervised process, the experiments in the paper differ from this intent.

Other works are based on unsupervised neural networks, namely autoencoders. An early approach, deep neural maps [Pesteie et al., 2018], combines a DAE, the

soft clustering loss from DEC and a SOM. Their procedure alternates between (1) assigning and updating cluster centers using the SOM stochastic algorithm (2) optimize the representations, regularized by the KL-divergence clustering term. They compared this approach with other DR and visualization tools, however they did not quantitatively assess the clustering or self-organization performance. [Ferles et al., 2018] proposes the denoising autoencoder self-organizing map (DASOM) and several variants in an extensive study on the combination of a non-linear DAE with SOM. Another series of work performing joint representation learning with a SOM is the SOM-VAE model, introduced in [Fortuin et al., 2019]. Their model is based on the VQ-VAE (Vector Quantization Variational Autoencoder) model which enables to train VAEs with a discrete latent space [van den Oord et al., 2017]. The principle of the VQ-VAE is to learn a finite set of embeddings $\{\mathbf{e}_k\}_1^K$ using vector quantization (VQ). Latent codes $\mathbf{z}_e$ are discretized by assigning them to their nearest embedding $\mathbf{z}_q$. [Fortuin et al., 2019] have added a topology constraint on the discrete latent space by modifying the loss function of VQ-VAE. The loss of SOM-VAE is composed of three terms: the ELBO reconstruction loss $\mathcal{L}_{\text{R}}$ is the discrete case (equivalent to a deterministic AE, without the (here constant) KL part), the vector quantization loss $\mathcal{L}_{\text{VQ}}$ and the SOM loss $\mathcal{L}_{\text{SOM}}$. The latter only considers the direct neighbors on the map and the neighborhood is fixed. For an individual input $\mathbf{x}$ the proposed loss is

$$\begin{aligned}\mathcal{L}_{\text{SOM-VAE}}(\mathbf{x}) &= \mathcal{L}_{\text{R}}(\mathbf{x}) + \alpha\mathcal{L}_{\text{VQ}}(\mathbf{x}) + \beta\mathcal{L}_{\text{SOM}}(\mathbf{x}) \\ &= ||\mathbf{x} - \tilde{\mathbf{x}}_q||_2^2 + ||\mathbf{x} - \tilde{\mathbf{x}}_e||_2^2 + \alpha||\mathbf{z}_e - \mathbf{z}_q||_2^2 + \beta \sum_{\mathbf{e}\in \text{N}(\mathbf{z}_q)} ||\mathbf{e} - \text{sg}[\mathbf{z}_e]||_2^2\end{aligned}$$

where $\tilde{\mathbf{x}}_e$ and $\tilde{\mathbf{x}}_q$ are respectively the reconstructions of $\mathbf{z}_e$ and $\mathbf{z}_q$, $\text{N}(\cdot)$ is the set of direct neighbors on the map and $\text{sg}[\cdot]$ is the stop-gradient operator. Later, the Deep Probabilistic SOM (DPSOM) [Manduchi et al., 2020], a very recent unpublished work improved on the SOM-VAE, achieving state-of-the-art clustering results and proposing an interpretable application in the healthcare domain. It combines elements from SOM-VAE and from DEC, and uses soft cluster assignments. Their loss function combines the ELBO (this time without discrete quantization) $\mathcal{L}_{\text{ELBO}}$ (see Equation 2.3), a KL-divergence clustering loss (the same as DEC) $\mathcal{L}_{\text{DEC}}$ and a soft SOM loss $\mathcal{L}_{\text{S-SOM}}$ with fixed neighborhood:

$$\begin{aligned}\mathcal{L}_{\text{DPSOM}} &= \gamma\mathcal{L}_{\text{ELBO}} + \mathcal{L}_{\text{DEC}} + \beta\mathcal{L}_{\text{S-SOM}} \\ &= \gamma\mathcal{L}_{\text{ELBO}} + \sum_{i=1}^{N}\sum_{k=1}^{K} p_{ik}\log\frac{p_{ik}}{q_{ik}} - \beta\sum_{i=1}^{N}\sum_{k=1}^{K} q_{ik}\sum_{k'\in\text{N}(k)}\log q_{ik'}\end{aligned}$$

where the distributions p and q are defined using Student's t-distribution kernel as in DEC, and $\text{N}(k)$ is the set of neighbors of unit k.

2.5 Conclusion

This chapter tackled representation learning without supervision with neural networks. One major family of approaches, explored in this thesis, are autoencoders. We introduced them in depth, with their regularizations and more sophisticated variants such as VAEs. We also mentioned other representation learning methods that were not applied during this PhD, namely adversarial and self-supervised learning with pretext tasks. Then, we have seen how these representations can be biased specifically towards a clustering objective, in order to learn a clustering-friendly feature space. So-called deep clustering methods optimize jointly an information preservation and a clustering loss. Furthermore, self-organized models can be enhanced in the same manner. Very recently, a few works have performed deep clustering with the SOM. We have presented a state-of-the-art of this new research area, before introducing our own contribution, subject of the next chapter.

3 Deep Embedded SOM (DESOM)

This chapter is based on the contributions:

- Forest, F., Lebbah, M., Azzag, H., & Lacaille, J. (2019). Deep Embedded SOM: Joint Representation Learning and Self-Organization. *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*.
- Forest, F., Lebbah, M., Azzag, H., & Lacaille, J. (2019). Deep Architectures for Joint Clustering and Visualization with Self-Organizing Maps. *Workshop on Learning Data Representations for Clustering (LDRC), Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*.
- Forest, F., Lebbah, M., Azzag, H., & Lacaille, J. (2020). Carte SOM profonde : Apprentissage joint de représentations et auto-organisation. *CAp: Conférence d'Apprentissage*.

One of the main contributions of this thesis is the Deep Embedded SOM (DESOM) model, introduced in [Forest et al., 2019b, Forest et al., 2019a, Forest et al., 2020c]. We propose an approach where self-organization of the prototypes and representation learning through a deterministic autoencoder are performed jointly by stochastic gradient descent (SGD).

The DESOM model has already been applied in literature shortly after its publication, to analyze energy consumption of buildings for smart energy management [Ullah et al., 2020]. It was used in [Medeiros et al., 2020] and extended to jointly learn feature relevance weights. Finally, it has been applied to seismic facies data in [Liu et al., 2020]. DESOM outperformed the other two-stage DR + SOM methods (such as AE+SOM), and authors introduced a variant called SDESOM (Sparse DESOM) with a sparsity constraint on the latent space, which improved feature extraction and clustering performance.

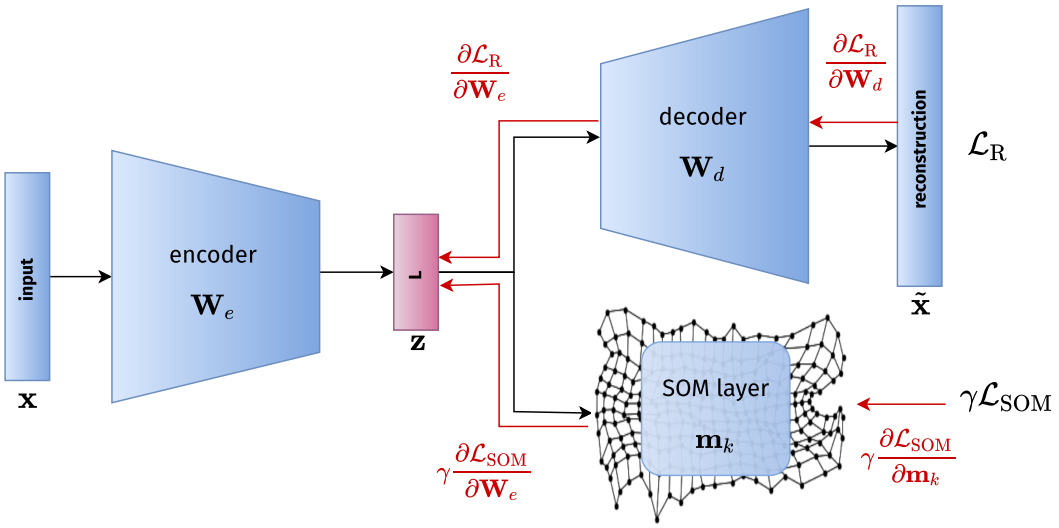


Fig. 3.1.: DESOM architecture and gradients paths. The input $\mathbf{x}$ is projected into latent space by the encoder $\mathbf{W_e}$, where the SOM prototypes $\mathbf{m}_k$ are learned via the SOM loss $\mathcal{L}_\mathrm{SOM}$. Latent samples $\mathbf{z}$ are reconstructed using the decoder $\mathbf{W_d}$ via the reconstruction loss $\mathcal{L}_\mathrm{R}$.

3.1 Architecture

The architecture is illustrated in Figure 3.1. The encoder and decoder networks are generic and can be fully-connected, convolutional or even recurrent. Experiment have been conducted with a fully-connected (DESOM) and a convolutional (ConvDESOM) version.

The architecture is composed of three neural network modules: an encoder, a decoder and a SOM layer. The encoder projects the inputs onto a latent, intermediate space. The SOM is trained in this latent space and receives encoded inputs. The decoder reconstructs the latent code back into original space, trying to match the input as closely as possible.

3.2 Loss function

The encoder and decoder parameter weights are respectively noted $\mathbf{W_e}$ and $\mathbf{W_d}$. The encoding function is denoted by $\mathbf{f_{W_e}}$ and the decoding function by $\mathbf{g_{W_d}}$. Thus, $\mathbf{z}_i = \mathbf{f_{W_e}}(\mathbf{x}_i) \in \mathbb{R}^L$ is the embedded version of $\mathbf{x}_i$ in the intermediate latent space, and $\tilde{\mathbf{x}}_i = \mathbf{g_{W_d}}(\mathbf{f_{W_e}}(\mathbf{x}_i)) \in \mathbb{R}^P$ is its reconstruction by the decoder. Our goal is to jointly optimize the autoencoder network weights and the SOM prototype vectors.

For this task, we define a hybrid loss function composed of two terms, that can be written as

$$\mathcal{L}(\mathbf{W_e}, \mathbf{W_d}, \{\mathbf{m}_k\}_1^K) = \mathcal{L}_{\text{R}}(\mathbf{W_e}, \mathbf{W_d}) + \gamma \mathcal{L}_{\text{SOM}}(\mathbf{W_e}, \{\mathbf{m}_k\}_1^K). \tag{3.1}$$

The first term $\mathcal{L}_R$ is the autoencoder reconstruction loss. We use a MSE loss, which corresponds to reconstructing the mean of a Gaussian output distribution:

$$\mathcal{L}_{\text{R}} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_{\text{R}}^i = \frac{1}{N} \sum_{i=1}^{N} ||\tilde{\mathbf{x}}_i - \mathbf{x}_i||_2^2.$$

The second term is the self-organizing map loss, denoted $\mathcal{L}_{\text{SOM}}$. It depends on the set of parameters $\{\mathbf{m}_k\}_{1 \leq k \leq K}$ and on the best-matching units, denoted b_i, assigning a latent data point to its closest prototype according to Euclidean distance, i.e.

$$b_i = \underset{k}{\text{argmin}} \, ||\mathbf{z}_i - \mathbf{m}_k||_2^2.$$

The expression of the self-organizing map loss is

$$\mathcal{L}_{\text{SOM}} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_{\text{SOM}}^i = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \mathcal{K}^T \left(\delta(b_i, k)\right) ||\mathbf{z}_i - \mathbf{m}_k||_2^2.$$

The gamma γ coefficient is a hyperparameter that trades off between minimizing the AE reconstruction error and the SOM error. Therefore, the SOM loss acts as a SOM-guided regularizer. The SOM loss can be decomposed into two terms, the first one being the squared distance between the best matching unit and the latent point, and the second one corresponding to the topographic relationship with neighboring units:

$$\begin{aligned}
\mathcal{L}_{\text{SOM}} &= \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \mathcal{K}^T \left(\delta(b_i, k)\right) ||\mathbf{z}_i - \mathbf{m}_k||_2^2 \\
&= \frac{1}{N} \sum_{i=1}^{N} \left[\mathcal{K}^T \left(\delta(b_i, b_i)\right) ||\mathbf{z}_i - \mathbf{m}_{b_i}||^2 + \sum_{k \neq b_i} \mathcal{K}^T \left(\delta(b_i, k)\right) ||\mathbf{z}_i - \mathbf{m}_k||_2^2 \right] \\
&= \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{z}_i - \mathbf{m}_{b_i}||_2^2 + \frac{1}{N} \sum_{i=1}^{N} \sum_{k \neq b_i} \mathcal{K}^T \left(\delta(b_i, k)\right) ||\mathbf{z}_i - \mathbf{m}_k||_2^2.
\end{aligned}$$

For large values of T, the second term is prevalent and leads to topographic organization. When the temperature approaches zero, the first term prevails and the SOM

loss becomes identical to a K-means loss, where the centroids correspond to the map prototypes:

$$\lim_{T\to 0} \mathcal{L}_{\text{SOM}} = \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{z}_i - \mathbf{m}_{b_i}||_2^2 = \mathcal{L}_{K\text{-means}}.$$

Thus, when temperature is close to zero, the hybrid loss function (Equation 3.1) can be written as follows:

$$\lim_{T\to 0} \mathcal{L} = \mathcal{L}_{\text{R}} + \gamma \mathcal{L}_{K\text{-means}}.$$

Hence, our model becomes identical to the DCN model [Yang et al., 2017b] or the DKM model [Fard et al., 2018] (at the end of their hyperparameter annealing).

3.3 Training procedure

We use a joint training procedure, optimizing both the network parameters and the prototypes by backpropagation and stochastic gradient descent. The assignments to the best-matching units are fixed between each optimization step, as it is non-differentiable. Thus, the weighting terms $w_{i,k} := \mathcal{K}^T(\delta(b_i, k))$ become simple coefficients for each input and prototype, constant w.r.t. the network parameters and the prototypes. The gradients of the loss function w.r.t. AE weights and prototypes are easy to derive if we consider assignments to be fixed at each step:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{W_e}} &= \frac{\partial \mathcal{L}_{\text{R}}}{\partial \mathbf{W_e}} + \gamma \frac{\partial \mathcal{L}_{\text{SOM}}}{\partial \mathbf{W_e}} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{W_d}} &= \frac{\partial \mathcal{L}_{\text{R}}}{\partial \mathbf{W_d}} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{m}_k} &= \gamma \frac{\partial \mathcal{L}_{\text{SOM}}}{\partial \mathbf{m}_k}.
\end{aligned}$$

The gradients for a single data point $\mathbf{x}_i$ are

$$\begin{aligned}
\frac{\partial \mathcal{L}^i_{\text{R}}}{\partial \mathbf{W_e}} &= 2(\mathbf{g}_{\mathbf{W_d}}(\mathbf{f}_{\mathbf{W_e}}(\mathbf{x}_i)) - \mathbf{x}_i)\frac{\partial \mathbf{g}_{\mathbf{W_d}}(\mathbf{f}_{\mathbf{W_e}}(\mathbf{x}_i))}{\partial \mathbf{W_e}} \\
\frac{\partial \mathcal{L}^i_{\text{R}}}{\partial \mathbf{W_d}} &= 2(\mathbf{g}_{\mathbf{W_d}}(\mathbf{f}_{\mathbf{W_e}}(\mathbf{x}_i)) - \mathbf{x}_i)\frac{\partial \mathbf{g}_{\mathbf{W_d}}(\mathbf{f}_{\mathbf{W_e}}(\mathbf{x}_i))}{\partial \mathbf{W_d}} \\
\frac{\partial \mathcal{L}^i_{\text{SOM}}}{\partial \mathbf{W_e}} &= 2\sum_{k=1}^{K} w_{i,k}(\mathbf{f}_{\mathbf{W_e}}(\mathbf{x}_i) - \mathbf{m}_k)\frac{\partial \mathbf{f}_{\mathbf{W_e}}(\mathbf{x}_i)}{\partial \mathbf{W_e}} \\
\frac{\partial \mathcal{L}^i_{\text{SOM}}}{\partial \mathbf{m}_k} &= 2w_{i,k}(\mathbf{m}_k - \mathbf{f}_{\mathbf{W_e}}(\mathbf{x}_i)).
\end{aligned}$$

The paths of the gradients of the loss function are illustrated on Figure 3.1. We optimize Equation 3.1 using minibatch SGD, with a learning rate l_r (in our experiments Adam is used instead, but the equations are derived for vanilla SGD). Given a batch $\mathcal{B}$ of n_b samples, the encoder's weights are updated by

$$\mathbf{W_e} \leftarrow \mathbf{W_e} - \frac{l_r}{n_b} \sum_{i \in \mathcal{B}} \left(\frac{\partial \mathcal{L}_{\text{R}}^i}{\partial \mathbf{W_e}} + \gamma \frac{\partial \mathcal{L}_{\text{SOM}}^i}{\partial \mathbf{W_e}} \right), \tag{3.2}$$

the decoder's weights are updated by

$$\mathbf{W_d} \leftarrow \mathbf{W_d} - \frac{l_r}{n_b} \sum_{i \in \mathcal{B}} \frac{\partial \mathcal{L}_{\text{R}}^i}{\partial \mathbf{W_d}}, \tag{3.3}$$

and finally, the map prototypes are updated by the following update rule:

$$\mathbf{m}_k \leftarrow \mathbf{m}_k - \frac{l_r}{n_b} \sum_{i \in \mathcal{B}} \gamma \frac{\partial \mathcal{L}_{\text{SOM}}^i}{\partial \mathbf{m}_k}. \tag{3.4}$$

By expanding the prototypes update rule in Equation 3.4, we obtain an expression somewhat in between of the stochastic SOM and the batch SOM algorithms presented in the previous section (see Equations 1.13 and 1.15), that we can call *minibatch stochastic SOM*:

$$\mathbf{m}_k \leftarrow \mathbf{m}_k + 2\gamma \frac{l_r}{n_b} \sum_{i \in \mathcal{B}} \mathcal{K}^T \left(\delta(b_i, k) \right) (\mathbf{z}_i - \mathbf{m}_k). \tag{3.5}$$

As in batch SOM, we alternate between BMU assignments and minimization, but minimization happens via a gradient descent step as in stochastic SOM. Thus, we think optimizing SOM with our procedure is a sound choice.

The main novelty of our model is a new custom layer called *SOM layer*. Its parameters are the set of SOM code vectors in latent space, i.e. a $K \times L$ matrix where K is the number of prototypes (e.g. 64 for an 8×8 map) and L is the dimensionality of the latent space. The outputs of this layer are defined as the pairwise squared Euclidean distances between the input batch and the prototypes: this allows to express the SOM loss as a simple weighted sum, using the neighborhood weight terms $w_{i,k}$.

First, the autoencoder is initialized either randomly using the Glorot uniform initializer, or by pretraining, and SOM parameters are initialized either at random with an encoded data sample, or by a standard SOM (this choice is studied in the experiments). At each iteration, the temperature is updated using exponential decay. Then, we perform inference on the current batch to obtain the pairwise distances between latent samples and SOM prototypes, in order to compute the weights $w_{i,k}$

using the neighborhood function. Finally, we perform a training step to update all parameters. In addition, we collect losses and performance metrics at a fixed interval on the training and test sets, using the library SOMperf[1] [Forest, 2020, Forest et al., 2020b]. The training procedure of DESOM is detailed in Algorithm 3.1, omitting the

Algorithm 3.1: DESOM training procedure.

Input: training set $\mathbf{X}$; AE architecture; SOM map size; temperatures T_{max}, T_{min}; $iterations$; $batchSize$

Output: AE weights $\mathbf{W_e}$, $\mathbf{W_d}$; SOM code vectors $\{\mathbf{m}_k\}$

Initialize AE weights $\mathbf{W_e}$, $\mathbf{W_d}$ (random or pretrain) ;

Initialize SOM parameters $\{\mathbf{m}_k\}$ (random data sample or pretrain) ;

for $t = 1, \dots, iterations$ **do**

$T \leftarrow T_{max} \left(\frac{T_{min}}{T_{max}}\right)^{t/iterations}$;

Load next training batch $\mathcal{B}$;

Encode current batch ;

Compute assignments and weights $w_{i,k}$ on batch ;

Train DESOM on batch by taking a SGD step (by Equations 3.2, 3.3 and 3.4) ;

test set for sake of brevity. We have also tried updating the self-organizing map not at every training iteration, but only every `update_interval` iterations, introducing an additional hyperparameter. This follows remarks from [Guo et al., 2017a] and [Ma et al., 2019] and should help better training the encoder. To achieve this, we set all gradients coming from the SOM loss to zero between each update interval. The impact of this update interval will be mentioned in the next section.

3.4 Training parameters

Across all experiments, we use the Adam optimizer with $l_r = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. The T_{min} parameter defines the final neighborhood radius at the end of training. It has a direct impact on the trade-off between quantization and topographic error: by choosing a T_{min} smaller than 1, the prototype vectors will be finetuned locally and improve quantization, however, it may harm the topology of the map. This choice depends on the priority of the practitioner. In our experiments, we set T_{min} to 0.1, in order to obtain good quantization and clustering, and we observed no visual degradation of the map. This will be further discussed. T_{max} is always set equal to the map size, in order to organize all units in the early stage of

[1] `https://github.com/FlorentF9/SOMperf`

training. All other parameters such as map size, latent space dimension, gamma γ, pretraining, initialization and batch sizes are studied in the experiments.

As there is a large number of parameters, we recapitulate them in Table 3.1. The last column indicates whether the value of the parameter is modified and studied in the experiments section; otherwise, it is fixed to the value indicated in the default value column.

Tab. 3.1.: DESOM training parameters.

Parameter	Notation	Default value	Studied in experiments
Gamma	γ	10^{-3}	✓
Latent code dimension	L	10	✓
Map size	-	8×8	✓
Initial temperature	T_{max}	8.0	✓
Final temperature	T_{min}	0.1	✓
Batch size	n_b	256	✓
Learning rate	l_r	0.001	✗
First moment decay	β_1	0.9	✗
Second moment decay	β_2	0.999	✗

3.5 Comparison with other deep SOM models

There are many differences between DESOM and SOM-VAE. First, SOM-VAE utilizes a discrete latent space to represent the SOM prototypes, whereas in DESOM, the SOM is learned in a continuous latent space. Secondly, they use a fixed window neighborhood to update the map prototypes, whereas we use a Gaussian neighborhood with exponential radius decay. Finally, the DESOM model presented in this work is based on a deterministic AE and not a VAE. Table 3.2 summarizes the properties of the different deep SOM models in terms of latent space, AE architecture, loss function and training.

3.6 Data sets

We experiment, evaluate and compare models on four different classification benchmark data sets, three image data sets and one text data set:

Tab. 3.2.: Comparison of the properties of deep SOM models.

Model	Latent	AE	Rec. loss	SOM loss	Joint	Pretraining
ConvSOM [Elend and Kramer, 2019]	continuous	AE	MSE	SOM	✗	✓
DASOM [Ferles et al., 2018]	continuous	DAE	MSE	SOM	✓	✓
DESOM [Forest et al., 2019b]	continuous	AE	MSE	SOM	✓	✗
Deep neural maps [Pesteie et al., 2018]	continuous	AE	MSE	KL+SOM	✓	✓
SOM-VAE [Fortuin et al., 2019]	discrete	VQ-VAE	ELBO	VQ+SOM	✓	✓
DPSOM [Manduchi et al., 2020]	continuous	VAE	ELBO	KL+SOM	✓	✓

- **MNIST** [LeCun et al., 1998]: the MNIST data set consists in 70000 grayscale images of handwritten digits, of size 28-by-28 pixels. We used the data set available in the Keras library, divided the pixel intensities by 255 to obtain floating-point values between 0 and 1, and flattened the images to 784-dimensional vectors (except for the convolutional architecture).
- **Fashion-MNIST** [Xiao et al., 2017]: the Fashion-MNIST data set was designed as a drop-in replacement for the original MNIST data set, but with images of clothing instead of digits, and provides a more challenging classification task. The data set is also available in Keras and we applied the same preprocessing.
- **USPS**: this data set also consists in images of grayscale handwritten digits, and contains 9298 16-by-16 pixel digits. We downloaded it from the Kaggle website and did not perform any preprocessing.
- **Reuters-10k** [Lewis et al., 2004]: the Reuters-10k data set is built from the RCV1-v2 corpus, that contains 804,414 English news stories labeled with a category tree, with a total of 103 topics. Reuters-10k is created by restricting the documents to 4 root categories (corporate/industrial, government/social, markets and economics), excluding documents with multiple labels, then sampling a subset of 10000 examples and computing TF-IDF features on the 2000 most frequently occurring words. We downloaded the raw RCV1-v2 topics and tokens and used the same code as in [Guo et al., 2017a] to build the data set.

The properties of each data set are described in Table 3.3. In particular, we use the default train/test splits. These data sets were selected because they all have a high dimensionality (256 to 2000) and can benefit greatly from the representation learning through a deep neural network. Using Euclidean distance directly on a high-dimensional space, as is done in traditional SOM, is known to be problematic, as explained in Chapter 1. Image data sets also have the advantage of being easily visualized on a self-organizing map. Finally, these data sets were used in

Tab. 3.3.: Data set statistics of MNIST, Fashion-MNIST, USPS and Reuters-10k.

Data set	Description	Total	Train	Test	Classes	Dimension
MNIST	images (digits)	70000	60000	10000	10	784
Fashion-MNIST	images (clothing)	70000	60000	10000	10	784
USPS	images (digits)	9298	7291	2007	10	256
Reuters-10k	text (tf-idf)	10000	7769	2231	4	2000

many previous works on deep learning-based clustering models, allowing direct comparisons.

3.7 Architecture and hyperparameter study

The DESOM model is governed by hyperparameters that are coupled in the training process. A thorough exploration of architecture and training hyperparameters and their influence on performance metrics is the goal of the experiments detailed in the next paragraph. We study the influence of five fundamental parameters: the gamma γ hyperparameter, the latent code dimension, the map size, and the nature of the autoencoder (fully-connected or convolutional). All other parameters that are either related to initialization or to training dynamics are fixed to reasonable values and will be studied later. The subsequent paragraph discusses initialization strategies for the AE and SOM weights. In the third paragraph, we study the learning dynamics of DESOM, in particular the evolution of learning curves and the interaction between its two components (AE and SOM), with different parameters. Then, the next paragraph studies the visual quality of prototypes for image data sets. The performance metrics used here are all introduced in Chapter 4. All experiments are run 10 times to obtain meaningful means and standard deviations, displayed on the graphs. We use the standard train/test split (see Table 3.3) and always report results on the test set (unless specified otherwise).

3.7.1 Gamma γ hyperparameter study

This parameter defines the relative weight of reconstruction and SOM in the loss function. We evaluate external clustering metrics on four data sets for different values of γ, ranging from 10^{-4} to 10 (see Table 3.4). Our goal is not to cross-validate and find the best value according to some external quality metric, as we

Tab. 3.4.: Comparison of purity and NMI with different values of DESOM hyperparameter γ. Best performance in bold underlined. Bold values correspond to results with no statistically significant difference to the best (p-value > 0.05 after pairwise t-test).

	MNIST		Fashion-MNIST	
γ	Pur	NMI	Pur	NMI
10^{-4}	.929 ± .004	.652 ± .003	**.759 ± .006**	**.543 ± .004**
10^{-3}	**.934 ± .004**	**.658 ± .004**	.751 ± .009	**.541 ± .004**
10^{-2}	.911 ± .006	.639 ± .006	.737 ± .007	.529 ± .005
0.1	.876 ± .008	.609 ± .006	.721 ± .005	.520 ± .004
0.5	.836 ± .033	.584 ± .016	.718 ± .009	.517 ± .006
1.0	.810 ± .025	.566 ± .019	.715 ± .012	.517 ± .007
10.0	.114 ± .000	.006 ± .003	.678 ± .008	.484 ± .007
	USPS		Reuters-10k	
γ	Pur	NMI	Pur	NMI
10^{-4}	.837 ± .014	.573 ± .009	.795 ± .019	.352 ± .012
10^{-3}	**.857 ± .011**	**.592 ± .010**	**.808 ± .017**	**.364 ± .011**
10^{-2}	.839 ± .013	.583 ± .008	.801 ± .017	.352 ± .014
0.1	.815 ± .014	.566 ± .007	**.809 ± .014**	**.365 ± .016**
0.5	.806 ± .018	.561 ± .012	**.819 ± .020**	**.371 ± .012**
1.0	.806 ± .013	.559 ± .006	**.804 ± .030**	**.354 ± .027**
10.0	.760 ± .029	.523 ± .021	.466 ± .101	.062 ± .078

are in an unsupervised setting, but to find a reasonable order of magnitude across multiple data sets. The hyperparameter γ trades off between preserving information (obtaining good reconstructions) and SOM clustering. As the SOM loss takes larger values than the reconstruction loss, γ must be set smaller than one. A good value is $\gamma = 10^{-3}$ across all data sets; it represents the optimum for MNIST, USPS, and is within the variance interval for Fashion-MNIST and Reuters-10k (for the latter data set, variance of our AE is very high). Higher values of γ lead to degenerate solutions for the encoder and decoder, which translates into low scores across all data sets, and the AE being unable to produce any reconstructions. This is due to the fact that the model tries hard to optimize the SOM loss, which is much easier to optimize than the reconstruction loss as we observed during our experiments, thus neglecting code quality.

Figure 3.2 represents SOM quantization and topographic error as a function of γ (for MNIST). Quantization error exhibits the trade-off between the reconstruction and SOM clustering: it decreases with γ, as high γ values result in the SOM being more finetuned. As a drawback, this finetuning also increases topographic error, as can been seen on the rightmost graph. Behavior is similar for other data sets.

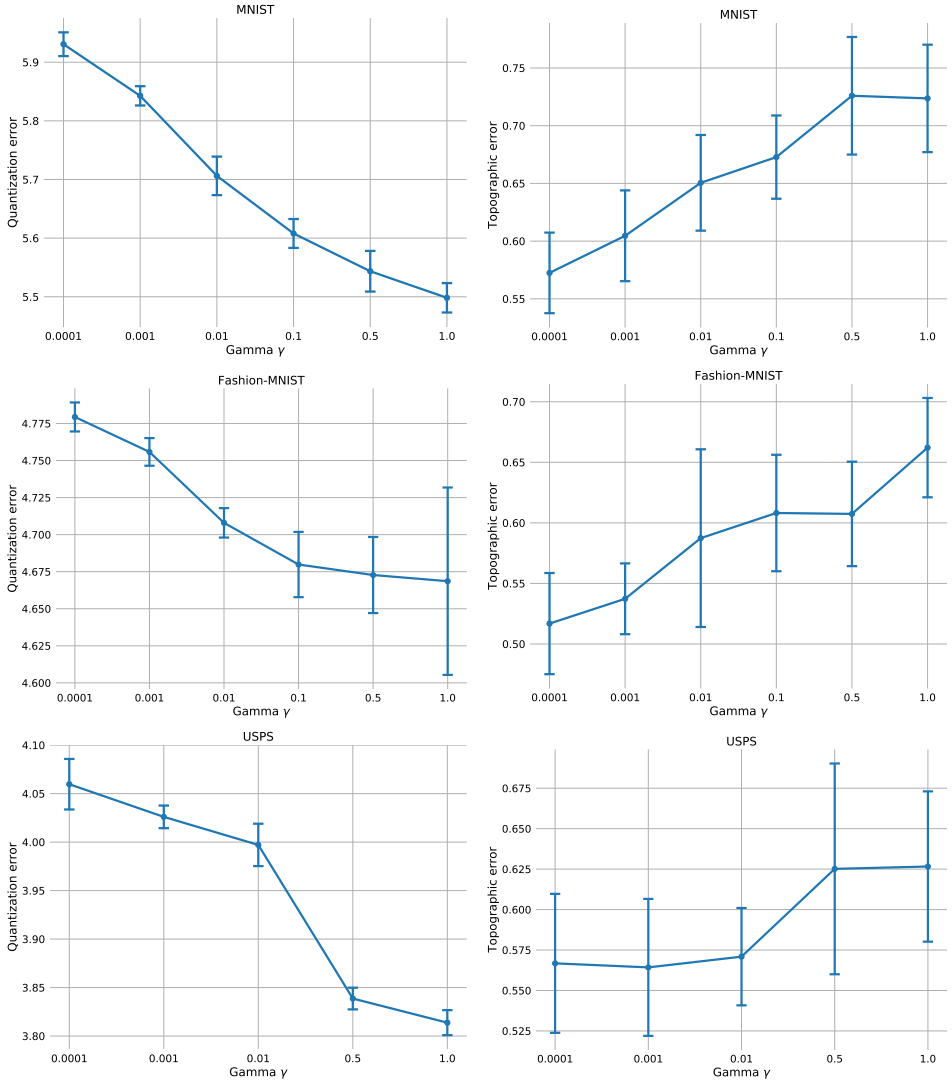


Fig. 3.2.: Quantization and topographic error as a function of DESOM hyperparameter γ on MNIST, Fashion-MNIST and USPS.

As a conclusion, DESOM is not very sensitive to γ as long as it stays in the right order of magnitude, and we select $\gamma = 10^{-3}$ for the rest of the paper, in accordance with the unsupervised setting (even if better results could be obtained by choosing an adapted value for each data set).

3.7.2 Latent code dimension study

Authors in [Manduchi et al., 2020] have found that DEC performed better with a lower-dimensional AE latent space ($L = 10$), while their VAE performed better with a higher code dimension ($L = 100$). Let us assume that the size of the SOM

has been fixed by the user. The dimensionality of the latent space where this SOM will be learned is expected to be a determining parameter. Concretely, we expect following behavior: on one hand, a latent space too small will result in a loss of information and a lower performance as measured by external indices (label-based). However the SOM may fit the latent code space very well and produce high latent quality metrics (e.g. latent quantization error). On the other hand, with a large latent space, the AE will not use all latent variables to extract useful features and likely overfit the training set, and in addition, the low-dimensional SOM will have difficulties to fit this high-dimensional space (translating into low latent quality metrics). A straightforward experiment consists in comparing performance metrics with different latent space dimensions, leaving all other parameters unchanged.

Tab. 3.5.: Comparison of purity and NMI with different latent code dimensions L in DESOM. Best performance in bold underlined. Bold values correspond to results with no statistically significant difference to the best (p-value > 0.05 after pairwise t-test).

	MNIST		Fashion-MNIST	
L	Pur	NMI	Pur	NMI
2	.768 ± .012	.552 ± .011	.688 ± .023	.499 ± .010
5	.901 ± .010	.628 ± .007	.736 ± .011	.530 ± .005
10	**.931 ± .007**	**.654 ± .006**	**.756 ± .008**	**.542 ± .003**
20	**.925 ± .006**	.647 ± .006	**.752 ± .008**	**.542 ± .003**
50	.921 ± .006	.641 ± .005	.747 ± .009	**.541 ± .004**
100	.921 ± .004	.643 ± .004	.747 ± .006	**.541 ± .003**
	USPS		Reuters-10k	
L	Pur	NMI	Pur	NMI
2	.800 ± .013	.563 ± .007	**.840 ± .010**	**.375 ± .012**
5	.843 ± .004	.583 ± .006	.808 ± .016	.360 ± .012
10	**.855 ± .010**	**.591 ± .008**	.800 ± .024	.358 ± .017
20	.828 ± .010	.571 ± .007	.794 ± .028	.353 ± .022
50	.817 ± .012	.562 ± .011	.789 ± .030	.346 ± .017
100	.806 ± .021	.558 ± .012	.787 ± .019	.344 ± .015

These intuitions are confirmed by Table 3.5, showing that a latent space dimension too low or too high both hurt the model's performance. An optimal value exists: it is $L = 10$ for MNIST, Fashion-MNIST and USPS, but not for Reuters-10k, suggesting that the optimal value depends on the intrinsic dimensionality of the latent factors of variation in the data distribution. Latent quantization error, represented as a function of L on Figure 3.3, naturally varies like $\sqrt{L}$. When not mentioned, we use a latent dimension equal to 10, even if better results could be obtained by tuning the dimension for each data set (to remain in an unsupervised setting).

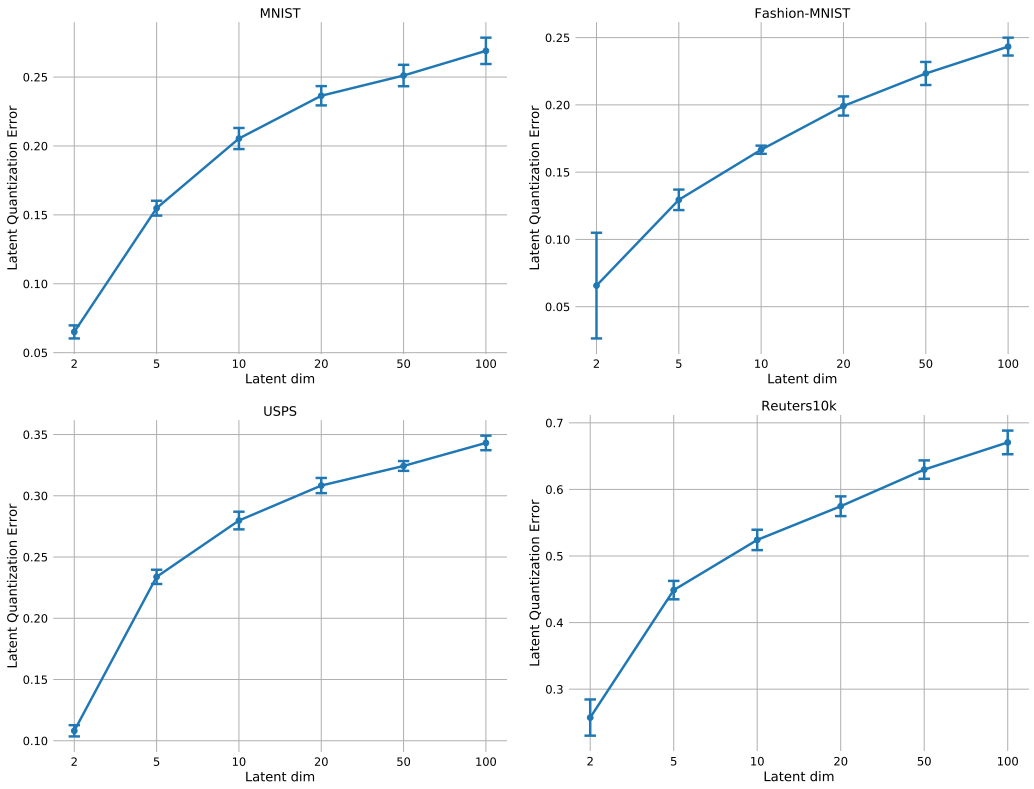


Fig. 3.3.: Latent quantization error as a function of latent space dimension.

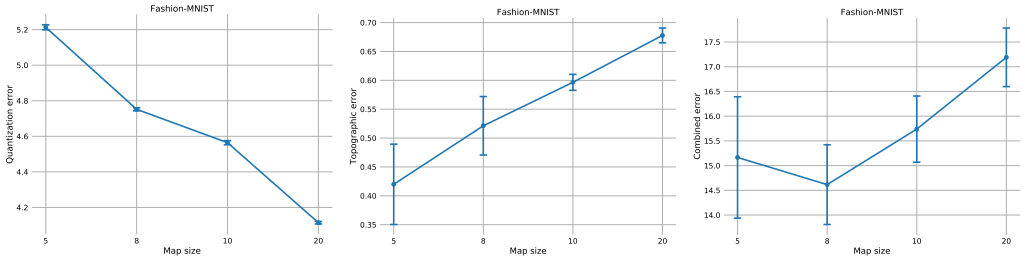


Fig. 3.4.: Quantization, topographic and combined errors as a function of the map size (5×5, 8×8, 10×10 and 20×20) for Fashion-MNIST. Quantization improves with map size while topographic error increases. Combined error acts as a trade-off and indicates 8×8 as a good compromise.

3.7.3 Map size study

This paragraph studies the influence of the SOM map size (number of neurons) on the results. The shape is kept square, with equal width and height, varying between 5 and 20 units. Metrics behave as expected, i.e. clustering purity naturally increases with map size, as there are more clusters, but NMI decreases. Quantization error improves while topographic error increases; combined error shows a minimum around size 8×8 on MNIST and Fashion-MNIST, which corresponds to the size we use throughout the paper, to compare with previous works (see Figure 3.4). On USPS and Reuters-10k, the underlying manifold seems to require fewer units. SOM entropy and class scatter index (the number of label groups formed on the map), vary proportionally to the number of units.

3.7.4 Convolutional architecture study

The fact that CNN-based deep clustering models tend to outperform similar approaches using dense AEs can be seen in [Aljalbout et al., 2018]. We compare the standard DESOM with a $[500, 500, 2000, 10]$ fully-connected encoder with a convolutional version, ConvDESOM [Forest et al., 2019a]. The ConvDESOM AE architecture is similar to the one used in [Manduchi et al., 2020]: 4 convolutional layers with $[32, 64, 128, 256]$ filters of size 3×3, and 2×2 max pooling after each convolution. We use no batch normalization and activations are basic ReLUs. We also apply it to MNIST and Fashion-MNIST data sets.

From the comparison in Table 3.6, the convolutional architecture is superior in terms of clustering purity and NMI, but slightly hurts quantization and topographic errors. On most other metrics we compared, ConvDESOM is equivalent to DESOM. As a conclusion, a convolutional AE performs better on images, but makes little

Tab. 3.6.: Comparison between DESOM and ConvDESOM in terms of purity, NMI, quantization and topographic errors, for MNIST and Fashion-MNIST.

		MNIST		
Method	Pur	NMI	QE	TE
DESOM	.934 ± .004	.658 ± .004	**5.843 ± 0.016**	**0.605 ± 0.039**
ConvDESOM	**.948 ± .004**	**.673 ± .005**	5.980 ± 0.038	**0.610 ± 0.034**
		Fashion-MNIST		
Method	Pur	NMI	QE	TE
DESOM	.751 ± .009	.541 ± .004	**4.756 ± 0.009**	**0.537 ± 0.029**
ConvDESOM	**.758 ± .004**	**.546 ± .002**	4.777 ± 0.017	**0.538 ± 0.045**

difference on toy data sets, as the fully-connected version is able to learn sufficient representations, but we believe that for larger, more complex and high-dimensional data, ConvDESOM or other architectures should produce superior results.

3.8 Initialization and pretraining

We now study the influence of initialization and pretraining. Usually, we seek a good initial solution for the model parameters and avoid local minima. In our model, two components must be initialized: AE and SOM.

3.8.1 AE pretraining

Pretraining the autoencoder consists in training it with only the reconstruction loss before performing the joint task. There are several ways to pretrain an AE: traditional end-to-end training, SDAE [Vincent et al., 2010] (also called layer-wise pretraining), RBM pretraining [Hinton and Salakhutdinov, 2006] etc. End-to-end training can be problematic because it could learn the identity function (but not an issue in case of undercomplete AEs), is less robust and prone to overfitting. Pretraining is used in most deep clustering approaches, either layer-wise [Xie et al., 2016, Yang et al., 2017b, Jiang et al., 2017], RBM [Song et al., 2014] or end-to-end [Fard et al., 2018], and improves results. We compared the two following strategies for DESOM:

- No pretraining.

- Pretraining the AE in a simple end-to-end fashion for 100 epochs using MSE reconstruction loss.

3.8.2 SOM initialization

The SOM weights are initialized using one of the following strategies:

- Random initialization: SOM weights are initialized with a random sample of encoded samples (taken without replacement).
- SOM initialization: a standard SOM is trained for 10 epochs on the encoded data set (we used the `minisom` package).

Other more sophisticated initialization schemes do exist (using for instance principal components), but we limit ourselves to these two simple strategies.

3.8.3 Initial temperature

In the case when the AE is pretrained and SOM is initialized, we may not want to disturb the map topology and only finetune the prototypes and representations. Thus, we might try to use a very small initial temperature (e.g. $T_{max} = 0.1$). We try following initial temperatures:

- $T_{max} = 0.1$ (local finetuning)
- $T_{max} = 8.0$ (self-organization across the entire map size)

To summarize, this results in 8 combinations of possible initializations. Some results are displayed in Figure 3.5. As for all other experiments, we performed 10 runs for meaningful means and standard deviations. The first observation is that SOM initialization has no effect at all on the final results in terms of clustering (purity and NMI) or SOM quality (quantization and topographic errors). Second, AE pretraining deteriorates the model's performance. The only improvement is the NMI when $T_{max} = 0.1$, but a small initial temperature naturally leads to a higher topographic error, as the prototypes are locally finetuned and global topology is not preserved well. It suggests that the best solution is found when reconstruction and SOM loss are optimized jointly from the beginning. The lowest topographic error is achieved by random AE and SOM initialization with $T_{max} = 8.0$, which is the setting we use throughout all other experiments.

To conclude, initialization and pretraining do not lead to performance improvements; in our benchmarks, none will be used. This allows to cut training time drastically,

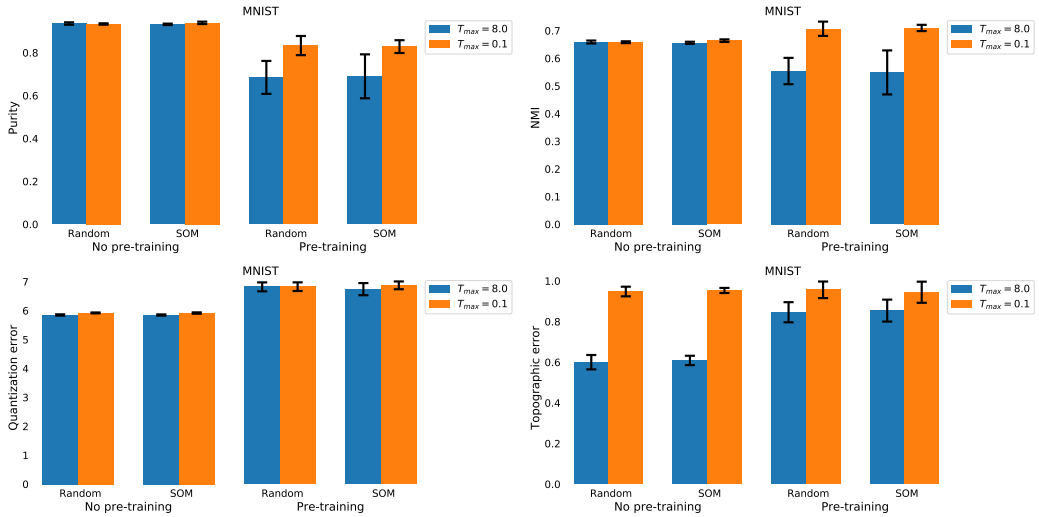


Fig. 3.5.: Performance metrics (purity, NMI, quantization and topographic errors) with different combinations of pretraining (with or without) and initialization (random or SOM).

as AE pretraining time has the same order of magnitude DESOM full joint training (about the half, see Table 3.12).

3.9 Training parameters and learning dynamics

3.9.1 Learning curves

Learning curves representing the evolution of losses and metrics during training on the training and test set (see Figure 3.6) show that our model converges, and does not overfit. To preserve space, curves are only included for the MNIST data set, and for $\gamma = 10^{-3}$, unless specified. An interesting behavior is that of topographic error, which first rapidly decreases, but then increases back until convergence (when temperature T reaches T_{min}). It shows the trade-off between self-organization and the autoencoder's reconstruction quality (very low at the beginning).

3.9.2 Latent space evolution

We visualize the evolution of latent space during training using the UMAP dimensionality reduction [McInnes et al., 2018]. We choose this method instead of the widely-used t-SNE, because it runs orders of magnitude faster (the MNIST test set

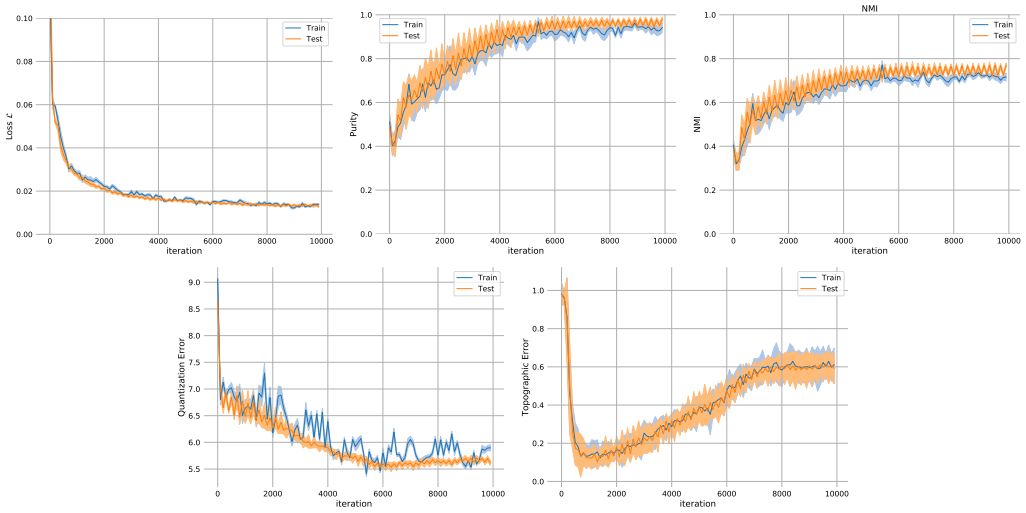


Fig. 3.6.: Learning curves of DESOM for loss, purity, NMI, quantization and topographic error for MNIST.

with 10000 points is projected in a few seconds only, compared with several minutes for t-SNE), and it is also able to effectively visualize the local and global structures of the data distribution.

Figure 3.7 displays the MNIST test set and DESOM map after 0, 10, 20 and 40 epochs. Points are colored according to their target class (digit). We clearly see that the DESOM objective (with SOM regularization) pushes points together to form clusters, and that map prototypes are self-organized in latent space. Note that we cannot interpret the SOM grid topology because of the UMAP projection, but we can still note that "similar" digits have strong connections: 4 and 9, 1 and 7, etc.

The visualizations on Figure 3.8 show the joint representation learning and self-organization: epoch after epoch, the reconstruction quality of the prototypes improves (upper part of the figure), and well-organized regions are emerging on the map, with samples of the same class gathering in the same units (bottom part).

3.9.3 SOM update interval

When using hybrid loss functions composed of terms with different optimization dynamics, it is common to update each term at different intervals. To prevent one term to prevail too much on the other, it may be optimized less frequently than the other. In our case, the SOM term is optimized faster, thus we tried to update it only every 10 or 100 SGD steps, while the reconstruction loss is updated at every step. However, no impact was observed.

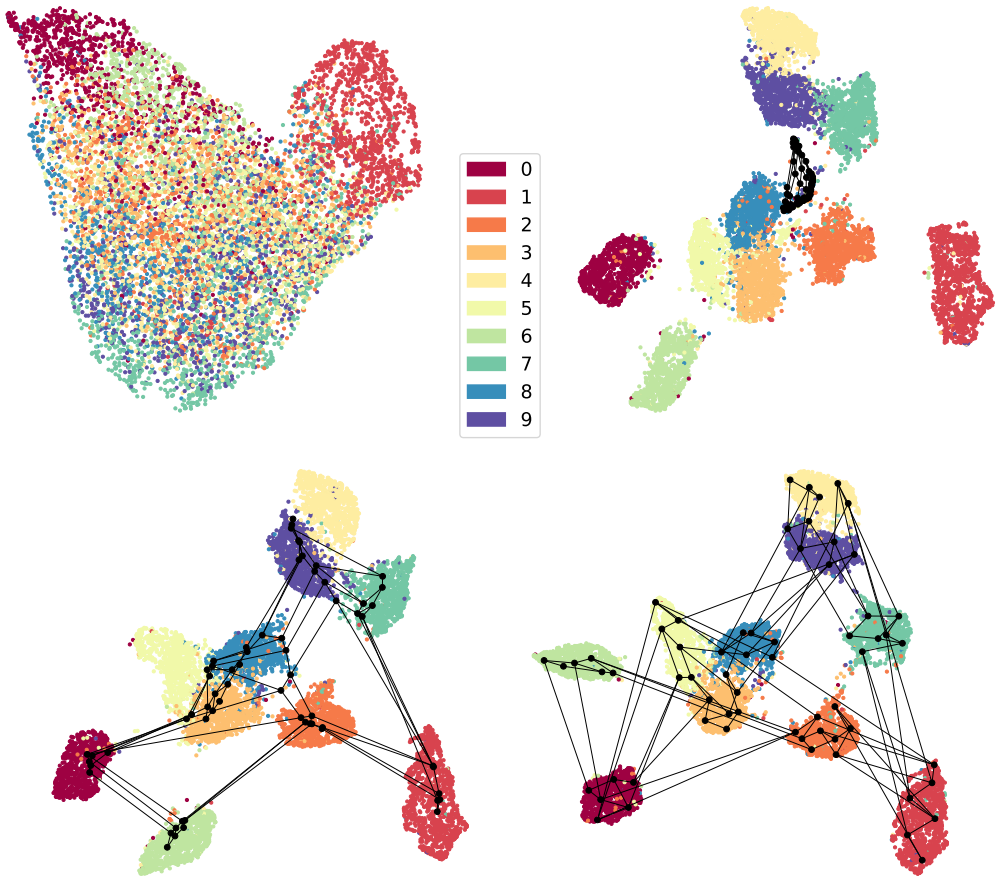


Fig. 3.7.: UMAP visualization of latent space after 0, 10, 20 and 40 training epochs.

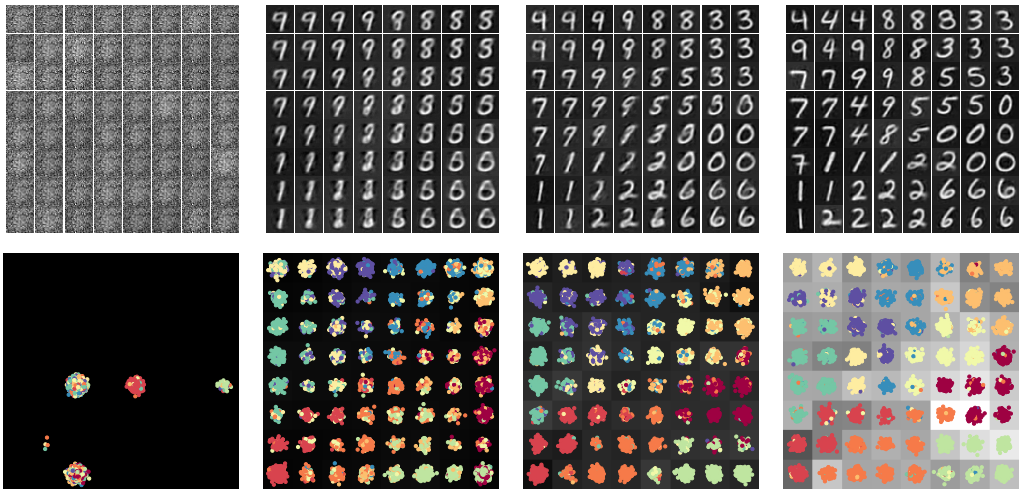


Fig. 3.8.: (Top) decoded prototypes (bottom) samples projected on U-matrix and colored by class after 0, 10, 20 and 40 training epochs.

3.9.4 Gamma γ hyperparameter

The hyperparameter γ controls the relative weight of reconstruction and SOM in the optimization of the DESOM loss function. Its influence on training can be visualized on Figure 3.9, representing $\mathcal{L}$ (total loss), $\mathcal{L}_{\text{R}}$ and $\mathcal{L}_{\text{SOM}}$ learning curves for different values of gamma. We see that the SOM loss has a higher amplitude, hence using $\gamma < 1$.

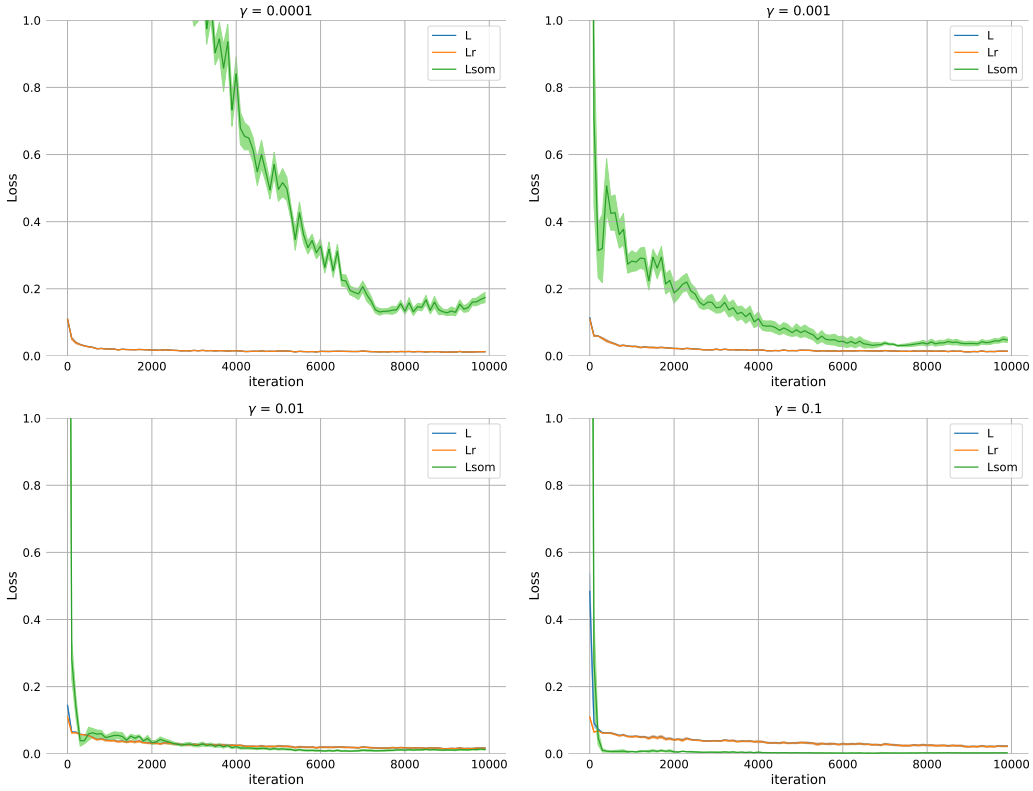


Fig. 3.9.: Evolution of DESOM losses for different values of hyperparameter γ.

If its value is too high, $\mathcal{L}_{\text{SOM}}$ is optimized too quickly, leaving a higher reconstruction loss. The comparison of both loss terms on Figure 3.10 makes clear the trade-off on γ, leading to different final values for each term. As we have seen previously, $\gamma = 10^{-3}$ is an optimal value across data sets in terms of clustering quality, as we can see on Figure 3.11 which presents the learning curves of purity and NMI for different values of γ.

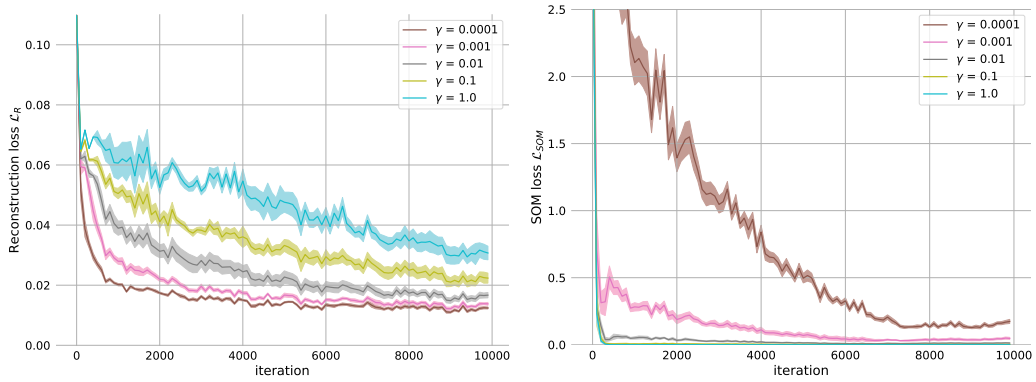


Fig. 3.10.: Evolution of reconstruction and SOM losses for different values of hyperparameter γ, which trades off between optimizing each of them.

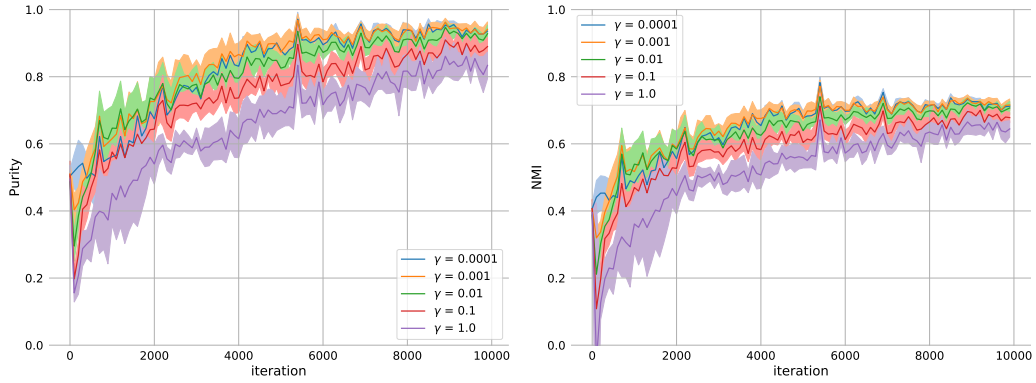


Fig. 3.11.: Evolution of DESOM clustering quality (purity and NMI) for different values of hyperparameter γ. Small values lead to better clustering, but at cost of topology.

3.9.5 Batch size

Training a SOM using minibatch stochastic gradient descent (here with the Adam optimizer) is unusual as it corresponds neither to the original stochastic Kohonen algorithm nor the batch version. However, it produces good results, even better than the standard SOM training (this was found in [Fortuin et al., 2019] and [Forest et al., 2019b]). Let us denote n_b the batch size, i.e. the number of samples in each minibatch. The Kohonen algorithm would correspond to $n_b = 1$, and the batch algorithm to $n_b = N$ with N the total number of training samples.

Throughout our experiments, we use $n_b = 256$, as it is a common practice in deep learning to use the largest possible batch size in order to exploit GPU parallelization and accelerate training (the limit being graphics memory). In this experiment, we studied the impact of batch size on DESOM training, and made n_b vary in powers

of two from 16 to 256. To be comparable, we adapted the number of iterations to keep the overall number of epochs constant (i.e. 10000 iterations for $n_b = 256$, 20000 for $n_b = 128$, etc.). Final external clustering metrics on the test set are presented in Figure 3.12, and are clearly in favor of using the largest possible batch size. On the other hand, SOM quality metrics such as quantization, topographic

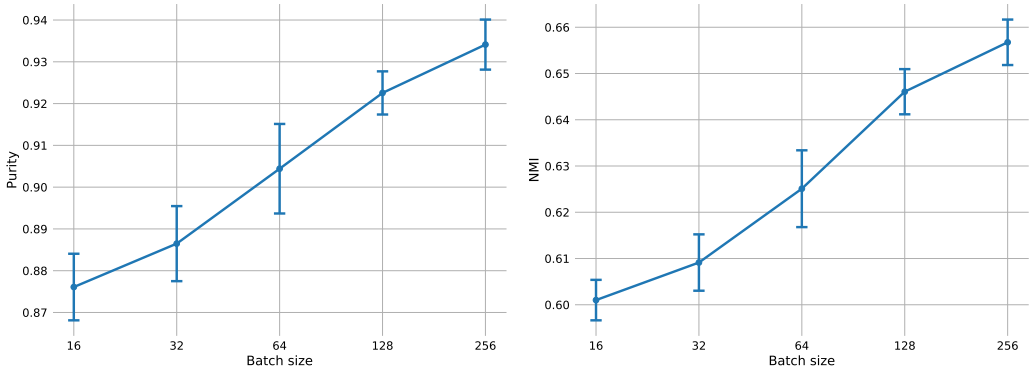


Fig. 3.12.: Purity and NMI for different batch sizes in DESOM optimization. Larger batch sizes improve clustering results overall.

and combined errors are not sensitive to the batch size (see Figure 3.13). A large

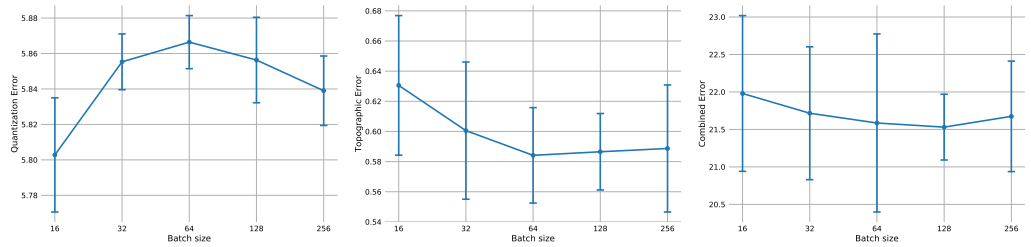


Fig. 3.13.: Quantization, topographic and combined errors for different batch sizes in DESOM. Medium values (64 or 128) seem preferable, but difference is negligible.

batch size is preferable as it improves both the solution and training speed due to parallelization.

3.10 Prototype image sharpness

As explained in the previous section, in case of image data sets, it is important for visualization that the SOM prototypes are realistic and not suffering from the blurring caused by averaging. It is clear that K-means and standard SOM will produce a certain amount of blur, because they compute a (weighted) average of the input images. Thus, the visualization will be of poor quality, even if quantization

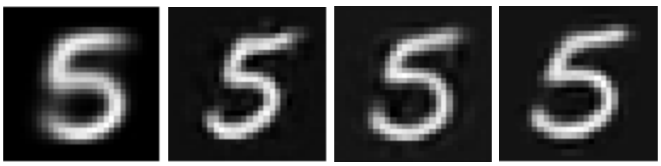


Fig. 3.14.: Examples of (reconstructed) prototype images for SOM, AE+SOM, DESOM and ConvDESOM (from left to right). With standard SOM, images are blurry due to averaging in original space. The best visual quality is obtained by ConvDESOM.

error is low (because QE is only an average Euclidean distance). DESOM also computes a weighted average, but in the latent space of autoencoders. Our intuition is that the reconstructed prototype images won't suffer as much from this issue, because autoencoders learn a *flattened* latent space where linear operations (addition, interpolation, etc.) are meaningful (for example, [Devries and Taylor, 2017] use linear interpolation and extrapolation for data augmentation and produce realistic outputs). In Figure 3.14 we represented an example of a prototype image of a 5 digit from the MNIST data set, obtained by four different models. First, a standard SOM applied on the raw pixels; then, a SOM applied on the encoded data set, using a standard AE (AE+SOM); thirdly, the DESOM model; and finally, ConvDESOM. This basic visual inspection confirms that the prototypes learned by the deep models have better quality and are less blurred than the standard SOM, and with no surprise, the best visual quality is obtained with the convolutional variant ConvDESOM.

As we want the prototype images to be realistic and as *sharp* as the original images, we need to quantify this "blurriness" or, equivalently, the "sharpness" of prototype images compared with the original samples. We have chosen a very simplistic way to compute the sharpness of an image, which is the average norm of pixel gradients, measured in two dimensions. The sharpness of a SOM is defined as the average sharpness of its prototypes. This sharpness measure can then be compared with the average sharpness of images in the original data set. We introduce the *prototype sharpness ratio* (PSR), defined as follows:

$$\text{PSR}(\{\tilde{\mathbf{m}}_k\}_1^K, \mathbf{X}) := \frac{\text{average prototype sharpness}}{\text{average dataset sharpness}} = \frac{\frac{1}{K}\sum_{k=1}^{K} ||\nabla_{\text{2D}}\tilde{\mathbf{m}}_k||_2^2}{\frac{1}{N}\sum_{i=1}^{N} ||\nabla_{\text{2D}}\mathbf{x}_i||_2^2}$$

A score lower than 1 means that the prototypes are in average blurrier than the original images; on the contrary, if it is larger than 1, they are less blurry (i.e. more crisp or noisy) than the originals. The closer the PSR is to 1, the better the score. PSR scores on benchmark image data sets for SOM, DESOM, ConvDESOM and variants are presented in Table 3.10, Section 3.11. On MNIST, SOM obtains a PSR of 0.720, while DESOM and ConvDESOM obtain respectively 1.030 and 1.045.

3.11 Benchmark results

This section is dedicated to evaluation and comparison of various clustering and SOM-based methods in a large benchmark, in terms of clustering, visualization and classification performance. As previously, all experiments are run 10 times to obtain meaningful means and standard deviations, and use the standard train/test split (see Table 3.3), always reporting results on the test set (unless specified otherwise). On the four benchmark data sets, we evaluate different aspects and tasks. First, we assess clustering quality with respect to external labels, using purity and NMI. Then, we measure clustering and self-organization through SOM's quality indices in original and latent space. More qualitatively, we also assess the visual quality of the obtained maps, using the reconstructed prototypes. Lastly, we compare different methods on a classification task, where the goal is to discriminate classes when the number of clusters equals the number of target classes, using unsupervised clustering accuracy.

3.11.1 Compared methods

The benchmarks compare following SOM-based methods:

- **SOM**: our implementation of a SOM in Keras (equivalent to DESOM with identity encoder and decoder) and trained by SGD.
- **AE+SOM**: two-stage approach, where SOM is learned on the encoded data set using a pretrained AE, resulting in the DESOM model without joint optimization of the AE and SOM.
- **DESOM-AE+SOM**: same approach, but using the AE from a pretrained DESOM model, to study the impact of the SOM-guided regularization.
- **DESOM**: Deep Embedded SOM with joint representation learning and self-organization.

The maps are always 8×8-sized, excepted in the classification task, where we also compare a one-dimensional #classes $\times$ 1 map. The AE has always the same architecture as in previous experiments, i.e. $[500, 500, 2000, 10]$ fully-connected symmetric, except in ConvDESOM. We also include K-means (from scikit-learn module), AE+K-means and DESOM-AE+K-means, with 64 prototypes, as a baseline, even if it cannot be directly compared because it lacks self-organization.

When using SOM, practitioners often need interpretable high-level regions on the map, that discriminate between different classes or behaviors of the studied phe-

nomenon. A common technique is to apply a subsequent clustering on the obtained prototypes, in order to reduce the number of clusters [Vesanto, 1999]. They will often use a hierarchical clustering (HC) algorithm, as in [Ambroise et al., 2000, Faure et al., 2017]. Thus, we evaluate the class discrimination power of the previously listed models, but without removing the topology constraint between the prototypes. In order to do so, the number of clusters must be reduced to the number of target classes. We compare three different methods:

- Perform K-means clustering on the map prototypes with $K = \#\text{classes}$.
- Perform Ward clustering (HC) on the map prototypes with #classes clusters.
- Directly train a one-dimensional, wire-shaped map of size #classes $\times$ 1.

On this task, we measure purity, NMI and unsupervised clustering accuracy, the unsupervised counterpart of supervised classification accuracy.

Results are put in perspective with baseline state-of-the-art SOM-based models (SOM-VAE, DPSOM) and deep clustering models (K-means, DEC, IDEC , DCN, DKM and VaDE) introduced in Chapter 2, keeping in mind that these are pure clustering models and do not produce a self-organized map. Values are taken directly from the papers (references in the results Table 3.7 and 3.11), and were in some cases reported for the entire data set, and not only the hold-out test set.

3.11.2 Clustering benchmark

Results of the clustering benchmark are displayed in Table 3.7. The first statement that is not surprising but clearly confirmed here, is that reducing dimensionality with neural networks improves clustering quality, also in the case of SOM. If we compare SOM vs. AE+SOM, every time there is a considerable performance gain.

Overall, AE+SOM (two-stage training) and DESOM (joint training) have similar performance across the first three data sets, with an advantage for AE+SOM. However, DESOM outperforms AE+SOM by a fair margin (approx. +3% in purity and +10% in NMI) on Reuters-10k, where the AE alone struggles to find good representations for the high-dimensional text data (the AE even decreases performance for K-means). Thus, the joint training of DESOM does not bring consistent quantitative benefits in terms of purity or NMI, as seen in deep clustering approaches, but it is at least close to the two-stage approach and much faster to train (no pretraining). Using the AE of DESOM in a two-stage approach yields similar or lower scores than joint training with DESOM. On Reuters-10k, we see that DESOM-AE has learned a better code than the standard AE.

Tab. 3.7.: Clustering performance of K-means and SOM-based models according to purity and NMI. Best performance among SOM-based models in bold underlined. Bold values correspond to results with no statistically significant difference to the best (p-value > 0.05 after pairwise t-test).

	MNIST		Fashion-MNIST	
Method	Pur	NMI	Pur	NMI
K-means ($k = 64$)	.845 ± .011	.581 ± .006	.718 ± .006	.514 ± .002
AE+K-means	.946 ± .004	.672 ± .005	.764 ± .005	.548 ± .003
DESOM-AE+K-means	.932 ± .005	.656 ± .006	.754 ± .008	.542 ± .003
SOM (8×8)	.832 ± .010	.576 ± .005	.712 ± .002	.513 ± .002
AE+SOM	**.935 ± .005**	**.666 ± .005**	**.758 ± .007**	**.542 ± .005**
DESOM-AE+SOM	**.933 ± .005**	.655 ± .005	**.756 ± .007**	**.543 ± .003**
DESOM (8×8)	**.934 ± .004**	.658 ± .004	**.751 ± .009**	**.541 ± .004**
SOM-VAE (8×8)	.868 ± .003	.595 ± .002	.739 ± .002	.520 ± .002
DPSOM (8×8)	.964 ± .001	.705 ± .001	.764 ± .003	.571 ± .001
	USPS		Reuters-10k	
Method	Pur	NMI	Pur	NMI
K-means ($k = 64$)	.858 ± .007	.598 ± .004	.895 ± .007	.439 ± .007
AE+K-means	.874 ± .007	.611 ± .006	.856 ± .016	.392 ± .014
DESOM-AE+K-means	.858 ± .010	.592 ± .009	.798 ± .028	.360 ± .018
SOM (8×8)	**.848 ± .009**	.595 ± .007	.554 ± .046	.225 ± .063
AE+SOM	**.849 ± .015**	**.611 ± .010**	.782 ± .020	.323 ± .018
DESOM-AE+SOM	**.852 ± .007**	.589 ± .007	**.799 ± .021**	**.355 ± .017**
DESOM (8×8)	**.857 ± .011**	.592 ± .010	**.808 ± .017**	**.364 ± .011**

As noted by [Fortuin et al., 2019], the SOM trained by SGD with Adam achieves much higher clustering quality than standard SOM implementations, and benefits from GPU acceleration. Thus we did not include another SOM implementation in this work.

Comparing with other SOM-based models, DESOM consistently outperforms SOM-VAE on MNIST and Fashion-MNIST according to purity and NMI, but is second to the very recent DPSOM model, achieving state-of-the-art results on both data sets. However, it is difficult to compare with these models, because they use a VAE, and the authors do not measure other metrics than purity and NMI (e.g. topographic organization). Tables 3.8 and 3.9 present quantization, topographic and combined

Tab. 3.8.: Comparison between SOM and DESOM using internal quality indices in original space. Best performance in bold underlined. Bold values correspond to results with no statistically significant difference to the best (p-value > 0.05 after pairwise t-test).

			MNIST	
Method	QE	TE	CE	TP
SOM (8×8)	**5.345 ± 0.004**	**0.518 ± 0.037**	**15.78 ± 0.662**	**-0.073 ± 0.004**
DESOM (8×8)	5.848 ± 0.016	0.597 ± 0.033	21.74 ± 0.643	-0.104 ± 0.004
			Fashion-MNIST	
Method	QE	TE	CE	TP
SOM (8×8)	**4.537 ± 0.008**	**0.477 ± 0.037**	**12.40 ± 0.654**	**-0.026 ± 0.007**
DESOM (8×8)	4.755 ± 0.007	0.536 ± 0.035	15.22 ± 0.941	-0.046 ± 0.005
			USPS	
Method	QE	TE	CE	TP
SOM (8×8)	**3.693 ± 0.005**	**0.474 ± 0.025**	**10.35 ± 0.378**	**-0.055 ± 0.007**
DESOM (8×8)	4.025 ± 0.018	0.556 ± 0.041	14.62 ± 0.649	-0.082 ± 0.005
			Reuters-10k	
Method	QE	TE	CE	TP
SOM (8×8)	42.70 ± 0.108	**0.595 ± 0.311**	**102.4 ± 21.26**	-0.206 ± 0.015
DESOM (8×8)	**41.81 ± 0.102**	**0.754 ± 0.072**	**113.8 ± 7.927**	**-0.147 ± 0.005**

errors as well as topographic product in original and latent space for all data sets. In original space, comparison between SOM and DESOM shows that generally DESOM obtain inferior quantization and topology (excepted on Reuters-10k). Joint representation learning implies a trade-off on SOM training: deep representations enable learning meaningful clusters with respect to latent factors (e.g. classes), but has an impact on the self-organization of SOM. The same metrics can be defined in latent space, to compare the AE+SOM, DESOM-AE+SOM and DESOM approaches. Here, results clearly point towards the advantage of the regularized, SOM-friendly

Tab. 3.9.: Comparison between SOM, AE+SOM and DESOM using internal quality indices in latent space. Best performance in bold underlined. Bold values correspond to results with no statistically significant difference to the best (p-value > 0.05 after pairwise t-test).

			MNIST	
Method	$\hat{\text{QE}}$	$\hat{\text{TE}}$	$\hat{\text{CE}}$	$\hat{\text{TP}}$
AE+SOM (8×8)	1.231 ± 0.029	**0.510 ± 0.047**	4.429 ± 0.287	-0.066 ± 0.002
DESOM-AE+SOM (8×8)	**0.205 ± 0.008**	**0.514 ± 0.031**	**0.713 ± 0.017**	**-0.055 ± 0.003**
DESOM (8×8)	**0.205 ± 0.008**	**0.534 ± 0.026**	**0.727 ± 0.058**	**-0.057 ± 0.005**
			Fashion-MNIST	
Method	$\hat{\text{QE}}$	$\hat{\text{TE}}$	$\hat{\text{CE}}$	$\hat{\text{TP}}$
AE+SOM (8×8)	0.960 ± 0.026	**0.532 ± 0.022**	3.973 ± 0.250	-0.059 ± 0.007
DESOM-AE+SOM (8×8)	**0.166 ± 0.003**	0.572 ± 0.037	**0.664 ± 0.041**	**-0.044 ± 0.003**
DESOM (8×8)	**0.167 ± 0.003**	**0.556 ± 0.035**	**0.661 ± 0.036**	**-0.045 ± 0.005**
			USPS	
Method	$\hat{\text{QE}}$	$\hat{\text{TE}}$	$\hat{\text{CE}}$	$\hat{\text{TP}}$
AE+SOM (8×8)	3.926 ± 0.151	0.689 ± 0.104	19.88 ± 2.737	-0.098 ± 0.007
DESOM-AE+SOM (8×8)	**0.278 ± 0.009**	**0.554 ± 0.033**	**1.174 ± 0.085**	**-0.065 ± 0.005**
DESOM (8×8)	**0.280 ± 0.007**	**0.563 ± 0.031**	**1.184 ± 0.037**	**-0.069 ± 0.003**
			Reuters-10k	
Method	$\hat{\text{QE}}$	$\hat{\text{TE}}$	$\hat{\text{CE}}$	$\hat{\text{TP}}$
AE+SOM (8×8)	30.00 ± 0.867	0.934 ± 0.017	270.7 ± 13.72	-0.146 ± 0.010
DESOM-AE+SOM (8×8)	**0.527 ± 0.017**	**0.710 ± 0.035**	**3.391 ± 0.401**	**-0.071 ± 0.007**
DESOM (8×8)	**0.524 ± 0.015**	**0.696 ± 0.065**	**3.102 ± 0.289**	**-0.069 ± 0.004**

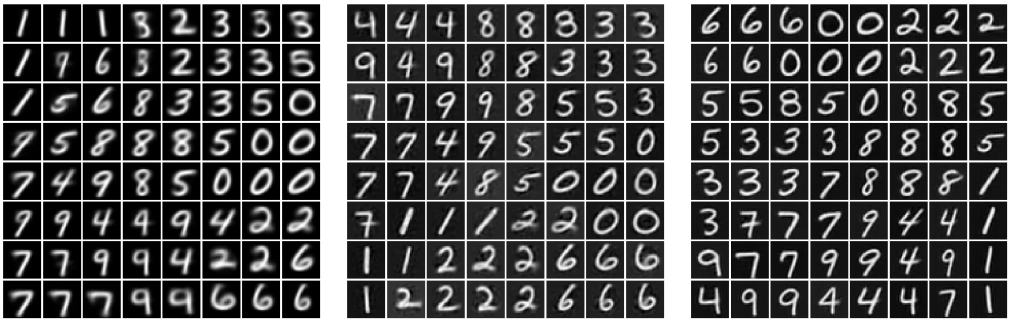


Fig. 3.15.: Prototypes (decoded) visualized on SOM, DESOM and ConvDESOM maps for MNIST.

latent space, because quantization and topology are by far superior with DESOM's autoencoder than with the non-regularized AE. Latent quantization, topographic and combined errors are similar for DESOM-AE+SOM and DESOM, but much higher for AE+SOM. Latent topographic products are closer to zero for every data set, meaning that the map is less stretched or distorted in DESOM's latent space.

3.11.3 Visualization

In this paragraph, we visualize and compare maps of image data sets obtained by SOM, DESOM and ConvDESOM. For image data sets, we can directly visualize the prototypes or reconstructed prototypes using the decoder. Maps for MNIST and Fashion-MNIST are shown on Figures 3.15 and 3.16. We clearly see the regions corresponding to different classes and smooth transitions between them. The advantage of fitting the map in latent space instead of the original high-dimensional space is visible when comparing the prototypes learned by SOM and DESOM. With SOM, they are very blurred (caused by averaging of data vectors in original space), compared with the decoded prototypes of DESOM. A few SOM prototypes are not realistic samples; however the topographic organization looks smoother with SOM (this was quantified in previous paragraphs). Visually, the best map is obtained with ConvDESOM, as the reconstruction quality is higher. In order to assess quantitatively the visual quality of prototypes, we measure the PSR for each model (see Table 3.10). A value close to 1 indicates sharpness close to the original samples, whereas a lower value points towards blurriness. SOM prototypes are blurry and obtain a low PSR, around 0.7. Models equipped with an AE obtain scores close to 1 and statistically equivalent, with an advantage for DESOM. Visualizations of larger maps are available in Appendix A.

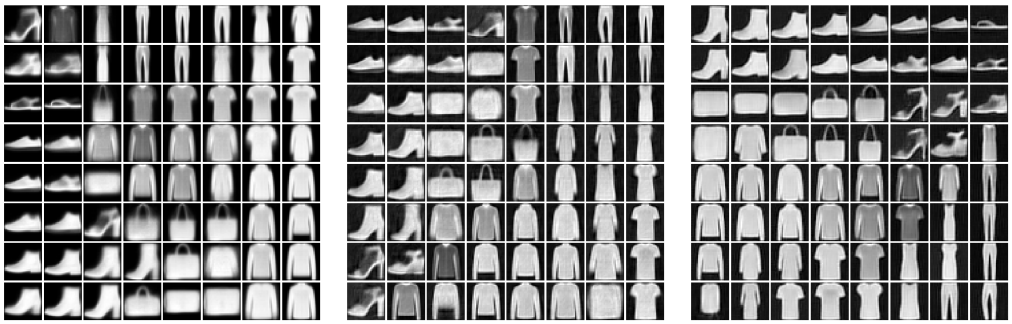


Fig. 3.16.: Prototypes (decoded) visualized on SOM, DESOM and ConvDESOM maps for Fashion-MNIST.

Tab. 3.10.: Prototype sharpness ratio of SOM, AE+SOM and DESOM variants on image data sets. Closest to 1 is best. Best performance in bold underlined. Bold values correspond to results with no statistically significant difference to the best (p-value > 0.05 after pairwise t-test).

Method	MNIST	Fashion-MNIST	USPS
SOM (8×8)	0.720 ± 0.005	0.703 ± 0.005	0.769 ± 0.005
AE+SOM (8×8)	1.083 ± 0.012	$\underline{\mathbf{0.834 \pm 0.011}}$	0.777 ± 0.016
DESOM-AE+SOM (8×8)	$\mathbf{1.034 \pm 0.011}$	$\mathbf{0.825 \pm 0.010}$	$\mathbf{1.041 \pm 0.021}$
DESOM (8×8)	$\underline{\mathbf{1.030 \pm 0.005}}$	$\mathbf{0.832 \pm 0.013}$	$\underline{\mathbf{1.027 \pm 0.020}}$
ConvDESOM (8×8)	$\mathbf{1.045 \pm 0.027}$	$\mathbf{0.829 \pm 0.009}$	-

3.11.4 Classification benchmark

The classification task consists in discriminating classes when the number of clusters equals the number of target classes. The number of clusters is reduced by applying a subsequent K-means or hierarchical clustering on top of the 8×8 map prototypes. Purity, NMI and unsupervised clustering accuracy results are displayed in Table 3.11. The HC approach, often used by practitioners, is more efficient than the K-means approach in most cases.

On MNIST and Fashion-MNIST, the DESOM+HC approach consistently outperforms every other method on all three metrics. It obtains 81% accuracy on MNIST, which challenges deep clustering methods without self-organization constraints, such as DEC. The AE+SOM+HC method is also very competitive. On Fashion-MNIST, the DESOM-based methods are clearly superior. Generally, reducing dimensionality with an AE greatly improves results, but USPS is an exception with SOM+HC performing better than AE+SOM+HC. We assume this is due to the relatively low dimension of this data set (256). However, DESOM+HC still achieves the best unsupervised clustering accuracy, which is the most important metric for this task. Lastly, on Reuters-10k the DESOM-based approaches again produce the best classification results. This time, DESOM+K-means performs best, but DESOM+HC is statistically equivalent.

These results demonstrate that DESOM's self-organizing map prior has enabled to learn a *SOM-friendly* representation that improves classification accuracy when classifying the map prototypes with HC or K-means. DESOM performs best on all four data sets is terms of purity, NMI and accuracy, the only exception being NMI on USPS.

The one-dimensional DESOM achieves decent results and even the best on Reuters-10k, showing that the very high-dimensional TF-IDF features benefit from joint learning. It even outperforms K-means, which has no topology constraint. However, such a map provides much less information on the data distribution and topology than a larger two-dimensional map.

3.11.5 Training time

We report training times of the compared methods for MNIST in Table 3.12. All models were trained on a Nvidia RTX 2080 GPU card in our lab, with a batch size of 256. Autoencoder end-to-end pretraining for 100 epochs lasts 2 minutes. The training time of SOM (our SGD-based Keras implementation) is also 2 minutes for

Tab. 3.11.: Classification performance when number of clusters equals number of classes. Best performance among SOM-based models in bold underlined. Bold values correspond to results with no statistically significant difference to the best (p-value > 0.05 after pairwise t-test).

Method	MNIST Pur	MNIST NMI	MNIST Acc	Fashion-MNIST Pur	Fashion-MNIST NMI	Fashion-MNIST Acc
K-means ($k = \#$classes)	.591 ± .026	.501 ± .020	.533 ± .038	.583 ± .018	.513 ± .012	.549 ± .040
AE+K-means	.820 ± .019	.754 ± .013	.801 ± .027	.544 ± .010	.524 ± .012	.489 ± .017
DESOM-AE+K-means	.770 ± .025	.701 ± .018	.744 ± .045	.625 ± .014	.590 ± .009	.586 ± .014
DEC [Guo et al., 2017a]	-	.837	.866	-	-	-
IDEC [Guo et al., 2017a]	-	.867	.881	-	-	-
DCN [Yang et al., 2017b]	-	.810	.830	-	-	-
DKM [Fard et al., 2018]	-	.796 ± .009	.840 ± .022	-	-	-
VaDE [Jiang et al., 2017]	-	-	.945	-	-	-
SOM (8×8)+KM	.559 ± .053	.485 ± .041	.510 ± .071	.538 ± .023	.499 ± .024	.498 ± .035
SOM (8×8)+HC	.641 ± .030	.612 ± .027	.598 ± .035	.550 ± .032	.536 ± .022	.491 ± .041
AE+SOM+KM	.773 ± .051	.728 ± .032	.728 ± .073	.485 ± .041	.461 ± .032	.439 ± .048
AE+SOM+HC	**.822 ± .024**	**.788 ± .030**	**.791 ± .026**	.528 ± .029	.550 ± .026	.480 ± .032
DESOM-AE+SOM+KM	.743 ± .049	.690 ± .033	.720 ± .057	**.588 ± .024**	.583 ± .012	.535 ± .033
DESOM-AE+SOM+HC	.747 ± .041	.681 ± .036	.721 ± .056	**.598 ± .042**	**.586 ± .036**	**.553 ± .047**
DESOM (8×8)+KM	.751 ± .048	.696 ± .036	.717 ± .065	**.587 ± .044**	.582 ± .025	**.536 ± .053**
DESOM (8×8)+HC	**.824 ± .024**	**.793 ± .025**	**.810 ± .032**	**.613 ± .035**	**.604 ± .019**	**.571 ± .036**
DESOM (#classes $\times$ 1)	.790 ± .017	.720 ± .020	.779 ± .033	.563 ± .019	.553 ± .011	.546 ± .025

Method	USPS Pur	USPS NMI	USPS Acc	Reuters-10k Pur	Reuters-10k NMI	Reuters-10k Acc
K-means ($k = \#$classes)	.703 ± .026	.585 ± .019	.660 ± .032	.647 ± .082	.373 ± .085	.589 ± .096
AE+K-means	.720 ± .033	.604 ± .026	.680 ± .063	.589 ± .044	.235 ± .046	.538 ± .041
DESOM-AE+K-means	.698 ± .027	.575 ± .024	.648 ± .032	.615 ± .067	.257 ± .062	.533 ± .069
DEC [Guo et al., 2017a]	-	.753	.741	-	.498	.737
IDEC [Guo et al., 2017a]	-	.785	.761	-	.498	.756
DCN [Yang et al., 2017b]	-	-	-	-	.760	.800
DKM [Fard et al., 2018]	-	.776 ± .011	.757 ± .013	-	.331 ± .049	.583 ± .038
VaDE [Jiang et al., 2017]	-	-	-	-	-	.798
SOM (8×8)+KM	.659 ± .029	.571 ± .016	.629 ± .043	.443 ± .070	.107 ± .099	.434 ± .072
SOM (8×8)+HC	**.711 ± .019**	**.650 ± .014**	.666 ± .024	.444 ± .028	.105 ± .041	.439 ± .030
AE+SOM+KM	.615 ± .027	.540 ± .027	.585 ± .049	.456 ± .058	.115 ± .067	.415 ± .042
AE+SOM+HC	**.673 ± .056**	.591 ± .043	**.649 ± .070**	.469 ± .057	.128 ± .058	.441 ± .071
DESOM-AE+SOM+KM	.639 ± .063	.545 ± .048	.621 ± .067	**.571 ± .050**	**.206 ± .050**	**.478 ± .061**
DESOM-AE+SOM+HC	.631 ± .041	.540 ± .035	.610 ± .040	**.536 ± .063**	**.197 ± .061**	**.467 ± .065**
DESOM (8×8)+KM	.636 ± .043	.543 ± .032	.611 ± .058	**.573 ± .059**	**.210 ± .063**	**.510 ± .073**
DESOM (8×8)+HC	**.710 ± .023**	.633 ± .017	**.698 ± .030**	**.521 ± .079**	**.186 ± .095**	**.486 ± .094**
DESOM (#classes $\times$ 1)	.683 ± .035	.556 ± .033	.649 ± .038	.661 ± .039	.322 ± .041	.596 ± .045

Tab. 3.12.: Training times of AE, SOM and DESOM on the MNIST data set (60 000 samples, batch size 256).

Model	Average training time
AE pretraining (100 epochs)	2 minutes
SOM (10 000 iterations)	2 minutes
AE+SOM (10 000 iterations)	4 minutes
DESOM (10 000 iterations)	4 minutes

10000 iterations, giving a total training time of about 4 minutes for the two-stage AE+SOM approach. This is identical to DESOM, which trains in 4 minutes. If AE pretraining was necessary for DESOM, training time would jump to a total of 6 minutes, a +50% increase. Overall, these SGD-based, GPU-accelerated methods are orders of magnitudes faster than standard SOM implementations that cannot handle data sets of this size in a reasonable time. Finally, we think that these low training times make SOM and DESOM very effective tools for surveying large, high-dimensional data sets.

3.12 Software implementations

The DESOM model was implemented using Python and the Keras deep learning framework, and is available as an open-source project[2].

3.13 Conclusion

DESOM is an unsupervised learning algorithm that jointly trains an autoencoder and the code vectors of a self-organizing map in a continuous latent space in order to survey, cluster and visualize large, high-dimensional data sets. It is one of the first members of what we could call the *deep SOM* family, along with several other recent concurrent works. Joint optimization allows to integrate dimensionality reduction to SOM learning, and to seek a *SOM-friendly* latent space that improves the performance of SOM. The model is governed by several hyperparameters impacting training and performance, in particular the trade-off between clustering and self-organization. This chapter presented experiments and visualizations in order to

[2] `https://github.com/FlorentF9/DESOM`

understand these effects. In our experiments, we found that reducing dimension with an AE vastly improves SOM clustering with respect to latent factors of variation. We also found that the learned latent space does not in general improve quantitative clustering quality compared with the representations produced by a pure AE model; however, DESOM results are on par with the two-stage approaches (AE+SOM), while requiring no pretraining at all and thus cutting down training time, which is an important criterion when quickly exploring large data sets. Training time is only a few minutes on medium-sized benchmark data sets. On a classification task where the goal is to discriminate between target classes, after post-clustering the SOM code vectors, DESOM consistently outperforms comparable models.

This chapter concludes the first part of this thesis. Until now, we have always assumed that we knew in advance the number of clusters to discover, denoted K. However, this is not the case in real life. Every parameter of a clustering needs to be selected in order to maximize the quality of the resulting solution. This process, called model selection, is far from being trivial in unsupervised learning, and is the topic of Part II.

Part II

Model selection in clustering

4 Model selection in clustering

> *Le modèle doit suivre les données et non l'inverse.*
>
> — **Jean-Paul Benzécri**
> 1973

> *All models are wrong [, but some are useful].*
>
> — **George Box**
> 1976

4.1 Introduction

In general, validating clustering solutions means evaluating results of cluster analysis in a quantitative and objective fashion [Roth et al., 2002]. This allows to perform *model selection*, in order to select the "right" number of clusters in a data set, or to tune any hyperparameter of an algorithm. Model selection is a major challenge in non-parametric clustering. There is no universally admitted way to evaluate clustering results for the obvious reason that there is no ground truth against which results could be tested, as in supervised learning. The difficulty to find a universal evaluation criterion is a direct consequence of the fundamentally ill-defined objective of clustering. In supervised learning, generalization performance can be evaluated using a hold-out set or cross-validation. This is impossible in clustering due to the lack of a proper loss function. For instance, the objective minimized in K-means cannot be used to trade off the quality of the clustering against the number of clusters, since it is a decreasing function of the latter: a high number of cluster will always yield a lower value of this objective.

To evaluate the results of a clustering algorithm, one can use so-called *clustering validity indices* (CVIs) [Milligan and Cooper, 1986, Arbelaitz et al., 2013]. In this work, we consider a clustering as being a partition of the input data set or underlying space. We will not work, for instance, on hierarchies of nested partitions (dendrograms), as done in [Carlsson and Mémoli, 2010]. Evaluation a partition can be based on two types of criteria:

- **External criteria**: a clustering solution is matched to a priori information, i.e. external information that is not contained in the data set such as ground-truth labels.
- **Internal criteria**: the quality measure is exclusively based on features inherent to the data set itself.

In a sense, external criteria are more objective than internal criteria because they incorporate no prior on the geometry of the clusters; instead, they evaluate how well the clustering was able to recover the ground-truth classes (an external classification task). However, ground-truth labels are generally unavailable in unsupervised data exploration, making internal criteria the only possible choice. Internal criteria can roughly be subdivided into two families of methods:

- Methods which assess the fit between the data and an expected structure. This is straightforward for model-based clustering, where the likelihood allows to build principled criteria. In model-free clustering, those methods rely on a combination of *between-cluster* and *within-cluster* distances [Dunn, 1974, Caliński and Harabasz, 1974, Davies and Bouldin, 1979, Rousseeuw, 1987, Ray and Turi, 1999, Desgraupes, 2013]. Between-cluster distance measures how distinct clusters are dissimilar or *far apart*, while within-cluster distance measures how elements belonging to the same cluster are similar, or the *coherence* of the cluster. Unfortunately, this incorporates a prior on the geometry of clusters.
- Methods which focus on the robustness of the solution, or the agreement of different cluster structures obtained on the same data. One approach is called cluster stability analysis [Von Luxburg, 2009], and is studied in this thesis. It relies on the principle that a good clustering solution should be a stable structure on the data distribution. Other methods compare the model with a null data set from a unimodal distribution without cluster structure (e.g. [Tibshirani et al., 2001]). Finally, combining an ensemble of several clusterings is the goal of consensus methods [Strehl and Ghosh, 2003].

The first section introduces external criteria, and the second the internal ones. Finally, the last section is about validating self-organized models.

4.2 External clustering validation

We note $\mathbf{X} = \{\mathbf{x}_i\}, i = 1 \dots N$ a data set and $\mathcal{C}_K = \{C_k\}, k = 1 \dots K$ a partition composed of K clusters. Let $|C_k|$ be the cardinality of cluster C_k. In order to define the external clustering criteria, we assume that labels are associated to each data point of $\mathbf{X}$, and compute a similarity between $\mathcal{C}_K$ and the labels. Equivalently, we can define a similarity between any two partitions $\mathcal{C}_K$ and $\mathcal{C}'_{K'}$, where the partition does not have to be ground-truth labels. This section provides formulae for every measure used in this work to compute a similarity (or distance) between two partitions. These can be broadly divided into two families: pair counting-based measures (e.g. Rand index) and information theoretic measures (e.g. mutual information).

4.2.1 Pair-counting-based measures

An important class of criteria for comparing clusterings is based on counting pairs of samples on which two clusterings agree or disagree.

Contingency matrix

The measures for comparing two clusterings $\mathcal{C}_K$ and $\mathcal{C}'_{K'}$ can be obtained from the *contingency* matrix. A contingency matrix reports the intersection cardinality for every pair of cluster assignments. It provides sufficient statistics for all clustering metrics where the samples are i.i.d. and one does not need to account for some instances not being clustered. It is a $K \times K'$ matrix, whose kk'-th element $n_{kk'}$ is the number of points belonging to cluster C_k in clustering $\mathcal{C}_K$ and to cluster $C'_{k'}$ in clustering $\mathcal{C}'_{K'}$, i.e. $n_{kk'} = |C_k \cap C'_{k'}|$:

$\mathcal{C}_K$ \ $\mathcal{C}'_{K'}$	C'_1	C'_2	$\dots$	$C'_{K'}$	Sums
C_1	n_{11}	n_{12}	$\dots$	$n_{1K'}$	a_1
C_2	n_{21}	n_{22}	$\dots$	$n_{2K'}$	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
C_K	n_{K1}	n_{K2}	$\dots$	$n_{KK'}$	a_K
Sums	b_1	b_2	$\dots$	b_s	

Any pair of samples falls into one of four cases:

1. N_{11} the number of pairs that are in the same cluster under both $\mathcal{C}_K$ and $\mathcal{C}'_{K'}$
2. N_{00} the number of pairs in different clusters under both $\mathcal{C}_K$ and $\mathcal{C}'_{K'}$

3. N_{10} the number of pairs in the same cluster under $\mathcal{C}_K$ but not under $\mathcal{C}'_{K'}$
4. N_{01} the number of pairs in the same cluster under $\mathcal{C}_K$ but not under $\mathcal{C}'_{K'}$

Rand index (RI)

The Rand index [Rand, 1971] is defined as

$$RI = \frac{N_{00} + N_{11}}{N_{00} + N_{11} + N_{01} + N_{10}} = \frac{N_{00} + N_{11}}{\binom{N}{2}}.$$

The Rand index is a value between 0 and 1, a value of 0 indicating that the clusterings do not agree on any pair of points, and a value of 1 indicating an agreement for every pair of points.

Adjusted Rand index (ARI)

The Adjusted Rand index is a version of the RI that is corrected by the expected index value. It was introduced in two different variants in [Morey and Agresti, 1984] and [Hubert and Arabie, 1985]. In this work, we use the first version of [Hubert and Arabie, 1985] that expresses as follows:

$$\overbrace{ARI}^{\text{Adjusted Index}} := \frac{\overbrace{\sum_{ij}\binom{n_{ij}}{2}}^{\text{Index}} - \overbrace{[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}]/\binom{n}{2}}^{\text{Expected Index}}}{\underbrace{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}]}_{\text{Max Index}} - \underbrace{[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}]/\binom{n}{2}}_{\text{Expected Index}}}$$

or using different notations

$$\text{ARI} = \frac{2(N_{00}N_{11} - N_{01}N_{10})}{(N_{00} + N_{01})(N_{01} + N_{11}) - (N_{00} + N_{10})(N_{10} + N_{11})}.$$

It can yield negative values if the index is lower than the expected value.

Fowlkes-Mallows

The Fowlkes and Mallows index [Fowlkes and Mallows, 1983] is defined as

$$\text{FM} := \frac{N_{11}}{\sqrt{(N_{11} + N_{10})(N_{00} + N_{01})}}.$$

Jaccard

$$\text{JACC} := \frac{N_{11}}{N_{01} + N_{10} + N_{11}}.$$

Purity

To compute the purity of a clustering, each cluster is assigned to the class which is most frequent in the cluster, and then the accuracy of this assignment is measured by counting the number of correctly assigned points and dividing by the total number of points. Formally:

$$\text{Pur}(\mathcal{C}_K, \mathcal{C}'_{K'}) := \frac{1}{N} \sum_{k=1}^{K} \max_{k'=1\ldots K'} |C_k \cap C'_{k'}|.$$

High purity is easy to achieve when the number of data points per cluster is small; in particular, purity is equal to 1 if all points get their own cluster. Thus, purity cannot be used to trade off the validity of the clustering against the number of clusters.

Unsupervised clustering accuracy

Accuracy in supervised classification corresponds to the number of samples assigned to the correct class divided by the size of the data set. It can also be used in clustering (i.e. unsupervised classification) as an external quality measure if labels are available and if the number of clusters is equal to the number of classes (or to compare partitions where $K = K'$). It consists in the accuracy of the resulting classification using the best one-to-one mapping between clusters and class labels. Denoting this mapping by π, the expression of unsupervised clustering accuracy is

$$\text{Acc}(\mathcal{C}_K, \mathcal{C}'_K) := \frac{1}{N} \max_{\pi} \sum_{k=1}^{K} |C_k \cap C'_{\pi(k)}|.$$

The best mapping can be found using the Hungarian assignment algorithm, also known as the Kuhn-Munkres algorithm [Kuhn, 1955, Munkres, 1957].

4.2.2 Information theoretic measures

Information theoretic measures are based on information theory. They calculate *how much information* knowing one partition gives on the other. See for instance [Vinh et al., 2010] for a survey.

Mutual Information and variants

We define $\mathrm{I}(\mathcal{C}_K, \mathcal{C}'_{K'})$ the mutual information between the clusterings $\mathcal{C}_K, \mathcal{C}'_{K'}$ to be equal to the mutual information between the associated random variables:

$$\mathrm{I}(\mathcal{C}_K, \mathcal{C}'_{K'}) := \sum_k \sum_{k'} P(C_k \cap C'_{k'}) \log \frac{P(C_k \cap C'_{k'})}{P(C_k)P(C'_{k'})} = \sum_k \sum_{k'} \frac{|C_k \cap C'_{k'}|}{N} \log \frac{N|C_k \cap C'_{k'}|}{|C_k||C'_{k'}|}.$$

Mutual information measures the information that $\mathcal{C}_K$ and $\mathcal{C}'_{K'}$ share: it tells how much knowing one of these clusterings reduces our uncertainty about the other. Let us call $\mathrm{H}(\mathcal{C}_K)$ the entropy of partition $\mathcal{C}_K$:

$$\mathrm{H}(\mathcal{C}_K) = -\sum_k P(C_k) \log P(C_k) = -\sum_k \frac{|C_k|}{N} \log \frac{|C_k|}{N}.$$

Entropy is always positive. It takes value 0 only when there is no uncertainty, namely when there is a single cluster. We can now define the different variants of the Normalized Mutual Information (NMI):

$$\mathrm{NMI}_1(\mathcal{C}_K, \mathcal{C}'_{K'}) := \frac{\mathrm{I}(\mathcal{C}_K, \mathcal{C}'_{K'})}{\max\left(\mathrm{H}(\mathcal{C}_K), \mathrm{H}(\mathcal{C}'_{K'})\right)} \qquad \mathrm{NMI}_2(\mathcal{C}_K, \mathcal{C}'_{K'}) := \frac{\mathrm{I}(\mathcal{C}_K, \mathcal{C}'_{K'})}{\min\left(\mathrm{H}(\mathcal{C}_K), \mathrm{H}(\mathcal{C}'_{K'})\right)}$$

$$\mathrm{NMI}_3(\mathcal{C}_K, \mathcal{C}'_{K'}) := \frac{\mathrm{I}(\mathcal{C}_K, \mathcal{C}'_{K'})}{\sqrt{\mathrm{H}(\mathcal{C}_K)\mathrm{H}(\mathcal{C}'_{K'})}} \qquad \mathrm{NMI}_4(\mathcal{C}_K, \mathcal{C}'_{K'}) := \frac{2\mathrm{I}(\mathcal{C}_K, \mathcal{C}'_{K'})}{\mathrm{H}(\mathcal{C}_K) + \mathrm{H}(\mathcal{C}'_{K'})}$$

$$\mathrm{NMI}_5(\mathcal{C}_K, \mathcal{C}'_{K'}) := \frac{\mathrm{I}(\mathcal{C}_K, \mathcal{C}'_{K'})}{\mathrm{H}(\mathcal{C}_K, \mathcal{C}'_{K'})}.$$

Finally, the adjusted mutual information (AMI) is defined as

$$\mathrm{AMI}(\mathcal{C}_K, \mathcal{C}'_{K'}) := \frac{\mathrm{I}(\mathcal{C}_K, \mathcal{C}'_{K'}) - \mathbb{E}[\mathrm{I}(\mathcal{C}_K, \mathcal{C}'_{K'})]}{\max\left(\mathrm{H}(\mathcal{C}_K), \mathrm{H}(\mathcal{C}'_{K'})\right) - \mathbb{E}[\mathrm{I}(\mathcal{C}_K, \mathcal{C}'_{K'})]}$$

where $\mathbb{E}[\mathrm{I}(\mathcal{C}_K, \mathcal{C}'_{K'})]$ is the expected mutual information between two clusterings $\mathcal{C}_K, \mathcal{C}'_{K'}$ as defined in [Vinh et al., 2010].

Variation of Information

The variation of information (VI) is defined as

$$\mathrm{VI}(\mathcal{C}_K, \mathcal{C}'_{K'}) := \mathrm{H}(\mathcal{C}_K, \mathcal{C}'_{K'}) - \mathrm{I}(\mathcal{C}_K, \mathcal{C}'_{K'}),$$

and its normalized version (NVI) as

$$\mathrm{NVI}(\mathcal{C}_K, \mathcal{C}'_{K'}) := 1 - \frac{\mathrm{I}(\mathcal{C}_K, \mathcal{C}'_{K'})}{\mathrm{H}(\mathcal{C}_K, \mathcal{C}'_{K'})}.$$

Unlike all previously introduced measures, the VI/NVI measure dissimilarity instead of similarity.

Information Distance

The expression of Information Distance (ID) is:

$$\mathrm{ID}(\mathcal{C}_K, \mathcal{C}'_{K'}) := \max(\mathrm{H}(\mathcal{C}_K), \mathrm{H}(\mathcal{C}'_{K'})) - \mathrm{I}(\mathcal{C}_K, \mathcal{C}'_{K'}).$$

The normalized variant, NID, is both a distance and a normalized measure:

$$\mathrm{NID}(\mathcal{C}_K, \mathcal{C}'_{K'}) := 1 - \frac{\mathrm{I}(\mathcal{C}_K, \mathcal{C}'_{K'})}{\max(\mathrm{H}(\mathcal{C}_K), \mathrm{H}(\mathcal{C}'_{K'}))}.$$

4.3 Internal clustering validation

In the first paragraph, we define the within- and between-cluster distances, which are the building blocks of clustering validity criteria and algorithms.

4.3.1 Cluster distances

Let $d(\cdot, \cdot)$ be the distance function between two elements (generally the squared Euclidean distance). The *centroid* $\mathbf{c}_k$ of cluster C_k is the arithmetic mean of the vectors in C_k: $\mathbf{c}_k := \frac{1}{|C_k|} \sum_{\mathbf{x} \in C_k} \mathbf{x}$. Within-cluster and between-cluster distances are defined respectively in Tables 4.1 and 4.2. Each metric has different properties and sensitiveness to noise and outliers. As we said, in clustering we seek a small within-cluster distance and a large between-cluster distance. We will first discuss

some properties of the within-cluster distances, and then between-cluster distances.

Tab. 4.1.: Within-cluster distances.

Diameter	$S_M(C) = \max_{\mathbf{x},\mathbf{x}' \in C_k} d(\mathbf{x}, \mathbf{x}')$
Average distance	$S_A(C) = \frac{1}{\lvert C\rvert(\lvert C\rvert-1)} \sum_{i \neq i', \mathbf{x}_i, \mathbf{x}_{i'} \in C} d(\mathbf{x}_i, \mathbf{x}_{i'})$
Centroid distance	$S_C(C) = \frac{1}{\lvert C\rvert} \sum_{\mathbf{x} \in C} d(\mathbf{x}, \mathbf{c}_k)$
Nearest neighbor distance	$S_{NN}(C) = \frac{1}{\lvert C\rvert} \sum_{i, \mathbf{x}_i \in C} \min_{i' \neq i, \mathbf{x}_{i'} \in C} d(\mathbf{x}_i, \mathbf{x}_{i'})$

Tab. 4.2.: Between-cluster distances.

Single linkage	$D_S(C_k, C_l) = \min_{(\mathbf{x},\mathbf{x}') \in C_k \times C_l} d(\mathbf{x}, \mathbf{x}')$
Complete linkage	$D_M(C_k, C_l) = \max_{(\mathbf{x},\mathbf{x}') \in C_k \times C_l} d(\mathbf{x}, \mathbf{x}')$
Average linkage	$D_A(C_k, C_l) = \frac{1}{\lvert C_k\rvert\lvert C_l\rvert} \sum_{(\mathbf{x},\mathbf{x}') \in C_k \times C_l} d(\mathbf{x}, \mathbf{x}')$
Centroid linkage	$D_C(C_k, C_l) = d(\mathbf{c}_k, \mathbf{c}_l)$
Ward linkage	$D_W(C_{k1} \cup C_{k2}, C_l)$
	$= \sqrt{\alpha_1 D_W(C_{k1}, C_l)^2 + \alpha_2 D_W(C_{k2}, C_l)^2 - \beta D_W(C_{k1}, C_{k2})^2}$
	where $\alpha_1 = \frac{\lvert C_{k1}\rvert+\lvert C_l\rvert}{\lvert C_{k1}\rvert+\lvert C_{k2}\rvert+\lvert C_l\rvert}$, $\alpha_2 = \frac{\lvert C_{k2}\rvert+\lvert C_l\rvert}{\lvert C_{k1}\rvert+\lvert C_{k2}\rvert+\lvert C_l\rvert}$, $\beta = \frac{\lvert C_{k1}\rvert+\lvert C_{k2}\rvert}{\lvert C_{k1}\rvert+\lvert C_{k2}\rvert+\lvert C_l\rvert}$
	and $D_W(\{\mathbf{x}_i\}, \{\mathbf{x}_j\}) = \lvert\lvert\mathbf{x}_i - \mathbf{x}_j\rvert\rvert_2$ (Euclidean distance)

The diameter is the maximum distance between two points of a cluster. Average within-cluster distance is the average pairwise distance between elements. Centroid distance, the average distance between elements of a cluster and its centroid. These distances will be low when the cluster is compact. Nearest neighbor distance only takes into account the distance between neighboring points; as a consequence, in order to have a low S_{NN}, the cluster must have no "holes", but not necessarily be compact. To take a concrete example, for a ring-shaped cluster (and generally for clusters where points are concentrated along a connected manifold), the average and centroid distances will typically be large but the nearest neighbor distance will be small. Among between-cluster distances, which express how two clusters C_k and C_l are dissimilar, an important notion is sensitiveness to noise. A metric is sensitive to noise if small perturbations, e.g. adding new points to the data set, can change its value by a great amount. Average linkage is the average pairwise distance between points belonging to the two clusters, so its value is high when their elements are all far from each other in average. As it is an average, it is not sensitive to local perturbations. Centroid linkage is simply the distance between the two centroids, so is it much faster to compute and is also insensitive to noise. However, it

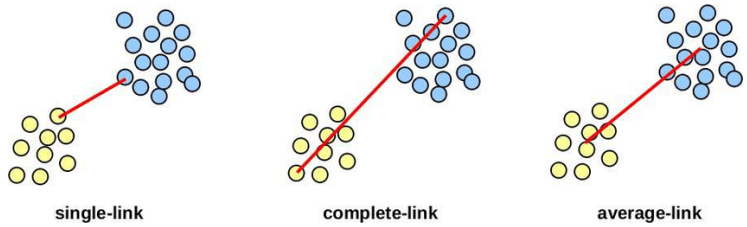


Fig. 4.1.: Single, complete and average linkage cluster distances.

ignores the cluster geometries. Single linkage is the distance between the two closest points belonging to a different cluster. Maximizing this distance guarantees that the clusters are well-separated (no points belonging to different clusters are close). This distance is very noise-sensitive, because adding a single point can completely change it. Similarly, complete linkage is the maximum pairwise distance between points of the two clusters. A low value means that points of both clusters are all relatively close together. However, it is very sensitive to outliers, because an outlier far from a cluster will result in a very high distance. These distances are used in agglomerative hierarchical clustering algorithms to merge clusters with minimal between-cluster distances in a bottom-up approach. Single, complete and average linkages are illustrated on Figure 4.1.

4.3.2 Distance-based internal criteria

Calinski-Harabasz index

The Calinski-Harabasz index [Caliński and Harabasz, 1974] is the ratio of the between-cluster *dispersion* of the data set, with the average cluster cohesion, measured via the centroid distance:

$$\mathrm{CH}(\mathcal{C}_K) := \frac{N-K}{K-1} \frac{\sum_{k=1}^{K} |C_k| d(\mathbf{c}_k, \bar{\mathbf{c}})}{\sum_{k=1}^{K} |C_k| S_C(C_k)}.$$

where $\bar{\mathbf{c}} := \frac{1}{N} \sum_{\mathbf{x} \in \mathbf{X}} \mathbf{x}$ is the global centroid of the data set. A good clustering in the sense of Calinski-Harabasz must maximize the value of this index.

Davies-Bouldin index

The Davies-Bouldin index [Davies and Bouldin, 1979] measures the validity of a clustering by computing the average maximum ratio between cluster cohesion

(generally using centroid distance) and separation (using centroid linkage) for each cluster:

$$\mathrm{DB}(\mathcal{C}_K) := \frac{1}{K}\sum_{k=1}^{K}\max_{l\neq k}\frac{S_C(C_k)+S_C(C_l)}{D_C(C_k,C_l)}.$$

A low value of the index indicates a better clustering quality, i.e. the elements of all clusters are close to their centroids and the centroids are well separated. Thus, it assumes that the data density is a decreasing function of distance to centroid, as in spherical or ellipsoidal clusters. For instance, a ring-shaped cluster is not adapted to Davies-Bouldin and will yield a high value of this index.

Dunn index

The Dunn index [Dunn, 1974] corresponds to the ratio between the minimal between-cluster distance (generally using single linkage) and the maximal within-cluster distance (generally using the diameter). Its expression is:

$$\mathrm{Dunn}(\mathcal{C}_K) := \frac{\min_{k\neq l} D_S(C_k,C_l)}{\max_k S_D(C_k)}.$$

It privileges compact (small value of the denominator) and well-separated (large value of the numerator) clusters. Unlike Davies-Bouldin, it must be maximized.

Ray-Turi index

The Ray-Turi index [Ray and Turi, 1999] is also a ratio index, between the average centroid distance and the minimum distance between the centroids of two distinct clusters:

$$\mathrm{RT}(\mathcal{C}_K) := \frac{\sum_{k=1}^{K} S_C(C_k)}{\min_{k\neq l} d(\mathbf{c}_k,\mathbf{c}_l)}.$$

Naturally, it needs to be minimized: a small numerator ensures compact clusters, and a large denominator enforces separation of clusters.

Silhouette index

The silhouette value [Rousseeuw, 1987] of an object is a measure of how similar this object is to its own cluster (cohesion) compared to other clusters (separation). For any data point $\mathbf{x}$, let a be the average distance of $\mathbf{x}$ with all other points within

the same cluster, and b be the lowest average distance of $\mathbf{x}$ to all points in any other cluster, of which it is not a member. The silhouette of $\mathbf{x}$ is defined as:

$$\text{Sil}(\mathbf{x}) := \frac{b - a}{\max\{a, b\}}.$$

We have $-1 < \text{Sil}(\mathbf{x}) < 1$. Silhouette is close to 1 when the point is very similar to its own cluster compared to other clusters. The silhouette coefficient of a cluster C_k is the average silhouette of the data points within the cluster:

$$\text{Sil}(C_k) := \frac{1}{|C_k|} \sum_{\mathbf{x} \in C_k} \text{Sil}(\mathbf{x}).$$

Finally, the silhouette of a clustering $\mathcal{C}_K$ can be defined as

$$\text{Sil}(\mathcal{C}_K) := \frac{1}{K} \sum_{k=1}^{K} \text{Sil}(C_k).$$

The maximum value of the index is used to determine the optimal clustering.

Wemmert-Gancarski index

The Wemmert-Gancarski index [Desgraupes, 2013] is built using for each input $\mathbf{x}_i$ the ratio between the distance to its own centroid $\mathbf{c}_k$ and the distance to the nearest centroid of a different cluster:

$$R_i := \frac{d(\mathbf{x}_i, \mathbf{c}_k)}{\min_{l \neq k} d(\mathbf{x}_i, \mathbf{c}_l)}.$$

Then, the ratios are combined for each cluster by taking either 0 or their complement to 1:

$$J_k := \max\{0, 1 - \frac{1}{|C_k|} \sum_{i, \mathbf{x}_i \in C_k} R_i\}.$$

Finally, the Wemmert-Gancarski index is the average of these values:

$$\text{WG}(\mathcal{C}_K) := \frac{1}{N} \sum_{k=1}^{K} |C_k| J_k.$$

A value close to 1 indicates the best partition. It is one of the best-performing indices compared in the next chapter.

Xie-Beni index

The Xie-Beni index [Xie and Beni, 1991] is defined as the ratio between the average centroid distance, and the minimum single linkage distance between two clusters:

$$\mathrm{CH}(\mathcal{C}_K) := \frac{1}{N} \frac{\sum_{k=1}^{K} |C_k| S_C(C_k)}{\min_{k \neq l} D_S(C_k, C_l)}.$$

Its value is to be minimized.

This list has included all indices used throughout this thesis, but is of course no exhaustive (see for instance [Arbelaitz et al., 2013, Desgraupes, 2013] for extensive reviews). Additionally, it has been shown in [Kaczynska et al., 2020] that many traditional CVIs can yield improved results by using a different decision rule, such as the maximum increment (or decrement) instead of the minimum (respectively the maximum) value. In particular, authors demonstrate that such alternate decision rules are more robust across different degrees of cluster separation (as measured by the method introduced in [Qiu and Joe, 2006]). In a similar spirit, the slope statistic [Fujita et al., 2014] is based on Silhouette and works better in presence of a dominant cluster.

4.3.3 Validation of model-based clustering

In model-based clustering, several criteria based on the likelihood are used. The most common may be the BIC (Bayesian Information Criterion) [Schwartz, 1978]. The BIC score penalizes the likelihood of the model by its complexity. Given a model with parameter θ, with a complexity (or *size*) C, the BIC is defined as

$$\mathrm{BIC}(\mathbf{X};\ \theta) := C \log N - 2 \log L(\mathbf{X};\ \theta)$$

where L is the likelihood of observing the data set $\mathbf{X}$ under this model. Other widely used criteria are the AIC (Akaike Information Criterion) [Akaike, 1973] and the Integrated Completed Likelihood (ICL) [Biernacki et al., 2000]. A study on several criteria for Gaussian mixtures is available in [Hu and Xu, 2003]. Another strategy is to avoid fixing the number of clusters by using Bayesian non-parametric approaches [Ferguson, 1973], such as the Dirichlet Process, which introduce a non-parametric prior for the number of clusters.

4.3.4 Other internal criteria

Many criteria are not based explicitly on the geometry of clusters, but on some type of statistical robustness. For instance, the *gap statistic* [Tibshirani et al., 2001] measures the strength of the clustering obtained on the data relative to the clustering obtained on data from a null distribution that does not contain any cluster structure. It is used to find the number of clusters K. It is calculated by generating D data sets following a null reference distribution (e.g. uniform noise), producing the partitions $\{\mathcal{C}_K^d\}_1^D$. Then, the sums of within-cluster distances $W(\mathcal{C}_K) := \sum_{k=1}^K \frac{1}{2|C_k|} S_A(C_k)$ (where the d is generally the squared Euclidean distance) are being computed. The gap statistic expresses as

$$\text{Gap}(\mathcal{C}_K) := \frac{1}{D}\sum_{d=1}^{D} \log W(\mathcal{C}_K^d) - \log W(\mathcal{C}_K).$$

The optimal number of clusters is chosen via finding the smallest K such that

$$\text{Gap}(\mathcal{C}_K) \geq \text{Gap}(\mathcal{C}_{K+1}) - s_{K+1}$$

where $s_K := \sigma_K\sqrt{1+1/D}$ and σ_K is the biased standard deviation estimate of the $\log W(\mathcal{C}_K^d)$. Other methods are CLEST [Dudoit and Fridlyand, 2002] the prediction strength [Tibshirani and Walther, 2005]. These methods are different but related to the *stability* of a clustering, which is the central concept of the next chapter.

Another family of methods to determine the number of clusters automatically are iterative refinement methods. This family of methods consists in an iterative procedure, wrapping around a standard algorithm (often K-means):

1. Start from a solution with few clusters (e.g. $K = 2$).
2. Evaluate the model to determine if K should be increased. Evaluation may take the form of an internal criterion, or a statistical test of unimodality.
3. Increase K repeatedly as long clusters can be split.

Representatives of these methods are X-means [Pelleg and Moore, 2000] (based on the BIC criterion), G-means [Hamerly and Elkan, 2004] (using a statistical test of Gaussianity, testing if each cluster comes from a Gaussian distribution), and PG-means [Feng and Hamerly, 2007]. Drawbacks of these methods is the assumptions behind statistical test (Gaussianity, spherical clusters, etc.), and the weakness of tests in high dimensions, requiring to use univariate projections (e.g. projection on each coordinate or on the direction of highest variance). The dip-means algorithm [Kalogeratos and Likas, 2012] makes no assumption of Gaussian clusters and works

with pairwise distance matrices. It is based on the dip test [Hartigan and Hartigan, 1985]. Skinny-dip [Maurus and Plant, 2016] improves on this method in presence of noisy distributions. [Hess and Duivesteijn, 2019] tests for unimodality by deriving a concentration inequality, and propose a SpecialK algorithm for selecting K in spectral clustering.

4.4 Validation of self-organized models

This section is about performance metrics for the evaluation of topographic map algorithms, in particular SOM [Forest et al., 2020b]. In every application, practitioners need to know whether they can *trust* the resulting mapping, and perform model selection to select algorithm parameters (e.g. the map size, learning rate and number of iterations). Concretely, two questions need to be answered:

- Do the SOM code vectors approximate well the data distribution?
- Does the mapping preserve neighborhood relationships between the map and the original data space?

Quantitative evaluation of SOM is a subset of clustering validation, which is typically achieved using validity indices. While these also apply to SOM, they ignore the topology of the map, only answering the first question. The second question brings in the additional challenge of assessing their topology. The problem of assessing SOM performance has already been tackled quite thoroughly in literature, giving birth to a family of quality indices incorporating neighborhood constraints, qualified as *topographic* indices [Kiviluoto, 1996, Polzlbauer, 2004]. Concerning the related but more general topic of evaluating quality in DR, see for instance [Lee and Verleysen, 2009].

While open-source software for SOM are available in various languages (see Section 1.3 in Chapter 1), implementations for quality indices are almost impossible to find. This is the issue we aim to solve in this work: after a survey of existing SOM performance metrics, we implemented them in Python, one of the most popular languages for data mining today, and provide them as an open-source library, `SOMperf`[1] [Forest, 2020].

An overview of all metrics implemented in `SOMperf` is depicted in Figure 4.2. These can be categorized into two families:

[1] `https://github.com/FlorentF9/SOMperf`

1. Clustering metrics. Any clustering quality measure that relies solely on the prototype vectors and not on their topological organization. This encompasses all quality indices used in clustering literature.
2. Topographic metrics. Under this term, we coin quality measures that, on the contrary, assess the topological organization of the model. Some indices also evaluate the clustering quality, but we call it topographic as soon as it incorporates the map topology. In particular, they must detect neighborhood violations such as *foldings*.

On another level, we can classify them into two well-known families, depending on the use of ground-truth labels:

1. Internal indices, using only intrinsic properties of the model and the data.
2. External indices, relying on external ground-truth class labels to evaluate results.

For instance, quantization error falls into the clustering metric category (as it measures how SOM cluster centers fit the data distribution, without using any topology information) and is an internal index (not depending on external labels). On the other side, the Class Scatter Index is a topographic metric and an external index, as it measures how ground-truth class labels are organized into groups of neighboring map units. We emphasize that all clustering indices introduced before this section also apply to self-organized models, thus we will only present new and specific indices.

4.4.1 Internal validation

Quantization error

Quantization error is the average error made by projecting data on the SOM, as measured by euclidean distance, i.e. the mean euclidean distance between a data sample and its best-matching unit:

$$\mathrm{QE}(\{\mathbf{m}_k\}, \mathbf{X}) := \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{x}_i - \mathbf{m}_{b_i}||_2.$$

Topographic error

Topographic error [Kiviluoto, 1996] assesses the self-organization of a SOM model. It is calculated as the fraction of samples whose best and second-best matching

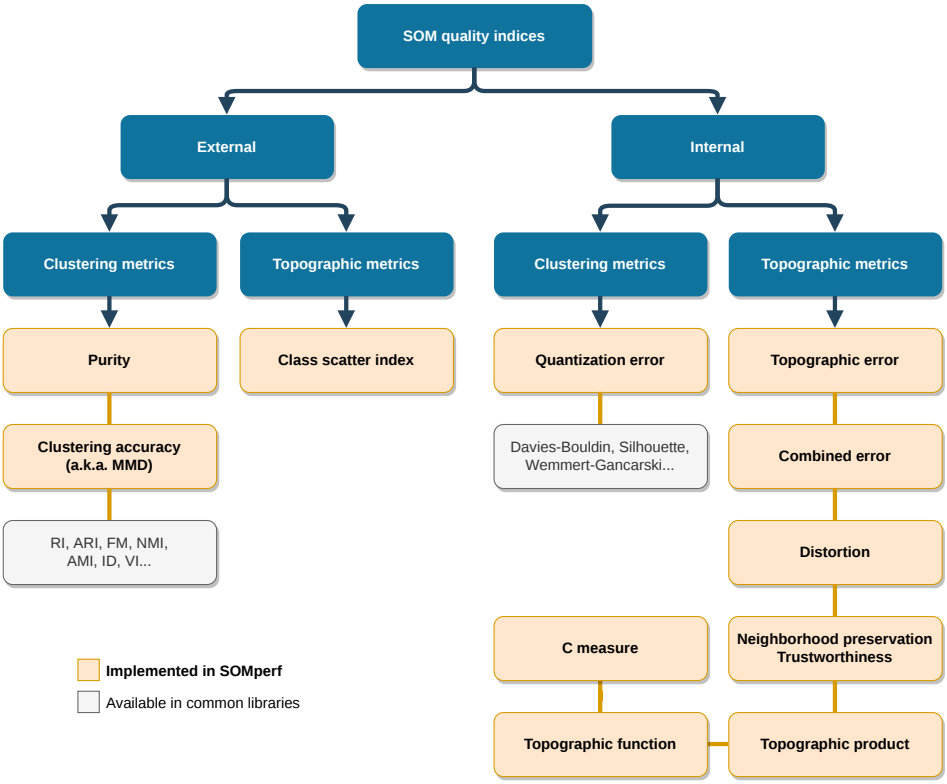


Fig. 4.2.: SOM performance metrics can be classified into external (label-based) or internal indices, and based on whether they evaluate topology (topographic metrics) or not (clustering metrics).

units are not neighbors on the map. In other words, this error quantifies the smoothness of projections on the self-organized map. Using the notation b_i^k for the k-th best-matching units of $\mathbf{x}_i$, we define the topographic error:

$$\text{TE}(\{\mathbf{m}_k\}, \mathbf{X}) := \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{\delta(b_i^1, b_i^2)>1}.$$

Combined error

Combined error [Kaski and Lagus, 1996] is an error measure that combines and extends quantification and topographic errors. Its computation is more complex than the previous indices. For a given data sample $\mathbf{x}_i$, we first compute its two best matching units b_i^1 and b_i^2. Then, we compute a sum of euclidean distances from $\mathbf{x}_i$ to the second BMU's prototype vector $\mathbf{m}_{b_i^2}$, starting with the distance from $\mathbf{x}_i$ to $\mathbf{m}_{b_i^1}$, and thereafter following a shortest path until $\mathbf{m}_{b_i^2}$, *going only through neighboring units on the map*. Let p be a path on the map of length $P \geq 1$, from $p(0) = b_i^1$ to $p(L) = b_i^2$, such that $p(k)$ and $p(k+1)$ must be neighbors for $k = 0 \dots P-1$. The distance along the shortest path on the map is computed as:

$$\text{CE}_i := ||\mathbf{x}_i - \mathbf{m}_{b_i^1}||_2^2 + \min_p \sum_{k=0}^{P-1} ||\mathbf{m}_{p(k+1)} - \mathbf{m}_{p(k)}||_2^2.$$

Finally, combined error (CE) is the average of this distance over the input samples:

$$\text{CE}(\{\mathbf{m}_k\}, \mathbf{X}) := \frac{1}{N} \sum_{i=1}^{N} \text{CE}_i.$$

Neighborhood preservation and trustworthiness

The neighborhood preservation and trustworthiness [Venna and Kaski, 2001] measure how the projection preserves neighborhoods in the input (respectively output) space by ranking the k-nearest neighbors (k-NN) of each sample before and after projection. For a given k, each sample contributes negatively by the difference between its rank and k. See Figure 4.3 for an illustration. An important issue with implementing neighborhood preservation and trustworthiness is handling ties in the projected k-NN. As the SOM projections are discrete, all samples projected onto the same map unit will have a distance equal to 0. In the continuous input space, exact ties are very unlikely but still possible. Four methods to tackle this problem are exposed in [Polzlbauer, 2004]. We adopt a weighted averaging approach as

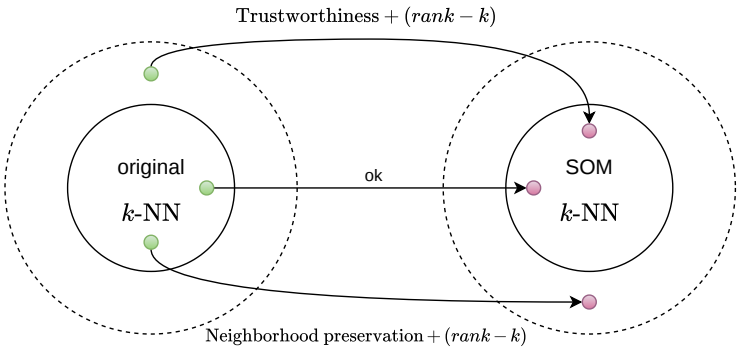


Fig. 4.3.: Illustration of neighborhood preservation and trustworthiness measures.

it is reasonable and deterministic. We include all ties in the set of k-NN, possibly producing more than k neighbors (in particular in the discrete projected space). To stay in the $[0, 1]$ range, every error term is weighted by:

- For trustworthiness: the ratio between the number of elements in the original k-NN (most often, exactly k), and the number of elements in the projected k-NN (most often, larger than k).
- For neighborhood preservation: the inverse of this ratio.

Topographic product

The topographic product (TP) [Bauer et al., 1992] measures the preservation of neighborhood relations between input space and the map. It depends only on the prototype vectors and map topology, and is able to indicate whether the dimension of the map is appropriate to fit the data set, or if it introduced neighborhood violations, induced by *foldings* of the map. We note d the euclidean distance in input space, and δ the topographic distance on the map. The computation of TP starts by defining two ratios between the distance of a prototype j to its k-th nearest neighbor on the map $|C_k|^\delta(j)$, and to its k-th nearest neighbor in input space $|C_k|^d(j)$:

$$Q_1(j,k) := \frac{d\left(\mathbf{m}_j, \mathbf{m}_{|C_k|^\delta(j)}\right)}{d\left(\mathbf{m}_j, \mathbf{m}_{|C_k|^d(j)}\right)},\ Q_2(j,k) = \frac{\delta\left(j, |C_k|^\delta(j)\right)}{\delta\left(j, |C_k|^d(j)\right)}.$$

Naturally, we always have $Q_1 \geq 1$ and $Q_2 \leq 1$. The ratios are combined into a product in order to obtain a symmetric measure and mitigate local magnification factors:

$$P_3(j,k) := \left[\prod_{l=1}^{k} Q_1(j,l)Q_2(j,l)\right]^{\frac{1}{2k}}.$$

Finally, TP is obtained by taking the logarithm and averaging over all map units and neighborhood orders:

$$\text{TP}(\{\mathbf{m}_k\}) := \frac{1}{K(K-1)} \sum_{j=1}^{K} \sum_{k=1}^{K-1} \log P_3(j,k). \tag{4.1}$$

TP < 0 indicates the map dimension is too low to correctly represent the data set; TP $= 0$ means the dimension is adequate; and TP > 0 indicates a dimension too high and neighborhood violations. However, as the TP only uses the map prototypes, it is unable to distinguish between foldings of a non-linear data manifold, and foldings due to neighborhood violations. As a consequence, it is limited to linear data manifolds.

Topographic function

The topographic function (TF) [Villmann et al., 1994] intends to overcome the limitation of the topographic product, by distinguishing between natural foldings of a non-linear data manifold and incorrect foldings due to neighborhood violations. The main difference is that is uses not only the prototype vectors, but the receptive fields of each unit c, defined as $R_c = \{\mathbf{x} \in \mathbf{X} \mid \underset{k}{\text{argmin}} ||\mathbf{x} - \mathbf{m}_k||_2^2 = c\}$. For each unit c and integer $k \in \{1, \ldots, K\}$, the number of units having adjacent receptive fields and a distance to c on the map larger than k, is computed:

$$f_c(k) := |\{c' \in \{1, \ldots, K\} \mid R_c \cap R_{c'} \neq \emptyset \ \wedge \ \delta(c, c') > k\}|.$$

The topographic function is defined by summing over all map units:

$$\text{TF}(k) := \sum_{c=1}^{K} f_c(k). \tag{4.2}$$

A normalization is necessary to compare maps of different sizes, replacing k by k/δ_{max} (the maximum distance on the map) and dividing by $K(K - 3^p)$ where p is the number of dimensions of the SOM (generally $p = 2$). In practice, the receptive fields can be easily estimated without computing the full Voronoi tessellation, by building a connectivity matrix connecting each pair of units that are the BMU and second-BMU of a given data point.

Kruskal-Shepard error

Kruskal-Shepard error, introduced for multi-dimensional scaling [Kruskal, 1964], measures the preservation of pairwise distances between two different spaces, and was used for the SOM in [Elend and Kramer, 2019]. It is computed by the squared Frobenius norm between the pairwise distance matrix of the data set, and the distance matrix between units on the SOM. Both distance matrices are scaled to the $[0, 1]$ range. We consider the normalized error, by dividing it by $N(N-1)$:

$$\text{KSE}(\{\mathbf{m}_k\}, \mathbf{X}) := \frac{1}{N(N-1)} ||\mathbf{D}^{\mathbf{X}} - \mathbf{D}^{\text{SOM}}||_F^2$$
$$\text{where } \mathbf{D}_{ij}^{\mathbf{X}} := \frac{||\mathbf{x}_i - \mathbf{x}_j||_2^2}{\max\limits_{i',j'} ||\mathbf{x}_{i'} - \mathbf{x}_{j'}||_2^2} \text{ and } \mathbf{D}_{ij}^{\text{SOM}} := \frac{\delta(b_i, b_j)}{\max\limits_{k,l} \delta(k, l)}.$$

C measure

The C measure [Goodhill and Sejnowski, 1996] measures neighborhood preservation between two spaces similarly to the Kruskal-Shepard error, but using the element-wise products between pairwise distances in the input and output space:

$$\text{C}(\{\mathbf{m}_k\}, \mathbf{X}) := \sum_{i=1}^{N} \sum_{j<i} d(\mathbf{x}_i - \mathbf{x}_j)\ \delta(b_i, b_j).$$

This quantity must be maximized, meaning that both distances must take large values at the same time, or in other words, if two points are far apart in the input space, they must also be far apart in the output space.

4.4.2 External validation

Class scatter index

The class scatter index (CSI), introduced in [Elend and Kramer, 2019], is an external SOM quality index that measures the scattering of ground-truth class labels on the map. It is based on the idea that on a *good* map, classes should be distributed into few distinct groups of neighboring units, and not scattered all over the map. This is important for interpretation, because it allows to associate map areas with particular classes. In practice, it computes the number of *groups of neurons* on the map, where for a given class, a group is defined as a set of neighboring neurons that have at

least one sample belonging to that class assigned to them. Finally, CSI is the average number of groups over all classes.

4.4.3 Software implementations

We have gathered and implemented all these metrics into a Python module, `SOMperf`[2] [Forest, 2020, Forest et al., 2020b]. It depends on the libraries `numpy`, `scipy`, `pandas` and `scikit-learn`. `SOMperf` is divided into following sub-modules:

- `metrics`: performance metric functions.
  - `external`: external indices.
  - `internal`: internal indices.
- `utils`: utility functions.
  - `neighborhood`: neighborhood kernel functions (Gaussian, window, etc.). Only used in `distortion` for now.
  - `topology`: distance functions on grid topologies (rectangular, hexagonal, etc.).

Usage examples and experiments are deferred to Appendix B.

4.5 Conclusion

The first chapter of this part on model selection in clustering reviewed the standard methods used in literature. These can be divided into external criteria, comparing a clustering with a ground-truth partition, and internal criteria, relying only on the data itself. The latter are the only choice when labels are unavailable. In the last section, we reviewed various performance metrics for SOM and introduced the `SOMperf` Python module, enabling practitioners to easily evaluate their models. Most internal CVIs optimize a certain combination of between-cluster distances (i.e. cluster separation) and within-cluster distances (i.e. cluster coherence) However, this makes strong geometrical assumptions, such as compact or spherical clusters, on the solution. Other approaches exist, but none is fully convincing. This why many practitioners still use heuristic qualitative tricks such as looking at the elbow curve. In the next chapter, we explore an elegant set of methods called stability analysis, based on the statistical robustness of the solution, and propose a novel principle and effective criterion for selecting the number of clusters in an unlabeled data set.

[2] `https://github.com/FlorentF9/SOMperf`

5 Selecting the number of clusters with a stability trade-off

This chapter is based on the contribution:

- Mourer, A., Forest, F., Lebbah, M., Azzag, H., & Lacaille, J. (2020). Selecting the Number of Clusters K with a Stability Trade-off: an Internal Validation Criterion. `https://arxiv.org/abs/2006.08530`

A second line of work of this thesis is to tackle model selection in clustering, in particular selecting the number of clusters K. We have chosen to follow an elegant and promising approach, yet not very successful in practice: cluster stability analysis. In this chapter, we introduce a novel stability-based method, called *Stadion* (stability difference criterion) [Mourer et al., 2020].

5.1 Cluster stability analysis

The previously introduced criteria incorporate strong assumptions on the geometry of clusters (e.g. compact, spherical clusters) or on the underlying distribution, which are specific to the algorithm or to an application. Hence, there is a need for a general, model-agnostic evaluation method. Clustering stability has emerged as a principle stating that "to be meaningful, a clustering must be both *good* and the only *good* clustering of the data, up to small perturbations. Such a clustering is called stable. Data that contains a stable clustering is said to be clusterable" [Meilă, 2018]. Hence, a clustering algorithm should discover stable structures in the data. In statistical learning terms, if data sets are repeatedly sampled from the same underlying distribution, an algorithm should find similar partitions. As we do not have access to the data-generating distribution in model-free clustering, perturbed data sets are obtained either by sampling or injecting noise into the original data.

Stability analysis for clustering validation is a long-established technique. It can be traced back as far as 1973 [Strauss et al., 1973], took off with influential works [Ben-

Hur et al., 2002, Lange et al., 2004] and from there has drawn increasing attention culminating with [Ben-David et al., 2006, Ben-David et al., 2007, Ben-David and Von Luxburg, 2008] and [Von Luxburg, 2009]. Albeit significant theoretical efforts, few empirical studies have been conducted. Each study focuses on specific aspects of clustering stability. For example, [Levine and Domany, 2001, Dudoit and Fridlyand, 2002, Ben-Hur et al., 2002, Lange et al., 2004] investigated perturbation by random subsampling of the original data set without replacement. Stability in a model-based framework was studied in [Kerr and Churchill, 2001]. Perturbation by random projections [Smolkin and Ghosh, 2003] and random noise [Fridlyand and Dudoit, 2001, Möller and Radke, 2006] were also considered.

5.2 Definitions and limitations

Clustering stability is analyzed in a standard statistical learning setting. A data set $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ consists in N independent and identically distributed (i.i.d.) samples, drawn from a data-generating distribution $\mathcal{P}$ on an underlying space $\mathcal{X}$. Formally, a clustering algorithm $\mathcal{A}$ takes as input the data set $\mathbf{X}$, some parameter $K \geq 1$, and outputs a clustering $\mathcal{C}_K = \{C_1, \ldots, C_K\}$ of $\mathbf{X}$ into K disjoint sets. Thus, a clustering can be represented by a function $\mathbf{X} \to \{1, \ldots, K\}$ assigning a label to every point of the input data set. Some algorithms can be extended to construct a partition of the entire underlying space. This partition is represented by an extension operator function $\mathcal{X} \to \{1, \ldots, K\}$ (e.g. for center-based algorithms, we compute the distance to the nearest center).

Let $\mathbf{X}$ and $\mathbf{X}'$ be two different data sets consisting in N independent and identically distributed (i.i.d.) samples, drawn from the same distribution $\mathcal{P}$. A clustering algorithm $\mathcal{A}$ takes as input the data set $\mathbf{X}$, some parameter $K \geq 1$, and outputs a clustering $\mathcal{C}_K = \{C_1, \ldots, C_K\}$ of $\mathbf{X}$ into K disjoint sets. Note $\mathcal{C}_K$ and $\mathcal{C}'_K$ their respective clusterings. Let s be a similarity measure such that $s(\mathcal{C}_K, \mathcal{C}'_K)$ measures the agreement between the two clusterings. Possible choices for this measure are detailed below. Then, for a given sample size N, the stability of a clustering algorithm $\mathcal{A}$ is defined as the expected similarity between two clusterings $\mathcal{C}_K$, $\mathcal{C}'_K$ on different data sets $\mathbf{X}$ and $\mathbf{X}'$, sampled from the same distribution $\mathcal{P}$,

$$\text{Stab}(\mathcal{A}, K) := \mathbb{E}_{\mathbf{X}, \mathbf{X}' \sim \mathcal{P}^N} \left[s(\mathcal{C}_K, \mathcal{C}'_K) \right]. \tag{5.1}$$

The expectation is taken with respect to the i.i.d. sampling of the sets from $\mathcal{P}$. This quantity is unavailable in practice, as we have a finite number of samples, so it needs

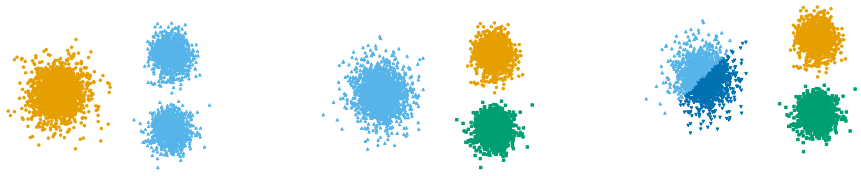
to be estimated empirically. Various methods have been devised to estimate stability using perturbed versions of $\mathbf{X}$.

The first methods used in literature are based on resampling the original data set, with or without replacement (splitting in half [Strauss et al., 1973], subsampling [Ben-Hur et al., 2002], cross-validation [Wang, 2010], bootstrapping [Falasconi et al., 2010, Fang and Wang, 2012], jackknife [Yeung et al., 2001]...). Another method consists in adding random noise either to the original data points [Möller and Radke, 2006] or to their pairwise distances (additive or multiplicative) [Bilu and Linial, 2012, Awasthi et al., 2012, Dutta et al., 2017, Balcan and Liang, 2016, Balcan et al., 2020]. For high-dimensional data, other alternatives are random projections or randomly adding or deleting variables [Strauss et al., 1973]. Once the perturbed data sets are generated, there are several ways to compare the resulting clusterings. With noise-based methods, it is possible to compare the clustering of the original data set (reference clustering) with the clusterings obtained on perturbed data sets, or to compare only clusterings obtained on the latter. With sampling-based methods, we can compare overlapping subsamples on data points where both clusterings are defined [Falasconi et al., 2010], or compare clusterings of disjoint subsamples (using for instance an extension operator or a supervised classifier to transfer labels from one sample to another [Lange et al., 2004]). Finally, to compute a similarity score between two partitions, common choices are the ARI [Falasconi et al., 2010, Zhao et al., 2011], FM, Jaccard [Ben-Hur et al., 2002], NMI, Minimum Matching Distance [Lange et al., 2004], or VI.

Before discussing in details the mechanisms of stability, we introduce a trivial example to illustrate its main issue: it cannot detect in general whenever K is too small. Consider the example presented in Figure 5.1 with three clusters, two of them closer to each other than to the third one. On any sample from such a distribution, as soon as we have a reasonable amount of data, K-means with $K = 2$ always constructs the solution separating the left cluster from the two right clusters. Consequently, it is stable despite $K = 2$ being the wrong number of clusters. This situation was pointed out in [Ben-David et al., 2006].

Discussion

Stability is determined by the number of data points changing clusters. In the case of algorithms that minimize an objective function (e.g. center-based or spectral clustering), two different sources of instability have been identified [Von Luxburg, 2009]. First, *jittering* is caused by data points changing side at cluster boundaries after perturbation. Therefore, strong jitter is produced when a cluster boundary cuts through high-density regions. Second, *jumping* refers to the algorithm ending



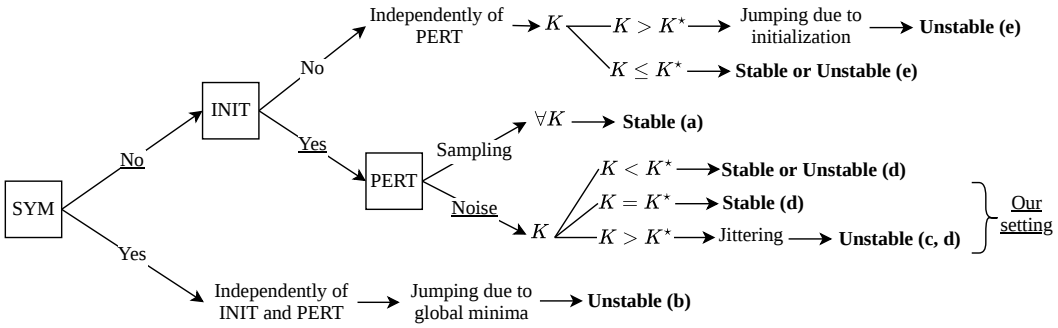
(a) $K = 2$ (stable) **(b)** $K = 3$ (stable) **(c)** $K = 4$ (unstable, jittering)

Fig. 5.1.: Example data set with three clusters. The labels correspond to the K-means clustering result for $K = 2$, 3 and 4. K-means is stable even if the number of clusters is too small.

up in different local minima. The most important cause of jumping is initialization (see Figure 5.6 for an example). Another cause is the existence of several global minima of the objective function on the underlying distribution. This happens only if there are perfect symmetries in the distribution (see Figure 5.5), which is extremely unlikely for real-world data sets.

However, practitioners mainly use algorithms with consistent initialization strategies. For instance with K-means, we keep the best trial over a large number of runs and use the K-means++ seeding heuristic [Arthur and Vassilvitskii, 2007]. This initialization tends to make K-means deterministic and its effectiveness has been proven in practice. Thus, it is different from the initialization proposed in [Von Luxburg, 2009, Bubeck et al., 2012] which allows jumping to occur whenever $K > K^\star$, where $K^\star$ is the true number of clusters. Throughout this work, we consider a setting with large enough sample size, without perfect symmetries and with effective initialization. Thus, we do not consider jumping as the main source of instability even when $K > K^\star$, and rather believe that jittering plays a major role. It captures useful information about a clustering, i.e. densities at boundaries, and also seems fundamental and related to supervised learning. As a consequence, we need a perturbation process that produces jittering. Unfortunately, as soon as N is reasonably large, resampling methods become trivially stable whenever there is a single global minimum [Ben-David et al., 2006, Von Luxburg, 2009]. See Figure 5.7 for an example where sampling methods fail. We summarize important results in the diagram Figure 5.2.

To conclude, for K-means in our setting, the perturbation process causes jittering and more rarely jumping (in experiments, we seldom observed jumping when K is too large), enabling stability to indicate whenever K is too large. On the other hand, stability cannot in general detect when K is too small. Despite a lack of theoretical



Justified by or related to:
(a) **Theorem 10** (Stability theorem) [1]
Lemma 1 (Stability and global optima of the objective function) [3]
(b) **Theorem 15** (Instability from symmetry) [1]
Lemma 1 (Stability and global optima of the objective function) [3]
(c) **Theorem 4** (High instability implies cut in high density region) [2]
Conclusion 3 (Instable clusterings) [3]
(d) **Conjecture 4** (Stable clusterings) [3]
Conclusion 5 (Stability of idealized K-means detects whether K is too large) [3]
(e) **Conjecture 8** (Stability of the actual K-means algorithm) [3]

Notations:
SYM: Symmetries in the data distribution
INIT: Effective initialization scheme
PERT: Perturbation process

References:
[1] Ben-David and von Luxburg (2006)
[2] Ben-David and von Luxburg (2008)
[3] von Luxburg (2010)

Fig. 5.2.: Diagram explaining the various sources of instability in different settings for K-means with large sample size, assuming $K \ll N$ and the underlying distribution has $K^\star$ well-separated clusters that can be represented by K-means. We consider no symmetries, effective initialization and noise-based perturbation, thus instability (due to jittering) arises when K is too large, and sometimes when K is too small whenever cluster boundaries are in high-density regions.

guarantees, concepts should apply to other algorithms. In order to overcome this limitation of stability, we introduce a novel concept of *within-cluster stability*.

5.3 Between-cluster and within-cluster stability

A clustering algorithm applied with the same parameters to perturbed versions of a data set should find the same structure and obtain similar results. The stability principle described by Equation 5.1 relies on between-cluster boundaries and we thus call it *between-cluster stability*. Therefore, it cannot detect structure within clusters. In Figure 5.1, $K = 2$ is stable, whereas one cluster contains two sub-clusters. This sub-structure cannot be detected by between-cluster stability alone. Obviously, this implies that stability is unable to decide whether a data set is clusterable or not (i.e. when $K^\star = 1$), which is a severe limitation. For this very reason, we introduce a second principle of *within-cluster stability*: clusters should not be composed of several sub-clusters. This implies the absence of stable structures inside any cluster. In other words, any partition of a cluster should be unstable. The combination of these two principles leads to a new definition of a clustering:

Definition 1. *Clustering: A clustering is a partitioning of data into groups (a.k.a. clusters) so that the partition is stable, and within each cluster, there exists no stable partition.*

Then, a clustering should have a high between-cluster stability and a low within-cluster stability. Despite their apparent simplicity, implementing these principles is a difficult task. As seen in the last section, between-cluster stability can be estimated in many different ways, however not all are effective. On the other hand, within-cluster stability is a challenging quantity to define and estimate. We propose a method to estimate both quantities, and then we detail and discuss our choices.

5.4 Stadion: a novel stability-based validity index

Let $\{\mathbf{X}_1, \dots, \mathbf{X}_D\}$ be D perturbed versions of the data set obtained by adding random noise to the original data set $\mathbf{X}$. Between-cluster stability of algorithm $\mathcal{A}$ with parameter K estimates the expectation in Equation 5.1 by the empirical mean of the similarities s between the reference clustering $\mathcal{C}_K = \mathcal{A}(\mathbf{X}, K)$ and the clusterings of the perturbed data sets,

$$\text{Stab}_{\text{B}}(\mathcal{A}, \mathbf{X}, \mathcal{C}_K, K) := \frac{1}{D} \sum_{d=1}^{D} s\left(\mathcal{C}_K, \mathcal{A}(\mathbf{X}_d, K)\right). \tag{5.2}$$

Since s is a similarity measure, this quantity needs to be maximized, and conversely with a dissimilarity measure. In order to define within-cluster stability, we need to assess the presence of stable structures inside each cluster. To this aim, we propose to *cluster again* the data within each cluster of $\mathcal{C}_K$. Formally, let $\mathbf{\Omega} \subset \mathbb{N}^*$ be a set of numbers of clusters. The k-th cluster in the reference clustering is noted C_k, its number of elements N_k, and $\mathcal{Q}_{K'}^{(k)} = \mathcal{A}(C_k, K')$ denotes a partition of C_k into K' clusters. Within-cluster stability of algorithm $\mathcal{A}$ is defined as

$$\text{Stab}_{\text{W}}(\mathcal{A}, \mathbf{X}, \mathcal{C}_K, K, \mathbf{\Omega}) := \sum_{k=1}^{K} \left(\frac{1}{|\mathbf{\Omega}|} \sum_{K' \in \mathbf{\Omega}} \text{Stab}_{\text{B}}(\mathcal{A}, C_k, \mathcal{Q}_{K'}^{(k)}, K') \right) \times \frac{N_k}{N}. \tag{5.3}$$

As a good clustering is unstable within each cluster, this quantity needs to be minimized. Hence, we propose to build a new validity index combining between-cluster and within-cluster stability. A natural choice is the difference between both

quantities. We call this index *Stadion*, standing for *stability difference criterion* and for the sake of brevity, $\mathcal{A}$, K and $\mathbf{X}$ are omitted in the notations:

$$\text{Stadion}(\mathcal{C}_K, \mathbf{\Omega}) := \text{Stab}_{\text{B}}(\mathcal{C}_K) - \text{Stab}_{\text{W}}(\mathcal{C}_K, \mathbf{\Omega}). \tag{5.4}$$

Since we use an effective initialization scheme, the same partition $\mathcal{C}_K$ is used in both terms of Equation 5.4. Thus, Stadion evaluates the stability of an algorithm w.r.t. a reference partition.

How to perturb data? In our realistic setting (see Figure 5.2), neither jumping nor jittering will occur if the data are perturbed under sampling processes, as soon as there is enough data. We show on a simple example that sampling-based methods such as [Ben-Hur et al., 2002, Lange et al., 2004] cannot work in the general case. Therefore, only noise-based perturbation is considered here. Among them, we adopt the ε-Additive Perturbation (ε-AP) with Gaussian or uniform noise for this work, assuming variables are scaled to zero mean and unit variance. The number of perturbed samples D can be kept very low and still gives reliable estimates. An analysis has also been conducted on the influence of D and showed that even very small numbers ($D = 1$) lead to great performance.

How to choose ε? A central trade-off has to be taken into account when perturbing the data set. If ε-AP is too strong, we might alter the very structure of the data. If on the contrary ε-AP is too small, the clustering algorithm will always obtain identical results, inevitably leading to stability. Although setting this value is not crucial according to [Möller and Radke, 2006], compared with choosing a subsample size, we still believe it is somewhat arbitrary and in a way implicitly defines what a clustering is. As in Example 5.9, if ε is too large, the two closest clusters will be merged under our stability principle. Hence, in a way, ε-AP defines a threshold distance below which two data points are similar and should belong to the same cluster. We propose to circumvent this issue by *not* choosing a single value for the level of noise ε, but a grid of possible values. By gradually increasing ε from 0 to a value ε_{max}, we obtain what we call a *stability path*, i.e. the evolution of stability as a function of ε. This method has one crucial advantage: it allows to compare partitions for different values of ε without the necessity of choosing one. However, it comes with two drawbacks: setting both the fineness and the maximum value of the grid. In our experiments, the fineness does not play a major role in the results. A straightforward method to fix a maximum value ε_{max} beyond which comparisons are not meaningful anymore is as follows. The perturbation corresponding to ε_{max} is meant to destroy the cluster structure of the original data. This corresponds to the value where the data are no longer clusterable, i.e. $K = 1$ becomes the best

solution w.r.t. Stadion. A first guess at $\varepsilon_{\max} = \sqrt{P}$ (where P is the data dimension) works well in practice. We found that visualizing the stability paths (see Figure 5.3) is appealing and greatly helps interpreting the structures found by an algorithm, hence improving the *usefulness* of results.

Which data to compare? A strong ε-AP can alter the data, but it can also destroy structure and close-by clusters can merge faster than others. Therefore, pairwise comparison between perturbed data sets may become unreliable, and we consider only comparison between the original and perturbed data sets. As stated in Equation 5.2, we compute similarities between the reference and perturbed partitions.

How to compare partitions? The similarity measure s chosen to compare two partitions is the ARI. A total of 16 different similarity and distance measures (such as the NMI or FM) are compared in a following section, and ARI achieved the best results. Its value is in $[0, 1]$, thus the Stadion has a value in the $[-1, 1]$ range, with 1 corresponding to the best clustering and -1 to the worst.

How to aggregate the Stadion path? To compute a scalar validity index for model selection, the Stadion path must be aggregated on the noise strength ε from 0 to $\varepsilon_{\max}$ (when the solution for $K = 1$ has the highest Stadion among all other solutions). Two aggregation strategies, the maximum (Stadion-max) and the mean (Stadion-mean), are evaluated in our experiments. A a consequence of this aggregation step, the criterion adapts to different degrees of cluster separation in the data set.

The within-cluster stability is governed by parameter $\mathbf{\Omega}$, which detects stable structures inside clusters of $\mathcal{C}_K$. As these are unknown, averaging several different values in $\mathbf{\Omega}$ gives a better estimate. In absence of sub-clusters, all partitions will be unstable because cluster boundaries will be placed in high-density regions. For the opposite reason, in presence of sub-clusters, at least some partitions will result in higher stability, thus increasing the within-cluster stability. The analysis of influence conducted in a following section shows that $\mathbf{\Omega}$ has low impact on Stadion results and can be set easily.

An important assumption behind our implementation of within-cluster stability is that, for non-clusterable structures (w.r.t. an algorithm), the algorithm must place cluster boundaries in high-density regions to produce instability through jittering. This encompasses a wide range of algorithms such as center-based, spectral or Ward linkage clustering which, for the sake of saving cost, would cut through dense clouds of points. If this requirement is not fulfilled, it is unclear whether this method will work. For instance, single linkage cannot be evaluated this way, since it may build two-cluster partitions of size 1 and $N - 1$, where the boundary lies at the frontier

of the cluster. Finally, the motivation for using the same algorithm to cluster again each cluster is that an algorithm should evaluate itself. For instance, one could use a clustering algorithm to estimate within-cluster stability different from the one used to compute stability between clusters, or one could train a supervised classifier on the clusters labels and then assess its stability [Dudoit and Fridlyand, 2002, Lange et al., 2004, Tibshirani and Walther, 2005]. However, it is not obvious what kind of bias would be introduced with this approach.

5.5 Pseudo-code

Algorithm 5.1: Between-cluster stability procedure.

Input: algorithm $\mathcal{A}$; data set $\mathbf{X}$; reference clustering $\mathcal{C}_K$; parameter K; perturbations D; similarity measure s; noise amplitude ε

Output: between-cluster stability $\text{Stab}_{\text{B}}(\mathcal{A}, \mathbf{X}, \mathcal{C}_K, K, \varepsilon)$

bstab $\leftarrow 0$;

for $d = 1 \dots D$ **do**

  Generate random noise $\epsilon \sim U(-\varepsilon, +\varepsilon)$ or $\mathcal{N}(\mathbf{0}, \varepsilon\mathbf{I})$;

  $\mathbf{X}_d \leftarrow \mathbf{X} + \epsilon$;

  bstab $\leftarrow$ bstab $+ s\left(\mathcal{C}_K, \mathcal{A}(\mathbf{X}_d, K)\right)$;

Return bstab$/D$;

Algorithm 5.2: Within-cluster stability procedure.

Input: algorithm $\mathcal{A}$; data set $\mathbf{X}$; set of parameters $\mathbf{\Omega}$; reference clustering $\mathcal{C}_K$; perturbations D; similarity measure s; noise amplitude ε

Output: within-cluster stability $\text{Stab}_{\text{W}}(\mathcal{A}, \mathbf{X}, \mathcal{C}_K, K, \mathbf{\Omega}, \varepsilon)$

wstab $\leftarrow 0$;

$N \leftarrow |\mathbf{X}|$;

for $k = 1 \dots K$ **do**

  $C_k \leftarrow k$-th cluster of $\mathbf{X}$ in reference clustering $\mathcal{C}_K$;

  $N_k \leftarrow |C_k|$;

  bstab $\leftarrow 0$;

  for K' *in* $\mathbf{\Omega}$ **do**

    $\mathcal{Q}^{(k)}_{K'} \leftarrow \mathcal{A}(C_k, K')$;

    bstab $\leftarrow$ bstab $+ \text{Stab}_{\text{B}}(\mathcal{A}, C_k, \mathcal{Q}^{(k)}_{K'}, K')$;

  bstab $\leftarrow$ bstab$/|\mathbf{\Omega}|$;

  wstab $\leftarrow$ wstab $+$ bstab $\times \frac{N_k}{N}$;

Return wstab ;

Algorithm 5.3: Complete procedure for selecting the number of clusters $\hat{K}$ using Stadion paths, with max (Stadion-max) or mean (Stadion-max) aggregation.

Input: algorithm $\mathcal{A}$; data set $\mathbf{X}$; maximum number of clusters $K_{\max}$; perturbations D; similarity measure s; grid of noise amplitudes $\{\varepsilon_i\}_{1 \leq i \leq M}$ with $\varepsilon_1 = 0$ and $\varepsilon_M = \varepsilon_{\max}$

Output: selected number of clusters $\hat{K}$

for $K = 1 \dots K_{max}$ **do**
  $\mathcal{C}_K \leftarrow \mathcal{A}(\mathbf{X}, K)$;
  for $i = 1 \dots M$ **do**
    $\text{bstab}_i \leftarrow \text{Stab}_{\text{B}}(\mathcal{A}, \mathbf{X}, \mathcal{C}_K, K, \varepsilon_i)$;
    $\text{wstab}_i \leftarrow \text{Stab}_{\text{W}}(\mathcal{A}, \mathbf{X}, \mathcal{C}_K, K, \mathbf{\Omega}, \varepsilon_i)$;
    $\text{stadion}_i \leftarrow \text{bstab}_i - \text{wstab}_i$;

$\hat{K} \leftarrow \text{argmax}_K \max_i \text{stadion}$ *or* $\text{argmax}_K \text{mean}_i \text{stadion}$;

Return $\hat{K}$;

5.6 Complexity study

Let $A(K, N)$ be the time complexity of the algorithm with parameter K and a data set of size N, assuming the data dimension is fixed. In addition, let $S(K, N)$ be the complexity of the similarity measure s, D the number of perturbations and M the length of the stability path.

Between-cluster stability The complexity for a given parameter K (assuming the complexity of perturbation is negligible) is $\mathcal{O}\left((A(K, N) + S(K, N))DM\right)$.

Within-cluster stability For a given parameter K and a set of parameters $\mathbf{\Omega} = \{2, \dots, K'\}$, the amount of operations is $\sum_{k=1}^{K} \sum_{k'=2}^{K'} (A(k', N_k) + S(k', N_k))DM$, which can be bounded by $\mathcal{O}\left(KK'(A(K', N) + S(K', N))DM\right)$. In the case of K-means, we have $A(K, N) = \mathcal{O}(KNTI)$, where T is the number of iterations until convergence of the algorithm, and I the number of runs. Then, ARI is linear: $S(K, N) = \mathcal{O}(N)$. Overall, we obtain a complexity for Stadion with K-means and ARI equal to $\mathcal{O}(KK'^2NTIDM)$.

The influence studies showed that $\mathbf{\Omega}$ can be set to a small range, e.g. $\{2, \dots, 5\}$ or $\{2, \dots, 10\}$, and that D can be kept very low. Thus, complexity in K' and D is manageable. Thus, complexity of Stadion is mainly driven by $\mathcal{O}(KNTIM)$. The extended version avoids running the algorithm again for each perturbation D, getting rid of the T and I factors. For K-means, we only have to find the closest centers, which is $\mathcal{O}(KN)$. Thus, we have an overall complexity of $\mathcal{O}(KK'^2NDM)$.

In regard, internal indices relying on between-cluster and within-cluster distances have a complexity of $\mathcal{O}(N^2)$ or $\mathcal{O}(KN)$ with centroid distance. Thus, the cost of having to run the algorithm several times may be smaller than a quadratic index, if N is large and the algorithm is linear.

5.7 Some experiments and examples

5.7.1 A simple example with stability paths

We begin this section by illustrating our method with K-means and uniform ε-AP on the example data set discussed previously (see Figure 5.1). Figure 5.3 displays between-cluster stability, within-cluster stability and Stadion as a function of the noise strength ε. For reasonable amounts of noise, the solutions $K = 1$, $K = 2$ and $K = 3$ are all perfectly stable, showing the insufficiency of between-cluster stability alone to indicate whenever K is too small. The solutions for $K \geq 4$ cut through the clusters and are thus unstable due to jittering. However, the solutions for $K = 1$ and $K = 2$ both have high within-cluster stability, caused by the presence of sub-clusters, which is not the case for $K \geq 3$. By computing a difference, our criterion Stadion combines this information and is able to indicate the correct number of clusters ($K = 3$) by selecting the Stadion path with the highest maximum or mean value. Through its formulation, Stadion is acting as a *stability trade-off*. The stability paths also give additional insights about the data structure. For example, we can read from the between-cluster stability path how the clusters successively merge together as ε increases. Finally, the last graph (called stability trade-off plot) represents Stadion-mean for different values of the parameter K.

5.7.2 Finding $K = 1$: the case of non-clusterable data

Is a data set clusterable? Between-cluster stability is unable to answer this question, as the solution with a single cluster ($K = 1$) is trivially stable. Some stability methods are not even defined for $K = 1$ because of normalization [Lange et al., 2004]. Moreover, many internal indices use between-cluster distance and are not defined for a single cluster neither. We verified empirically that our criterion consistently outputs $K = 1$ in the case when the algorithm does not find any cluster structure. Table 5.1 contains results for non-clusterable artificial data sets. Stadion

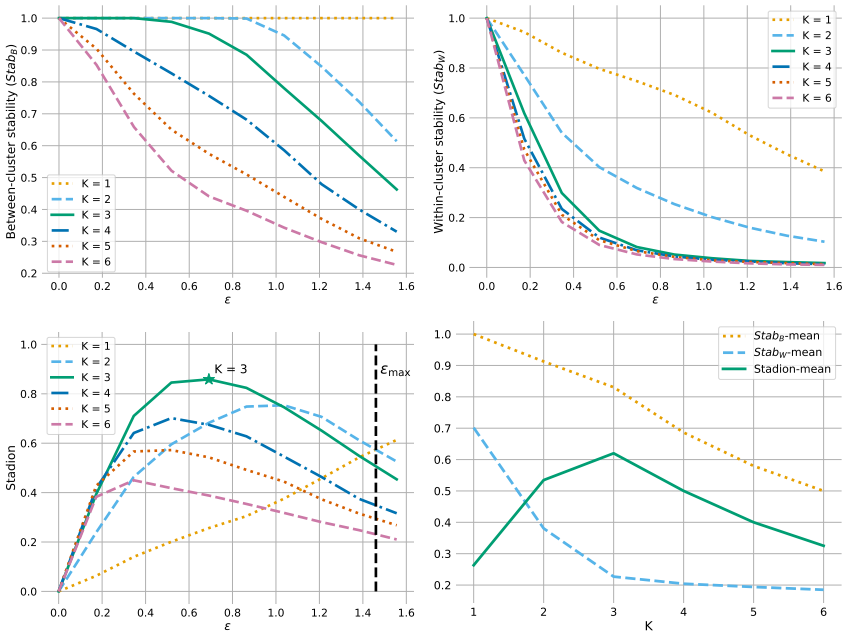


Fig. 5.3.: Between-cluster stability paths (top left), within-cluster stability paths (top right), Stadion paths (bottom left) and stability trade-off curve (bottom right) for K-means on the data set Figure 5.1, for $K \in \{1 \dots 6\}$. ε is the amplitude of the uniform noise perturbation. The best solution $K = 3$ is selected either by taking the maximum or by averaging Stadion over ε until $\varepsilon_{\max}$. The trade-off plot represents the averaged Stadion, between- and within-cluster stability as a function of K.

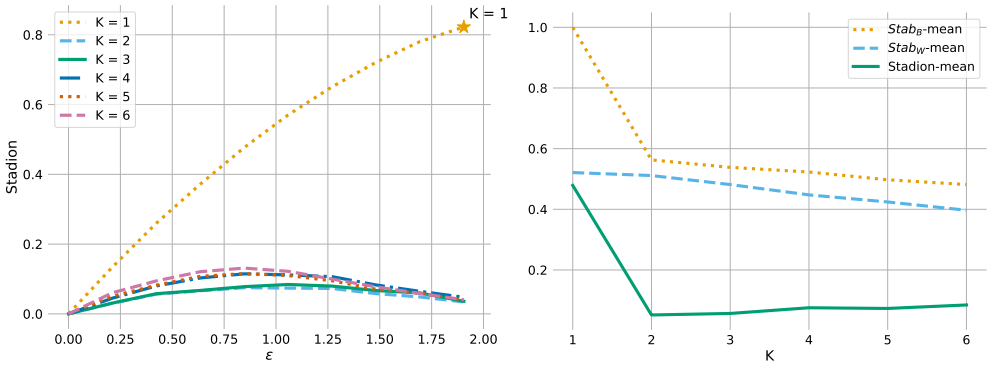
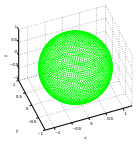


Fig. 5.4.: Stadion path (left) and stability trade-off plot (right) on the golfball data set with K-means. $K = 1$ is clearly the best solution found by Stadion-max/mean (uniform noise, $\mathbf{\Omega} = \{2, \ldots, 10\}$).

outputs $K = 1$ in all cases. An example of Stadion paths and trade-off curve for the golfball data set is provided on Figure 5.4 (results are similar for other data sets).

Tab. 5.1.: Number of clusters found by Stadion on non-clusterable artificial data sets.

Data set		N	P	K selected by Stadion
Uniform cube (2d)		1000	2	1
Uniform cube (10d)		1000	10	1
Gaussian (2d)		1000	2	1
Gaussian (10d)		1000	10	1
Golfball [Ultsch, 2005]		4002	3	1

5.7.3 Examples of jumping between local minima

As explained in Section 5.2, two sources of instability are jumping and jittering. We have already stated that our method leverages jittering of cluster boundaries in high-density regions due to perturbation. Jumping, on the other hand, happens when the algorithm finds very different solutions on different samples; in case of objective-minimizing algorithms, it ends up in different local minima. Two main effects lead to jumping: first, symmetries in the data distribution, and second,

initialization. Finally, subtle geometrical properties of the distribution might also cause jumping [Von Luxburg, 2009]. An example of jumping of K-means due to

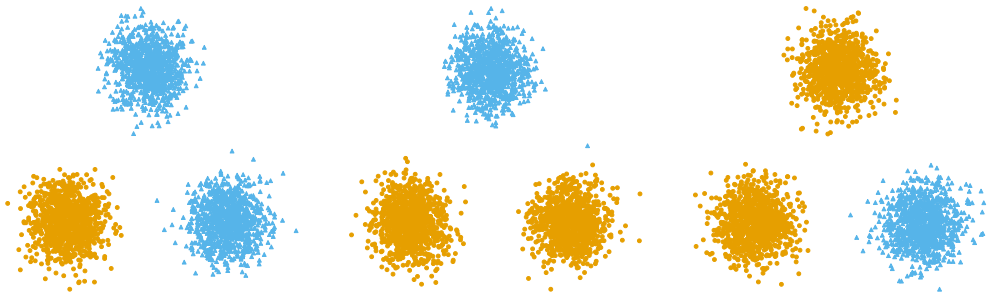


Fig. 5.5.: Example of K-means jumping between three global minima for $K = 2$ on a symmetric distribution with three Gaussians, despite effective initialization (K-means++ and best of 10 runs). Under slight perturbation (here uniform ε-AP, but resampling gives identical results), the algorithm jumps between grouping two random clusters together.

symmetries is shown on Figure 5.5: clearly, there are several global minima, and even if the algorithm is deterministic, slight perturbations of the distribution (noise or sampling) make the algorithm jump between solutions. The second cause of jumping is due to initialization. As illustrated by Figure 5.6 for K-means, if a single random initialization is used, depending on the initial position of centers, four different configurations occur randomly, even without any perturbation of the data. We place ourselves in a realistic setting without perfect symmetries and an effective

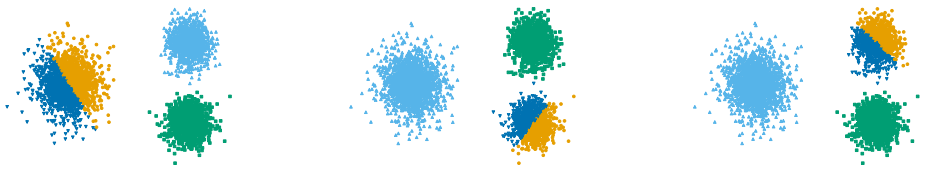


Fig. 5.6.: Example of K-means jumping between three local minima for $K = 4$, when a single random initialization is used. Depending on the initial centers configuration, the algorithm jumps between splitting a random cluster in two (a, b, c).

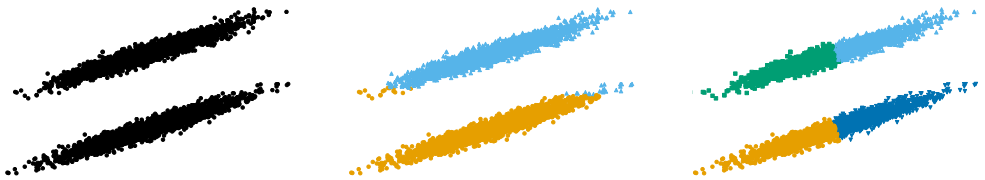
algorithm initialization strategy, thus jumping is not the main source of instability.

5.7.4 Failure of sampling-based stability methods

In this section, we will see on a trivial example why stability methods based on sampling are not reliable to detect the presence of structure in the data. Four methods are compared:

1. Stadion based on ε-Additive Perturbation.
2. Stadion based on bootstrapping.
3. The model explorer algorithm [Ben-Hur et al., 2002] based on subsampling.
4. The model order selection method [Lange et al., 2004] based on splitting data in two halves and transferring labels from one half onto the other using a supervised nearest-neighbor classifier.

We demonstrate that only the first method is successful on a simple example consisting in a mixture of two correlated Gaussians, represented on Figure 5.7. Data are scaled to zero mean and unit variance as for every other data set. K-means is used to cluster the data. As illustrated on the plot, K-means with $K = 2$ separates almost perfectly the two Gaussians. All other solutions split the two Gaussians into several sub-clusters of equal sizes, with cluster boundaries lying in the regions of highest density, as can be seen from the example for $K = 4$ (where the boundaries are in the middle of the Gaussians). Thus, in addition to being the best solution, $K = 2$ is the only acceptable one. However, sampling-based methods fail in assessing its stability, since they estimate $K = 4$ as the most stable solution. This result can be explained because the data set is not symmetric and for each K there is one global minimum so no jumping occurs, even with a poor initialization scheme. Thus the only possible source of instability stems from jittering. As expected in theory, our experiments showed here that the different sampling processes did not succeed in creating jittering. Conversely, ε-AP has indeed produced jittering, where a small amount of noise produced very different partitions. In details, the model order selection method [Lange et al., 2004] selects $K = 4$, followed by $K = 6$. The model explorer [Ben-Hur et al., 2002] finds $K = 6$ as the best solution, followed by $K = 4$. These results are consistent across initialization schemes (random, K-means++, best of several runs). Hence, random initialization will not help creating instability by jumping. Furthermore, our stability criterion Stadion was able to find $K = 2$ among the set of tested values $\{1, \ldots, 6\}$ (here with uniform noise and $\mathbf{\Omega} = \{2, \ldots, 6\}$). This is not only due to adding the within-cluster stability. As evidence, we replaced ε-AP by a bootstrap perturbation: Stadion with bootstrapping also fails, selecting $K = 1$ as the best solution followed by $K = 4$, and this for all initialization schemes.



(a) **(b)** $K = 2$ **(c)** $K = 4$

Fig. 5.7.: Example data set of two correlated Gaussians, scaled to zero mean and unit variance. With the K-means algorithm, all sampling-based methods select $K = 4$ or $K = 6$, whereas with ε-Additive Perturbation, $K = 2$ is the only stable solution.

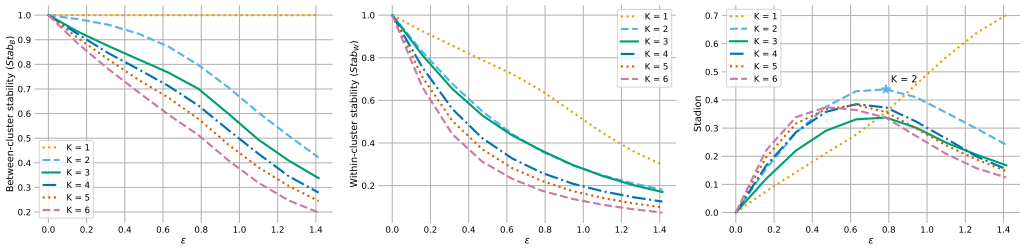


Fig. 5.8.: Between-cluster stability, within-cluster stability and Stadion paths (uniform noise, $\mathbf{\Omega} = \{2, \ldots, 6\}$) on the example of two correlated Gaussians where all sampling-based methods fail to select $K = 2$. Stadion clearly finds $K = 2$ by taking the max or mean of the path curve.

5.7.5 Example of Stadion behavior with K-means

This example illustrates the behavior of our stability criterion Stadion and how to interpret the stability paths, using the data set 2d-4c shown in Figure 5.9. It consists in four clusters with different variance and size, where two clusters are closer to each other while the other clusters are at a greater distance. At first glance, this example looks trivial, but the majority of internal indices fail. For instance, the Dunn and Silhouette indices both select $K = 3$. The stability paths are presented

Fig. 5.9.: Example data set 2d-4c consists in four clusters of different variance and size.

in Figure 5.10, where we observe that Stadion is able to detect the structure of the data and selects $K = 4$. The only difference between the solutions with $K = 4$ and $K = 5$ is that the largest cluster (in green) is split, thus leading to a much lower between-cluster stability but the same within-cluster stability. Solutions $K = 2$ and $K = 3$ group clusters together without any splitting. Therefore, those solutions have a high between-cluster stability and also a high within-cluster stability. Altogether, on the Stadion path (Figure 5.10), the path corresponding to $K = 4$ is similar to $K = 5$ whereas $K = 2$ and $K = 3$ have an equivalent behavior. This is due to the structure of the data, and especially because the two rightmost clusters are close to each other. The moment when the path of solution $K = 3$ becomes the best solution is the moment when these two clusters merge because of a high ε-AP, and this is also the moment where $K = 1$ prevails. Finally, Stadion paths (with stability and instability paths) give useful additional information on a clustering and on the structure of the data. When $K > K^\star$, the paths are similar to the path of $K^\star$ but with a smaller scale, as they have the same within-cluster stability but lower between-cluster stability. On the other hand, when $K < K^\star$, the paths are shifted towards the right, and may become superior for larger ε values.

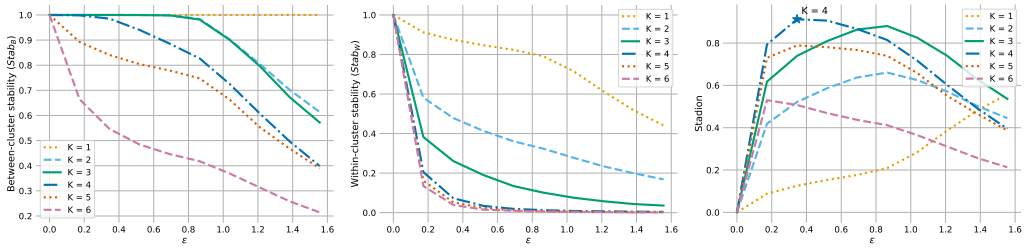


Fig. 5.10.: Between-cluster stability, within-cluster stability and Stadion paths (uniform noise, $\mathbf{\Omega} = \{2, \ldots, 6\}$) on the 2d-4c data set. Stadion selects $K = 4$, followed by $K = 3$.

5.7.6 Whenever $K^\star$ is not the best partition

Sometimes, the best solution is not the partition obtained with the true number of clusters $K^\star$, because the algorithm is unable to recover the ground-truth partition. This is the case for the 4clusters_corner data set, depicted on Figure 5.11. While obviously the best solution is to separate the four clusters, it is not achievable by K-means: with $K^\star = 4$, it will cut through the large cluster instead of separating the two small green clusters, for the sake of saving the cost induced by the variance and the size of this cluster. Among the proposed solutions, the highest ARI (w.r.t. the ground-truth partition) is obtained with $K = 3$ (ARI = 0.92), followed by $K = 2$ (0.74), $K = 5$ (0.65) and lastly $K^\star = 4$ (0.58). All internal indices, excepted the

(a) $K = 2$ (ARI = 0.74) Most internal indices

(b) $K = 3$ (ARI = 0.92) Stadion

(c) $K^\star = 4$ (ARI = 0.58) [Ben-Hur et al., 2002, Lange et al., 2004]

(d) $K = 5$ (ARI = 0.65)

Fig. 5.11.: Partitions found by K-means on the 4clusters_corner data set for $K \in \{2, \ldots, 5\}$.

Gap, select $K = 2$. Stability methods based on sampling ([Ben-Hur et al., 2002], [Lange et al., 2004]) selected the ground-truth $K^\star = 4$, earning them a "win", although it is the worst partition among the four. We explain it by the fact that these methods are not leveraging jittering inside the large cluster. Finally, the Stadion always selects the solution $K = 3$ having the highest ARI. Moreover, the criterion outputs solutions in the same order than ARI. This examples clearly exhibits the

stability trade-off occurring in Stadion: it tries to preserve a high between-cluster stability while keeping within-cluster stability as low as possible (see Table 5.2). Stadion paths on Figure 5.12 also show how the three smaller clusters merge as the noise level increases.

Tab. 5.2.: Stability trade-off leveraged by Stadion on the 4clusters_corner data set.

K	ARI	StabB	StabW	Stadion	
1	0.00	++	- -	0	✗
2	0.74	++	-	+	✗
3	0.92	++	+	+++	✓
4	0.58	- -	+	-	✗
5	0.65	- -	++	0	✗

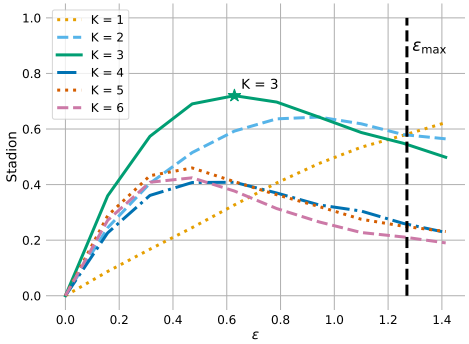


Fig. 5.12.: Stadion paths on the 4clusters_corner data set. $K = 3$ is selected although $K^\star = 4$.

5.7.7 Example of K approaching N

This paragraph introduces the behavior of Stadion when the numbers of clusters K evaluated become as large as the number of samples N. Even if this is beyond the common setting in clustering, the criterion is still valid. Figure 5.13 displays the stability trade-off for K-means on an example with three Gaussians, using ARI as the similarity metric. As K approaches N:

- Between-cluster stability decreases towards 0, except for $K = N$ where it jumps back to 1, because all partitions with one sample per cluster are perfectly similar to ARI.
- Within-cluster stability increases towards 1, as clusters with few samples become trivially stable.
- Stadion still indicates the correct solution $K = 3$, while decreasing towards -1, only jumping back to 0 when $K = N$.

Note that the borderline case $K = N$ depends on the similarity used, for instance with Fowlkes-Mallows the between-cluster stability does not jump back to 1, staying at 0. In addition, with the extended version, the perturbed partition will not have one sample per cluster, thus it will also stay at 0. Nevertheless, for all similarity measures, Stadion's behavior is consistent and valid even for large values of K.

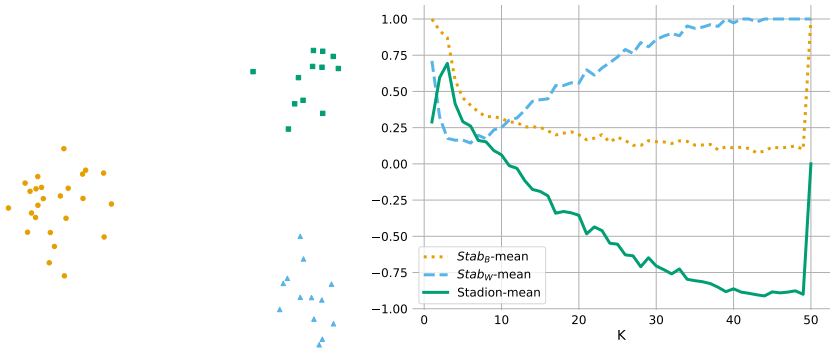


Fig. 5.13.: Stability trade-off plot for K-means on three Gaussians with $N = 50$ for $K \in \{1, \ldots, 50\}$ (uniform noise, $\mathbf{\Omega} = \{2, \ldots, 10\}$). Stadion is still valid when the tested K becomes large.

5.8 Selecting K in K-means, GMM and Ward clustering

In this benchmark experiment, three algorithms are considered: K-means [MacQueen, 1967], Gaussian Mixture Models (GMM) [Banfield and Raftery, 1993] and Ward hierarchical clustering [Ward, 1963]. For K-means, two versions of Stadion are evaluated: the first one using the stability computation described in section 4 (referred to as the *standard* version), and the second one using the extension operator (referred to as the *extended* version). As seen in section 2, an extension operator extends a clustering to new data points. K-means extends naturally by computing the euclidean distance to centers. Hence, instead of re-running K-means for each perturbation of the data, we directly predict the cluster assignments of perturbed data points. This approximation is sensible since we consider jittering as the main source of instability, and spares computation time (see complexity study). GMM allows a similar extension, by assigning points to the cluster with the highest posterior probability. It is the only version we consider here due to the high computational cost of GMM which makes the standard version prohibitive. Albeit first experiments looked promising, the same limitation was encountered for spectral clustering [Von Luxburg, 2007], but unfortunately it has no straightforward extension operator.

We evaluate clustering validation methods on a large collection of 73 artificial benchmark data sets, most of them extensively used in literature. Data sets were selected so that the algorithms can achieve good partitions w.r.t. the true clusters. We also ensured different difficulty levels for model selection, obtained by varying the numbers of clusters, sizes, variances, shapes and the presence of noise and close-by or overlapping clusters. In addition, we present results on 7 real data sets, labeled into $K^\star$ ground-truth classes. It was surprising to discover how difficult it is to find

real-world data that are clusterable into $K^\star$ clusters without preprocessing. First, the original features seldom have a cluster structure. Second, it may happen that the labels do not represent a natural partitioning of the data, unlike for artificial data sets. Thus, is was necessary to preprocess most real data sets. For instance, Crabs was preprocessed by a PCA [Pearson, 1901] keeping only components two and three, as described in [Bouveyron and Brunet-Saumard, 2014]. High-dimensional data sets such as images (MNIST, USPS) were reduced beforehand, using an autoencoder network in order to extract clusterable features, followed by UMAP [McInnes et al., 2018] to obtain a two-dimensional representation, as introduced in [McConville et al., 2021]. This way, we ensure the labels truly represent clusters. Identical preprocessing settings were used across every method and data set, without any further tuning. An exhaustive description of the experimental setting, including data sets and preprocessing steps, is provided in Appendix C.

Table 5.3 summarizes results for all three algorithms. We compare Stadion to the partitions obtained with the true number of clusters $K^\star$, best-performing internal clustering indices (see [Desgraupes, 2013, Hämäläinen et al., 2017] for reviews), Gap statistic [Tibshirani et al., 2001] (K-means only), BIC (GMM only) and stability methods [Ben-Hur et al., 2002, Lange et al., 2004]. To ensure a fair comparison, all internal indices were computed on the same partition, that was also the reference partition in Stadion. We report the number of data sets where each method found $K^\star$, which we refer to as the number of *wins*. However, only checking whether $K^\star$ is selected is not always related to the goodness of the partition w.r.t. the ground-truth. Results strongly depend on the performance of algorithms, which do not necessarily succeed in finding a good partition into $K^\star$ clusters. Thus, a more adequate performance measure is the similarity between the selected partition and the ground-truth. As a performance measure, the ARI is a standard choice [Romano and Bailey, 2016] when clusters are mostly balanced. Let us note $\mathcal{Y}_{K^\star} = \{Y_1, \ldots, Y_{K^\star}\}$ the ground-truth partition. The performance of each validation method is assessed by computing $\text{ARI}(\mathcal{Y}_{K^\star}, \mathcal{C}_{\hat{K}})$, where $\hat{K}$ is the estimated number of clusters. In order to compare methods on multiple data sets, we compute the average ranks in terms of ARI, denoted $\overline{R_{\text{ARI}}}$. Since data sets have different difficulties, their results are not comparable. Thus, average ARI or numbers of wins are meaningless [Demšar, 2006]. Nonetheless, wins are reported for reference. The benchmark uses uniform ε-AP, $D = 10$, $\mathbf{\Omega} = \{2, \ldots, 10\}$, $s =$ ARI and evaluates solutions for $K \in \{1, \ldots, K_{\max}\}$ where $K_{\max}$ is $K^\star + 20$ rounded down to the nearest tenth. Stadion-max achieves the best results overall. On K-means, it is even ranked higher than $K^\star$ in terms of ARI. The second-best performing index is Wemmert-Gancarski. It was shown in [Balcan and Liang, 2016] that agglomerative clustering is not robust to noise, which

Tab. 5.3.: Benchmark results on 80 artificial and real data sets for K-means, Ward and GMM. Average rank of the ARI with the ground-truth classes ($\overline{R_{\text{ARI}}}$) and number of times $K^\star$ was selected (wins).

	Artificial data sets						Real data sets					
	K-means		Ward		GMM		K-means		Ward		GMM	
Method	$\overline{R_{\text{ARI}}}$	wins	$\overline{R_{\text{ARI}}}$	wins	$\overline{R_{\text{ARI}}}$	wins	$\overline{R_{\text{ARI}}}$	wins	$\overline{R_{\text{ARI}}}$	wins	$\overline{R_{\text{ARI}}}$	wins
$K^\star$	6.47	73	4.77	73	5.05	73	4.50	7	3.36	7	3.93	7
Stadion-max	**6.02**	50	**5.25**	**54**	-	-	**4.93**	**5**	5.86	**4**	-	-
Stadion-mean	6.12	51	5.80	49	-	-	6.57	4	7.64	3	-	-
Stadion-max (extended)	6.13	**56**	-	-	**5.59**	**56**	6.29	3	-	-	**4.43**	**5**
Stadion-mean (extended)	6.42	48	-	-	6.79	43	6.29	3	-	-	5.50	3
BIC	-	-	-	-	6.45	48	-	-	-	-	7.29	2
Wemmert-Gancarski	6.62	53	5.40	**54**	5.77	52	6.00	**5**	**5.36**	**4**	5.79	4
Silhouette	7.51	46	6.47	45	7.01	45	7.21	4	5.86	**4**	6.50	4
[Lange et al., 2004]	7.93	45	6.53	51	6.99	48	8.64	3	5.86	**4**	7.14	3
Davies-Bouldin	8.11	40	6.45	41	7.29	34	8.29	4	7.29	3	8.57	3
Ray-Turi	8.19	37	6.97	40	7.68	33	8.29	4	6.29	3	7.36	4
Calinski-Harabasz	8.71	41	7.14	39	7.43	37	12.21	1	8.86	1	5.79	3
Dunn	10.11	26	7.77	33	7.92	34	10.57	1	7.79	2	9.07	2
Xie-Beni	10.27	22	7.61	34	8.19	28	11.50	1	7.57	2	9.93	2
Gap statistic	10.38	26	-	-	-	-	10.57	2	-	-	-	-
[Ben-Hur et al., 2002]	10.99	20	7.86	31	8.85	28	8.14	1	7.93	2	9.71	2

explains inferior Stadion results with Ward. Moreover, results are slightly biased in favor of the indices that are only valid for $K \geq 2$, unlike Stadion that will select $K = 1$ on non-clusterable distributions, as shown in Table 5.1. Full result tables and statistical tests are provided in Appendix C along with a more thorough analysis. In particular, the ranking is unchanged by using other external performance measures such as AMI or NMI instead of the ARI. Beyond selecting K, Stadion may also be used to select the kernel parameter in spectral clustering, the radius in density-based clustering or select between different algorithms.

5.9 Hyperparameter study

The stability difference criterion (Stadion) introduced in this work is governed by several hyperparameters:

- D: the number of perturbed samples used in the stability computations.
- noise: the type of noise for ε-AP. We experimented with uniform and Gaussian noise.
- $\mathbf{\Omega}$: the set of parameters K' used in within-cluster stability computation.

- s: the similarity measure used in stability computation is a special hyperparameter, and is treated specifically in the last paragraph of this section.

The goal of this section is to study their importance and impact on the performance of Stadion for clustering model selection, using the three studied algorithms (K-means, Ward linkage and GMM). Only the extended versions of Stadion for K-means and GMM are included, for the sake of saving computational cost.

5.9.1 Importance study with fANOVA

Ideally, practitioners would like to know how hyperparameters affect performance in general, not just in the context of a single fixed instantiation of the remaining hyperparameters, but across all their instantiations. The fANOVA (functional ANalysis Of VAriance) framework for assessing hyperparameter importance introduced in [Hutter et al., 2014] is based on efficient marginalization over dimensions using regression trees. The importance of each hyperparameter is obtained by training a Random Forest model of 100 regression trees to predict the performance of Stadion in terms of ARI given the set of hyperparameters. Then, the variance of the performance due to a given hyperparameter is decomposed by marginalizing out the effects of all other parameters. It also allows to assess interaction effects. Hence, the fANOVA framework provides insights on the overall importance of hyperparameters and their interactions.

The maximum amount of noise ε_{max} and the fineness of the grid are not included in the study, because it is data-dependent and one can easily check if values are appropriate by looking at the paths. We study the following discrete hyperparameter space:

- $D \in \{1, \dots, 10\}$
- noise is uniform or Gaussian
- $\mathbf{\Omega} \in \{2, 3, 5, 10, \{2, \dots, 5\}, \{2, \dots, 10\}, \{10, \dots, 20\}, \{2, \dots 20\}\}$
- $s =$ ARI

Before going any further, we would like to add a clarification on the causes of jumping. In section 5.2, we stated that the two causes of jumping are symmetries in the data and initialization. Indeed, they are the only ones possible in our setting. But one aspect has not been addressed, whenever K becomes large w.r.t. N. In this case, the effective initialization strategy no longer prevents jumping. Furthermore, if the size of clusters is very small, then small perturbations can drastically change the solution, unlike when $K << N$ with N sufficiently large. The latter can undeniably

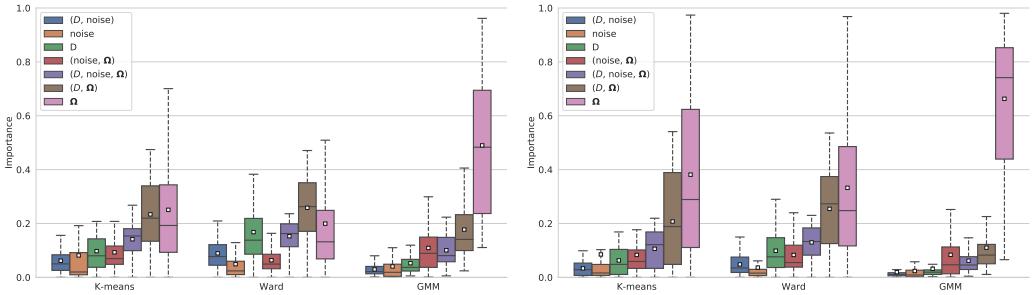


Fig. 5.14.: Box plots of the fANOVA importance of parameters and their interactions for Stadion-max (left) and mean (right) across 73 artificial data sets, for three algorithms.

cause jumping. That brings us to the conclusion that Stadion should not be used, with this formulation, for applications with large K w.r.t N, for instance deduplication [Kushagra et al., 2018], and that further investigations are needed in this context.

Figure 5.14 shows the contributions of hyperparameters and their interactions to the variance of ARI performance, across 73 artificial data sets. $\mathbf{\Omega}$ is by far the most important parameter, and this for two reasons.

First, the size of the data N needed to obtain good estimations is relative to the number of clusters K. More precisely, in our setting we only consider $K << N$. Whenever K is large w.r.t. N, jumping due to "large K" can occur, which sometimes happens in within-cluster stability, where small clusters might be split into up to 20 sub-clusters. This implies that even in presence of sub-clusters, high values of K' in $\mathbf{\Omega}$ will create instability and thus lead to low within-cluster stability. More precisely, if $K \geq K^\star$, then in general Ω will not affect within-cluster stability because it is already low. But whenever $K \leq K^\star$, within-stability is more impacted by large values of K' in Ω, and within-stability paths for these specific values of K shift down, leading to higher Stadion paths. Second, ARI has decreasing performance for large numbers of small clusters [Romano and Bailey, 2016].

The second most important parameter is the interaction $(D, \mathbf{\Omega})$, for the same reason: large numbers of clusters make estimating the within-cluster stability more difficult, and thus a higher number of perturbations D is needed to obtain a good approximation.

5.9.2 Influence of D

The D hyperparameter defines the number of perturbed samples used in the stability computation in Equations 5.2 and 5.3. In our benchmark, we used $D = 10$. Surprisingly, a number of samples as low as $D = 1$ already gives a good estimate of the expectation and the performance only slightly increases with larger values of D. We perform an experiment by making D vary from 1 to 10, keeping other hyperparameters fixed (uniform noise, $\mathbf{\Omega} = \{2, \ldots, 10\}$), for the three algorithms and both Stadion path aggregation strategies (max and mean), and measure performance in terms of ARI over 73 artificial benchmark data sets. Results on Figure 5.15 show that low D values have a higher variance and slightly lower performance. To

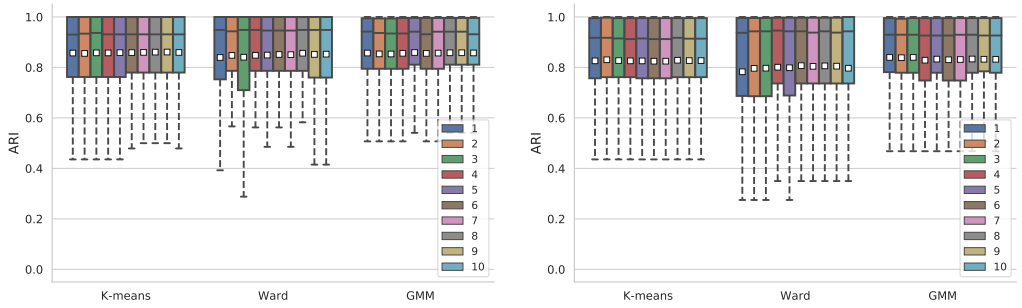


Fig. 5.15.: Box-plot of the ARI of partitions selected by Stadion-max (left) and mean (right) across 73 data sets, for three algorithms and different values of D, the number of samples in the stability computation.

quantify further the influence of this parameter, we followed the recommendation in [Demšar, 2006] and used the Friedman test for comparisons on multiple data sets, in order to test against the null hypothesis $\mathcal{H}_0$ stating that all parameters have equivalent performance. After rejecting $\mathcal{H}_0$, we performed the pairwise post-hoc analysis recommended by [Benavoli et al., 2016] where the average rank comparison (e.g. Nemenyi test) is replaced by a Wilcoxon signed-rank test at $\alpha = 5\%$ with a Holm-Bonferroni correction procedure to control the family-wise error rate (FWER) [Holm, 1979, García and Herrera, 2008]. To visualize post-hoc test results, we use the critical difference (CD) diagram [Demšar, 2006], where a thick horizontal line shows groups (cliques) of classifiers that are not significantly different in terms of performance. In all but one case, the Friedman test could not reject the null hypothesis. Only for the GMM algorithm and max aggregation, the null hypothesis was rejected, leading to the critical difference diagrams on Figure 5.16. The number of samples D has a negligible impact on the performance of our method. We assume it is due to the fact that in our setting, instability is caused by jittering at cluster boundaries, which does not vary much from one perturbation to another with rea-

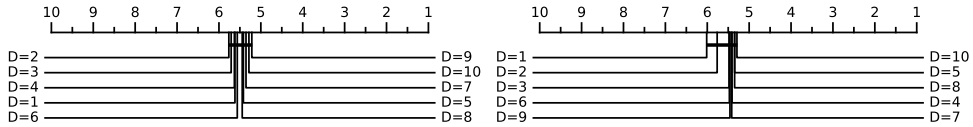


Fig. 5.16.: Critical difference diagrams after Wilcoxon-Holms test ($\alpha = 5\%$) on GMM performance, for Stadion-max with uniform (left) and Gaussian (right) noise, for different values of D, the number of perturbations in the stability computation.

sonable amounts of data. On the contrary, sampling-based stability methods that rely on jumping require a much higher number of samples (for instance, [Ben-Hur et al., 2002] use 100 samples and [Lange et al., 2004] use 20 samples). As a conclusion, we recommend using $D \geq 5$, but if computation time is costly, $D = 1$ can be used safely to cut down complexity.

5.9.3 Influence of noise type

We experiment with two types of ε-additive noise perturbation: uniform noise and Gaussian noise. As previously, we report the distributions of performance in terms of ARI across 73 artificial data sets for both noise types on Figure 5.17 (with $D = 10$ and $\mathbf{\Omega} = \{2, \ldots, 10\}$). To assess the difference between both noise types, we perform the

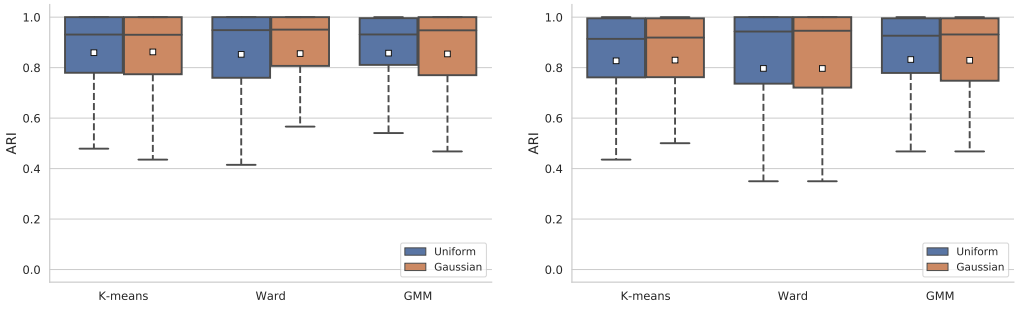


Fig. 5.17.: Box plots of the ARI of partitions selected by Stadion-max (left) and mean (right) across 73 artificial data sets, for three algorithms, using uniform or Gaussian noise perturbation.

Wilcoxon signed-rank test on the performance results (at confidence level $\alpha = 5\%$). For every algorithm and Stadion path aggregation, the test did not reject the null hypothesis. Thus, either uniform or Gaussian noise can be used.

5.9.4 Influence of $\mathbf{\Omega}$

The $\mathbf{\Omega}$ hyperparameter is a set defining the numbers of clusters used to cluster again each cluster of the original partition. We perform an experiment by varying $\mathbf{\Omega}$, keeping other hyperparameters fixed (uniform noise, $D = 10$), for both Stadion path aggregation strategies (max and mean), and measure performance in terms of ARI over 73 artificial benchmark data sets. Results in Figure 5.18 demonstrate that

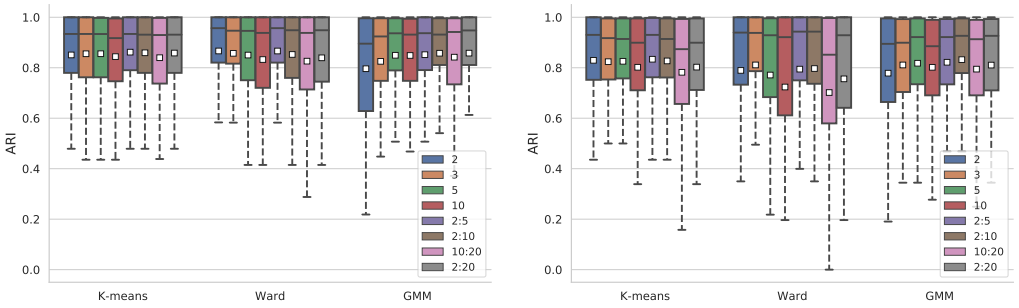
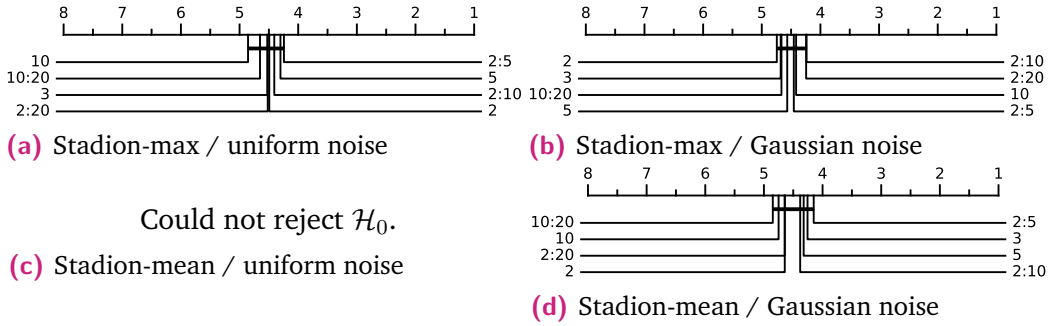


Fig. 5.18.: Box plots of ARI of partitions selected by Stadion-max (left) and mean (right) across 73 artificial data sets, for three algorithms and different sets $\mathbf{\Omega}$ (numbers of clusters used in within-cluster stability).

$\mathbf{\Omega}$ does not have, in most cases, a big impact on the performance of Stadion. This does not contradict the results of the fANOVA. Indeed, $\mathbf{\Omega}$ has the largest variance contribution to the performance, but this variance remains small, and overall Stadion is robust for reasonable choices of the parameter, such as $\{2, \ldots, 5\}$ or $\{2, \ldots, 10\}$. Ward linkage is the most influenced by the choice of $\mathbf{\Omega}$. We guess the main reason is that agglomerative clustering algorithms are not robust to noise [Balcan and Liang, 2016]. Critical difference diagrams after Wilcoxon-Holms test on performance are given in Figures (5.19, 5.20, 5.21). None of them showed significant differences, indicating that there is not enough data to conclude. However, small values in $\mathbf{\Omega}$ seem to perform better, which confirms the previous claim that large values of K' in $\mathbf{\Omega}$negatively impact performance. In particular, the range $\{2, \ldots, 10\}$ used in our benchmark performs well across all algorithms.

5.9.5 Similarity measure analysis

An extensive study was conducted to compare similarity measures (or distances) between partitions, noted s in the stability computations (Equations 5.2, 5.3). Note that all definitions and formulae are available in Section 4.2 of Chapter 4. The first



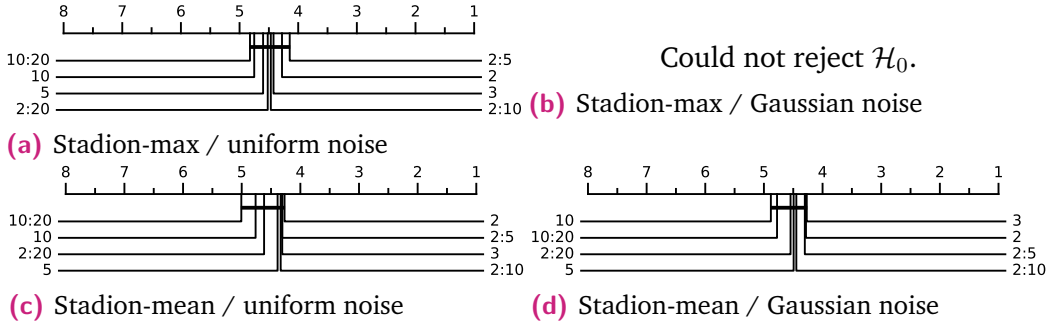
(a) Stadion-max / uniform noise

(b) Stadion-max / Gaussian noise

(c) Stadion-mean / uniform noise

(d) Stadion-mean / Gaussian noise

Fig. 5.19.: Critical difference diagrams after Wilcoxon-Holms test on K-means performance, for different values of $\mathbf{\Omega}$, the set of parameters used in within-cluster stability computation.



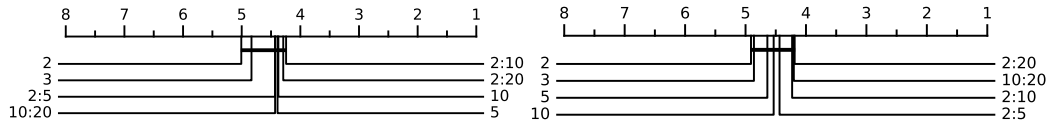
(a) Stadion-max / uniform noise

(b) Stadion-max / Gaussian noise

(c) Stadion-mean / uniform noise

(d) Stadion-mean / Gaussian noise

Fig. 5.20.: Critical difference diagrams after Wilcoxon-Holms test on Ward performance, for different values of $\mathbf{\Omega}$, the set of parameters K used in within-cluster stability computation.



(a) Stadion-max / uniform noise

(b) Stadion-max / Gaussian noise

Fig. 5.21.: Critical difference diagrams after Wilcoxon-Holms test on GMM performance, for different values of $\mathbf{\Omega}$, the set of parameters K used in within-cluster stability computation. With Stadion-mean, the Friedman test could not reject $\mathcal{H}_0$.

five are count-based measures and were compared in a study [Milligan and Cooper, 1986]:

- **RI**: Rand Index [Rand, 1971]
- **ARI**$_1$: Hubert and Arabie's Adjusted Rand Index [Hubert and Arabie, 1985]
- **ARI**$_2$: Morey and Agresti's Adjusted Rand Index [Morey and Agresti, 1984]
- **FM**: Fowlkes and Mallows index [Fowlkes and Mallows, 1983]
- **JACC**: Jaccard index

Throughout the paper, ARI was referring to ARI$_1$ and the two terms are now used interchangeably. Following information theoretic measures [Vinh et al., 2010] are also compared in this work:

- **MI**: Mutual Information
- **AMI**: Adjusted Mutual Information
- **VI**: Variation of Information
- **NVI**: Normalized Variation of Information
- **ID**: Information Distance
- **NID**: Normalized Information Distance
- **NMI**$_1$: Normalized Mutual Information, with max normalization
- **NMI**$_2$: Normalized Mutual Information, with min normalization
- **NMI**$_3$: Normalized Mutual Information, with geometric mean normalization
- **NMI**$_4$: Normalized Mutual Information, with arithmetic mean normalization
- **NMI**$_5$: Normalized Mutual Information, with joint entropy normalization

Table 5.4 compares measures by counting the number of data sets where Stadion selected the true number of clusters. Our results confirm that adjusted measures are generally preferable [Vinh et al., 2010]. However, for particular applications, for instance large numbers of clusters or small number of observations, other measures might be better suited. On average, the best-performing measure is ARI$_1$, but the average number of wins is not sufficient to conclude. In order to asses which measures are significantly different, we perform a statistical test. However, we cannot use a signed-rank test as previously, because we can no longer use the ARI score as an external performance measure. Our experiments have shown that the choice of the performance metric introduces a bias, favoring different similarity or distance measures used inside Stadion. For instance, using ARI as the performance measure has lead to higher performance for $s = ARI$ and equivalently for the other measures. Thus, the only way to compare a partition with the ground-truth is whether it has found the correct number of clusters $K^\star$ or not. Under this limitation, the only test at our disposal is the sign test, which compares the number of successes/losses/ties for each pair of methods, where success indicates if a method

Tab. 5.4.: Comparison of similarity measures s used in Stadion computation. Number of correct numbers of clusters for each algorithm and aggregation (with uniform noise, $D = 10$ and $\mathbf{\Omega} = \{2, \ldots, 10\}$).

	Stadion-max			Stadion-mean			
measure	K-means	Ward	GMM	K-means	Ward	GMM	average wins
ARI_1	**56**	54	**56**	48	49	43	51.0
ARI_2	**56**	54	**56**	48	49	43	51.0
AMI	54	52	55	48	49	45	50.5
NID	54	52	55	48	49	45	50.5
NMI_1	54	52	55	48	49	45	50.5
NMI_4	54	51	55	48	49	46	50.5
NMI_3	54	51	**56**	48	46	38	48.8
NMI_2	53	52	55	**50**	47	34	48.5
NMI_5	53	48	**56**	43	**50**	41	48.5
NVI	53	47	**56**	42	**50**	41	48.2
FM	47	**56**	51	48	41	45	48.0
JACC	45	55	50	38	44	45	46.2
ID	32	55	39	47	35	**47**	42.5
VI	35	**56**	34	47	34	45	41.3
RI	23	21	46	43	35	31	33.2
MI	8	11	18	17	15	13	13.7

selected $K^\star$ and not the other. The sign test uses a binomial test, assuming that if two methods are equivalent, they should each succeed on approximately half of the data sets. The results of the sign test is represented on Figure 5.22. As before, we control the FWER at $\alpha = 5\%$ using the Holm-Bonferroni procedure for multiple comparisons.

The matrix of p-values exhibits a block structure with on one hand, the majority of measures that perform well with Stadion, and on the other hand, the MI and RI, which perform poorly (because they scale with K). In addition, ARI are significantly superior to JACC, ID and VI. However, due to the high number of ties, the low power of the sign test and insufficiency of data, we cannot reach any further conclusions. This structure remains across all three tested algorithms and Stadion path aggregations. As a conclusion, we recommend using ARI_1 with the Stadion criterion, but several similarity measures between partitions are well-suited to measure stability.

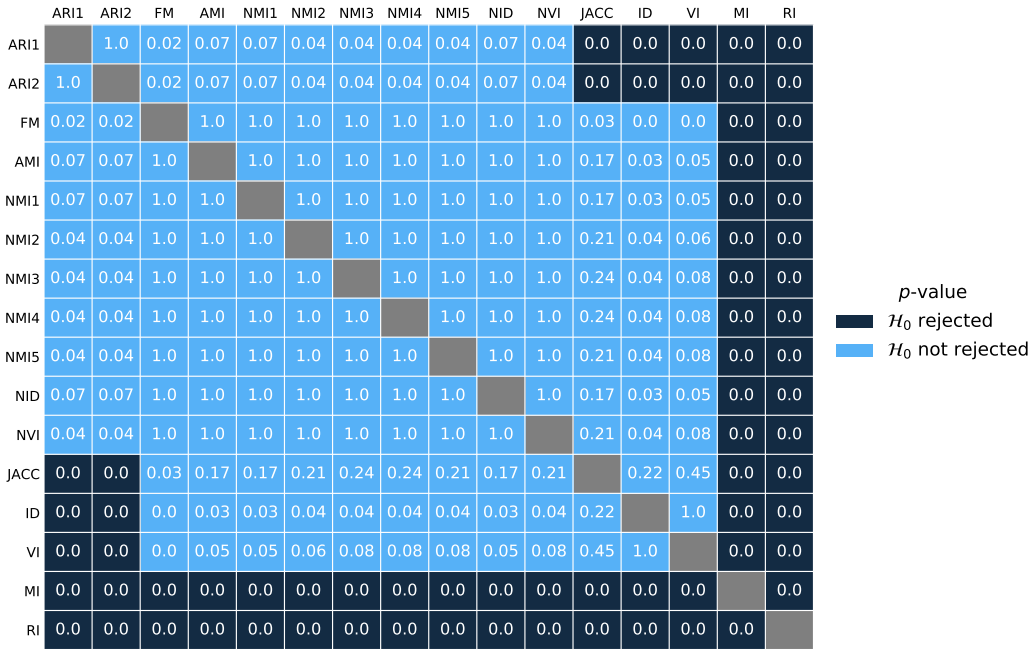


Fig. 5.22.: Matrix of p-values after a pairwise sign test comparing different similarity measures and distances between clusterings used in Stadion, here with K-means (uniform noise, $D = 10$ and $\mathbf{\Omega} = \{2, \ldots, 10\}$). The null hypothesis $\mathcal{H}_0$, that two measures are equivalent, is tested at $\alpha = 5\%$ confidence, using Holm-Bonferroni correction to control the FWER.

5.10 Software implementations

We developed the `skstab`[1] [Forest and Mourer, 2020] for cluster stability analysis in Python with a `scikit-learn` compatible API. The class hierarchy of `skstab` is represented on Figure 5.23.

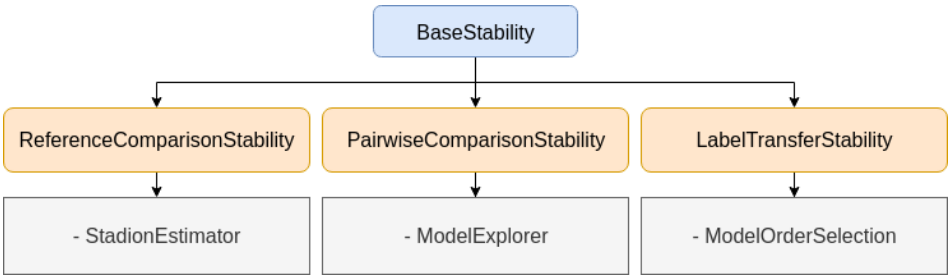


Fig. 5.23.: Class hierarchy of the `skstab` package.

[1] `https://github.com/FlorentF9/skstab`

5.11 Conclusion

Stability is a universal tool to assess the quality of solutions obtained by supervised and unsupervised learning algorithms. In this chapter, we introduce the concept of cluster stability and related works, with their limitations. One major drawback is that when there is a single global minimum to the algorithm's objective, solutions with K too small are always stable. Our main contribution is a newly introduced concept of within-cluster stability, and Stadion (stability difference criterion), a novel clustering validation criterion acting as a trade-off between the traditional between-cluster stability. This principle emerged from the fact that jittering of cluster boundaries is crucial to measure densities at boundaries and assess the stability of a solution, in a realistic setting we have described. We also advocate for using additive noise perturbation instead of re-sampling. Furthermore, our method to control the amount of perturbation provides an interpretable visualization tool called stability paths.

We evaluated Stadion and many methods of literature for three standard algorithms against a benchmark of 73 artificial and 7 real-world data sets (the largest benchmark ever conducted in stability analysis)). Performance is superior or on par with internal clustering indices that were designed with specific cluster geometries in mind (e.g. specifically for center-based algorithms), while relying on more general assumptions. This comes at a computational cost, requiring to run the algorithm many times. Nevertheless, an extensive hyperparameter study has shown that it can be drastically reduced, the impact of down-sizing the hyperparameters being negligible. Still, there remains some limitations, left for future work (see the final conclusion of the thesis). In the next chapter, we will apply this work to time series clustering.

6 Validation of time series clustering with an invariance-guided criterion

This chapter is based on the contribution:

- Forest, F., Mourer, A., Lebbah, M., Azzag, H., & Lacaille, J. (2020). An Invariance-guided Stability Criterion for Time Series Clustering Validation. *International Conference on Pattern Recognition (ICPR)*.

Following the criterion introduced in [Mourer et al., 2020], we now address the specific problem of model selection in time series clustering [Forest et al., 2021].

Time series clustering has been mostly studied under the angle of finding efficient algorithms and distance metrics adapted to the specific nature of time series data. Much less attention has been devoted to the general problem of model selection. We propose to apply stability analysis (subject of the last chapter) to time series by leveraging prior knowledge on the nature and invariances of the data. These invariances determine the perturbation process used to assess stability. Based on the previously introduced Stadion criterion combining between-cluster and within-cluster stability, we propose an invariance-guided method for model selection, applicable to a wide range of clustering algorithms. Experiments conducted on artificial and benchmark data sets demonstrate the ability of our criterion to discover structure and select the correct number of clusters, whenever data invariances are known beforehand.

6.1 Introduction

Time series are a type of data naturally organized as sequences with a temporal dimension, such as values collected by sensors. Large volumes of unlabeled data are ubiquitous across various domains such as healthcare, industry, biology, astronomy, economy, the Internet of things (IoT) and many others. Clustering, a widely used technique to gain insights from such data, consists in finding groups of elements

called clusters such that elements sharing the same cluster are similar, and elements belonging to different clusters are dissimilar. Time series clustering (TSC) [Warren Liao, 2005, Aghabozorgi et al., 2015] is a challenging task due to the temporal nature of the data, which implies high dimensionality [Verleysen and François, 2005], temporal feature correlation, invariance to transformations, and different lengths. Model selection for TSC in particular is not well studied in literature [Aghabozorgi et al., 2015]. For instance, methods to select the number of clusters are rarely provided, although selecting the *best* or *natural* number of clusters is known to be one of the crucial problems in cluster analysis [Ng and Han, 1994, Ben-David et al., 2006, Von Luxburg, 2009]. When external labels are unavailable, model selection is done using internal clustering validity indices [Arbelaitz et al., 2013]. Most indices are based on between-cluster and within-cluster distances, and could be used with any distance other than Euclidean (e.g. Silhouette with Manhattan distance [Ng and Han, 1994]), but their application to time series has not been well studied [Warren Liao, 2005]. These indices are generally used on extracted features, not raw time series (e.g. Davies-Bouldin in [Neel, 2005]). Heuristic methods with cross-correlation dissimilarity have been developed in [Baragona, 2001]. For TSC based on autoregressive models, distances between ARMA/ARIMA models have been devised [Maharaj, 2000, Piccolo, 1990]. In case of model-based clustering, such as mixture models, the AIC, BIC and ICL criteria have been widely used [Biernacki et al., 2000, Bouveyron et al., 2015, Goffinet et al., 2020b, Goffinet et al., 2020a]. Still, the validation of time series clustering is unsolved in general.

Clustering stability [Ben-David et al., 2006, Von Luxburg, 2009] has emerged as a natural and model-agnostic principle: an algorithm should find stable structures in the data. "To be meaningful, a clustering must be both good and the only good clustering of the data, *up to small perturbations*. Such a clustering is called *stable*. Data that contains a stable clustering is said to be clusterable" [Meilă, 2018]. In statistical learning terms, if data sets are sampled from the same underlying distribution, an algorithm should find similar partitions. The data-generating distribution is unavailable in model-free clustering, thus perturbed data sets are obtained either by resampling or injecting noise into the original data. Limitations of this principle, in particular its ability to select the number of clusters, have been studied in [Mourer et al., 2020]. It has been shown that a novel criterion called *Stadion* (stability difference criterion) is able to successfully discover structure and select the number of clusters when using additive noise perturbation. We base ourselves onto this work and extend it to time series which have their own specificities. It is known that temporal data are resilient to particular perturbations, which depend on the application and the physical nature of the observed phenomena. Thus, we leverage

prior knowledge on the invariances of the data in order to assess stability of a clustering.

Invariant perturbations are already used for data augmentation, to improve the generalization capability of supervised classifiers. Suitable perturbations for various applications can be found in this literature, for example for time series [Pan et al., 2020, Fu et al., 2020] or images [Fawzi et al., 2016, Shorten and Khoshgoftaar, 2019]. Transformation-invariant clustering algorithms have also been developed [Frey and Jojic, 2000, Monnier et al., 2020]. In particular, warping-invariant time series embeddings are learned in [Mathew and Sahely, 2019]. To our knowledge, the first application of stability analysis to time series clustering comes from the financial field [Marti et al., 2016]. In their work, authors study the price of financial derivatives, namely credit default swaps. They compare the stability of weighted linkage clustering with different dissimilarities (Euclidean distance, Pearson and Spearman correlations, and a combination of correlation and Hellinger distance between distributions). In order to assess stability, they devise a specialized perturbation framework for financial time series. This idea of leveraging prior knowledge on the nature and properties of the data is also what we would like to develop in this work. However, the approach remains focused on their business field and is application-specific. In addition, no quantitative stability scores are computed, and results are interpreted by visualizing the partitions. Finally, it does not tackle the problem of selecting the number of clusters. A second recent work [Klassen et al., 2020] uses stability to evaluate fuzzy *over-time* clustering to detect correlated subsequences in multivariate time series. This approach is interested in time-point clustering (i.e. clustering individual time points of several series) and in particular the evolution of cluster structure over time. Differently, our work focuses on whole time series clustering. Moreover, they compute stability scores based on a resampling approach [Roth et al., 2002], whereas we adopt the framework of [Mourer et al., 2020], using perturbation by noise.

6.2 Invariances and time series clustering

Clustering algorithms are always based on a notion of distance between elements of the data set. Distances between time series are only meaningful if they satisfy certain invariances: in other words, some sequences should be considered similar even if their raw feature values are different. It is not possible to choose an adequate distance measure without knowing what invariances are desirable for the specific task. For the same data set, several clustering solutions are possible, depending on

these invariances. Hence, the problem of multiple clusterings is amplified [Färber et al., 2010]. Example of invariances are:

- **Scale** or **offset invariance**. In many cases, we want two series to be considered similar if they differ by an affine transformation (for example, if a value was measured in different physical units, like Celsius and Fahrenheit degrees).
- **Shift invariance**. If a same phenomenon is observed at different time points in two series, they should be considered identical.
- **Warping invariance**. This invariance is necessary if the phenomenon may have different speeds or delays, which is ubiquitous in motion and biological signals. Series can be aligned and matched using Dynamic Time Warping (DTW) [Sakoe and Chiba, 1978].
- **Uniform temporal scaling invariance**. Unlike local scaling in warping, global scaling is necessary to match behaviors at different speeds or frequencies, yielding series with different lengths. A solution is to stretch series by a constant factor.
- **Occlusion invariance**. Parts of the input being unobserved should not change cluster membership.
- **Complexity** or **noise invariance**. [Batista et al., 2011] have shown that time series can have different *complexities*, and that complex series tend to be closer to simpler series than to other complex series under Euclidean distance.

Euclidean distance, used in most traditional clustering algorithms that operate on tabular data (i.e. *flat* vectors of features), does not satisfy any of these invariances. Thus, a variety of dissimilarity measures between time series has been devised [Giusti and Batista, 2013].

Time series clustering methods can be broadly divided into three categories [Aghabozorgi et al., 2015]. Whole time series clustering considers each series as an individual object. Subsequence clustering consists in clustering subsequences of a single time series, for example a measurement over a long period of time or real-time, streaming data. Time point clustering clusters the individual time observations, and is similar to segmentation. In this work, we only experiment with whole time series clustering.

On another level, clustering algorithms can be either based on raw time series, feature-based, or model-based [Warren Liao, 2005]. Raw time series clustering algorithms define a distance between raw values in the time domain. Agglomerative clustering with single, complete or average linkage, and K-medoids (also called PAM for Partitioning Around Medoids) [Kaufman and Rousseeuw, 1990, Ng and Han, 1994] can be used with any distance between time series. Other widely used methods require the computation of an average in the sense of specific distance,

such as K-DBA [Petitjean et al., 2011], K-SC [Yang and Leskovec, 2011] and K-shape [Paparrizos and Gravano, 2015]. Another approach uses *shapelets*, which are short salient subsequences that discriminate between classes. First proposed in supervised learning, unsupervised shapelets are also used for clustering [Zakaria et al., 2012, Zhang et al., 2018].

Feature-based approaches consist in removing the temporal dimension by extracting higher-level features and projecting the data into a space where euclidean distance and generic algorithms (e.g. K-means, agglomerative clustering, SVMs, decision trees) can be used. For instance, statistical features can be extracted, such as mean, variance, minimum and maximum values, number of peaks, etc. Then, a time series can be projected into the frequency domain using a Fourier transform, extracting spectral features. Wavelets are another option. Another approach is to discretize the values taken by the series and aggregate the sequence into a *bag-of-features*, removing the temporal dimension, called piecewise aggregate approximation [Patel et al., 2003, Lin et al., 2007]. Many successful methods in classification and clustering are based on combining bags of multiple time- and frequency-domain features [Schäfer, 2015, Schäfer and Leser, 2016]. Finally, this includes deep learning approaches where a neural network learns representations from the raw time series values [Madiraju et al., 2018, Ma et al., 2019, Fortuin et al., 2019, Manduchi et al., 2020].

Another kind of approach learns the temporal behavior through autoregressive models, such as ARMA or recurrent neural networks, and cluster the resulting model parameters [Maharaj, 2000, Piccolo, 1990]. Finally, model-based clustering estimates cluster membership probabilities using probabilistic models such as functional mixture models [Chamroukhi and Nguyen, 2018].

In this work, we will experiment with two widely used algorithms: K-medoids and K-shape. K-medoids is a center-based algorithm, but differently from K-means, instead of computing the mean (*centroid*) of each cluster, the center is the element minimizing the sum of the distances to every other element (and is called the *medoid*). It can be used with any dissimilarity measure. Here, we will use Euclidean (EUC), Correlation (COR) and Dynamic Time Warping (DTW) [Sakoe and Chiba, 1978] distances, defined between two same-length series $\mathbf{x} = (x_1, \ldots, x_T)$ and $\mathbf{y} = (y_1, \ldots, y_T)$ as

$$\text{EUC}(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}||_2 = \sqrt{\sum_{t=1}^{T}(x_t - y_t)^2}$$

$$\text{COR}(\mathbf{x}, \mathbf{y}) = 1 - \text{NCC}_0(\mathbf{x}, \mathbf{y}) = 1 - \frac{\sum_{t=1}^{T}(x_t - \bar{x}_t)(y_t - \bar{y}_t)}{||\mathbf{x} - \bar{\mathbf{x}}||_2 ||\mathbf{y} - \bar{\mathbf{y}}||_2}$$

$$\text{DTW}(\mathbf{x}, \mathbf{y}) = \min_{W} \sqrt{\sum_{i=1}^{P} W_i}$$

where the warping path $W = \{w_1, \ldots, w_P\}$ with $P \geq T$ is obtained using a dynamic programming approach on the pairwise distance matrix between the two series, based on following recurrence: $d(i, j) = \text{EUC}(i, j) + \min\{d(i-1, j-1), d(i-1, j), d(i, j-1)\}$. It is common to constrain the warping path to a band around the diagonal, e.g. the Sakoe-Chiba band [Sakoe and Chiba, 1978]. The invariances of K-medoids depend on the distance used: no invariance with EUC, scale invariance with COR and warping invariance with DTW. K-shape is a center-based algorithm using the shape-based distance (SBD), based on normalized cross-correlation:

$$\text{SBD}(\mathbf{x}, \mathbf{y}) = 1 - \max_{w} \text{NCC}_w(\mathbf{x}, \mathbf{y})$$

where $w \in [-T, T]$ is the shifting of $\mathbf{x}$. K-shape is thus invariant to scaling and shifting, and is meant to be computationally efficient in its computation of averages. Invariances of each algorithm are summarized in Table 6.1.

Tab. 6.1.: Invariances of clustering algorithms to scaling, shifting and warping.

Method/Invariance	Scale	Shift	Warping
K-medoids + EUC	✗	✗	✗
K-medoids + COR	✓	✗	✗
K-medoids + DTW	✗	✓	✓
K-shape	✓	✓	✗

6.3 Invariance-guided stability by perturbing invariant latent factors

Stability methods based on resampling are data-independent and therefore directly applicable to time series. However, it has been shown in a realistic setting that these methods cannot work in the general case [Mourer et al., 2020]. On the contrary, noise-based perturbations such as uniform or Gaussian ε-Additive Perturbation produce instability through jittering of cluster boundaries. The underlying assumption is that a clustering should be resilient to low levels of noise (no points change clusters), unless a boundary lies in a high-density region, where a large number of points change clusters. While adding uniform or Gaussian noise to every dimension is meaningful for tabular vectors of normalized features where Euclidean distance is used, it is irrelevant for raw time series. Algorithms for clustering raw time series use different distance metrics, thus there is no reason that random noise would or would not make points change clusters, depending on the cluster boundaries. The notion of cluster boundary itself becomes unclear when using different distances, as it is no longer a simple hyperplane. Clusters of time series are not clusters in the sense of euclidean distance and are resilient to different types of perturbation. For example, if time series are invariant to shifting, clusters should be resilient to perturbation by random shifting. As another example, if a set of time series is clustered under DTW distance with Sakoe-Chiba band w, clusters should be resilient to perturbation by warping, with a warping level not exceeding w. Most importantly, these invariances are determined by the physical nature of the observed phenomenon and not by the data set itself. The practitioner needs to know in advance which transformations are invariant and which are not. Only then, a suited distance and algorithm can be used and evaluated for model selection.

The computation of stability needs to be adapted to the case of time series, and the perturbation process depends on the invariances of the data (shifting, scaling, offsetting, uniform or local warping, noise, etc). Let us illustrate this discussion with a simple artificial example, displayed on Figure 6.1. A data set consists in one-dimensional time series with "bumps" at two different time locations and with two different scales on the y-axis. The data are generated by only two underlying latent factors: location (z_1) and scale (z_2). Had we access to the variables underlying the time series data-generating process, the task would be traditional clustering in a two-dimensional Euclidean vector space: the latent data distribution is simply 4 Gaussians. At a first glance, the model selection task can now seem straightforward: take any clustering algorithm based on Euclidean distance (e.g. k-means or Ward

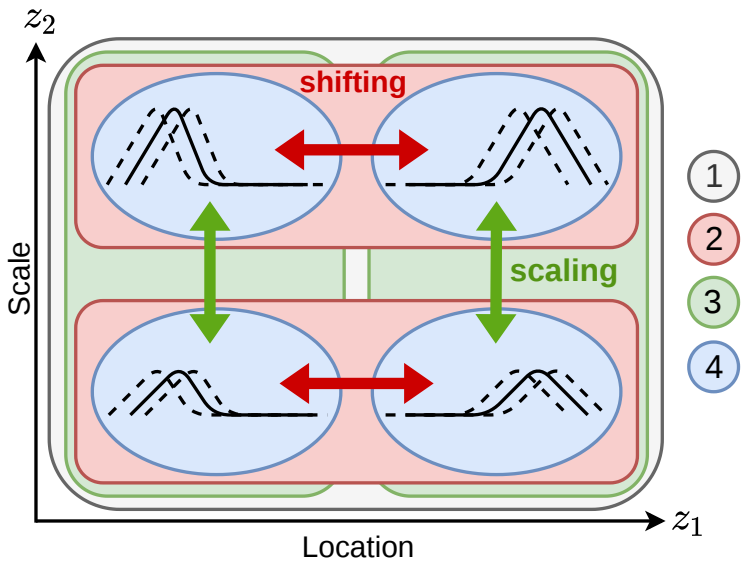


Fig. 6.1.: Artificial time series data set consisting in one-dimensional bumps at two different locations and scales. The data distribution is represented in the (*location*, *scale*) latent factor space. Invariance to perturbation by random shifting (red) or scaling (green) determines the cluster structure, leading to 4 different solutions with 1, 2 or 4 clusters.

linkage), with ε-AP, and the Stadion criterion surely outputs the solution with $K = 4$. However, it is clearly false, because the true solution depends on the invariances of the original time series. The perturbation used in latent space must also take into account these invariances. Shift (or scale) invariance implies the variable z_1 (respectively z_2) should be ignored in the perturbation. There is a duality between perturbations in original time series space and in latent factor space, represented on Figure 6.1. The true cluster structure consists in:

- Shift and scale invariance: solution (1) with $K = 1$
- Shift invariance only: solution (2) with $K = 2$
- Scale invariance only: solution (3) with $K = 2$
- No invariance: solution (4) with $K = 4$ clusters

In the next paragraphs, we will focus on two model selection tasks, using the stability principle introduced in the previous section. First, we show that stability indicates whether a distance is adapted to the data invariances, and second, we select the number of clusters K using the Stadion internal validity index.

6.4 Selecting the right distance with stability

Between-cluster stability can be used to select a distance or algorithm with appropriate invariances. An algorithm should obtain a high between-cluster stability when perturbing the data under invariant transformations. Concretely, we consider the toy data set shown in Figure 6.1, and the widely used K-medoids algorithm, where the number of clusters is fixed to $K = 2$. For effective initialization, required by [Mourer et al., 2020], we use K-medoids++ initialization and take the best result over 10 runs. In the first experiment, we assume the data is scale-invariant. Thus, we use perturbation by randomly scaling the whole time series by a factor drawn uniformly in the $[1/(1+\varepsilon), 1+\varepsilon]$ interval. The value ε controls the perturbation level, similarly to the noise level in [Mourer et al., 2020]. Then, we evaluate between-cluster stability for three distances: Euclidean, correlation and DTW. Figure 6.2 displays the resulting stability paths, and unsurprisingly, correlation distance (K-medoids+COR) is the most stable. The second experiment assumes shift-invariance of the data. The whole time series are shifted temporally by a fraction of the time series length, drawn uniformly in $[0, \varepsilon]$. The perturbation level ε now represents the maximum shift length. The between-cluster stability paths now indicate that K-medoids+DTW is the most stable algorithm. This toy task is rather a sanity check, because one

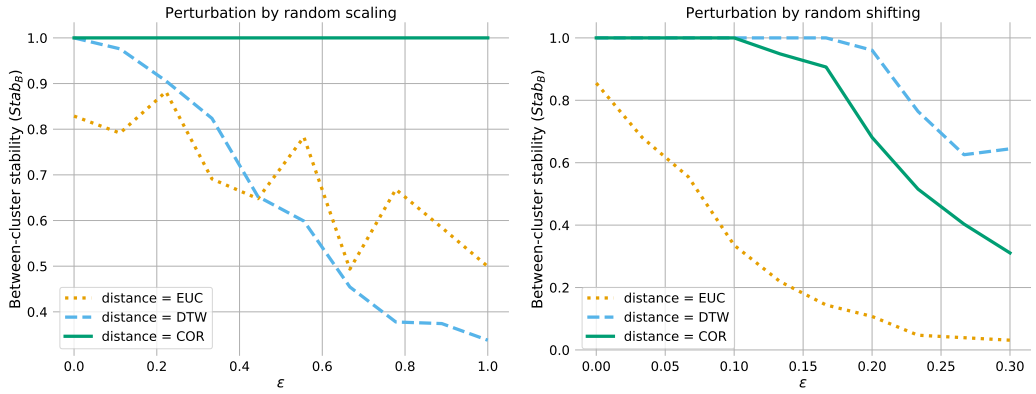


Fig. 6.2.: Between-cluster stability paths under perturbation by random scaling (top) and shifting (bottom) for the K-medoids algorithm, with euclidean (EUC), correlation (COR) and dynamic time warping (DTW) distances. COR is resilient to scaling and DTW is more resilient to shifting. ε controls the level of perturbation.

generally knows in advance which invariances an algorithm satisfies, but we can imagine more complex algorithms where invariances are not clearly determined.

6.5 Selecting the number of clusters

The second model selection task is the selection of the number of clusters K. First, we consider an artificial data set consisting in one, two or three bumps located around three different time locations in the series, displayed on Figure 6.3. The desired invariance is warping invariance. Thus, the true number of clusters is $K = 3$, corresponding to the number of bumps. We evaluate the K-medoids algorithm with

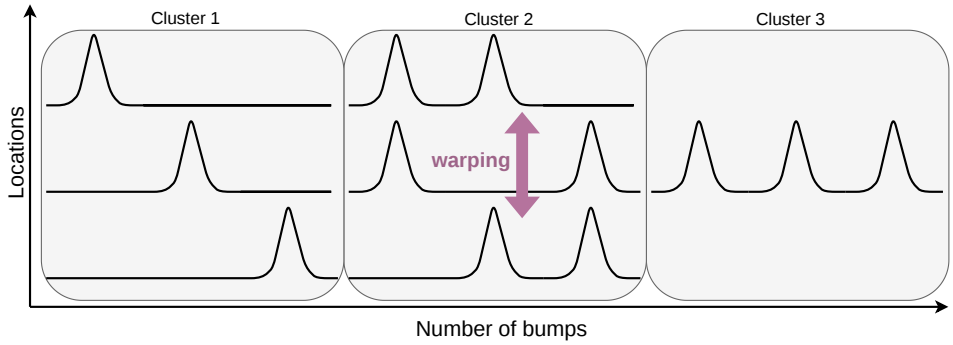


Fig. 6.3.: Artificial time series data set consisting in one, two or three bumps at different locations, represented in the (*number of bumps*, *locations*) latent factor space. Under the assumption of warping invariance, the true number of clusters is 3, corresponding to the number of bumps.

DTW distance using the Stadion criterion and warping perturbation, for $K = 1 \dots 9$. Warping level is controlled by two parameters: first, α controls the maximum fraction of the series that will be warped, and ε controls the warping level, drawn uniformly in $[1/(1+\varepsilon), 1+\varepsilon]$. We fix $\alpha = \varepsilon = 0.2$. The hyperparameters of Stadion are set to $D = 10$ and $\mathbf{\Omega} = \{2 \dots 5\}$ without need for any tuning (see [Mourer et al., 2020] for discussions on hyperparameters). Stadion scores and standard deviations over $D = 10$ perturbations are shown on Figure 6.4. Clearly, our method has selected the desired solution $K = 3$. This means that the most natural structure is three clusters, with respect to the considered algorithm and invariance. Whenever the algorithm is not able to find any structure resilient to the perturbation, our method outputs $K = 1$, i.e. the data is not clusterable. This happens if we use Euclidean distance instead of DTW, as shown in Figure 6.5. Interestingly, the second-best solution is $K = 7$, as there are 7 different configurations for the locations of the bumps.

Experiments were then conducted on univariate data sets from the UCR/UEA archive [Bagnall et al., 2018] (although our stability framework also applies in the multivariate case). We present results for the CBF and Trace data sets, with two algorithms: K-medoids+DTW and K-shape [Paparrizos and Gravano, 2015]. For each algorithm we keep the best out of 10 runs. In order to speed up computations, we use only a subsample of 50 time series with balanced ground-truth class labels.

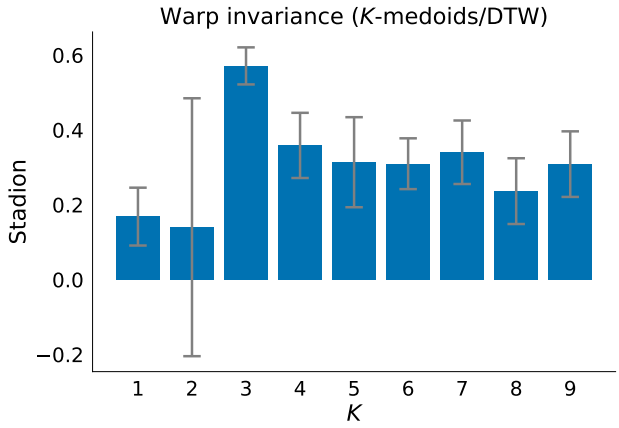


Fig. 6.4.: Stadion criterion with perturbation by random warping (here with $\alpha = \varepsilon = 0.2$ and $D = 10$) for the K-medoids algorithm with DTW distance, for $K = 1 \dots 9$. The correct solution $K = 3$ is selected.

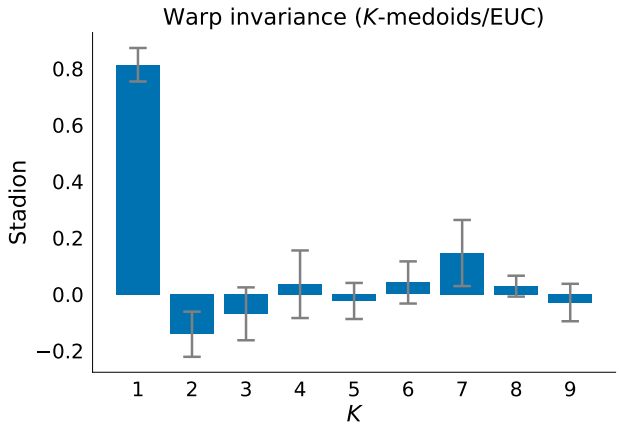


Fig. 6.5.: Stadion criterion with perturbation by random warping for K-medoids with Euclidean distance, for $K = 1 \dots 9$. Our method outputs $K = 1$, meaning that the data is not clusterable w.r.t. the considered algorithm and invariance.

First, we evaluate K-medoids+DTW on CBF (see Figure 6.6). We choose to perturb the data by random shifting and adding uniform noise, as CBF consists in three different noisy shapes at different locations. As previously, the shifting level is controlled by ε, varied from 0 to 0.3 to obtain the Stadion paths on Figure 6.7. The uniform noise is fixed and drawn in $[-0.3, 0.3]$. Choosing the right perturbation seems to be a difficult task and to require profound knowledge of the data set; however, it is not strictly necessary. On CBF, warping invariance could also be correctly used, but shifting is sufficient to discover the right structure. Results are presented on Figure 6.7: our method successfully selects the solution $K = 3$ (by taking the highest maximum or average Stadion value over the path until $\varepsilon_{\max}$, as explained in [Mourer et al., 2020]). It also corresponds to the partition with the best ARI (ARI $= 0.93$). A second experiment on the Trace data set with K-shape

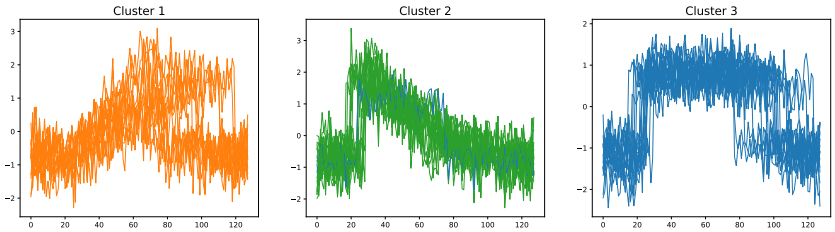


Fig. 6.6.: Partitions obtained on the CBF data set by K-medoids+DTW for $K = 3$. The best solution w.r.t. the ARI is $K = 3$ (ARI $= 0.93$).

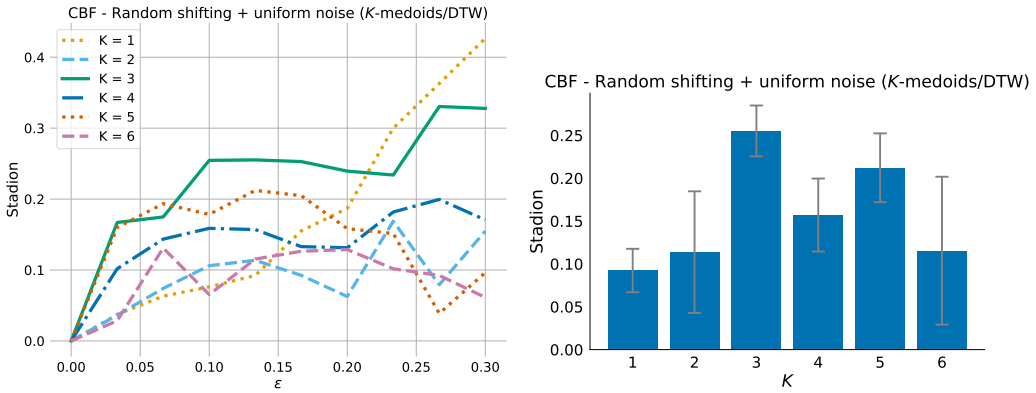


Fig. 6.7.: Stadion criterion for K-medoids+DTW on CBF under shifting and uniform noise perturbation, evaluated for $K = 1 \dots 6$. (Top) Stadion paths as a function of shifting level ε. The solution is selected by the highest maximum or average Stadion value. (Bottom) Stadion scores taken at $\varepsilon = 0.15$ with standard deviations over $D = 10$ perturbations.

presents a case where the algorithm cannot recover the ground-truth partition (see Figure 6.8). We choose a warping-based perturbation, with $\alpha = \varepsilon$ and $\varepsilon_{\max} = 0.5$, and evaluate parameters $K = 1 \dots 5$ ($K > 5$ produces clusters with too few points).

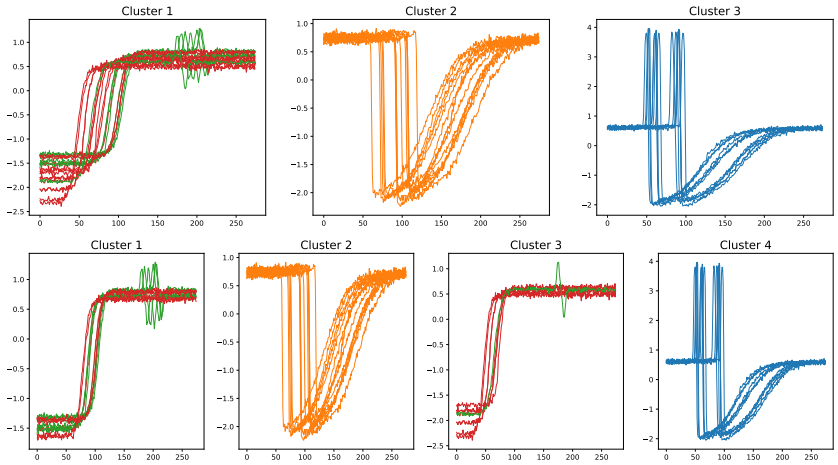


Fig. 6.8.: Partitions obtained on the Trace data set by K-shape for $K = 3$ (top) and $K = 4$ (bottom). The algorithm is unable to recover the ground-truth partition into 4 clusters. The best solution w.r.t. the ARI is $K = 3$ (ARI $= 0.80$), followed by $K = 4$ (ARI $= 0.75$).

As can been seen on the results Figure 6.9, Stadion selects the solution with $K = 3$, although the ground-truth partitions has 4 clusters. However, two of the clusters cannot be distinguished by K-shape, thus $K = 3$ is objectively the best solution (as measured by ARI with ground-truth labels). As a conclusion, our method evaluates the *quality of a given partition with respect to a given algorithm and a given set of invariances*, and yields sensible and interpretable results.

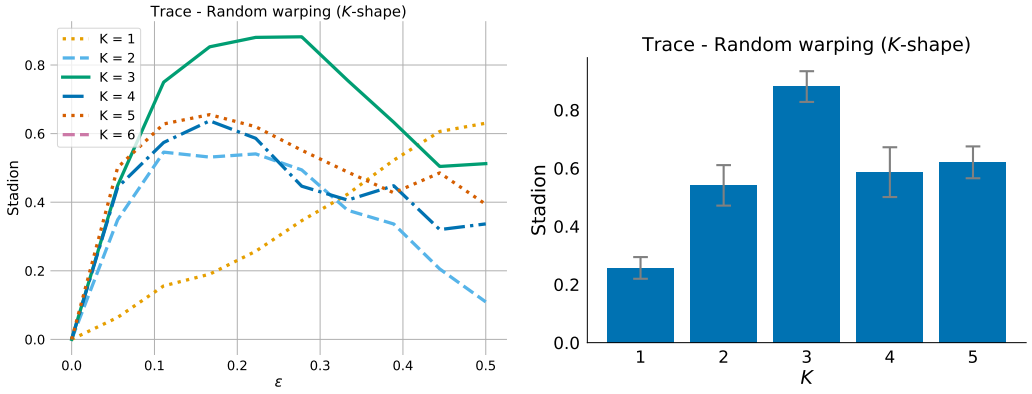


Fig. 6.9.: Stadion criterion for K-shape on Trace under warping perturbation, evaluated for $K = 1 \ldots 5$. (Top) Stadion paths as a function of warping level ε. (Bottom) Stadion scores taken at $\varepsilon = 0.25$ with standard deviations over $D = 10$ perturbations.

6.6 Software implementations

We used the algorithm implementations of the `tslearn` library[1] [Tavenard, 2017] for K-shape, and `sklearn-extra`[2] for K-medoids. The CBF and Trace data sets were taken from the UCR/UEA archive [Bagnall et al., 2018]. The stability analysis procedures, including perturbation function, are part of the `skstab`[3] module developed during this PhD.

6.7 Conclusion

In this chapter, we introduced an invariance-guided criterion for model selection in time series clustering. The method is based on the principle that a good clustering is stable under particular perturbations. We use prior knowledge on the invariances of time series data to compute stability scores, based on the recent Stadion criterion. Encouraging results were obtained on several toy and benchmark data sets, using well-known center-based time series clustering algorithms. The criterion was able to correctly determine the number of clusters given a set of invariances, and benefits from the interpretable visualization of stability paths. An important drawback is its high computational cost, as it requires to run the algorithm multiple times for each evaluated parameter, and time series algorithms generally have a high complexity.

This chapter concludes the theoretical parts around unsupervised learning. The coming two chapters will dive into the technical challenges of scalability, in order to process huge data sets in an industrial context, and solve business use cases. We begin with a chapter on distributed storage and computing across clusters of machines. This set of tools will allow to scale the previously introduced algorithms to very large numbers of samples.

[1] `https://github.com/tslearn-team/tslearn`
[2] `https://github.com/scikit-learn-contrib/scikit-learn-extra`
[3] `https://github.com/FlorentF9/skstab`

Part III

Industrial applications and scalability

7 Scaling to Big Data with distributed computing

> *. . . there are situation where the computer makes feasible what would have been wholly unfeasible.*
>
> **— John Tukey**
> (The Future of Data Analysis, 1962)

7.1 Introduction

Humanity generates two and a half quintillion bytes (or two and a half exabytes, or 2.5×10^{18} bytes), every single day in 2016, according to IBM[1]. And this amount is increasing exponentially: 90% of the data in the world has been generated within the last two years. This deluge of data, at the heart of the activities of tech industry companies, is also transforming more traditional industries, and the aerospace industry in particular. Big Data analytics is mentioned as a top three corporate priority in all industry sectors, and is the top one priority for 61% of companies in the aviation industry, more than in any other industry, tells a 2014 Accenture/General Electric survey[2] (see Figure 7.1). The term *Big Data* was reported formally for the first time in 2000 [Diebold, 2000], where the author stated that:

> Big data refers to the explosion in the quantity (and sometimes, quality) of available and potentially relevant data, largely the result of recent and unprecedented advancements in data recording and storage technology. In this new and exciting world, sample sizes are no longer fruitfully measured in "number of observations," but rather in, say, megabytes. Even data accruing at the rate of several gigabytes per day are not uncommon.

[1] https://www.ibm.com/blogs/watson/2016/07/10-industries-using-big-data-win-big/
[2] https://www.forbes.com/sites/louiscolumbus/2014/10/19/84-of-enterprises-see-big-data-analytics-changing-their-industries-competitive-landscapes-in-the-next-year/

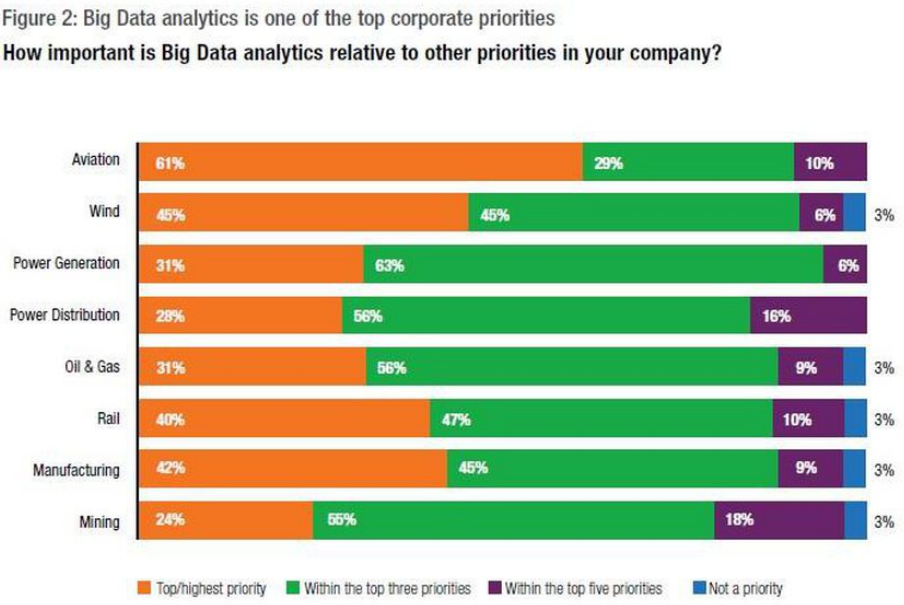


Fig. 7.1.: Big Data analytics: a top priority in industry.

As explained in this quotation, the Big Data phenomenon is the result of trends, which are the exponentially increasing number of data sources (sensors, connected devices, software, the Internet, Internet of Things (IoT), etc.), inexpensive storage, and the availability of tools to process the data. In addition, more and more people are involved in the process of generating, processing and consuming the data: this phenomenon is known as the *democratization of data* [Akerkar, 2014]. A common definition of Big Data uses the *three V's* [Laney, 2001]:

- **Volume**: large data sets and storage that range from gigabytes to petabytes or exabytes.
- **Velocity**: data are generated, collected and processed at a high speed. The data flow is massive, real-time and continuous.
- **Variety**: data can take various forms, structured or unstructured (sensor measurements, images, video, text, audio, speech, human- or machine-generated...).

Often, these properties are expanded to *four V's*, *five V's*[3] or even more, by adding:

- **Value**: useful insights and value contained in the data, that can be used to inform decision-making.
- **Veracity**: data integrity is a challenge. Data may contain errors, biases and noise and must be cleaned before processing.

[3] `https://www.ibm.com/blogs/watson-health/the-5-vs-of-big-data/`

- **Validity**: data must be correct and accurate for the intended use.
- **Variability**: data are captured at different times, places and by different people. The interpretation of data depends on the context.
- **Volatility**: volatility refers to how long data are relevant for an analysis and how long they should be stored.

Data are considered as Big Data if they pose a combination of the challenges expressed by these V's. There are dozens of definitions of Big Data, and they vary by sector and industry, but an objective definition was phrased in [TechAmerica Foundation, 2012]:

> Big data is a term that describes large volumes of high-velocity, complex, and variable data that require advanced techniques and technologies to enable the capture, storage, distribution, management, and analysis of the information.

At the same time, the world of science has changed with the emergence of a fourth scientific paradigm: *data-intensive science* [Bell et al., 2009]. The first two scientific paradigms are several hundred years old and are *empirical science* and *theoretical science*. More recently, science has been using scientific simulations to describe complex phenomena that cannot be captured by only the two first paradigms, giving birth to the *computational science* paradigm. Today, experiments and simulations generate large data sets that need to be analyzed, using what is called data-intensive science. Some of the first fields confronted with data-intensive science were astronomy (telescopes) and fundamental physics (particles).

7.1.1 The V's of aircraft engine data

The aim of this section is to explain why data from the aviation sector, and in particular data collected on civil aircraft engines, can be qualified as *Big Data*, by deriving some of their relevant *V's*. The focus is put on an important application of engine data analysis: *health monitoring*. Engine health monitoring (EHM) broadly consists in monitoring the state of an engine or a fleet of engines by using engine data and past events, in order to improve engine operation and availability [Blanchard et al., 2009]. The first objective is to avoid abnormal events as in-flight shutdowns (IFSD), aborted take-offs (ATO) and delays and cancellation (D&C). A second objective is optimizing maintenance operations to improve safety while reducing costs for aircraft and engine manufacturers and airline companies. In this thesis, we will mostly adopt the point of view of the engine manufacturer. EHM requires

calculating health indicators and running algorithms on data sets that are becoming larger and larger.

Volume

Today's highly instrumented aircraft produce huge amounts of data that range from gigabytes to terabytes per flight. An average flight generates between 500 and 1000 gigabytes [IBM Global Business Services, 2015]. The number of sensors installed on new aircraft models is exploding. Airbus 380-1000 aircraft are equipped with 10000 sensors on each wing and generate several terabytes of data per day [Marr, 2015]. The data concerning aircraft engines is also growing, as the number of sensors keeps growing. Several hundreds of variables are measured on an engine, and on more recent models, the sensor measures are recorded on the whole flight at high frequency. For example, continuous data of more than 500 variables are recorded on LEAP engines at Safran Aircraft Engines, and the sampling frequency can reach up to 60 Hz. Data relative to vibrations study have a particularly high frequency. Taking into account that flights can last for several hours and that thousands of engines are operating every day, it represents a considerable volume.

Velocity

Velocity is the speed at which data is produced and needs to be processed. One kind of data often used by airlines and even by the public is real-time aircraft route data including aircraft type and tail number, airline, geographical location, speed, altitude and route destination. In the USA, this kind of data is made available through the Aircraft Situation Display to Industry (ASDI) service[4], a data stream service of the US Department of Transportation. Today, the data sources are connected by a central platform part of the SWIM program[5] (System-Wide Information Management) and the data encompasses weather information and airport status. These data arrive in real-time from the thousands of aircraft in circulation. Most often, aircraft data recorded during flights are only retrieved occasionally (every batch of flights) for ad-hoc analysis, and do not have the property of high velocity. But in the future, more and more data will be streamed in-flight for real-time analysis, so velocity will become a challenge for companies and manufacturers.

[4] https://en.wikipedia.org/wiki/Aircraft_Situation_Display_to_Industry
[5] https://www.faa.gov/air_traffic/technology/swim/

Variety

Aviation data variety is immense. Most of the data is structured, but there are also cases of unstructured data. An example of unstructured data is text data coming from maintenance log reports. Recently, aircraft manufacturers have been starting to develop systems to automatically process log reports (written in English) using natural language processing techniques. However, the immense majority of the data is structured and consists in physical quantities: geographical position, speed, altitude, temperature, pressure, cockpit commands, aircraft configuration, etc. On an engine, examples of variables are the rotation speeds of the engine fan and core, temperatures and pressures at different parts of the engine, fuel flow, oil pressure, etc. When these quantities are measured at a sequence of time steps during a flight, we speak of time series data.

Veracity

Veracity of the data can be a challenge at several moments of the analysis. At data generation, errors can be introduced by sensor failures. Data extraction steps happening between the recording of the data and its use for analysis also have risks (e.g. decoding, ingestion, etc.). Then, sensor data is always noisy and the noise should be removed before further utilization, without loosing information. Additionally, when working with time series data, re-sampling and interpolation can also create artifacts and corrupt the data. Thus, data veracity must be verified throughout the process, in particular if the results are used for decision-making.

Volatility

Volatility really depends on the use case. However, as aircraft and their engines remain in operation for decades, their data should be preserved for a very extended period of time. Even years later, historical data may be needed for an analysis.

Value

Most of the data generated by aircraft remains untapped, but today this fact is changing rapidly, as aviation actors realize the value contained in these data. For engine manufacturers, analyzing engine data enables to detect potential failures before they happen and raise alerts. If necessary, these alerts are sent to customers

(aircraft operators) as a Customer Notification Report (CNR). For example, GE's predictive system generated 350000 alerts resulting in 9000 CNRs on their fleet in 2015 [Broderick, 2016]. An automatic data-driven CNR generation application has also been developed at Safran A.E. Beside health monitoring applications by manufacturers, companies are also developing data-driven products and services. For example, Airbus developed its Skywise platform, providing subscribing airlines with predictive analytics [Airbus, 2018]. The Safran group has developed the Cassiopée product [Safran, 2018]. Airline companies are already using Big Data analysis for various applications, including airline route optimization ([Diebold, 2000], [Kasturi et al., 2016]). Data analytics systems could also be used to provide pilots and crew with insights and predictive information while in flight. However, this poses human factors-related questions as, how to display this information on cockpit displays, and how it should be interpreted [Oh, 2017].

7.2 Hadoop and the Map-Reduce paradigm

This section will introduce a widely used paradigm for Big Data processing, *Map-Reduce*, which originated in the needs of large Internet companies to process the huge amounts of data they generate every day. The popular open-source Hadoop platform implements various components for handling huge data volumes. Mastering these technologies can enable to process efficiently the data generated by aircraft engines and develop data-driven applications such as engine health monitoring.

7.2.1 Map-Reduce

At the Internet era, petabytes of information are being generated by billions of users and can no longer be stored on single machines and processed in traditional ways. In response to the need for analyzing and processing these data faster, companies developed solutions to process data spread across clusters of machines. In particular, the web giant Google was at the forefront of Big Data processing and built upon a simple but efficient and cost-effective paradigm called Map-Reduce [Dean and Ghemawat, 2008] that allowed to process data in parallel on a cluster. The idea was that many jobs could be written as two operations: a *map* operation, and a *reduce* operation. The terms map and reduce come from the functional programming vocabulary, and are functions applicable on iterable data structures. To further explain how the Map-Reduce paradigm works, we first need to define some basic vocabulary:

- A *cluster* is a set of similar computers (called *nodes*) on the same local network (if the nodes are geographically distributed and have heterogeneous hardware, it is called a *grid* instead).
- In a cluster, we generally distinguish *worker* nodes, responsible for carrying out computations, and one *master* node, responsible for supervision, scheduling, sending tasks to the workers, allocating resources, etc.

The data are stored in a distributed filesystem or database. One of the first distributed filesystem is the GoogleFS (GFS). Before going into more details, the main idea is that large files are cut into chunks (64MB in GFS), replicated for redundancy, and distributed across the nodes of the cluster. The metadata, which contain the locations of the blocks composing each file, are managed by the filesystem's master node.

The key concept of Big Data processing that Map-Reduce is taking advantage of is *data locality*: bringing the processing to the data, rather than retrieving the data to a machine where the computations take place. In Big Data, a golden rule is to avoid transferring data between machines as much as possible, due to network latency. Moving data between nodes is called *shuffling*. The key feature of this framework is to abstract out to the application programmer the precise details of where the different parts of the data are stored: "the run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system" [Dean and Ghemawat, 2008].

The *map* function takes a key-value pair and returns an intermediate list of key-value pairs. The *reduce* function takes this list and merges all values having the same intermediate key. A Map-Reduce program is roughly composed of three steps:

1. **Map step**: the map worker nodes apply the map function to their local data (in parallel) and write the intermediate key-value pairs to disk.
2. **Shuffle step**: data are redistributed across the worker nodes so that the intermediate data belonging to the same key are located on the same reduce worker nodes.
3. **Reduce step**: the reduce workers recursively apply the reduce function per key (in parallel) and write the output on a reduce partition on disk.

See Figure 7.2 for an illustration of these three steps. Consider the common example of word counting, where the goal is to count the number of occurrences of words in

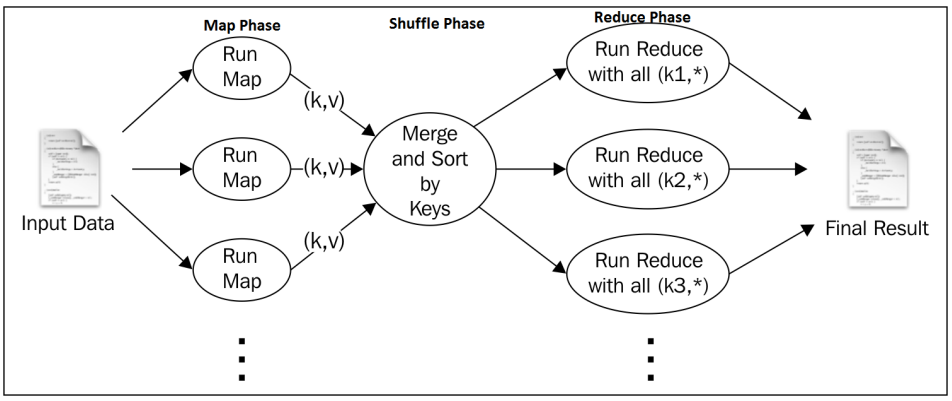


Fig. 7.2.: Illustration of the Map-Reduce paradigm.

a large set of documents (possibly billions). The key idea is that the word counts for subsets of the documents can be computed independently in parallel on different machines, and be combined to obtain the final result. In this example, the map operation splits the document into words, and for each word, emits a key-value pair with the word as the key and 1 as the value corresponding to the word count. Then, the reduce operation iterates through the key-value pairs (associated with the same key), sums the values corresponding to the partial word counts, and return a key-value pair with the word as the key and the sum of the partial counts as the value

```
function map(String input_key, String input_value):
    // input_key: document name
    // input_value: document contents
    for each word w in input_value:
        EmitIntermediate(w, 1)

function reduce(String output_key, Iterator intermediate_values):
    // output_key: a word
    // output_values: a list of counts
    int result = 0
    for each v in intermediate_values:
        result += v
    Emit(result)
```

Listing 7.1: Word count Map-Reduce pseudo-code.

See 7.1 for the pseudo-code of this example. Google's original MapReduce software was implemented in C++, and benchmarked on tasks like distributed sort and distributed *grep* (searching for character patterns in text data). In 2007, 1PB of data

was sorted in 12 hours; in 2011, this time was reduced to 33 minutes[6]. Since then, performance is continually improving.

The first usage of Map-Reduce was analyzing web logs, but it can be used for other computational tasks, especially those involving linearly computable statistical functions over the elements of the data set [Aggarwal and Reddy, 2013], which is the case of many data mining steps and machine learning algorithms, as we will see later.

7.2.2 The Hadoop platform

Hadoop[7] is an open-source platform for scalable distributed processing of large data sets, inspired from MapReduce, GoogleFS and BigTable (Google's distributed database management system), and supported by the Apache Software Foundation since 2009. It was invented by Doug Cutting, former software engineer at large Internet companies, now working at Cloudera and chairman of the Apache Software Foundation board of directors since 2010. The Hadoop platform is composed of many software components and tools, with at its core, HDFS (the Hadoop Distributed File System), Hadoop MapReduce (Hadoop's implementation of Map-Reduce), and, since Hadoop version 2, the YARN (Yet Another Resource Negotiator) scheduling and resource management system. Other important components are HBase (a column-oriented distributed database inspired from BigTable), ZooKeeper, Pig and Hive (a data warehousing software enabling to perform SQL-like requests on HDFS). Most Hadoop components were developed in Java and as a consequence, the whole ecosystem heavily relies on the JVM (Java Virtual Machine). The main drivers of Hadoop are the following:

1. **Scalability**: the goal is to be able to process huge volumes of data (terabytes/petabytes), structured or unstructured and collected from various sources.
2. **Distributed processing**: data-parallel processing.
3. **Fault tolerance**: the system must be tolerant to task failures and losing cluster nodes, using block replication.
4. **Cost-effectiveness**: running on clusters of *commodity hardware*, i.e. using a large number of already-available, low-performance and low-cost hardware, instead of fewer high-performance, high-cost computing components.

[6] `https://cloud.google.com/blog/big-data/2016/02/history-of-massive-scale-sorting-experiments-at-google`
[7] `https://hadoop.apache.org/`

For more information about the nuts and bolts of the main Hadoop components and how tasks are orchestrated on a cluster, refer to Appendix D. The Spark framework used throughout this PhD relies on these components, and is introduced right now.

7.3 Efficient analytics with Apache Spark

Apache Spark[8] [Apache Spark, 2014] is an open-source distributed general-purpose processing engine able to perform high-performance, in-memory and resilient computations on both batch and streaming data. The project started at Berkeley's AMPLab, a lab focused on Big Data analytics, by Matei Zaharia. In 2013, the creators founded the Databricks company, a company helping clients with cloud-based data processing with Spark, developing a web-based platform for using Spark, providing online courses and organizing the Spark Summit conferences. The project is now part of the Apache Software Foundation. Spark is probably the most used large-scale data processing framework and can be used for various tasks. Other projects of data-parallel computation engines are Apache Tez[9], Apache Flink[10], Apache Apex[11], Apache Storm[12], the Microsoft Dryad project[13], the Python Dask library[14], Apache Drill, Cloudera Impala, Apache Crunch, Cascading (and its Scala version Scalding), Google Cloud Dataflow with Apache Beam[15]... See for instance [Inoubli et al., 2018] and references therein for surveys on Big Data processing frameworks.

7.3.1 Apache Spark

The Apache Spark framework improves on some of the limitations of Hadoop and MapReduce. Indeed, Hadoop is not adapted for iterative tasks like, for example, machine learning, as it is only using disk for reading and writing results of operations. Disk access is slow and a bottleneck for such applications. On the contrary, Spark is capable of *in-memory* computations, meaning that it can perform computations by storing data in RAM. Furthermore, it provides an interactive interface called the *Spark shell*, allowing to manipulate large data sets interactively for ad-hoc, interactive

[8] https://spark.apache.org/
[9] http://tez.apache.org/
[10] https://flink.apache.org/
[11] http://apex.apache.org/
[12] https://storm.apache.org/
[13] https://en.wikipedia.org/wiki/Dryad_(programming)
[14] http://dask.pydata.org/
[15] https://beam.apache.org/

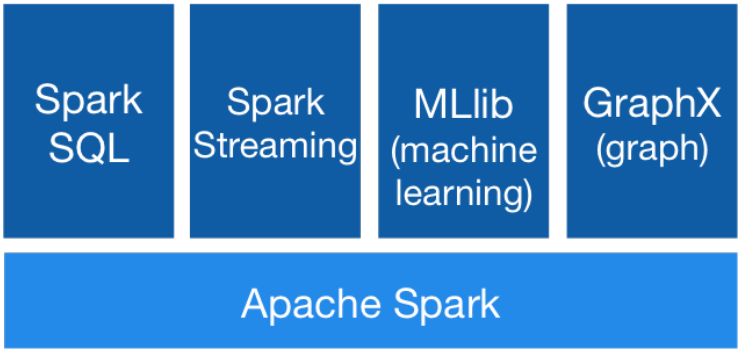


Fig. 7.3.: Apache Spark: a unified analytics stack. In this thesis, we use Spark SQL and MLlib in addition to the core functionalities.

analysis, as data scientists are used to for smaller data sets. The Spark framework is a stack of components composed of the Spark Core, Spark SQL (allowing to perform SQL queries on Spark data sets), Spark Streaming (for processing streaming data), MLlib (Spark's machine learning library) and GraphX (graph processing library). The stack is represented on figure 7.3. The framework is written in Scala, a modern JVM language presented in more details in the following paragraphs. It provides APIs to write applications in Java, Scala, Python or R. Spark runs on a cluster and can connect to 4 different cluster managers:

- **Standalone**, the simple cluster manager included with Spark.
- **YARN**, the resource manager of Hadoop v2.
- **Mesos**, another cluster manager.
- **Kubernetes**, experimental, since Spark 2.3.

The standalone cluster manager, included with Spark, has limited scheduling capabilities but has the enormous advantage that it can be run in *local* mode, i.e. without access to a cluster. This enables to run and test Spark jobs locally before running them on a cluster. In addition, it can run on cloud clusters like Amazon EC2. Finally, Spark is compatible with most of the data sources used on Hadoop (HDFS, Hive, HBase, Cassandra, etc.).

7.3.2 Functional programming

The functional programming paradigm in Big Data

Big data processing is tightly linked with the functional programming paradigm. The abstraction of functions is essential in programming in general: instead of

writing the same code dozens or even thousands of times, it can be written once and reused as needed, thus reducing the amount of code, reducing the risk of bugs and mistakes, and speeding up the development process. In functional programming, we use functions that are close to the concept of mathematical functions, called *pure functions*, that have the property of having no *side effects*, meaning that they cannot modify other variables, and that every time it is called with the same input, the output remains the same (as with a mathematical function). In distributed data processing, we want tasks to run in parallel on different machines without knowing in which order and how many times the tasks will be executed (Spark is able to re-run failed tasks for fault tolerance), so the functions used are close to pure functions.

Spark uses the same kind of operators as in functional programming, such as *map* (to apply a function on a collection of elements), *fold* (to iterate through a list), *reduce* (to combine elements recursively using an associative operator), etc. The functional programming concept of *higher-order functions* (i.e. functions that take another function as argument or return a function) is omnipresent in distributed programming. As we will see, Scala is particularly oriented towards the functional programming style.

Programming languages for writing Spark applications

The four languages for writing Spark applications are Scala, Java, Python and R. We will quickly present the pros and cons of each of the languages.

Data analytics is often an iterative, interactive process where data scientists and other engineers need to query and analyze the data without writing complete applications and with near real-time responses. This requirement is a major deal breaker for choosing the programming language. Java does not support REPL (Read-Evaluate-Print-Loop), whereas both Scala and Python offer such functionality.

Scala is a modern, multi-purpose programming language combining functional and object-oriented programming. It runs on the JVM (Java Virtual Machine), and is thus fully compatible with Java libraries. It is a typed language (unlike Python which is non-typed), but far more concise than Java, and with a type inference system. Like Java, it is compiled to bytecode that is interpreted by the JVM (once more, unlike Python that is interpreted). It offers powerful object-oriented features (classes, traits, class composition with mixins, etc.), and is particularly suited for functional programming (high-order functions, currying, etc.) and also has a pattern matching feature. It has a handy data structure called *case class* for classes that

only contain typed data fields and no methods. We will see that case classes can be useful for pattern matching and working with typed Spark data sets (note that Python 3.7, released in summer 2018, introduces a similar concept called *data class*). Furthermore, Scala is particularly adapted for concurrent and distributed applications by using the *actor* model[16], as well as asynchronous programming. The most used implementation of the actor model for JVM languages (Java and Scala) is the Akka framework[17]. Apache Spark itself is written mostly in Scala and based on the actor model for distributed processing. At the beginning, Spark was based on Akka, but this dependency was removed later by rewriting the needed functionalities from scratch. Scala applications are usually built using sbt (Scala Build Tool) or Maven. Here are the main advantages of using Scala for writing Spark applications:

- **Coding style**. Scala's functional and object-oriented programming style results in elegant code, as it is the **native** language of Spark.
- **Performance**. As a JVM language, it runs faster than interpreted Python code.
- **Safety**. Strong typing and compilation avoid runtime errors.
- **Dependency management**. Dependencies are added in the sbt or Maven build file and automatically downloaded if needed.
- **Deployment**. A Scala Spark application can be packaged into a single JAR file (including its dependencies).
- **Compatibility** with Java libraries.

Depending on the developer's background, it also comes with some hurdles:

- **Steep learning curve**. Scala comes with a lot of new concepts that can take time to be mastered.
- **Fewer libraries** are available compared to Python, as the community is considerably smaller. In particular, data analysis, machine learning and data visualization libraries are less rich and mature. However, the tendency is changing as more and more open-source Scala libraries are being developed and more and more companies are using it in industry.

Python is without a doubt the most used language among data scientists and can also be used to write Spark applications, thanks to the PySpark wrapper. We will not provide more details about the Python language as it is already well-known, but some pros and cons of using Python for developing Spark projects. Notable advantages are:

[16] `https://en.wikipedia.org/wiki/Actor_model`
[17] `https://akka.io/`

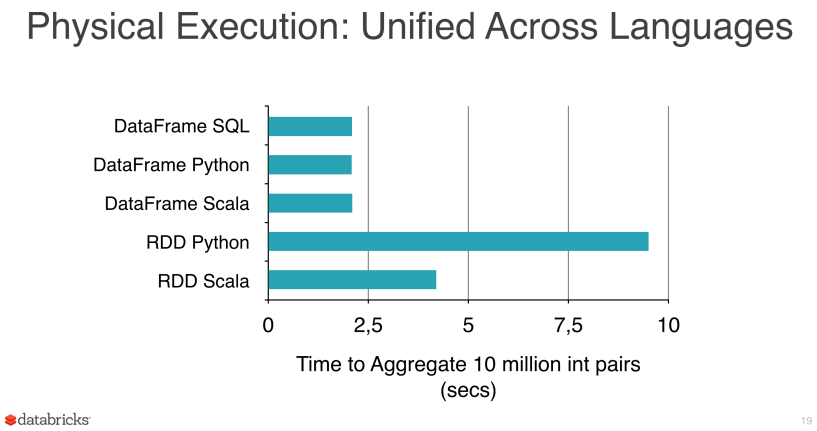


Fig. 7.4.: Benchmark of an aggregation task in Spark with RDD and DataFrame APIs in different languages.

- **Easy to learn, versatile and powerful**. Most data scientists and engineers master the Python language, and it can be used for a large variety of applications.
- **Thousands of powerful libraries** exist in Python for almost all kinds of use cases. Linear algebra, statistics, machine learning and data analysis libraries (e.g. `numpy`, `scipy`, `pandas`, `scikit-learn`, etc.) are particularly rich and efficient (relying on fast vectorized underlying C and C++ implementations).

Disadvantages are:

- **Performance**. PySpark is a wrapper around Spark, and Python is intrinsically slower than compiled languages.
- **Safety**. Runtime errors can easily happen as there is no type-checking.
- **Deployment** is not as easy as with Scala. Files and dependencies must be packaged in a `.zip` or `.egg` archive and may require virtual environments.

However, when working with a lot of cores, performance is not a major driving factor in choosing the programming language for Spark, as the global performance of the application will depend more on data-parallel processing than on pure language performance. Moreover, Spark SQL's DataFrame API for structured data sets performs special optimizations with the Catalyst optimizer that generate a physical execution plan in form of JVM bytecode, so there will be no performance difference between using Scala or PySpark (benchmark on Figure 7.4). The conclusion is that each developer can use the language they prefer, depending on the background and

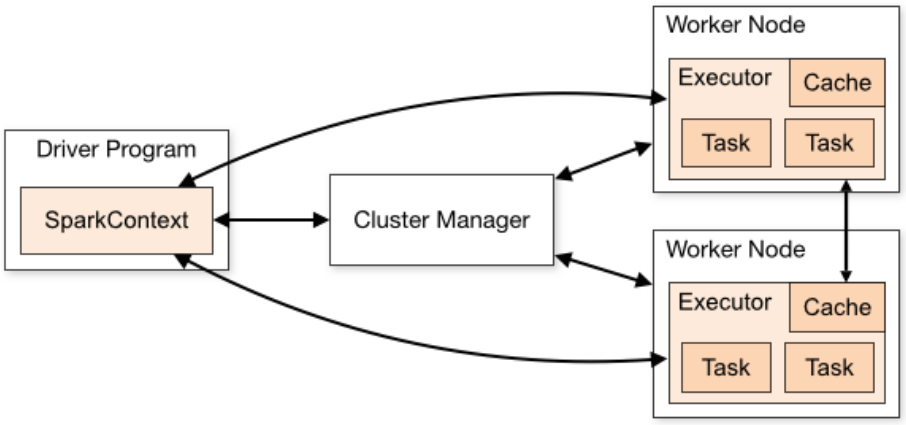


Fig. 7.5.: Overview of the Spark architecture. The SparkContext lives on the driver and talks to the cluster manager (e.g. YARN) to coordinate tasks executed on different nodes, called executors.

the context of the team. Data processing workflows can be composed of several components written in different languages.

7.3.3 Anatomy of a Spark job

As we explained in the paragraph on YARN, the resource manager negotiates resources to start containers on the worker nodes. In the case of a Spark application, the container processes are called *executors* and correspond to JVM instances that carry on computations and store data. As a consequence, there can be several executors running on a single node.

A Spark application is coordinated by the *SparkContext* that is located in the driver program and sends the application code (JAR or Python files) and *tasks* to the executors. On YARN, an application can be submitted in *client mode* (i.e. the driver runs on the client) or in *cluster mode* (i.e. the driver runs on a cluster node) using the `-deploy-mode` argument of `spark-submit`. This architecture is illustrated on Figure 7.5. Spark has a number of properties to configure application properties, execution behavior, scheduling, runtime environment, shuffle behavior, memory management, compression and serialization, networking, security and Spark UI settings. Additional properties exist to configure the different cluster managers and the components Spark SQL, Spark Streaming, GraphX and SparkR. On top of this, there are environment variables, logging configuration, and the configuration of Hadoop components (Hive). Inevitably, configuration is a huge part of the

work in successfully running Spark jobs. Some essential performance tunings are explained thereafter. The entire list of available properties is available in the Spark documentation[18].

7.3.4 The RDD API

Apache Spark's fundamental data abstraction is the RDD (Resilient Distributed data set). As contained in its name, a RDD is a data collection that is replicated on several nodes for fault-tolerance (*resilient*) and can be processed in parallel (*distributed*). A RDD can be created either by parallelizing an existing collection in the driver program (e.g. a Python `list` or Scala `Seq`), or by referring to an external data source (e.g. a local file, HDFS, a database, etc.). Spark divides the data into *partitions* which can be processed in parallel on several nodes. The number of partitions (called the *parallelism*) can be determined automatically by Spark (equal to 2 to 4 times the number of cores in the cluster), or manually specified by the user.

Listing 7.2 and 7.3 present an example where we create a RDD from a list of integers from 1 to 100, and calculate the sum of squares by first applying a square function using the `map` method, and then summing the intermediate results by passing the summation function to the `reduce` method. As we can see in this example, we will make a heavy use of anonymous functions.

```
1  val rdd = sc.parallelize(1 to 100)
2  val squares = rdd.map(x => x*x)
3  val result = squares.reduce((a,b) => a+b)
```

Listing 7.2: Calculating a sum of squares on a RDD (Scala).

```
1  rdd = sc.parallelize(range(1,101))
2  squares = rdd.map(lambda x: x*x)
3  result = squares.reduce(lambda a,b: a+b)
```

Listing 7.3: Calculating a sum of squares on a RDD (Python).

In this example, Spark will divide the numbers from 1 to 100 in several partitions and apply the map and reduce functions in parallel. It is important to note that the map function itself returns a new RDD: this is called a *transformation*. The reduce method combines the elements of the squares RDD and produces the final result (an "int" variable): this is called an *action*. The essential idea behind this is that Spark creates a Directed Acyclic Graph (DAG) of transformations and actions, and only performs the needed computations when an action is called. In this case, Spark does actually not

[18] `https://spark.apache.org/docs/latest/configuration.html`

perform any operations until the reduce function is called. This is called *lazy evaluation*. Eventually, we could made the code more concise by combining both operations and writing `result = rdd.map(lambda x: x*x).reduce(lambda a,b: a+b)`.

A very common type of RDD is the pair RDD, where each element is a key-value pair. Pair RDDs, for example, have a `reduceByKey` method that reduces values having the same key. An example that we already mentioned is the word count example, where we count the number of occurrences of each word in a document. In this case, the text data is read from a file and the elements of the RDD are strings corresponding to the lines in the file. The Spark code for this example is presented in 7.4. Here again, Spark only reads the text file and performs the RDD transformations when the `reduceByKey` method is called.

```
rdd = sc.textFile("document.txt")
# collect() is a RDD action that retrieves
# the data in the resulting RDD
result = rdd.flatMap(lambda s: s.split())\
            .map(lambda w: (w,1))\
            .reduceByKey(lambda a,b: a+b).collect()
```

Listing 7.4: Word count Spark example on a text file.

7.3.5 Spark SQL and the DataFrame/Dataset API

Spark SQL is the module for structured data processing. Processing structured data is different from processing unstructured data, because Spark can leverage the structure of the data to perform optimizations. Spark SQL also allows to execute SQL queries on data sets, and from version 2.0.0, there is a built-in support for Hive that allows to read an create tables, perform HiveQL queries and use Hive UDFs, even without a Hive installation.

To represent structured data sets, Spark provides the DataFrame/Dataset API. Unlike RDDs which are collections containing any type of unstructured data (e.g. text), these abstractions represent the data in the same way as relational tables. In recent Spark versions, the API varies slightly between Python and Scala. In Python, the only data type is the `DataFrame`. It is a data set organized into named columns, and is very similar to the dataframes used by the pandas data analysis library. Under the hood, a Spark DataFrame is implemented as a RDD of `Row` objects. A Row is a structured type composed of named fields defining the schema of the table. In Scala, structured data is now represented using the `Dataset` class, which is a collection of typed elements, defined by a Scala `case class`. DataFrames can also be used,

in fact they are only an alias for a data set of Row objects (thus less general and flexible than typed data sets).

Structured data can be manipulated using the same functional transformations and actions than RDDs (map, filter, etc.), but in addition, it is possible to perform all kinds of operations specific to relational tables, for example aggregations (count, sum, min, max, average...), grouping, ordering, UDFs (user-defined functions), UDAFs (user-defined aggregation functions), etc. Operations can be performed using Spark data set/DataFrame methods or directly expressed as SQL queries. Finally, Spark supports a variety of data sources for structured data, for instance text files in CSV or JSON format, optimized data formats like Parquet or ORC, as well as JDBC database connectivity. Results stored in a data set/DataFrame can eventually be saved to disk as a persistent table (e.g. in a file or in Hive), with the possibility of using partitioning and/or bucketing. Note that partitioning when writing a table to disk is a completely different concept that Spark's partitioning.

An essential point is that this APIs should be always preferred to the RDD API, because Spark SQL's optimization engine (called Catalyst) will lead to better performance in most cases. The optimization engine is capable of building a physical execution plan from the queries, and we mentioned in the previous paragraph that it allowed the same high performance in Python than in Scala. In example 7.5, we create a DataFrame from an existing Hive table using a SQL query, group the data set on a column and perform a count aggregation.

```
1    # df and result both are DataFrames
2    df = spark.sql("select * from my_table")
3    result = df.groupBy("column").count()
```

Listing 7.5: Spark SQL count aggregation example on a table.

7.4 Distributed machine learning

In contrast to traditional single machine (or local) clustering, parallel and distributed algorithms use multiple machines to speed up the computation and increase the scalability. A somewhat old survey is available is available in [Aggarwal and Reddy, 2013]. Many existing clustering algorithms can be generalized to the Map-Reduce framework. Nevertheless, not every algorithm can be adapted efficiently. Map-Reduce is particularly effective for *data-parallel* linear computations, i.e. linear computations that can run independently in parallel on parts of the data. According

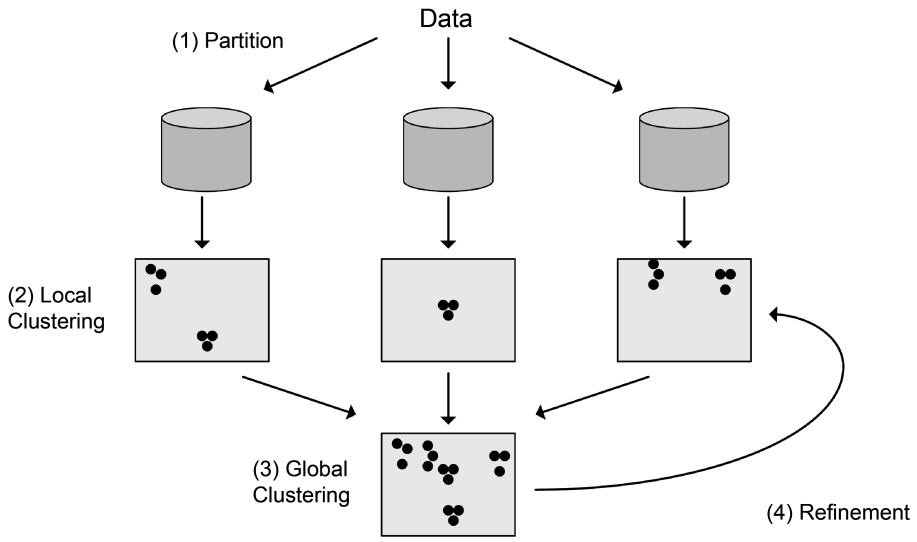


Fig. 7.6.: The general framework of most parallel and distributed clustering algorithms. [Aggarwal and Reddy, 2013]

to [Aggarwal and Reddy, 2013], most parallel and distributed clustering algorithms follow the general framework depicted in Figure 7.6:

1. **Partition**: data are partitioned and distributed over machines.
2. **Local clustering**: each machine performs local clustering on its partition of the data.
3. **Global clustering**: the cluster information from the previous step is aggregated globally to produce global clusters.
4. **Refinement** of local clusters: optionally, the global clusters are sent back to each machine to refine the local clusters.

K-means clustering can be easily distributed in Map-Reduce [Zhao et al., 2009]. First, data is partitioned across the nodes (step 1). This is already the case when using a distributed storage. The *map* operation assigns each data point to its cluster by computing the nearest centroid (step 2), assuming centroids are available on each node (e.g. using `broadcast` variables in Spark). The *reduce* operation groups the elements by cluster membership and aggregates them by computing the mean (step 3). Finally, as in standard K-means, we iterate between steps 2 and 3 until convergence (refinement step 4). This algorithm obtains exactly the same solution as the standard K-means. A distributed density-based clustering in [Januzaj et al., 2004] partitions the data, finds a small number of local representatives in step 2, merges the obtained representatives and clusters them using standard DBSCAN in step 3. A more recent Map-Reduce DBSCAN is MR-DBSCAN [He et al., 2011b]. Other algorithms, such as subspace, graph clustering or co-clustering, have also been

adapted to Map-Reduce (see [Aggarwal and Reddy, 2013], 11.4). An important challenge in distributed ML is to balance the load between the parallel workers. For more references on distributed ML and clustering in particular, refer to the theses [Sarazin, 2018, Beck, 2019].

7.4.1 Distributed SOM

The batch SOM algorithm (see Algorithm 1.2) can be easily distributed, as it expresses as an alternating procedure similar to K-means, that can be expressed as Map-Reduce. It was implemented and presented in previous work [Sarazin et al., 2014a, Sarazin et al., 2014b]. More precisely:

1. The data set is partitioned and the SOM prototype vectors are broadcasted, exactly as in K-means.
2. The *map* operation computes the BMU assignments b_i of each input $\mathbf{x}_i$ and computes the neighborhood-weighted coefficients necessary to perform the update operation in Equation 1.15: $\mathcal{K}^T(\delta(k, b_i))\mathbf{x}_i$ for the numerator and $\mathcal{K}^T(\delta(k, b_i))$ for the denominator.
3. Then, the *reduce* operation combines the intermediate results to compute the numerator and the denominator and finally update the prototypes.
4. In the refinement step, the previous steps are repeated using the updated prototypes until convergence.

7.4.2 Software implementations

Spark MLlib[19] is the official library for distributed machine learning on Spark. Many traditional classification and regression algorithms, as well as K-means clustering, and various preprocessing functions, are implemented. However, not all algorithms are available and unsupervised learning features are particularly limited. The community provides open-source implementations, such as the C4E (Clustering4Ever) initiative [LIPN, 2018] started at LIPN, gathering local and distributed implementations of clustering algorithms in Scala/Spark.

Implementations of the distributed batch SOM were already available, but not using the modern Spark ML API (Dataset/DataFrame). To better integrate with the rest of the code, the Spark ML SOM[20] [Forest, 2019] was developed during this PhD.

[19] `http://spark.apache.org/docs/latest/ml-guide.html`
[20] `https://github.com/FlorentF9/sparkml-som`

7.5 Conclusion

This chapter provided a general overview of Big Data processing beyond the buzzword, diving into the software tools that will be used in the industrial applications presented in the next chapter, in particular the Hadoop eco-system, Hive and the Spark framework. In addition, we introduced the Map-Reduce paradigm and how it allows to implement distributed machine learning algorithms. Hadoop, Spark and the distributed Spark ML SOM will be used in the engine fleet monitoring applications presented in the next chapter.

Industrial applications

8

This chapter is based on the contributions and patent:

- Forest, F., Lacaille, J., Lebbah, M., & Azzag, H. (2018). A Generic and Scalable Pipeline for Large-Scale Analytics of Continuous Aircraft Engine Data. *IEEE International Conference on Big Data.*
- Forest, F., Cochard, Q., Noyer, C., Cabut, A., Joncour, M., Lacaille, J., Lebbah, M. & Azzag, H. (2020). Large-scale Vibration Monitoring of Aircraft Engines from Operational Data using Self-organized Models. *Annual Conference of the PHM Society.*
- Lacaille, J., **Forest, F.**. Système d'environnement informatique pour la surveillance de moteurs d'aéronefs / Computer environment system for monitoring aircraft engines. *Patent No. FR3089501 (2020-06-12)*

8.1 Aircraft engine health monitoring

Nowadays, aviation industry and aircraft operation generate growing amounts of data that can be leveraged for various applications [Oh, 2017, Akerkar, 2014, Akpinar and Karabacak, 2017]. For engine manufacturers, an important one is *engine health monitoring* (EHM), part of the general field of Prognostics & Health Monitoring (PHM). The general aim is to improve availability and operation of engines [Blanchard et al., 2009, Bastard et al., 2016]. It consists in monitoring the state of an engine or a fleet of engines by using operational data and past events. The first objective is to avoid abnormal events as in-flight shutdowns, aborted take-offs and delays and cancellation. The second objective is optimizing maintenance operations to improve safety while reducing costs for manufacturers and airline companies. Maintenance is a crucial part of the engine life cycle (see Figure 8.1). Various costs are incurred by the possession of an engine throughout its life, represented on Figure 8.2, mainly linked to maintenance operations.

Maintenance operations often consists in preventively exchanging engine parts, after visual inspection by an operator, before serious wear or damage occurs. It also includes water-washing. Such maintenance is called *preventive*, as opposed to

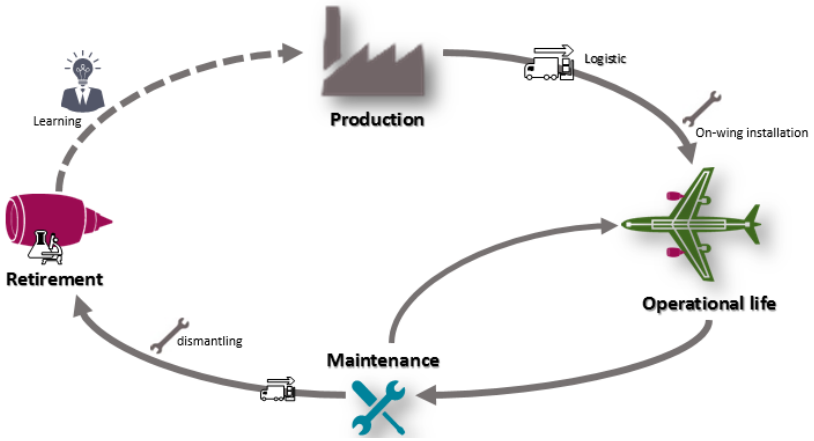


Fig. 8.1.: Life cycle of an aircraft engine from production to retirement. [Coupard et al., 2018]

Fig. 8.2.: Sources of costs for aircraft engine operators. [Coupard et al., 2018]

corrective maintenance, if an event as already occurred. In this case, maintenance is unexpected and thus very costly for the engine operator, because the engine will remain grounded. If maintenance happens at a fixed schedule, we speak of *time-based*, or equivalently, *scheduled* or *predetermined*) maintenance. Today, the standard scheduled maintenance plans are being enhanced by so-called *predictive* or *condition-based* maintenance. Figure 8.3 shows an overview of different maintenance types. Condition monitoring (CM) of industrial assets is a set of techniques that

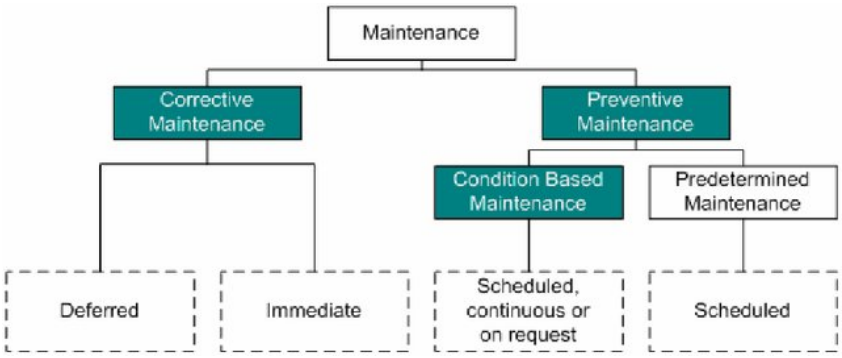


Fig. 8.3.: Overview of the different maintenance types.

aims at increasing machine availability and safety, while reducing maintenance costs (and thus the ownership cost). It is at the core of a predictive maintenance (PM) strategy (also called condition-based maintenance). Implementing a condition-based maintenance program requires in-depth knowledge of the machine's condition. This knowledge can be extracted from data, hence the term *data-driven* maintenance. Data-driven maintenance enables to detect faults and prevent failures before they happen, extending the life span of systems and reducing costs. PM for aircraft engines consists in adapting the maintenance plan to the actual state of each individual engine, unlike traditional time-based preventive maintenance, the state of each engine being the result of its actual use during its lifetime. This allows a more efficient scheduling of preventive and corrective actions (e.g. shop visits): time between actions can be increased if no maintenance is necessary (thus reducing costs), and actions can be taken earlier thanks to enhanced predictability of events (thus improving safety).

Concretely, CM combines historical data and physical models to raise alerts, build models that evaluate wear of parts and their residual useful life, probability of failure, etc. To build these models, the input data must provide information on the wear or an aspect of the engine state. Specific variables or features are generally selected by engine experts. For instance, the exhaust gas temperature (EGT) indicates engine wear. Then, these models can be based on thresholds, statistical models incorporating physical knowledge, or machine learning, i.e. statistical models whose

parameters are learned from historical data. In the following works, we tackle monitoring and raising alerts. Diagnosis and prognosis are then done by relevant experts.

From the operations point of view, data analytics are already used for fuel consumption and route optimization [Kasturi et al., 2016]. Aircraft manufacturers provide customized data-driven services to airlines [Airbus, 2018]. Engine manufacturers also provide such services, allowing client airlines to manage their operational data and use data analysis in the objective of improving safety, maintenance and reducing fuel consumption [Safran, 2018]. With the growth of air traffic, the volumes of data to be processed are growing exponentially and can no longer be handled in traditional ways: as an example, recent aircraft are equipped with tens of thousands of sensors and generate several terabytes of data per day [Marr, 2015].

EHM applications at Safran Aircraft Engines follow the OSA-CBM (Open Systems Architecture for Condition-based Maintenance) standard, described in [Bastard et al., 2016]. It is composed of several layers, represented on Figure 8.4, and aims at improving the clarity and modularity of applications, and enabling exchange within the company. The two use cases implemented during this thesis can be decomposed

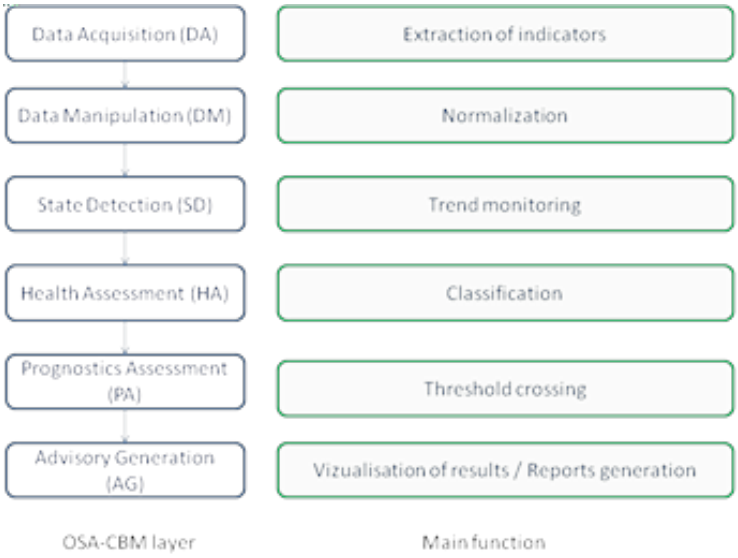


Fig. 8.4.: OSA-CBM architecture and examples of each step. [Bastard et al., 2016]

following this standard, as we will see later.

Applications of data mining and machine learning to PHM is a very active and promising research field. The methodology presented in [Cottrell et al., 2009, Côme et al., 2010, Côme et al., 2011] uses self-organizing maps to monitor the state of a fleet of engines based on expert indicators. We have chosen to develop this

methodology during this thesis. Features may also be learned from raw data. The application of deep learning architectures to PHM is discussed in [Zhao et al., 2019, Fink et al., 2020]. Several works applied deep neural networks to fault diagnosis, prognostics or anomaly detection from raw sensor data [Lv et al., 2016, Jing et al., 2017, Zhang et al., 2017b, Yan and Yu, 2019, Lee et al., 2020]. Even the recent deep clustering approaches (see Section 2.3 in Chapter 2) have already been applied in the PHM field [Qu et al., 2019].

8.2 Aircraft engine data sets

On an aircraft, data are collected during flights using flight recorders. The Quick Access Recorder[1] (QAR), for example, records over 2000 flight parameters at a maximum frequency of 1Hz, and its data is easily accessed by airlines or manufacturers using memory cards, USB or cellular network. It samples data at a higher rate compared to the Flight Data Recorder (FDR, also known as the "black box"), and its data is analyzed to improve safety and operational quality.

Another type of data is *snapshot reports*, which correspond to sets of parameters recorded on a short time interval (between 3 and 30 seconds) at key flight phases (e.g. during take-off, climb of cruise), aggregated and sent back to the ground near real-time using ACARS (Aircraft Communication Addressing and Reporting System) communication system.

Flight data are often combined with context or environmental data such as weather information, e.g. METAR (METeorological Aerodrome Report) or TAF (Terminal Aerodrome Forecast).

8.2.1 LEAP CEOD

Continuous Engine Operational Data (CEOD) are composed of several hundreds of parameters recorded during entire flights on recent aircraft. Safran A.E. receives the CEOD data sets for the LEAP engine family. Due to the large number of parameters recorded at high frequencies, these data contain much more information compared to lighter types of aircraft data sets, for instance snapshot reports. This leads to large volumes that can no longer be processed in traditional ways. For now, due to bandwidth limitations, continuous data are offloaded post-flight. In future, these

[1] https://en.wikipedia.org/wiki/Quick_access_recorder

data may be streamed in real-time, allowing for live predictive analytics. We quickly present the current acquisition process of CEOD:

1. Airline operators manually download raw data from aircraft flight recorder. Depending on the time since last download, raw CEOD may contain the concatenated recordings of several flights, as the flight recorder writes into memory in a sequential manner.
2. Raw data are decoded into a structured file format using a proprietary software.
3. Files are cut into distinct flights by detecting flight start and end based on sensor values, and ingested into a Safran A.E. Hadoop cluster. As we have large volumes of structured data, it is stored on HDFS using the Hive data warehouse in ORC format.

Concretely, each flight (identified by its `flight_id`) consists in a set of flight parameters (`param`) measured at given timestamps (`time`). The data used in our experiments concern a fleet of engines of the same type, identified by their *engine serial numbers* (ESN/`esn`). As a consequence, CEOD are a set of univariate time series. Because the flight recorders dynamically adapt their sampling frequency, time series can have very different frequencies, and they can vary throughout the flight. See Table 8.1 for the Hive table schema specification. In this format, observations

Tab. 8.1.: CEOD Hive table schema.

Column name	Type
`esn`	`string`
`flight_id`	`string`
`param`	`string`
`time`	`timestamp`
`value`	`float`

are stored row-wise, i.e. one row per flight, per parameter and per time step at which a value was recorded for this parameter. The main advantages of this format to represent CEOD time series are:

- The format is fixed and robust to evolutions in the data: in particular, parameter names are expected to change (e.g. with new versions of engines or flight recorders).
- It handles variable frequencies: some parameters have a high frequency, whereas others only have few measurements throughout the flight.

The summary of data set properties in Table 8.8 illustrates the fact that we already are in a high volume context. The volumes are expected to grow enormously in the

coming years. Health monitoring must provide insights on the incoming data quickly (and in a not so distant future, in real-time), thus we also have a high velocity. Altogether, the processing of CEOD for health monitoring requires the use of "Big Data" methods and tools.

8.2.2 Time series data representations

There are at least two ways to represent time series data in a relational table. We call *observation* the tuple consisting in a timestamp, a key identifying the variable (here, the key is composite and is composed of `esn`, `flight_id` and `param`), and the value of the variable. Given a single point in time, we call *instant* the values of all observed variables at this timestamp (if the variables are not measured at the same timestamps, there may be no value for some of the parameters). This defines two ways of representing a set of univariate time series in a table:

- The *observations* table representation has one column containing the timestamps, one column for the key (or several columns if there are several keys) and a column for the values. Each row contains the observation of a variable at each timestamp. The timestamps are not necessarily ordered. An example of an observations table is represented on Table 8.2, with only one key corresponding to the variable name for simplicity.
- The *instants* table representation has one column containing the timestamps and one column per variable. Each row contains an instant, i.e. the values of the variables at a given timestamp. The values of some variables may not exist for every timestamp. See Table 8.3 for the corresponding example. Actually, there are often several keys (e.g. group keys and the variable key), so there will be additional columns for the group keys.

Tab. 8.2.: Observations table of time series data.

Timestamp	Key	Value
2018-06-12 16:33:42	TEMP	1337.0
2018-06-12 16:33:44	TEMP	1342.5
2018-06-12 16:33:43	ALTITUDE	10057.0
2018-06-12 16:33:42	SPEED	145.0
2018-06-12 16:33:43	SPEED	147.5
2018-06-12 16:33:44	SPEED	151.0

Each representations has its pros and cons and are more or less practical depending on applications. The advantages of the observations are:

Tab. 8.3.: Instants table of time series data.

Timestamp	TEMP	ALTITUDE	SPEED
2018-06-12 16:33:42	1337.0	NA	145.0
2018-06-12 16:33:43	NA	10057.0	147.5
2018-06-12 16:33:44	1342.5	NA	151.0

- **Fixed schema**. The columns remain the same regardless of the number of keys and observations. This makes it easy to append new observations at the end of the table without rewriting the whole table. This is adapted for data storage formats that do not allow schema evolution.
- **No need for preprocessing or resampling**. Metrics can be stored in the table without further preprocessing (e.g. missing value imputation, resampling, time index alignment etc.).

However, notable disadvantages are:

- **Redundancy and inefficient storage**. The number of rows in an observations table is equal to the product of the number of keys times the number of timestamps per key, resulting in a very large number of rows. For aircraft data, the key typically consists in a flight ID and the name of the recorded parameter, and there are typically thousands of flights, hundreds of parameters and thousands or even hundreds of thousands of values recorded for a parameter per flight. The resulting number of rows is $\#flights \times \#parameters \times \#samples \sim 10^3 \times 10^2 \times 10^5 \sim 10^{10}$, representing many billion rows. In addition, the keys are unnecessarily repeated in each row. On the whole, this results in bulky storage volumes (mitigated by compression), and data warehousing systems will have to scan a large number of rows to retrieve values for a specific key or range of dates.
- **Unpractical for many operations**. In many cases we need to perform operations that require grouping the data by key or by timestamp, which is cumbersome and also computationally expensive because they require scanning the whole table. For example, to compute an aggregation (e.g. sum or an average) of a variable, we need to first filter the corresponding rows (using a `WHERE` clause in SQL). These operations would be much easier if the keys were stored column-wise. In particular, in machine learning, it is traditionally expected to have a column for each feature as in the instants representation, so we must first go through a preprocessing step before applying a machine learning algorithm. Turning observations into instants requires pivoting the table and is an expensive operation. Finally, if variables have unequal sampling frequencies, subsequent operations will have to deal with resampling.

The instants schema solves the main disadvantages of the observations schema but has also its limitations.

- **More efficient storage**. The instants representation has as many columns as variables ($\#parameters \sim 10^2$) but far fewer rows. Taking the same assumptions as previously, the order of magnitude of the number of rows is $\#flights \times \#samples \sim 10^3 \times 10^5 \sim 10^8$, less than one billion rows. Furthermore, storage is saved because the group keys are repeated on fewer rows and the variable names are not repeated on any row.
- **Practical for column operations and machine learning**. Variables can be selected conveniently.

This schema is also more difficult to obtain and to manage due to following disadvantages:

- **Variable schema**. The table schema is not fixed, as it depends on the variables. This can become difficult to manage when we do not know in advance what these variables will be. It is more difficult to add a new variable to an existing table.
- **Missing values**. If the sampling frequencies of the variables are very different, the resulting table will be sparse. Variables sampled at a low frequency will have mostly missing values. As a consequence, if there is an important disparity in the frequencies, the instants representation will actually store a lot more values than the observations and be *less efficient in terms of storage*! In our previous estimations, we supposed that the numbers of samples had the same order of magnitude, which may not be the case.
- **Downsampling and interpolation** may be necessary to limit the number of rows and for missing value imputation.

Lastly, alternative representations can be used to mitigate the disadvantages of the two aforementioned representations. We can propose a *series-observations* representation where an observation is not the value of a variable at a given timestamp, but the whole sequence of values of this variable along with the sequence of the timestamps (see Table 8.4). The fields thus contain vectors of values (`array` type in Hive). This representation has a fixed schema and the row dimension is rid of the temporal dimension, reducing it drastically. This representation is useful when variables have very different numbers of samples and frequencies (if their is a single frequency, the time indices would be unnecessarily repeated). Using the same orders of magnitude as before, the number of rows is reduced to approximately $\#flights \times \#parameters \sim 10^3 \times 10^2 \sim 10^5$. However, akin to the observations representation, it does not allow for easy column operations on variables. Moreover,

the table fields contain complex types (arrays) and may contain very long sequences, which is not optimal for data storage and performance (compression and serialization in particular). Each representation is best suited for different usages. For our

Tab. 8.4.: Series-Observations table of time series data.

Key	Timestamps	Values
TEMP	[2018-06-12 16:33:42, 2018-06-12 16:33:44]	[1337.0, 1342.5]
ALTITUDE	[2018-06-12 16:33:43]	[10057.0]
SPEED	[2018-06-12 16:33:42, . . . , 2018-06-12 16:33:44]	[145.0, 147.5, 151.0]

use cases, we will transform the original CEOD representation (observations) into the series-observations format.

8.3 Scalable and generic processing of aircraft engine data

8.3.1 Introduction

Choosing the adequate Big Data infrastructure and software tools for storage, analysis and visualization is a major challenge for actors in the web industry but also in more traditional ones such as as aerospace [Murugan et al., 2014, Li et al., 2017]. The Hadoop platform was chosen by a large number of companies, but its utilization is not straightforward for non-trained engineers. In aerospace, the domain knowledge is held by engineers who often have sufficient programming skills to implement algorithms in some programming language (e.g. Python) and run them on a small to moderately large, local, data set (e.g. a CSV file), but are not acquainted with Big Data tools and programming frameworks that are needed to efficiently query and process large, distributed data sets stored on a cluster.

Our objective is to design and implement a generic and scalable processing pipeline to power health monitoring applications based on operational aircraft engine data. The goal is to present a completely operational pipeline based on open-source software tools from the Hadoop ecosystem and ready to use in an industrial setting. Our main contributions are:

- Processing and analyzing real, large-scale, industrial data sets coming from thousands of operating aircraft engines.
- Combining recent programming techniques, open-source technologies and a Big Data infrastructure, which is not yet common in traditional industries such as aerospace and engine manufacturing in particular.

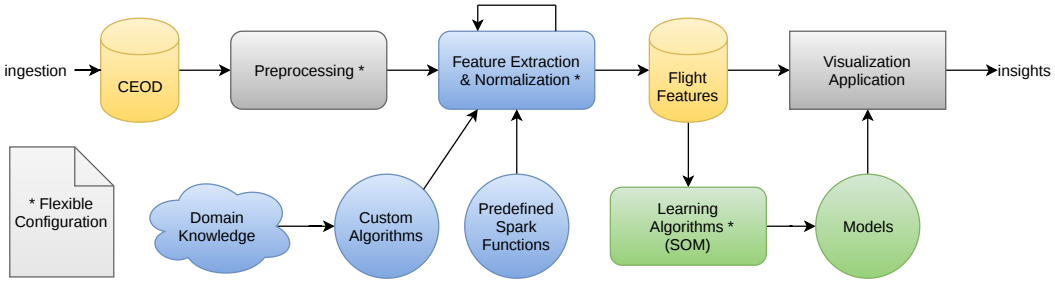


Fig. 8.5.: Diagram representing the analytics pipeline.

- Focusing on genericity and allowing domain engineers (who are not software engineers nor data scientists) to deploy their domain-specific computations and algorithms in an agile way.
- Demonstrating how scalable unsupervised learning algorithms can be integrated and used for visualization in engine health monitoring applications.

Data analysis and machine learning have already been used extensively for health monitoring applications across all industries. However, these approaches do not address the issue of large-scale processing of large volumes of data. In recent years, much effort has been put into designing and building Big Data architectures by aerospace industry actors. [Murugan et al., 2014] present a Big Data architecture for storing and analyzing operational and repair data at Honeywell. Their operational data come from multiple sources and supports, for example snapshot and summary data from the ACMS (Aircraft Condition Monitoring System). This typically includes parameters as speed, position, altitude and exhaust temperature at specific flight phases. Repair data consists in shop visit reports. They use a Hadoop cluster and store data in HDFS. For analytics, they use R along with packages to interact with Hadoop, and also provide analytics as a service through either an R Shiny web application, or an OpenCPU RESTful service for interacting directly with R. Unlike their approach, we will use the Spark framework for distributed computing. Reference [Ayhan et al., 2013] from Boeing use a Big Data architecture to analyze near real-time ASDI (Aircraft Situation Display to Industry) data. These data are a feed of XML messages, which have to be translated into a relational database. Their infrastructure relies on IBM software for message brokering, database management (DB2), processing and business intelligence. They present their use case and some optimizations for near real-time processing. Their data is quite different from our continuous operational data, and we use a Hadoop ecosystem.

8.3.2 Analytics pipeline

This section details the components of our processing pipeline. An overview of the different steps is represented as a diagram in Figure 8.5.

Technological stack

Data is stored on a Hadoop cluster using Hive. For computations, we use the Apache Spark distributed processing engine [Apache Spark, 2014]. In particular, we use Spark SQL and the Dataset/Dataframe API. The Spark jobs are written either in Scala or in Python (using the PySpark wrapper). For linear algebra, numerical analysis and machine learning, we used Spark ML, as well as Python and Scala libraries (numpy, scipy, pandas, scikit-learn, breeze, smile). Finally, for visualization, we developed web applications using HTML/JavaScript and the D3.js library or the Plotly/Dash framework, as well as a minimalistic Python server with Flask to interact with Hive via the JDBC connector.

CEOD preprocessing

The computations carried on during the feature extraction step will need to process the whole time series of a flight parameter, so it is not necessary to have one row per time step in our Spark Dataframe: the smallest entity to be processed in parallel is a parameter during an entire flight. Thus, we aggregate the CEOD on the time dimension into vectors of timestamps and values, using Spark SQL's `collect_list` aggregation. This drastically reduces the number of rows (from approximately 200 billion to 5 million rows) and accelerates subsequent processing. It only needs to be executed once on new incoming data. See Fig. 3 for the resulting schema.

Feature extraction

The first step in EHM is to select and compute a set of features that represent the health state of an engine or engine sub-system (for example fuel, oil or control system) at a given flight. Such features can be as simple as the value of a parameter at a specific instant of the flight (e.g. the EGT) but they may be more complex features that were engineered by domain experts. Computing these features is an essential step of our processing pipeline, as they will be the input for the subsequent

algorithms and define what aspects of the engine health will be monitored. The main requirements of the feature extraction step are:

1. **Scalability.** Features are extracted in a distributed data-parallel way, allowing to process a huge number of flights and engines in parallel.
2. **Genericity.** It is possible to use generic functions and already existing algorithms to compute features.
3. **Ease-of-use.** Engineers should be able to implement and use their own feature extraction algorithms without any knowledge of Hadoop cluster architecture and distributed computing with Spark.

Requirements 2 and 3 come from the observation that scaling and deploying algorithms created by domain experts to production is not straightforward. Since every engineer cannot be trained to these technologies (which is not their core profession and would require months if not years of experience), it is necessary to go through a long and costly industrial deployment step, often requiring to rewrite the algorithm from scratch using different programming techniques. Here, we take an alternative path: users only have to write their core algorithm as a generic function using a compatible programming language (in our case, Python or Scala), plug it into the pipeline, and it will be executed in parallel by the Spark engine, transparently for the user. The user's custom function has to follow a standard input and output schema. The user also specifies what entities are processed in parallel: for example, one might want to run a function on all flights in parallel in order to compute features for each flight. The same can be done on different levels (e.g. all flights for a given engine, a sliding window of flights, etc.), defining the parallelism. This allows an agile work flow where engineers can quickly deploy their computations. Our feature extraction API supports three types of functions:

1. **Native Spark code**, for users familiar with Spark and seeking optimal performance. Some commonly used feature extraction functions are already implemented and ready to use. For instance, for simple operations like aggregations, it is not efficient to write a dedicated custom function.
2. **Custom functions**, written in local (i.e. non-distributed) Python or Scala code, allowing to flexibly define any kind of algorithm. Common numerical, data analysis and machine learning libraries can be used. Parallelism is achieved by using Spark's `group by` and `map` or `apply` operators.
3. **Python modules.** Legacy algorithms are already packaged as Python modules, so an API allows to import them and act as an interface to use them the same way as custom functions.

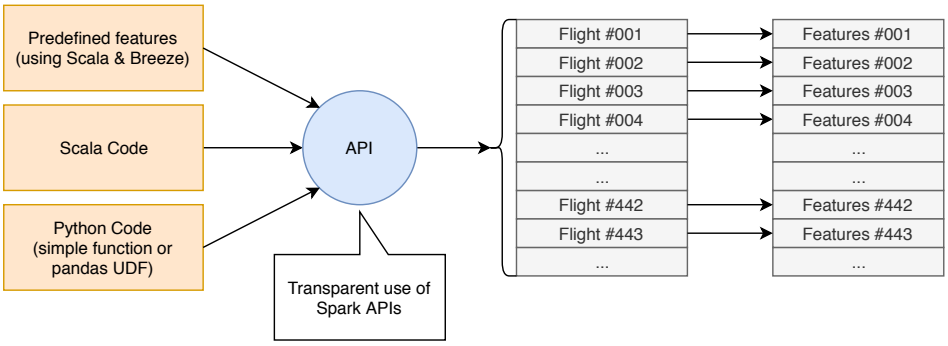


Fig. 8.6.: Diagram representing the data-parallel engine feature extraction step.

As Spark practitioners know, these custom functions are considered as black boxes and not optimized by Spark's execution engine, but it is the only solution allowing to express any processing logic (which may rely on external libraries) and without knowledge of distributed programming. However, when a custom algorithm is in its final version and ought to be used regularly in an industrial workflow, it can be rewritten in native code and optimized. Custom functions are applied by grouping on some key or sliding window, and using a map-like operation. Note that we use the most efficient Spark API and associated group/map methods whenever possible: Dataset API for Scala functions, DataFrame API with pandas vectorized UDFs or RDD API for Python functions. Figure 8.6 illustrates our generic feature extraction API for the flight-parallel case. Features are stored using a format almost identical to

Tab. 8.5.: Flight features table schema.

Column name	Type
`esn`	`string`
`flight_id`	`string`
`feature`	`string`
`time`	`timestamp`
`value`	`float`
`computed_on`	`timestamp`
`user`	`string`

the CEOD, only renaming the `param` column to `feature` and adding two columns for metadata: `computed_on`, a timestamp indicating when the feature was computed, and `author`, indicating the user (see Table 8.5). This additional information is useful to manage different versions of the features computed by different users. The time column is kept even if we no longer store time series, because engineers find it useful to associate a timestamp to the feature (e.g. for a maximum value, we also keep the timestamp at which the maximum was reached during the flight).

As already mentioned, this fixed-schema, row-wise format is practical due to its robustness and flexibility, allowing to easily append new features to an existing table. As a consequence, it allows to calculate features in several passes, by executing the feature extraction step on the output of the previous pass: this enables, for example, to compute more complex features from more basic ones (hence the loopback at the feature extraction step in Figure 8.5).

Final indicator computation

While the previously used row-wise format is flexible and robust, it is not adapted to analysis and fitting machine learning models, where the norm is to represent features as a matrix with the columns containing the features and the rows containing the individuals. Thus, we select the final features and apply a pivot operation on the table, producing the desired output format (see Table 8.6). Additional post-processing

Tab. 8.6.: Flight indicators table schema.

Column name	Type
`esn`	`string`
`flight_id`	`string`
`indicator1`	`float`
`...`	`...`
`indicatorP`	`float`

operation may be applied on the selected flight features, such as aggregations (e.g. smoothing with a moving average), or more complex transformations (e.g. normalization w.r.t. context variables).

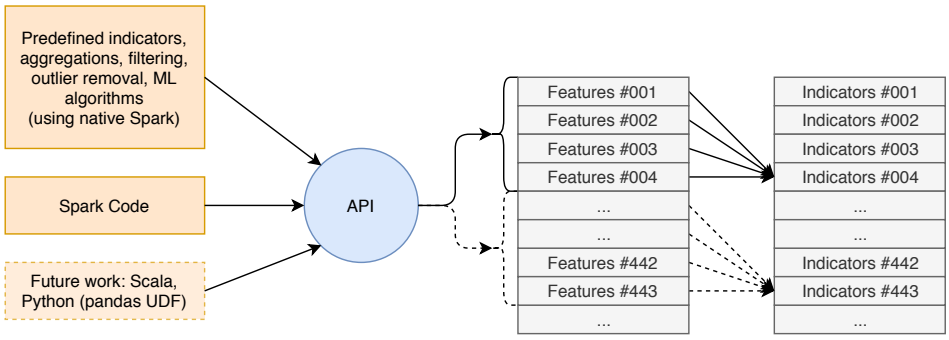


Fig. 8.7.: Diagram representing the final indicator computation step.

Learning algorithms

The next step of our pipeline is visualizing the previously calculated features. As the number of features describing an engine (or any other system) is usually larger than two, this requires a dimensionality reduction step, using unsupervised learning algorithms. In particular, we focus on self-organizing models that achieve simultaneous clustering and visualization of high-dimensional data sets. In order to test our analytics pipeline on the SOM-EHM use case, SOM is the first model we integrated. We used a Spark implementation of the SOM algorithm [Forest, 2019], part of the C4E project [LIPN, 2018]. Here again, genericity is a strong requirement, thus we also integrated the K-means algorithm, to demonstrate that any centroid-based clustering model could be plugged into the pipeline. For example, SOM could be replaced by a GMM or another self-organizing clustering model, as long as it is scalable w.r.t. the size of the data, to meet our scalability requirement. Supervised algorithms could also be used, but their output will not be adapted to the visualization tool presented in the next paragraph.

Model serialization

The resulting models are serialized to simple JSON objects containing metadata (including the name of the Hive table containing the training set and the projection of each flight) and the prototype vectors along with their cardinalities (number of training points belonging to each cluster). They are then saved into a MongoDB database on a local server.

Visualization interface

The last step of our analytics pipeline is a web interface developed specifically for displaying the results of centroid-based clustering algorithms such as K-means and SOM. The application is developed in HTML/JavaScript using the D3.js library. It can be easily accessed via a browser, which is adapted for widespread use in a company. Its main functionalities are:

- Providing a summary of the cluster centroids (also called prototype vectors): feature values, cardinalities, basic statistical properties of each cluster.
- Visualizing the self-organizing map, by displaying the previous quantities on the topology-preserving grid, as well as engine trajectories.

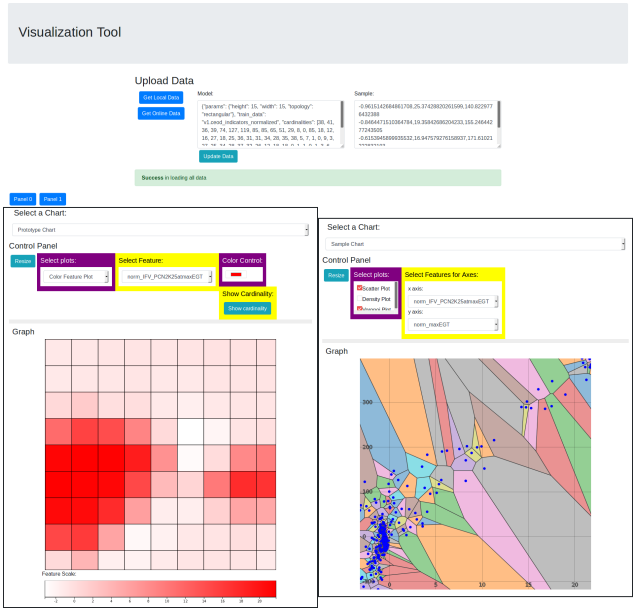


Fig. 8.8.: Screenshot of the visualization web application.

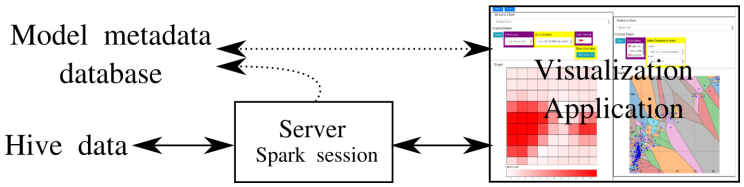


Fig. 8.9.: Architecture of the visualization application. The application server retrieves models from a metadata database, and queries Hive using JDBC.

- Projecting and visualizing a sample of the training set along with the model prototypes on a scatter plot, density plot or a Voronoï tessellation using PCA or t-SNE to reduce the dimensions to 2D.

The user interface takes as input the model output at the previous step. Along with the application, a Python server interacts with Hive through JDBC. When the users visualize the training data using PCA or t-SNE, the application calls the server that in turn queries a data sample from the training set. The name of this table is part of the model metadata. The whole architecture is summarized in Figure 8.9 and a screenshot is provided in Figure 8.8.

This analytics pipeline can be used by two target classes of users:

1. *Creators*: engineers who will adapt and create their own processing. They select the data (fleets, engines, flights, variables) and features relevant to them (i.e. relevant to the part of the engine they want to study), implement and deploy their own algorithms if needed, execute the processing and collect the

resulting features for their own utilization. They may also select and tune the learning algorithm, and visualize the results to obtain insights. In a nutshell, they may intervene at any step of the pipeline, so genericity is crucial.

2. *Consumers*: users who will not modify the pipeline, but rather consume already computed results for a specific health monitoring task, and interpret them to inform decision-making. They use the visualization application to observe results that are calculated automatically by a "frozen" version of the pipeline, where the features and algorithm are fixed and tuned for a specific use case. When new data from recent flights are ingested, the pipeline must run automatically in order to show up-to-date results, using for example a workflow manager to schedule the jobs. For these users, it is crucial that the web application is always available, shows relevant and up-to-date results and that the user interface is adapted to their use case. Specific versions of the visualization application might be needed to satisfy all needs.

The following sections will present concrete use cases realized using the analytics pipeline.

8.4 Engine state cartography using self-organized models

First, we reproduce the approach presented in [Cottrell et al., 2009, Côme et al., 2010, Côme et al., 2011, Lacaille and Côme, 2011] using the Kohonen SOM algorithm to visualize the state of a fleet of CFM56 aircraft engines based on performance indicators. In [Côme et al., 2011], a snapshot data set is used, containing 20 variables (15 context variables and 5 engine variables) measured on a fleet of 91 engines during approximately one year. Their methodology, summarized on Figure 8.10, consists in four modules:

1. Environmental condition normalization
2. Changes detection and smoothing
3. Self-Organizing Map learning
4. Search module based on edit distance to find similar engine trajectories

A trajectory is the name given to the sequence of SOM units corresponding to successive flights of an engine. The first two steps are normalization w.r.t. the context variables, smoothing and removal of abrupt changes and slow trends. Then, the residuals of the engine variables are used to fit a SOM model. It is shown that such a model is useful to monitor the state of an engine, its evolution flight after flight (trajectory), and detect potential faults and deterioration of engine parts.

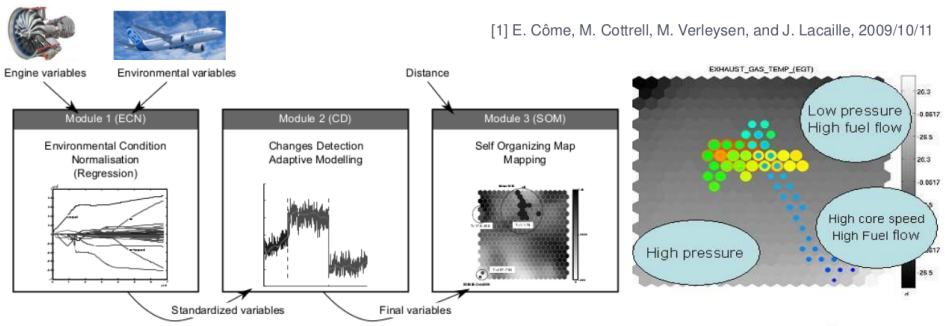


Fig. 8.10.: Engine state cartography methodology on performance indicators. [Côme et al., 2011]

We aim at reproducing this methodology with our analytics pipeline, with some major differences. First, the previous approach processed a small snapshot data set using Matlab, while our data is large-scale and stored on a cluster, and is processed using distributed software. Second, their variables are scalar values for each flight (snapshot data), whereas we process both snapshot reports and CEOD, which are time series and require preprocessing and feature extraction. Finally, we focus on genericity, customization and ease-of-use, in order to solve not just this use case but a wide range of applications. Note that we have not implemented the change detection algorithms and the search module during this PhD.

8.4.1 Context and environment normalization

The input flight parameters are selected in the preprocessing step and summarized in Table 8.7. We consider the values at take-off: these are directly available in the take-off snapshot reports, but for CEOD, we extract them in the feature extraction step.

Tab. 8.7.: Flight parameters at take-off describing engine performance state.

Parameter name	Description	Type
`temp`	Ambient air temperature	Context
`N1`	Fan speed	Context
`N2`	Core speed	Engine
`fuelflow`	Fuel flow	Engine
`EGT`	Exhaust gas temperature	Engine

In order to compare flights together, it is necessary to remove the effects of the context and environment of the flight. For example, is has no sense to compare engine state variables if the aircraft has been operated on different routes, in different

regions with all different ambient air temperatures, altitudes, speeds, utilizations of the engine, etc. Thus, the features are divided into two categories:

- *Engine state* variables, representing an aspect of the engine (here the performance state), which will be the inputs of the clustering and visualization algorithms.
- *Context* or *environment* variables, representing the operating conditions of the engine.

A simple approach to normalization is using a regression model and calculating the residuals of the state variables w.r.t. the context variables, taken at a specific flight phase. For instance, the LASSO regression [Tibshirani et al., 2001] method is used in [Cottrell et al., 2009, Côme et al., 2010]. This removes linear dependencies. The model can be expressed as follows, in its simplest form:

$$\mathbf{y}_i^j = \mu_j + \boldsymbol{\beta}_j^T \mathbf{x}_i$$

for each engine state variable $\mathbf{y}^j$ and each flight i, where μ are the intercepts and $\boldsymbol{\beta}$ are the regression coefficients for each state variable w.r.t. the context variables $\mathbf{x}_i$. In practice, additional terms must be taken into account, as for instance the engine effect representing individual variations between different engines. Context normalization, as well as smoothing, happens during the final indicator computation step. To fit the LASSO model, we used the Spark MLlib library. The residuals of the linear model are then used as normalized variables, and will be the input features of the clustering and visualization models in the next step of the pipeline. Figure 8.11 shows that the linear model has effectively removed the strong linear correlations between context and engine variables. Another normalization approach is taken in [Lacaille et al., 2014], using an online version of the mixture of probabilistic PCA (MPPCA), necessary in absence of stabilized phases (test bench experiments data).

8.5 Application to vibration monitoring

Vibration analysis is an important component of condition monitoring of rotating industrial equipment [Randall, 2004, Randall, 2011]. Vibration analysis provides knowledge on the condition of this equipment by enabling to look inside a rotating machine. Its applications include the detection of unbalance, misalignment, or flutter, due for instance to gears, rollings or bearings damage or even cracks or

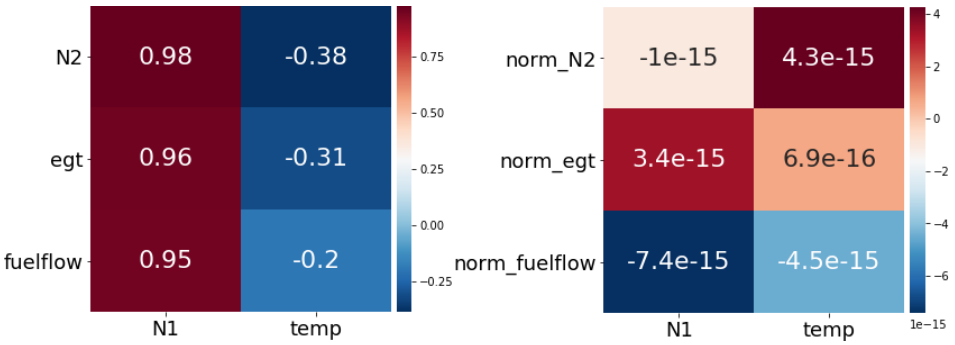


Fig. 8.11.: Pearson correlation between context and engine parameters. (Left) Raw engine parameters. (Right) Normalized parameters.

loose parts. Aircraft engines in particular are complex rotating machines where vibrations put engine parts under dynamic structural stress. In this work, we are interested in LEAP turbofan engines used in civil aircraft. We present a methodology for vibration monitoring of a fleet of civil aircraft engines using historical flight data and unsupervised learning algorithms. Every step, from ingestion to visualization, is made scalable through distributed processing on a cluster using the Spark framework, thanks to the already introduced generic analytics pipeline. Such global and large-scale approaches are yet uncommon in aerospace industry.

Our main contribution apart from the pipeline, is to extract, classify and visualize vibration signatures using interpretable self-organized clustering algorithms, yielding a visual cartography of vibration profiles. The resulting models can be used by domain experts for monitoring, anomaly detection, giving early warnings and other insights. As an example, we show it can be used to detect anomalies, compute anomaly scores, or find similar engines (which is useful to identify at-risk engines in a post-finding situation after an event has occurred). Our method has already been tested on real flight data from operating aircraft, and is intended to be part of the ground component of an EHM system [Bastard et al., 2016].

8.5.1 Related work

In this section, we will first provide a brief review of vibration analysis techniques and how they are applied to aircraft engines. Then, we present applications of unsupervised learning algorithms, and in particular self-organized maps for clustering and visualization of high-dimensional data. For a more thorough review of vibration analysis in the time and frequency domain, please refer to [Yang et al., 2003, Randall, 2011] and the various references provided in the thesis [Abdel-Sayed, 2016].

Vibration analysis on aircraft engines

A description of the turbofan engine is provided in the Introduction, see Figure 0.2 for a simplified diagram. Sensors are disposed to measure the rotation speed of each shaft (also called *regime*) and vibration amplitude. Vibration amplitude can be expressed in three different ways: displacement (unit: mm SI or $mils$), velocity (unit: mm/s SI or ips) or acceleration (unit: m/s^2 SI or g). In order to measure it on a machine, two possibilities exist. First, directly measuring displacement of moving parts, using eddy current (also known as Foucault's current) proximity sensors. This solutions is used for testing (e.g. tip-timing), but is unpractical in operating engines. Instead, the second solution is to measure the acceleration of non-moving parts (e.g. bearing or casing) using accelerometers (which are much easier to install on smaller parts), and integrate to obtain speed or displacement. In the following section, we will describe the acquisition process and properties of the sensor data that will be used in this work.

As part of aircraft engine health monitoring (EHM) [Bastard et al., 2016], vibration analysis tackles following issues: rotor unbalance (fan, compressors or turbines), rotor/stator contact [Peng et al., 2005], or defects due to wear affecting blades [Kharyton, 2009, Hazan et al., 2010], bearings [Orsagh et al., 2003] or gears [Wang et al., 2001].

Frequency analysis Vibrations signals are usually processed not in the time-domain, but in the frequency or time-frequency domain. When signals are stationary, i.e. when the engine rotation speed is constant, the Fourier transform is traditionally used to analyze the spectrum [Randall, 2011]. When rotation speed is varying, during an acceleration or deceleration, analysis takes place in the time-frequency domain and makes use of spectrograms. The works presented in [Hazan et al., 2010, Lacaille, 2013, Abdel-Sayed et al., 2015] tackle the problem of pattern recognition in high-frequency, high-bandwidth vibration data measured on aircraft engines on a test bench, as part of the production process. These data contain the complete spectral information on the engine and allow to prevent faults in new engines coming out of the production plant. Due to the high frequency of the measurements (51 kHz), the vibration data are represented as spectrograms. Traditionally, experts perform a visual analysis of the spectrograms to detect anomalous patterns. The goal is to automate this process using algorithms and numerical methods. In [Lacaille, 2013], spectrogram patches are queried against a database of reference patterns, using dimensionality reduction through non-negative matrix factorization (NMF). [Abdel-Sayed et al., 2015] propose an automatic anomaly detection procedure also based

on NMF. This line of work differs vastly from our contribution, firstly because we are interested in a fleet of operating engines, and not a test bench. The nature of our data is also different, as we have medium-frequency time-domain signals, already aggregated by the flight recorder, but measured during entire flights. Moreover, we are not interested in early detection of faults in young engines just coming out of the plant, but in the evolution of vibration signatures of operating engines, flight after flight.

Time-domain analysis In this work, we are not interested in the frequency information contained in the spectrum of the signal, but we will directly manipulate vibration amplitude signals already aggregated by the electronic flight recorder into medium-frequency time-domain signals. Amplitude is measured either by displacement, velocity or acceleration. Instead, we are interested in the vibratory response of specific parts of the engine as a function of regime, called a vibration signature [Randall, 2004]. In rotor dynamics, a vibration signature can describe intrinsic properties of parts. It is generally measured during an acceleration (monotonic increase of the regime) or a deceleration of the engine (monotonic decrease of the regime). Vibration signatures can then be represented as Campbell diagrams as a function of time or equivalently as a function of regime.

Unsupervised learning for engine data analysis

As more and more data are collected on modern aircraft, data-driven approaches and machine learning have become useful tools for condition monitoring. Supervised learning allows to build predictive models when target values or labels are available. Unsupervised learning, on the other hand, can be used for data exploration, anomaly detection, monitoring, etc. We have seen previously that dimensionality reduction allows to compress and extract information from high-dimensional data [Abdel-Sayed et al., 2015]. Another major tool is clustering, also known as unsupervised classification. Clustering is a family of unsupervised learning techniques that try to discover groups of similar elements in a data set, providing information on the structure of the underlying data distribution. The approach used in [Hazan et al., 2010] uses clustering to detect signatures of orders in spectrograms. In this work, we focus on a family of clustering algorithms called self-organizing maps (SOM). SOM algorithms enforce neighborhood constraints on the cluster centers and have the advantage of producing smooth, interpretable visualizations. High-dimensional data are clustered and projected onto a low-dimensional manifold (usually two-dimensional) with a grid topology, called a map. Each unit of the map corresponds

to a prototype vector in the original high-dimensional space, and new data points are projected on the map by finding the closest prototype vector w.r.t. euclidean distance. Originally introduced by Kohonen [Kohonen, 1982], there are many variants of SOM working with relational data defined by a distance matrix [Olteanu et al., 2013] or using unsupervised neural networks for joint representation learning [Forest et al., 2019b, Fortuin et al., 2019].

Self-organizing maps were already used for anomaly detection from the vibration spectrum of industrial systems [Harris, 1993] and from aircraft engine performance data [Bellas et al., 2014]. Aircraft engine fleet monitoring with SOM is tackled in [Cottrell et al., 2009, Côme et al., 2010, Côme et al., 2011, Forest et al., 2018]. These works focus on the performance health state of the engine and not the vibration aspects. In [Faure et al., 2017], SOM are used to classify transient flight phases.

8.5.2 Data description

This sensor data used in this work are CEOD. We begin by describing the sensors and parameters, and present the studied vibration signatures.

Sensors and acquisition

Two types of signals are necessary to compute vibration signatures: rotation speed (or regime), and vibration amplitude. On the regime side, two variables are considered:

- N1: rotation speed of the LP shaft.
- N2: rotation speed of the HP shaft.

Rotations speeds are recorded by two phonic wheels at an initial frequency of 51 kHz, before being down-sampled on-board to 66 Hz. On the vibration side, vibration peak amplitudes (displacement, speed or acceleration) are measured by two accelerometers. One of the sensors (ACC1) is located near #1 bearing, placed on the static frame as close as possible to the LP shaft, whereas the second (ACC2) is located at the turbine rear frame. Vibration is also sampled at 51 kHz and then aggregated to a lower frequency of 4 Hz. Through filtering, we obtain vibrations corresponding to N1 and N2 regimes, producing a total of four vibration variables:

- LP-ACC1 and LP-ACC2: vibration amplitude at N1 speed (in terms of displacement in $milsda$).

- HP-ACC1 and HP-ACC2: vibration amplitude at N2 speed (in terms of speed in $ipspk$).

A cross-section schema of the engine with sensor positions is displayed in Figure 8.12. The signals are measured during entire flights, from engine start to engine stop.

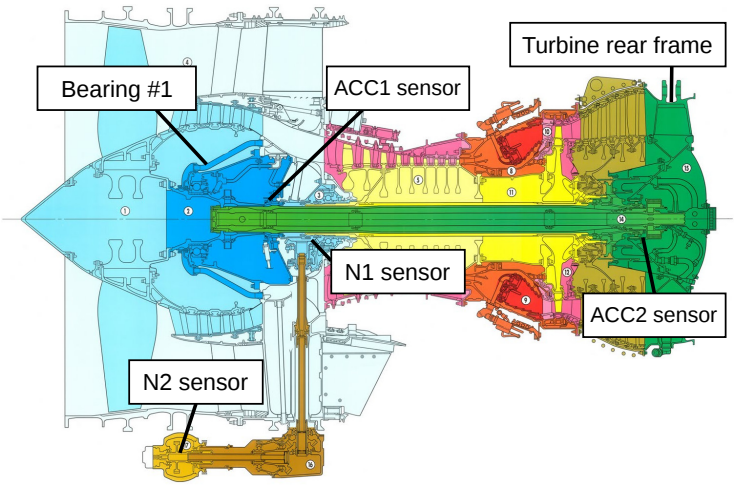


Fig. 8.12.: Engine cross-section with positions of the rotation (N1, N2) and vibration (ACC1, ACC2) sensors.

Figure 8.13 (left) shows an example of N1, LP-ACC1 and LP-ACC2 signals. The N1 rotation speed is directly controlled by the pilot pushing on the thrust lever, and corresponds to the engine thrust. It is expressed as a percentage of maximum thrust (this maximum depends on many factors and flight conditions). During a normal passenger flight, the N1 signal can be broadly divided into different phases: first, a strong acceleration during take-off and ascent, then a long stabilized phase during cruise, and finally a decrease during descent, with short peaks corresponding to maneuvers before landing. The LP-ACC1 signal follows N1 during the first part of the flight, increasing during acceleration, with a small mode at around 90% regime. However, the strongest vibrations are observed during deceleration, with several peaks showing an important mode at around 40% regime. The LP-ACC2 signal is interesting because it exhibits a very strong mode at low regimes. Signals N2, HP-ACC1 and HP-ACC2 for the same flight are displayed on Figure 8.13 (right). The behavior of N2 is similar to N1, with an acceleration until it reaches a plateau just over 100% regime, where vibration is higher, before entering the stabilized cruise regime. Both signals contain a very sharp and high peak at low regime.

The properties of the data analyzed in this work are described in Table 8.8.

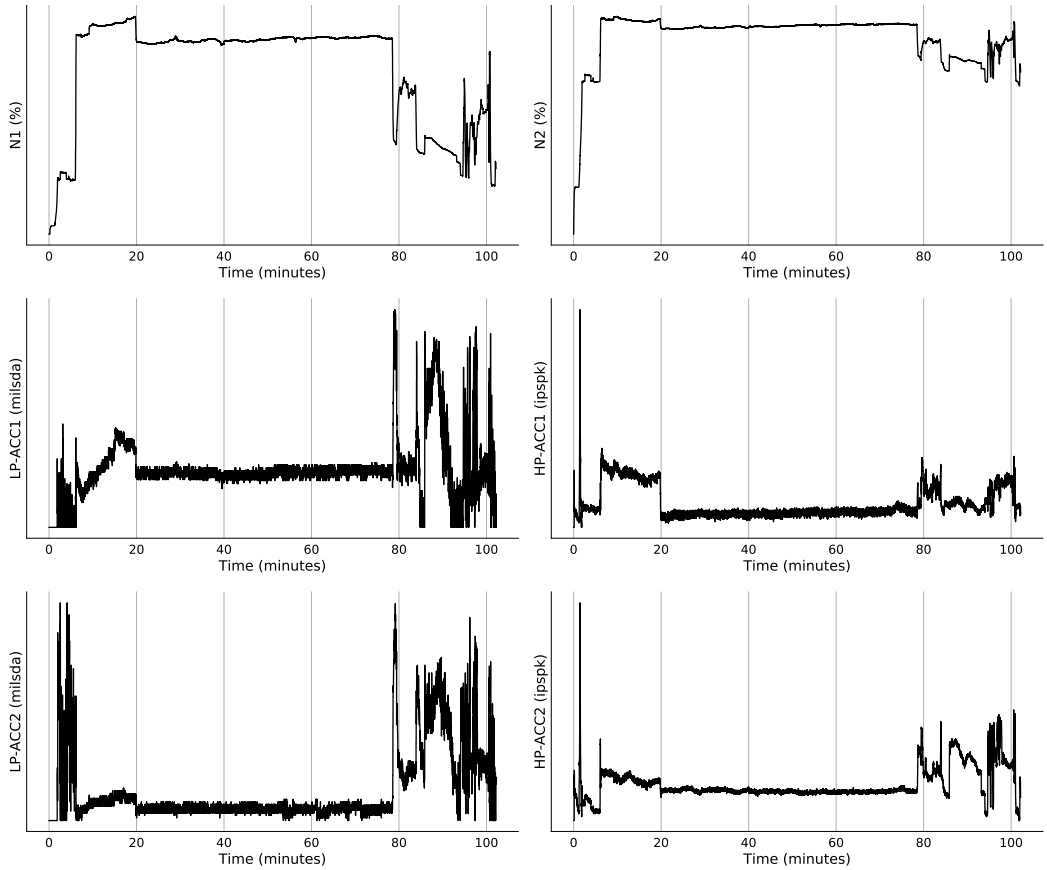


Fig. 8.13.: (Left) example of rotation speed N1 and vibration amplitude signals LP-ACC1 and LP-ACC2 during a flight. (Right) Example of rotation speed N2 and vibration amplitude signals HP-ACC1 and HP-ACC2 during a flight.

Tab. 8.8.: Data properties for the vibration monitoring use case.

Property	Approximate value
Number of engines	1000
Number of flights	1 million
Number of parameters	6
Frequency of parameters	4 Hz or 66 Hz
Total HDFS storage volume	1 TB

Vibration signatures

In order to represent the vibratory response of the engine, the raw time series are transformed into signatures that represent vibration as a function of regime. Thus, a signature can directly relate a given regime to a vibratory mode. The location and intensity of these modes are crucial to understand what happens inside the engine. Here are the four signatures studied in this work:

1. LP-ACC1 vs N1.
2. LP-ACC2 vs N1.
3. HP-ACC1 vs N2.
4. HP-ACC2 vs N2.

By observing these signatures, experts are able, for example, to detect unbalance at a specific location of the engine. As we define signatures in terms of entire flights, we are not in the standard setting of a monotonic acceleration or deceleration. Thus, a given regime is reached several times during a flight, and may correspond to different vibration amplitudes, producing a point cloud, as shown for signature 4 on Figure 8.14. To extract the modes and general shape, we cut the x-axis into bins of 5% regime, and aggregate values by taking the quantile at 75% (not the maximum because it is sensitive to outliers). It is clear that manual monitoring of

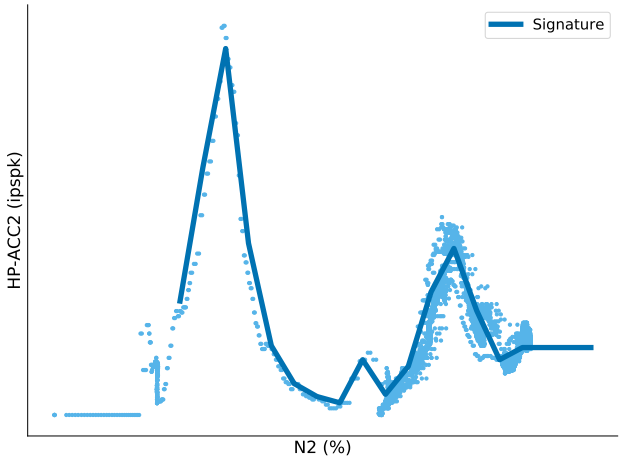


Fig. 8.14.: Vibration signature 4 (HP-ACC2 vs N2) on an example flight. Each point is a measurement during the flight (after re-sampling).

these signatures is impossible, because of their variability and the huge number of flights. The next sections present how we process data and use self-organized clustering models for efficient fleet CM.

8.5.3 Vibration signature extraction

For the large-scale computation of vibration signatures on a fleet of civil aircraft engines, we use the generic Big Data processing pipeline already introduced previously [Forest et al., 2018]. This pipeline has been designed to analyze operational flight data (here CEOD) on a Hadoop cluster and is based on the Apache Spark. It allows to deploy custom functions containing the engineers' business logic without knowledge of distributed programming. The first step in the pipeline is basic preprocessing and selection queries against the flight database. The second step is generic feature extraction, using predefined or user-provided functions to compute flight features. This is where domain knowledge is incorporated. Signature computation happens at this step. Signatures are computed by a custom function taking the parameters of

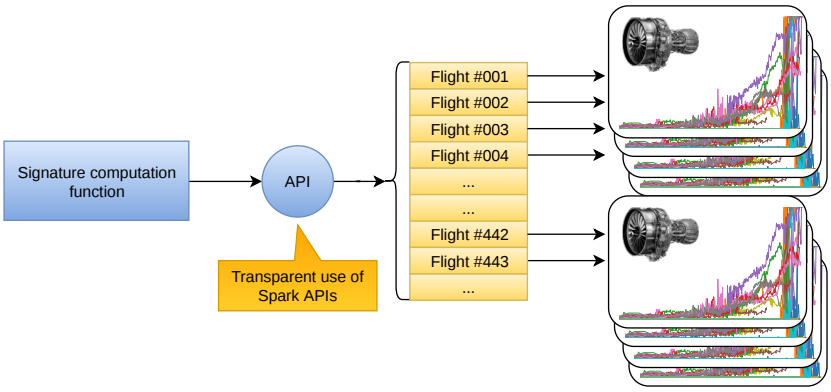


Fig. 8.15.: Data-parallel signature extraction on a collection of flights.

a single flight as input, and outputs the resulting signature. The code is written in Python with standard numerical and data analysis libraries. This function has five parameters: the names of both input signals (x- and y-axis), the period and range of the x-axis (e.g. the regime range), and the type of operation used to aggregate points in each bin of the x-axis (e.g. average, max, quantiles, etc.). Parameters are set in the configuration file, allowing to extract various signatures using the same generic code. The feature extraction module applies this function on flights in parallel across the cluster, as illustrated in Figure 8.15.

In this work, we use a period of 5% regime bins in the [25%, 100%] range, thus each signature can be viewed as a 15-dimensional vector, or a one-dimensional curve of length 15.

8.5.4 Clustering and visualization with self-organized models

Self-Organizing Maps

The algorithm takes as input the set of vibration signatures $\mathbb{S} = \{\mathbf{s}_i\}_{1 \leq i \leq N}, \mathbf{s}_i \in \mathbb{R}^D$, with $D = 15$, and outputs a square map composed of $K = 8 \times 8$ units. Each unit is associated to a prototype signature $\{\mathbf{m}_k\}_{1 \leq k \leq K} \in \mathbb{R}^D$. A new flight is projected onto the map by finding its closest prototype signature w.r.t. Euclidean distance. We call the corresponding map unit *best-matching signature* (BMS):

$$\text{BMS} := \underset{k}{\text{argmin}} ||\mathbf{s}_i - \mathbf{m}_k||_2^2$$

Before feeding into SOM, the data set is z-normalized to zero mean and unit variance to give each point of the signature an equal weight in euclidean distances. The resulting map for signature 4 (HP-ACC2 vs N2) is displayed Figure 8.16 and will be investigated further in the next paragraph. We use the distributed Spark ML SOM

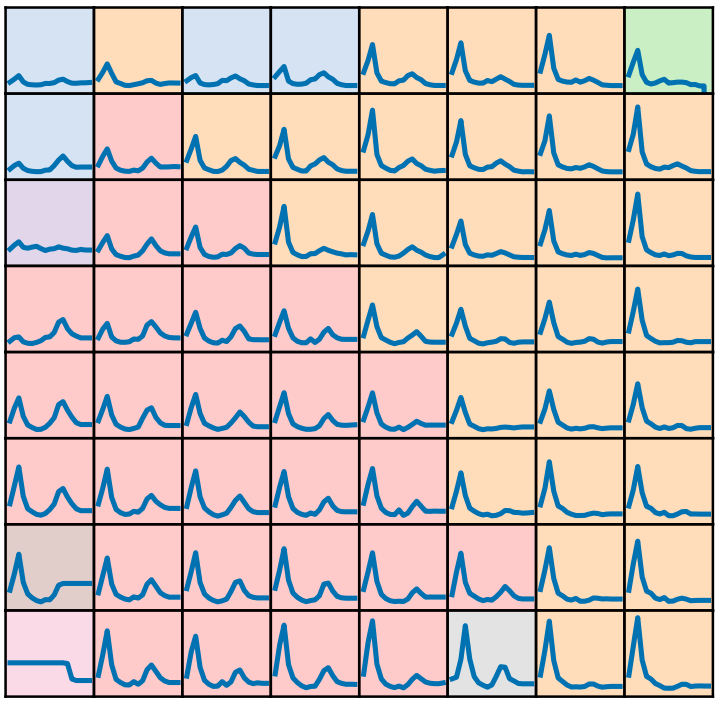


Fig. 8.16.: SOM map of signature 4 (HP-ACC2 vs N2). Each cell represents a vibration signature prototype. The background colors are higher-level profiles obtained by Ward hierarchical clustering (here with 8 clusters).

implementation of batch SOM [Forest, 2019]. This allows to leverage the production cluster to train SOM models on huge data sets of several million flights.

One year of historical flight data for 1000 engines, representing approximately 1 million flights and 1 TB of raw signal data, has been processed. After training a SOM for each signature, the resulting models are saved.

Vibration signatures exhibit several modes at particular regimes, visible on the visualizations provided in Figure 8.16 and appendix Figure E.1. The variability in locations and amplitudes of the modes translates into smooth transitions on the map. Experts in engine dynamics are able to identify these modes and link them, for example, to unbalance at a specific part of the engine. Moreover, certain vibratory behaviors are normal, such as vibrations of the whole aircraft structure, or temporary unbalance due to thermal conditions. On the other side, certain behaviors are due to wear and must be monitored closely.

In order to classify map cells into higher-level vibration profiles, we perform hierarchical clustering (HC) on the prototype signatures. This classification is materialized by the cell's background color. These profiles may correspond to very well-balanced engines (see the very low-amplitude signatures on Figure 8.16 and appendix), rotor unbalance at fan, compressor or turbine, but also issues not related to the engine at all (e.g. flat signals, such as the bottom-left cell on Figure 8.16, are due to a sensor switched off or some issue during data decoding or ingestion). As a result from this analysis, each map region has been interpreted and labeled by experts.

For EHM of operating aircraft, new flights are projected onto their best-matching signature (BMS). The (euclidean) distance between a flight and its BMS is a proxy for an anomaly score: a large distance means that a flight is dissimilar to previously observed behaviors, thus it needs to be investigated carefully. Flights that are projected onto abnormal regions of the map raise alerts and can be immediately investigated by engineers. The sequence of projections of a single engine, flight after flight, is called a *trajectory*. Because a signature describes intrinsic properties of an engine, it should not change drastically from one flight to another. Thus, a trajectory should stay within the same region or higher-level profile. A sudden jump, or a progressive trend towards a different region, can be a warning for abnormal wear. However, changes in vibration profiles may also be due to maintenance operations or a folding of the SOM map [Kiviluoto, 1996]. An engine trajectory is represented on Figure 8.17. The BMS of a flight is marked by a black circle, where the radius is proportional to the number of flights where the engine stayed on the same cell. The lower part of Figure 8.17 represents the sequence of higher-level profiles found by HC. Clearly, the vast majority of flights have similar vibration profiles. Out of

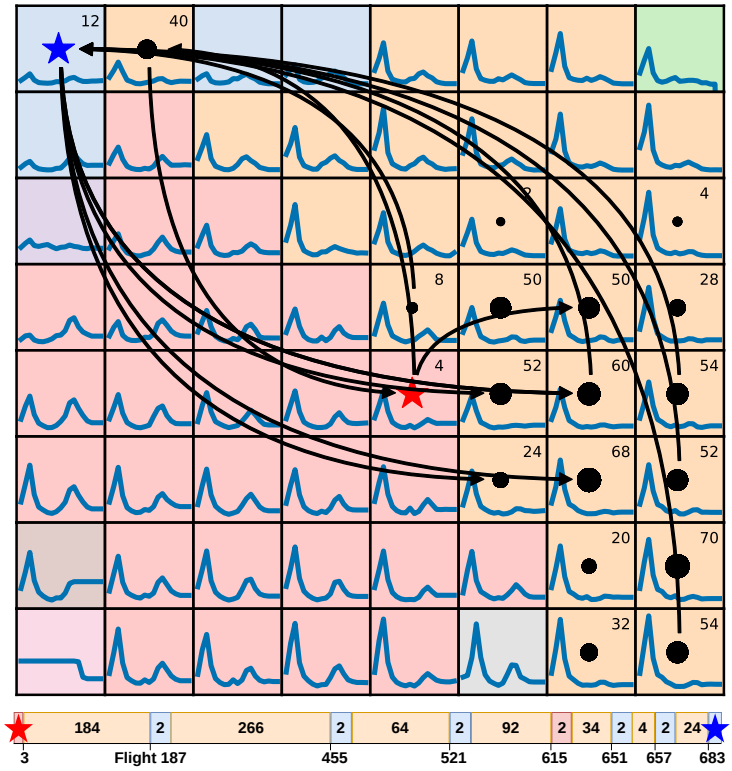


Fig. 8.17.: Trajectory of a single engine on the SOM of signature 4, for a total of 684 flights. Circles correspond to flight projections. The first and last flights are marked by red and blue stars. The radius of a circle is proportional to the number of flights the engine stayed on the same vibration profile (this number is also printed within each cell). Sudden jumps between non-neighboring cells (marked by arrows) indicate abrupt changes in vibration profiles, which may correspond to operational events or a SOM folding. This engine mostly stays within the orange region, as shown by the sequence of transitions between higher-level profiles (bottom diagram).

684 flights, only 16 fall outside the orange area, and most transitions occur within the higher-level profiles. For sake of readability, only transitions between non-neighboring cells were represented by arrows on the map. The fact that a signature is an individual property of engines is supported by a heatmap visualization of BMS counts, i.e. by representing the number of flights projected on each cell for different engines (Figure 8.18). Finally, in a post-finding situation, after an event has occurred, we can find *similar* engines that share the same vibration patterns or have similar temporal evolutions (e.g. with an edit distance on trajectories [Côme et al., 2011]). However, a map is only a snapshot of past flights. Periodically, models must be re-trained with up-to-date flight data, to account for new trends and the aging of the fleet. The complete methodology, summarized visually on Figure 8.19, can be analyzed under the OSA-CBM framework for EHM (described in

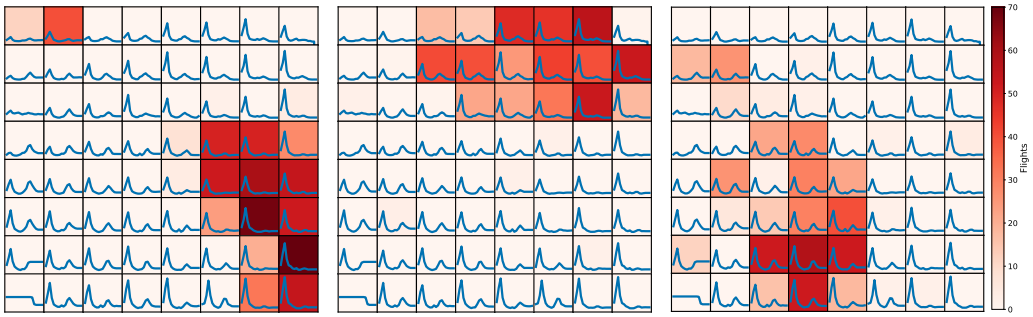


Fig. 8.18.: Heatmaps of projection counts on SOM map of vibration signature 4, for three different engines. Each individual engine has its vibration signatures concentrated in a single region, because a signature is an intrinsic property of engines.

[Bastard et al., 2016]): (1) signature computation corresponds to Data Acquisition & Manipulation; (2) State Detection assigns flights to vibration profiles as well as distances to the map; (3) Health Assessment consists in the classification of the profiles and anomaly detection; (4) analysis, prediction and search of similar engine trajectories is part of Prognostics Assessment and finally (5) Advisory Generation encompasses visualization and alerts generation.

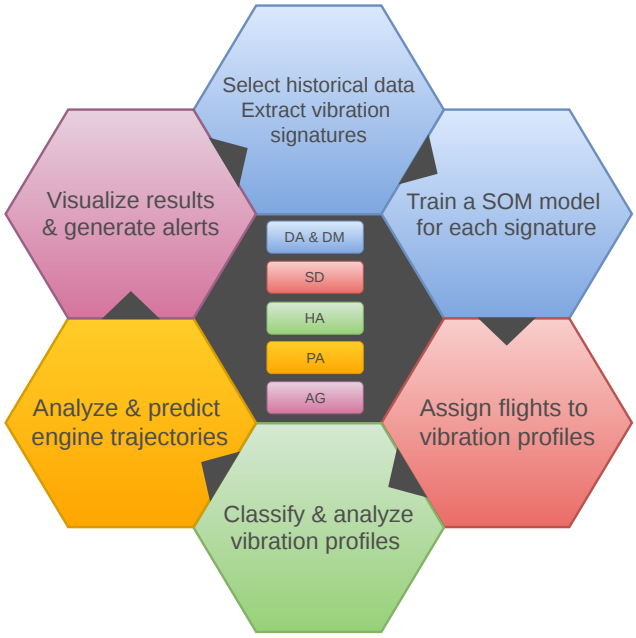


Fig. 8.19.: Vibration monitoring methodology under the OSA-CBM standard. Colors represent the steps of the OSA-CBM standard: Data Acquisition & Manipulation (DA), State Detection (SD), Health Assessment (HA), Prognostics Assessment (PA), and Advisory Generation (AG).

8.6 Conclusion

This chapter has begun by introducing health monitoring of aircraft engines, and the various data sets collected from the flight recorder and processed on ground. We have to deal with multivariate, high-frequency, time series data, posing a problem of format to be processed efficiently.

The first contribution in this chapter was to design and build an end-to-end pipeline for large-scale analytics of continuous operational aircraft engine data collected on aircraft engines, and stored on a Hadoop cluster. Using this pipeline, we implemented a minimal but fully operational version of a health monitoring application. Each step of the processing scales to large data sets thanks to Map-Reduce data-parallel processing using the Apache Spark framework. Moreover, each step is kept generic and customizable in order to adapt to a wide range of use cases. The pipeline integrates an unsupervised learning algorithm (in the present use case, a self-organizing map) and a visualization web application to analyze the results. We also enable domain engineers with no Big Data skills to deploy their algorithms at scale, using a simple API for custom functions.

The second contribution presents a methodology for large-scale vibration monitoring on thousands of operating civil aircraft engines, also based on the SOM algorithm and a database of LEAP flight recorder data. First, vibration signatures are extracted using the previous pipeline. These signatures are then classified and visualized using SOM, yielding a cartography of vibration profiles. As part of a CM strategy, these profiles are useful for experts who can quickly gain insights about the vibratory state of a fleet, and detect unbalance or other abnormal behaviors.

Conclusion and perspectives

Throughout this PhD thesis, we have tackled important problems in the field of unsupervised machine learning. First, the problem of representation in clustering. We have explored deep clustering approaches, using neural networks to learn high-level representations to improve clustering. As we are particularly interested in topographic map algorithms for their interpretable neighborhood preservation and visualization capabilities, we proposed a Deep Embedded Self-Organizing Map (DESOM) algorithm to jointly learn representations through an autoencoder and the SOM prototype vectors in latent space. Secondly, we addressed the problem of model selection in clustering, which is challenging because of the ill-posed objective of clustering and the lack of ground-truth labels. We adopted the clustering stability principle, due to its elegant formulation and assumptions in contrast with many clustering validation methods. We proposed a novel principle of within-cluster stability and the Stadion criterion, along with a concrete implementation, showing competitive results on a very large benchmark study we have conducted. An extension to time series clustering was proposed, based on data invariances.

Meanwhile, we have not put aside the challenge of solving real-world industry problems. Modern aircraft generate growing amounts of sensor data that can be leveraged for health monitoring and predictive maintenance, in order to improve safety, availability and reduce costs for the engine manufacturer and/or airline operators. These massive data sets are processed efficiently using Big Data tools, such as distributed storage and processing. We adopted the Apache Spark framework. To this end, it was necessary to develop specific tools to scale health monitoring applications. We developed a flexible processing pipeline to extract expert indicators from raw flight data and use algorithms to cluster and visualize results. This allowed to reproduce the engine fleet cartography methodology, based on the SOM algorithm, that we also implemented in Spark ML. In particular, we applied it to vibration monitoring of the LEAP engine and obtained encouraging results in collaboration with Safran A.E. engineering teams.

Here is a brief summary of our contributions:

1. Proposing to jointly learn representations and the SOM prototype vectors in the DESOM model (3 publications and 1 journal under submission).

2. Proposing a novel principle and method to evaluate clustering algorithms using cluster stability analysis, with an extension to time series (1 publication and 1 under submission).
3. Developing algorithmic and software tools enabling large-scale health monitoring methodologies on flight recorder data stored on a cluster. Conducting an industrial application to vibration monitoring in collaboration with Safran A.E. engineering teams (2 publications and 1 patent).
4. Developing several open-source software including DESOM, SOM performance evaluation, stability analysis, and a distributed Spark ML SOM implementation.

Future work

The various paths followed throughout this thesis open up perspectives for future work. First, not every aspect has been tackled in the work on DESOM. We conducted experiments using a fully-connected or convolutional AE network, but an extension to sequences with a recurrent AE is possible. In addition, the model could be extended to the variational or adversarial frameworks (VAE or GAN), which could improve the quality of learned representations and provide us with a generative model. We also did not try different SOM neighborhood functions and radius decays, assuming that they are only related to the SOM learning and should not fundamentally change the properties of DESOM. However, this cannot be excluded due to the interaction between SOM loss and reconstruction loss. These perspectives are left as future work.

Future developments for `SOMperf` include the computation of per-unit metrics, a SOM visualization module, as well as distance functions between self-organized models. In addition, other more recent SOM quality metrics could be implemented, such as the map embedding accuracy [Hamel, 2016].

Concerning our work on cluster stability analysis, many improvements work directions must be considered. An important concern is to speed up the Stadion computation. We believe that measuring the data density at cluster boundaries is sufficient to assess stability, therefore it could be estimated only by computing the distances of data points to the boundaries when these are known (e.g. in center-based methods). Then, our implementation of within-cluster stability is not functional for all classes of algorithms, for instance density-based clustering, although we already obtained promising results for DBSCAN. Furthermore, our work is empirical and still lacks theoretical guarantees, that could be established in future works.

In our application to time series clustering, there is a need for efficient algorithms or algorithms with an extension operator, able to assign new points without re-training. Future work will focus on reducing the computational burden, and exploring more complex time series data sets. Insights on data perturbations can be gained from the vast literature on invariant transformations and data augmentation. Finally, we are convinced that interesting links could be made between clustering stability and adversarial attacks [Fawaz et al., 2019].

On the industrial applications side, a next step is to automate the processing pipeline using workflow management software, for instance Oozie, Airflow or Azkaban, that allow to author workflows that schedule jobs sequentially, in parallel or based on conditions and triggers (e.g. availability of new data). Another important future development is data and model versioning. As the database is constantly growing with incoming flights, it is essential to keep track of what data was used to fit a model. Thus, the application should handle multiple models and compare them in order to analyze their evolution. Generally, efficient data and model versioning capability should be integrated into the pipeline. Then, in collaboration with teams of domain experts, we would like to extend the engine fleet cartography methodology to other engine families, different use cases, and for other engine aspects and subsystems (for instance the oil system, valves, etc.). Vibration signatures from the radial drive shaft (RDS) were not tackled in out work but their behavior is of great interest.

Based on the engine fleet cartography, we would like to develop a new application called *Weekly Report*. The principle would be to automatically monitor engine trajectories and generate reports and alerts describing abnormal trends or behaviors. Interpretable clustering and visualization for decision-making is crucial in aerospace industry but also in other fields such as healthcare. In healthcare, probabilistic models have been used to make temporal predictions on the state of a patient using *disease trajectories* with Gaussian processes [Schulam and Arora, 2016], sometimes in combination with recurrent neural networks [Lim and van der Schaar, 2018]. [Fortuin et al., 2019] apply their SOM-VAE model to time series from the intensive care unit. These ideas are considered for our use case in future work, in order to model and predict the future trajectory of an engine. In particular, the remaining number of flights before reaching a *risky* state (for example a state where an event has occurred in the past) could be estimated, a kind of remaining useful life. Lastly, instead of computing a one-dimensional curve from the point cloud (shown in Figure 8.14), we could extract multidimensional vectors from the distribution (in particular, the standard deviation or envelope). This might require to use deeper SOM architectures for dimensionality reduction.

Appendix to chapter 3 — DESOM visualizations

A

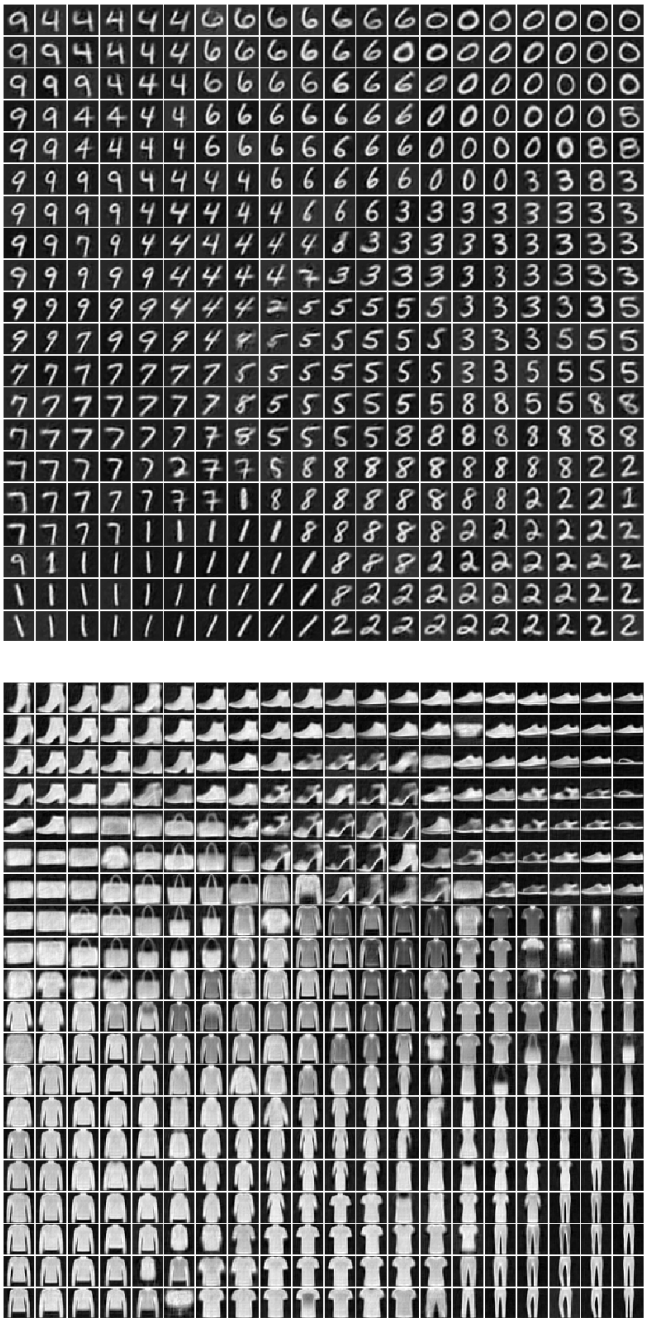


Fig. A.1.: 20-by-20 DESOM maps of MNIST and Fashion-MNIST data sets.

Appendix to chapter 4 — SOMperf usage examples

B

In the `SOMperf` module, metric functions usually take as arguments a matrix containing the SOM code vectors, the data matrix and/or the distance matrix, to avoid recomputing all pairwise distances. In addition, they need a distance function on the map. A code example for computing topographic error with a rectangular map topology is given below:

```
from somperf.metrics import topographic_error
from somperf.utils import rectangular_topology_dist
map_size = (10, 10)
dist_fun = rectangular_topology_dist(map_size)
x = ... # data matrix
som = ... # SOM code vectors matrix
te = topographic_error(dist_fun, som, x)
```

Figure B.1 shows three SOM maps trained on a square uniform distribution, with three different levels of topographic organization. Topographic, combined and Kruskal-Shepard error increase as the map becomes more disordered, and the C measure decreases.

Figure B.2 reproduces the experiments described in [Villmann et al., 1994]: the topographic function vanishes when its argument k approaches the length of a 1-dimensional map. On Figure B.3, we reproduced the example from [Kaski and Lagus, 1996], with three different solutions of a 1-dimensional SOM trained on a 2D rectangular stripe. Solution (1) approximates well the data (low quantization error), but is too *complex* and less smooth (high topographic error), whereas solution (3) has a bias (high QE) but is a simple straight line (low TE). Only combined error is able to indicate the best compromise, i.e. solution (2). A large number of additional experiments are available online as a notebook[1], including examples from the original papers.

[1] `https://github.com/FlorentF9/SOMperf/blob/master/tests/SOMperf-Tests.ipynb`

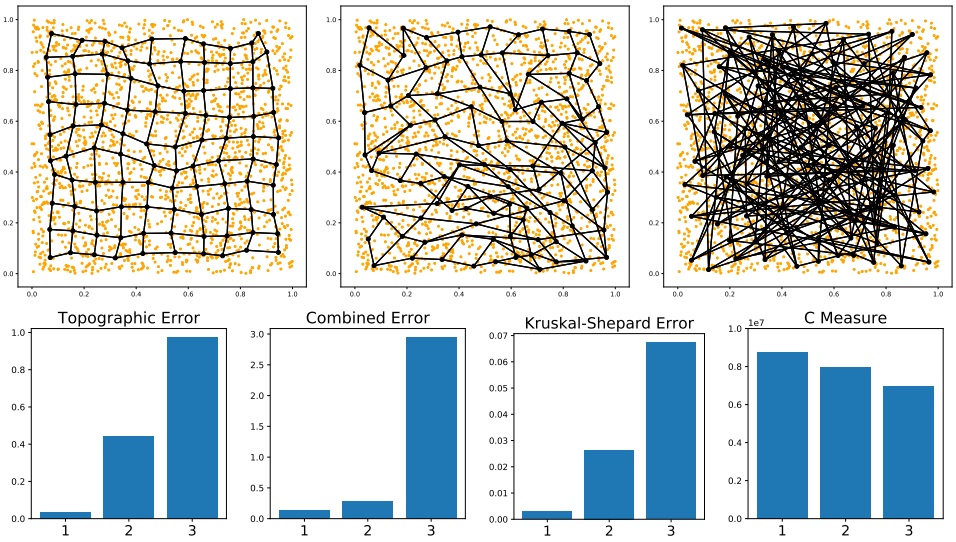


Fig. B.1.: SOM maps representing a uniform distribution with three different levels of topographic organization. Topographic, combined and Kruskal-Shepard errors and C measure behave as expected as a function of organization.

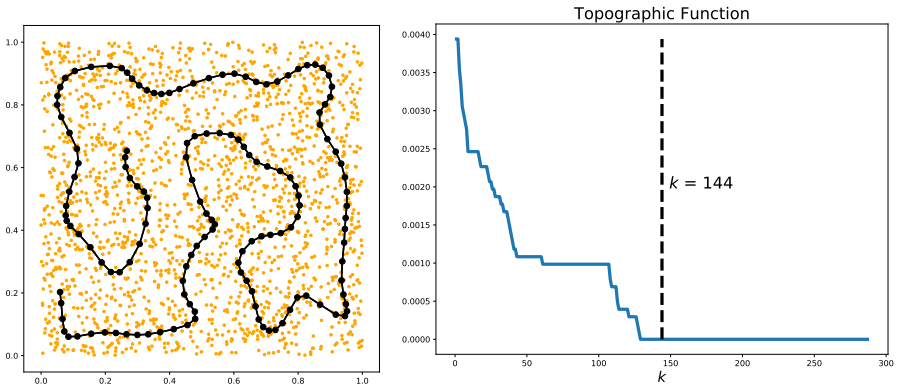


Fig. B.2.: Reproduction of the example in [Villmann et al., 1994]. The topographic function vanishes when k approaches the length of the 1D map.

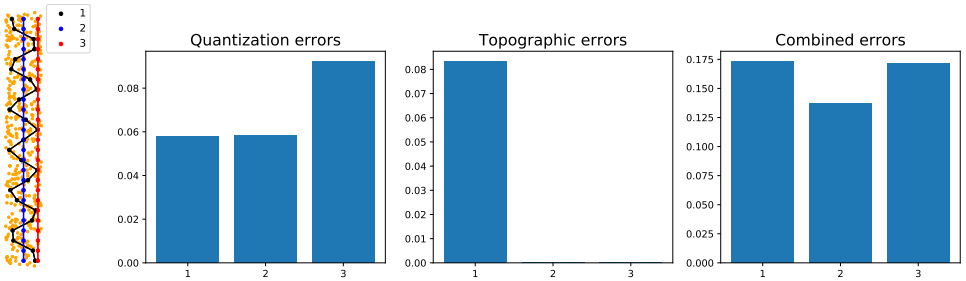


Fig. B.3.: Reproduction of the example in [Kaski and Lagus, 1996] with a stripe distribution and 1D SOM maps with three different configurations. Among quantization error, topographic error and combined error, only the latter is able to indicate the best solution, i.e. the straight line.

C

Appendix to chapter 5 — Detailed benchmark results and experimental settings

C.1 Results analysis

A large benchmark on 73 artificial and 7 real data sets compares the solutions selected by Stadion with the true number of clusters $K^\star$, a selection of internal indices including the Gap statistic for K-means, the BIC for GMM and two previous stability methods. As summarized in Table 5.3, Stadion achieves the best results overall, and other internal indices such as Wemmert-Gancarski and Silhouette also perform well. In particular, with GMM, it outperforms the BIC on both the artificial and real benchmark, although BIC is a standard choice and a strong baseline in model-based clustering. Note that Stadion and BIC were evaluated for $K \geq 1$, while all other methods used $K \geq 2$. This slightly favors the latter methods, because many data sets have $K^\star = 2$, thus on data sets where the algorithm fails to recover structure, Stadion will select $K = 1$ while most indices output $K = 2$ as a default. In order to assess the statistical significance of those results and determine which methods really are different, we adopt the same methodology as previously and carry out a Friedman test followed by Wilcoxon-Holms post-hoc analysis on the ARI performances of each method, across our benchmark of 73 artificial data sets and for the three algorithms considered in this work.

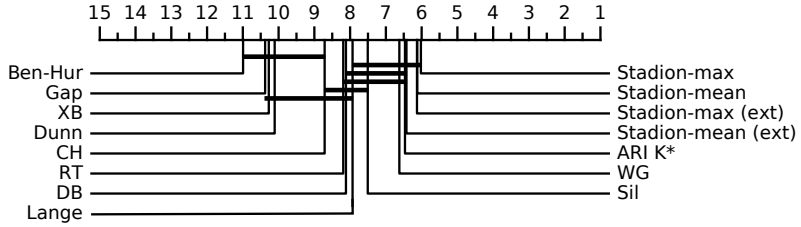


Fig. C.1.: Critical difference diagram after Wilcoxon-Holms test on ARI performance across 73 artificial data sets comparing several clustering validation methods for the K-means algorithm.

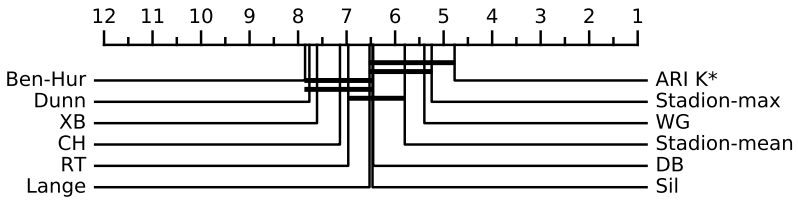


Fig. C.2.: Critical difference diagram after Wilcoxon-Holms test on ARI performance across 73 artificial data sets comparing several clustering validation methods for the Ward algorithm.

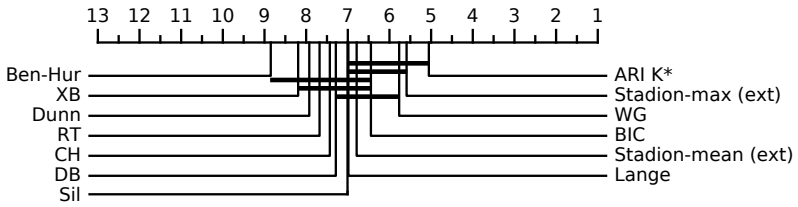


Fig. C.3.: Critical difference diagram after Wilcoxon-Holms test on ARI performance across 73 artificial data sets comparing several clustering validation methods for the GMM algorithm.

As shown on the CD diagrams in Figures (C.1, C.2, C.3), Stadion seems to outperform other indices. However, the signed-rank Wilcoxon-Holms test on ARI performance did not assess significant difference between the group of best-performing methods. The main reason is that ARI performance is evaluated on partitions that are fixed for any given K. In contrast to supervised learning (using accuracy), here the methods are often attributed the same ARI scores, since they often find the same number of clusters. In other words, methods mostly succeed and fail on the same data sets. This implies a large number of ties in terms of ARI score. Under these conditions, the experimental data is not sufficient to reach any conclusion regarding the statistical superiority of our method.

That said, some conclusions can still be drawn. Stadion (mean/max, extended or standard) performs significantly better than Ben-Hur, Gap, Xie-Beni, Dunn, Calinski-Harabasz, Ray-Turi and Davies-Bouldin indices for the K-means algorithm. Similarly for GMM, Stadion-max performs significantly better than the same groups of indices. Finally for Ward, Stadion-max performs significantly better than Ben-Hur, Xie-Beni, Dunn, Calinski-Harabasz and Ray-Turi. Overall, Stadion-mean had slightly inferior performance than Stadion-max, but this was not significant.

Note that performance is evaluated using the external index ARI (w.r.t. the ground-truth partition), while Stadion also uses s = ARI as its similarity measure to estimate stability. It would be reasonable to expect this situation to introduce some kind of bias. However, results are not biased in favor of Stadion, as shown by Tables (C.1,

Tab. C.1.: Performance rankings of clustering validation methods, evaluated with 16 external indices w.r.t. the ground-truth partitions, over 73 data sets for K-means. Rankings are mostly unchanged.

		Stadion		Stadion											
	$\text{ARI}_{K^\star}$	max	mean	max (ext)	mean (ext)	WG	Sil	Lange	DB	RT	CH	Dunn	XB	Gap	Ben-Hur
RI	6.43	6.15	6.26	**5.97**	6.35	6.66	7.71	8.51	8.09	8.36	8.40	9.82	9.95	10.36	10.99
ARI_1	6.47	**6.02**	6.12	6.13	6.42	6.62	7.51	7.93	8.12	8.19	8.71	10.11	10.27	10.38	10.99
ARI_2	6.47	**6.02**	6.12	6.13	6.42	6.62	7.51	7.93	8.12	8.19	8.71	10.11	10.27	10.38	10.99
FM	6.42	**5.96**	6.32	6.02	6.42	6.54	7.42	8.21	8.06	8.12	8.73	10.10	10.27	10.23	11.18
JACC	6.39	**5.96**	6.36	6.03	6.45	6.62	7.34	8.16	7.98	8.16	8.77	10.14	10.32	10.23	11.09
AMI	6.03	6.11	**6.07**	6.10	6.54	6.62	7.47	8.18	7.73	8.49	8.52	10.41	10.58	10.19	10.95
MI	7.68	7.62	7.55	6.94	6.84	8.23	9.09	9.90	8.27	8.60	**6.34**	7.66	6.80	9.86	8.61
NID	6.03	6.11	**6.07**	6.10	6.54	6.62	7.47	8.18	7.73	8.49	8.52	10.41	10.58	10.19	10.95
ID	5.88	**6.07**	6.04	6.20	6.68	6.60	7.25	7.93	7.80	8.58	8.84	10.52	10.70	10.07	10.86
NVI	6.32	6.29	6.25	**6.03**	6.35	6.58	7.51	8.32	7.91	8.27	8.42	10.18	10.34	10.32	10.91
VI	6.29	**6.04**	6.22	6.24	6.57	6.51	7.07	7.85	7.87	8.33	8.90	10.38	10.64	10.00	11.08
NMI_1	6.03	6.11	**6.07**	6.10	6.54	6.62	7.47	8.18	7.73	8.49	8.52	10.41	10.58	10.19	10.95
NMI_2	9.50	8.26	8.49	7.86	7.60	8.59	8.64	8.38	8.51	7.90	7.21	6.42	**5.89**	8.93	7.82
NMI_3	6.42	6.38	6.21	**6.08**	6.23	6.57	7.59	8.44	7.90	8.30	8.34	10.04	10.21	10.47	10.84
NMI_4	6.32	6.29	6.25	**6.03**	6.35	6.58	7.51	8.32	7.91	8.27	8.42	10.18	10.34	10.32	10.91
NMI_5	6.32	6.29	6.25	**6.03**	6.35	6.58	7.51	8.32	7.91	8.27	8.42	10.18	10.34	10.32	10.91

Tab. C.2.: Performance rankings of clustering validation methods, evaluated with 16 external indices w.r.t. the ground-truth partitions, over 73 data sets for Ward. Rankings are mostly unchanged.

	$\text{ARI}_{K^\star}$	Stadion-max	Stadion-mean	WG	Sil	Lange	DB	RT	CH	Dunn	XB	Ben-Hur
RI	4.92	**5.42**	5.93	5.60	6.58	6.49	6.60	7.10	6.87	7.53	7.34	7.63
ARI_1	4.77	**5.25**	5.80	5.40	6.47	6.53	6.45	6.97	7.14	7.77	7.61	7.86
ARI_2	4.77	**5.25**	5.80	5.40	6.47	6.53	6.45	6.97	7.14	7.77	7.61	7.86
FM	4.89	**5.15**	5.74	5.47	6.41	6.25	6.57	7.12	7.24	7.89	7.68	7.59
JACC	4.84	**5.15**	5.73	5.52	6.40	6.18	6.49	7.15	7.29	7.95	7.73	7.58
AMI	4.42	**5.45**	5.95	5.66	6.40	6.13	6.46	7.12	7.24	7.89	7.69	7.59
MI	5.88	6.44	6.39	6.40	7.44	7.45	6.47	6.61	**5.21**	6.38	5.62	7.73
NID	4.42	**5.45**	5.95	5.66	6.40	6.13	6.46	7.12	7.24	7.89	7.69	7.59
ID	4.40	**5.36**	5.85	5.61	6.36	6.02	6.54	7.13	7.35	7.98	7.78	7.63
NVI	4.95	**5.45**	5.92	5.51	6.48	6.25	6.45	6.99	7.09	7.74	7.54	7.64
VI	4.78	**5.30**	5.70	5.62	6.19	6.09	6.42	7.15	7.38	8.03	7.83	7.52
NMI_1	4.42	**5.45**	5.95	5.66	6.40	6.13	6.46	7.12	7.24	7.89	7.69	7.59
NMI_2	7.85	7.50	7.30	6.68	6.34	6.77	6.48	5.92	6.21	**5.21**	5.23	6.51
NMI_3	4.96	**5.44**	5.95	5.53	6.51	6.27	6.46	7.01	7.05	7.68	7.50	7.64
NMI_4	4.95	**5.45**	5.92	5.51	6.48	6.25	6.45	6.99	7.09	7.74	7.54	7.64
NMI_5	4.95	**5.45**	5.92	5.51	6.48	6.25	6.45	6.99	7.09	7.74	7.54	7.64

C.2, C.3). Indeed, the ranking almost never changed while using different external indices to evaluate performance of Stadion, and keeping $s = \text{ARI}$.

C.2 Complete results on real-world and artificial data sets

Due to the large number of tables, we do not include them in this manuscript. Please refer to the paper's appendix [Mourer et al., 2020].

Tab. C.3.: Performance rankings of clustering validation methods, evaluated with 16 external indices w.r.t. the ground-truth partitions, over 73 data sets for GMM. Rankings are mostly unchanged.

	$\text{ARI}_{K^\star}$	Stadion-max	Stadion-mean	BIC	WG	Sil	Lange	DB	RT	CH	Dunn	XB	Ben-Hur
RI	5.04	**5.55**	6.59	6.73	5.82	7.13	7.35	7.40	7.77	7.18	7.72	7.75	8.98
ARI_1	5.05	**5.59**	6.79	6.45	5.77	7.01	6.99	7.29	7.68	7.43	7.92	8.19	8.85
ARI_2	5.05	**5.59**	6.79	6.45	5.77	7.01	6.99	7.29	7.68	7.43	7.92	8.19	8.85
FM	5.15	**5.62**	6.85	6.84	5.64	6.87	6.92	7.18	7.47	7.29	7.90	8.10	9.17
JACC	5.15	**5.62**	6.85	6.84	5.64	6.87	6.92	7.18	7.47	7.31	7.90	8.09	9.17
AMI	4.89	**5.50**	6.78	6.53	5.75	6.86	7.21	7.48	7.68	7.40	7.82	8.14	8.95
MI	6.25	6.34	6.42	7.08	7.11	8.03	8.41	6.64	7.45	**5.94**	6.95	6.52	7.88
NID	4.89	**5.50**	6.67	6.37	5.75	6.86	7.21	7.62	7.82	7.40	7.82	8.28	8.82
ID	4.88	**5.57**	6.77	6.52	5.64	6.73	7.06	7.63	7.70	7.39	7.82	8.36	8.94
NVI	5.12	**5.58**	6.70	6.51	5.72	6.76	7.16	7.34	7.82	7.38	7.86	8.24	8.82
VI	5.29	**5.68**	6.92	6.64	5.54	6.62	6.94	7.15	7.60	7.42	7.87	8.31	9.02
NMI_1	4.89	**5.50**	6.78	6.53	5.75	6.86	7.21	7.48	7.68	7.40	7.82	8.14	8.95
NMI_2	7.86	7.14	6.88	8.08	7.23	6.65	7.76	6.47	6.95	6.87	**5.58**	5.77	7.76
NMI_3	5.12	**5.58**	6.92	6.78	5.72	6.76	7.13	7.14	7.66	7.32	7.80	8.05	9.02
NMI_4	5.12	**5.58**	6.81	6.67	5.72	6.76	7.16	7.21	7.68	7.38	7.86	8.10	8.95
NMI_5	5.12	**5.58**	6.81	6.67	5.72	6.76	7.16	7.21	7.68	7.38	7.86	8.10	8.95

C.3 Algorithm initialization

- K-means: in both Stadion versions (standard and extended), effective initialization is achieved using K-means++ [Arthur and Vassilvitskii, 2007] and keeping the best of 35 runs (w.r.t. the cost function).
- Ward linkage: agglomerative clustering is deterministic, no initialization strategy is needed.
- Gaussian Mixture Model: Two initializations were considered. The first one uses K-means to initialize the EM algorithm. The second one uses the approach discussed in [Scrucca and Raftery, 2015]. The main idea is to project the data through a suitable transformation which enhances separation of clusters. Among the investigated transformations, the scaled SVD transformation performed best in their experiments and so we used it. Then it applies model-based agglomerative hierarchical clustering at the initialization step. Once the hierarchy is obtained, the EM algorithm is run on the original data. However, there was no noticeable difference between the two initializations.

C.4 Preprocessing

Every data set was scaled to zero mean and unit variance on each dimension, which is essential for the algorithms to work but also for the additive noise perturbation. For the real data sets, additional preprocessing was necessary to ensure class labels truly represent the cluster structure.

Crabs The Crabs data set was decomposed using a PCA and keeping the principal components two and three, as described in [Bouveyron and Brunet-Saumard, 2014]. The standard scaling was applied both before and after the PCA.

Old Faithful & Iris No preprocessing apart from scaling.

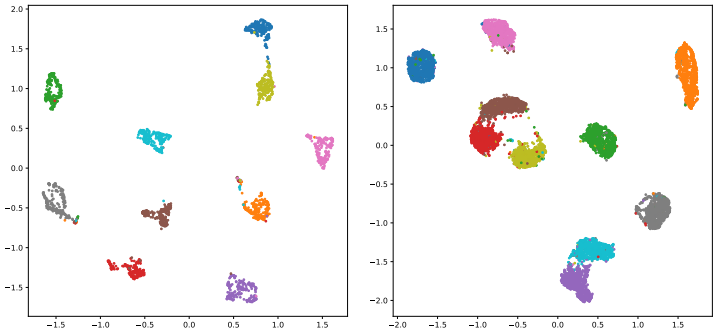
MFDS, MNIST, USPS On these high-dimensional data sets (respectively 649, 784 and 256 dimensions), we extracted a two-dimensional representation following a two-step process, as introduced in [McConville et al., 2021]. First, a fully-connected symmetric autoencoder with $[500, 2000, 2000, 10]$ units is trained to compress data into a 10-dimensional latent feature space. Then, the UMAP [McInnes et al., 2018] dimensionality reduction algorithm is applied on the latent features to obtain 2D representations. Finally, standard scaling is applied.

Wine This data set was reduced to two dimensions with UMAP, and then scaled. All UMAP-reduced data sets displayed in Figure C.4.

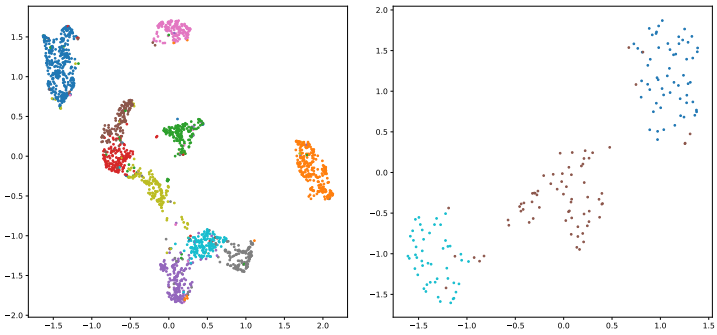
C.5 List of data sets

A complete list of the 80 artificial and real data sets used in this paper is provided in the paper's appendix [Mourer et al., 2020], indicating the number of samples, dimension, ground-truth number of clusters and references. It is not included here to avoid lengthening this manuscript. Some data sets are original and have been created for this work, in order to provide more challenging model selection tasks. All data set are available in the companion repository of this paper, `skstab`[1], to ensure reproducibility.

[1] `https://github.com/FlorentF9/skstab`



(a) MFDS_UMAP (b) MNIST_UMAP



(c) USPS_UMAP (d) wine_UMAP

Fig. C.4.: Data sets MFDS, MNIST, USPS and wine after UMAP dimensionality reduction.

Appendix to chapter 7 — Hadoop cluster components

D

D.1 Architecture of a cluster

D.1.1 HDFS

The Hadoop Distributed File System (HDFS) is at the heart of the Hadoop platform and is designed around the four aforementioned design considerations. HDFS is a virtual filesystem built on top of native Linux filesystems such as ext3 or ext4. It is virtual in the sense that it appears as a single entity, but in reality, the underlying data is spread across many different locations, disks and native filesystems. As in GFS, HDFS ingests large files by cutting them into fixed-sized blocks and distributes the blocks on the cluster, to enable data-parallel processing. HDFS is designed for large files taking several blocks, and not for many small files. The default block size is 128MB but is configurable. For example, a 300MB file will be split into 3 blocks of size 128MB, 128MB and 44MB respectively. For comparison, block sizes on Linux filesystems (ext3, ext4) and Windows filesystems (NTFS, FAT) are much smaller: 4KB. For fault-tolerance, blocks are replicated on different nodes of the cluster. This replication also increases the chances of data co-locality, reducing shuffle reads and writes. By default, each block is replicated 3 times. HDFS was originally developed to support only batch computation and was designed as a *Write Once, Read Many (WORM)* filesystem. This means that data blocks are immutable: they cannot be modified, and need to be re-written entirely. This property guarantees that the data cannot be modified during processing. Another property of HDFS is being *schema-less*. It means that no index or metadata are stored when data is written to HDFS, unlike with relational databases. Thus, it is not optimized for fast data retrieval, and the philosophy is always to take the computations to the data. Furthermore, it is tailored for large volumes of *unstructured* data (like text, images, video, etc.). However, many (if not most) of the use cases for aircraft data analytics involve *structured* data (with a more or less flexible schema). We will see that other tools are adapted for manipulating structured data and retrieving it faster using partitioning.

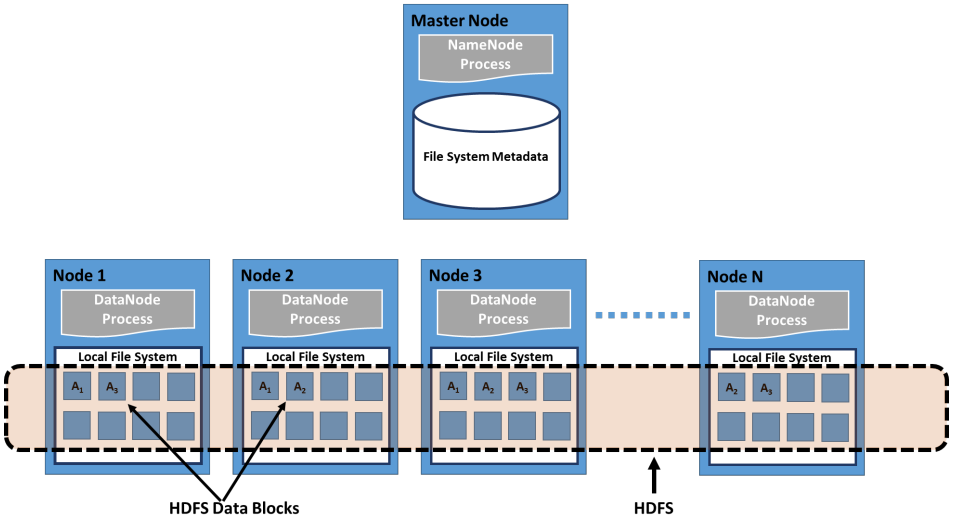


Fig. D.1.: HDFS architecture.

HDFS is composed of one *NameNode* and *DataNodes*. The NameNode is the process running on the master node whose most important function is managing the filesystem metadata catalog. The filesystem metadata resides in-memory and contains the directories, file attributes (such as user and group access control lists) and the locations of the blocks composing all the files in the filesystem. It is the only link between the virtual files and their physical block representation. The metadata is also journalized on disk. Before Hadoop v2, the NameNode was a single point of failure, because in case of a NameNode crash, the whole cluster would be unavailable. Since Hadoop v2, it is possible to achieve high availability by setting up one NameNode in Active state (receiving client operations) and another NameNode in Standby state (slave), and running JournalNode daemons on several nodes of the cluster to log modifications and use the Quorum Journal Manager to share the logs between the Active and Standby NameNodes[1]. The data are stored and managed on the DataNodes. Each node runs a DataNode process, and hosts the HDFS data blocks on their local filesystem. DataNodes are also responsible for replication, checking block integrity (computing checksums regularly) and sending reports to the NameNode in order to maintain the filesystem metadata catalog. The HDFS architecture is illustrated in Figure D.1.

[1] `https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html`

D.1.2 YARN

The first generation Hadoop v1 only supported MapReduce v1 as a data processing framework. The MapReduce v1 scheduling was built to schedule MapReduce workloads and optimize data locality, but could not schedule other types of jobs like iterative computations, real-time or graph processing, or SQL queries. It was not adapted for applications other than MapReduce like Spark, Tez, Impala, etc. Hadoop v2 introduced a new cluster manager called YARN (Yet Another Resource Negotiator), able to schedule a wide variety of jobs. YARN is composed of three main components:

- ResourceManager
- ApplicationMaster
- NodeManagers

The ResourceManager is the YARN master node that arbitrates the available resources of the cluster. It is aware of the cluster's topology (nodes, racks) and allocates *containers* by given them a certain amount of CPU cores and memory on a worker node. It is only responsible for scheduling and not monitoring of the tasks. The NodeManagers are processes that receive instructions from the ResourceManager, manage the resources on each node and send reports to the ResourceManager. Finally, the ApplicationMaster is the main container of an application, running on one of the NodeManagers. The ApplicationMaster is responsible for negotiating resources with the ResourceManager and monitoring the application. It is important to note that several applications, submitted by different users, can run on YARN at the same time; in this case, there will be one ApplicationMaster per application. The scheduling policy determines how the resources are shared between applications. For example, with the FIFO (First-In, First-Out) strategy, every job in the queue waits for the completion of the previous job before starting; with the FAIR policy, the resources are shared and the jobs run concurrently, allowing short jobs to finish before long jobs, even if they were submitted later. Here are the steps required to run an application on YARN:

1. The client first submits the application to the YARN ResourceManager.
2. The ResourceManager designates an ApplicationMaster on a NodeManager.
3. The ApplicationMaster negotiates resources for containers on the NodeManagers.

See Figure D.2 for an illustration of the YARN architecture.

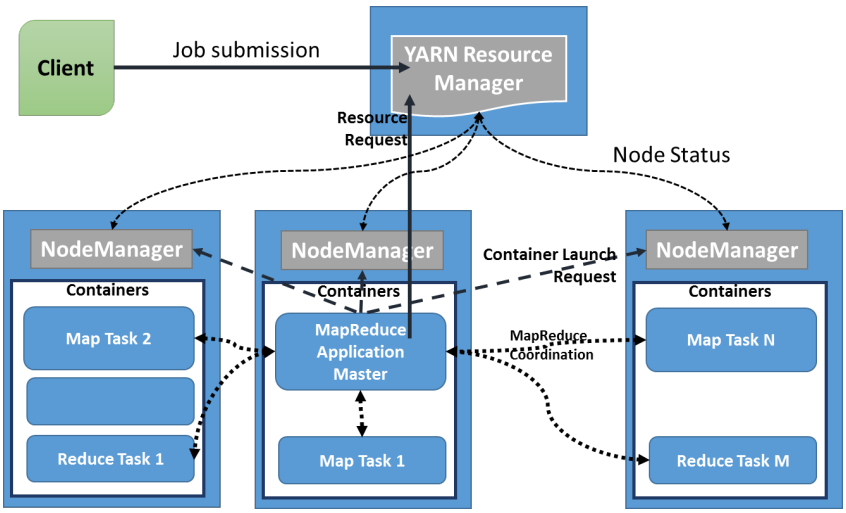


Fig. D.2.: YARN architecture.

D.1.3 Hive

Apache Hive [Apache Hive, 2010] is Hadoop's data warehousing system. Hive allows to read, write and manage large data sets on HDFS. As we will see, it is somewhat similar to a relational database management system. Its main functionality is querying data sets using the HiveQL language, which is very close to SQL. HiveQL allows to translate SQL queries into Map-Reduce jobs. Thus, Hive is an essential tool for managing structured, relational data sets on a cluster. Hive is composed of several components, presented thereafter.

D.1.4 Hive components

First, Hive is composed of a *metastore*, where it stores the metadata, i.e. information on the tables, schema, user permissions, and location of the files in HDFS. The metastore is simply a SQL database. By default, it uses Derby[2], a lightweight open-source Java RDBMS (Relational DataBase Management System). The derby metastore allows only one connection at a time, which quickly becomes a constraint when several driver programs need access to the metastore (a second user trying to access the metastore results in an error "`Failed to start database 'metastore_db'`"). The derby metastore is also called the *embedded metastore*, as it runs in the same JVM as the Hive driver. The solution is to use a standalone, fully-fledged SQL database that allows concurrent connections, like MySQL, PostgreSQL or Oracle (in fact, any

[2] `https://db.apache.org/derby/`

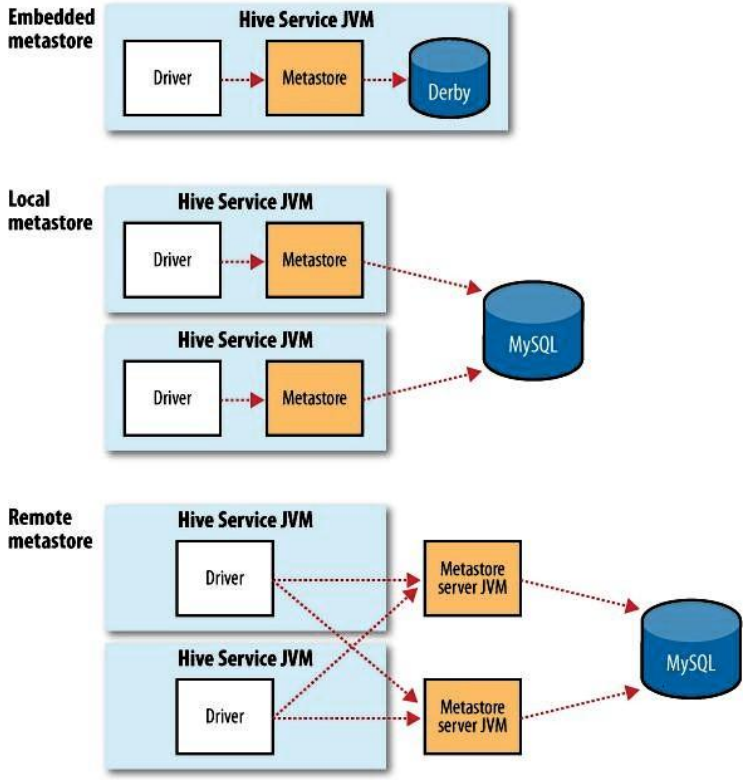


Fig. D.3.: Hive metastore architectures.

SQL database with JDBC connectivity). The metastore can be run in two ways: *local* or *remote*. The difference of a remote metastore is that a dedicated service handles the metastore and queries the metastore database. Details are illustrated in Figure D.3. Hive uses a Thrift API which allows to query the metastore using JDBC (Java DataBase Connectivity) or ODBC (Open DataBase Connectivity). Apache Thrift[3] is a RPC (Remote Procedure Call) framework, originally developed at Facebook and an Apache open-source project, that allows to create client/server services that connect applications in various languages and platforms using interfaces. As we already mentioned, Hive is designed towards distributed processing of large structured data sets. The metadata are stored in the metastore, and the underlying data are stored in files in HDFS (which is inherently schema-less). Supported file formats are TextFile, SequenceFile, RCFile, ORC Files, Parquet, Avro, and custom formats[4]. It is also possible to store data in non-native table formats and connect to data stores (HBase, Druid and Accumulo). The choice of the data format is of utmost importance and directly affects query performance, compression, and the

[3] `https://thrift.apache.org/`, `https://en.wikipedia.org/wiki/Apache_Thrift`
[4] `https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-StorageFormats`

Tab. D.1.: Hive data types.

Data type	Description

`TinyInt`
`SmallInt`
`Int/Integer`
`BigInt`	

`Float`
`Double`
`Decimal/Numeric`	

`Boolean` `Binary`	True or False Byte Array

`String`
`Char`
`Varchar`
	
	Fixed length string
	Variable length string

Complex types	`Timestamp`, `Date`, `Interval`, `Array`, `Map`, `Struct`, `Union`, ...

flexibility of schema evolution. Because it is such a crucial issue, this discussion is postponed to the next section, which concentrates on the storage of aircraft engine data.

The next component of Hive is the SerDe interface (Serialization and Deserialization). SerDe is the link between table data formats (in the metastore) and HDFS file formats. This library is used for IO (In/Out) between tables and HDFS, allowing to read and write the Hive data formats (listed in Table D.1), as well as custom data formats (needs to implement a custom SerDe interface). The most powerful component is the HiveQL engine, which allows to transform SQL-like queries (expressed in HiveQL language, similar to SQL) into MapReduce jobs. As a consequence, data analysts, business analysts and engineers familiar with SQL can query and manipulate Hive data without needing to write MapReduce jobs themselves. HiveQL automatically parses, plans and optimizes the queries, and executes them on MapReduce. Finally, Hive can be extended with User-Defined Functions (UDF) and User-Defined Aggregate Functions (UDAF), as well as with custom MapReduce code. An overview of the Hive components is shown on Figure D.4.

D.1.5 Tables and partitioning

Hive data are stored in *tables*. This paragraph will explain how these tables are organized hierarchically, presents the two types of tables, and explains the concept

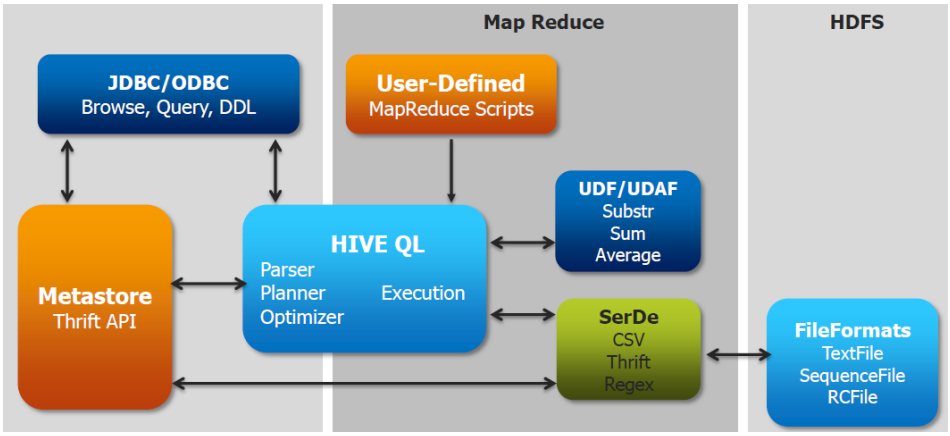


Fig. D.4.: Hive components architecture.

of *partitioning* that can be used to control the way data are read from and written to HDFS and optimize performance.

At the highest hierarchical level, a table is part of *database*, also called a *namespace* (because two tables cannot have identical names inside a database). In the filesystem, databases correspond to directories, and tables are stored in subdirectories. In HDFS, all Hive databases are located in `/user/hive/warehouse/`. There exists a *default* database that does not correspond to any directory. The syntax to access a table inside a database in HiveQL is `database.table`. Hive tables can be of two different kinds: *managed* or *external*. As its name suggests, managed tables are completely "managed" by Hive, i.e. Hive controls the data of the table and stores it inside directories as explained previously. If one drops a managed table, its data are deleted from disk. Managed tables are used in cases when only Hive uses the data and should completely manage its life cycle. Managed tables are also called *internal* tables, opposed to *external* tables. External tables, on the contrary, are used when the data is already used by another program and is stored in a custom location. Only their metadata are managed by Hive, not the underlying data. Dropping an external table does not delete the data in the table (only the metadata about the table). In order to create an external table, one must specify the location where the data can be read.

A common query on a table is to select a subset of the data set by filtering on the values of a column (using a `WHERE` clause). Without using partitioning, the entire table needs to be scanned. Partitioning allows to cut a table into chunks (called partitions) and to only read the necessary partitions. Suppose that a table "`engine_table`" in a database "`engine_db`" contains measurements concerning three different aircraft engines. A column "`engine_id`" contains the engine identification numbers `1`, `2` and

3. Partitioning this table on the basis of the values of the "`engine_id`" column results in three partitions:

```
/user/hive/warehouse/engine_db/engine_table/engine_id=1/<file>
/user/hive/warehouse/engine_db/engine_table/engine_id=2/<file>
/user/hive/warehouse/engine_db/engine_table/engine_id=3/<file>
```

This is achieved by adding `PARTITIONED BY(engine_id INT)` to the Hive query. If a user queries the data for a specific engine, only the corresponding partition will be read, and performance will benefit from it greatly.

Hive partitioning is hierarchical, meaning that it is possible to partition hierarchically on the combination of values of several columns. For example, suppose that our table has columns `year`, `month` and `day` for the date of the measurement. The obtained partitions are:

```
/.../engine_db/engine_table/engine_id=1/year=2018/month=1/day=1/<file>
/.../engine_db/engine_table/engine_id=1/year=2018/month=1/day=2/<file>
...
/.../engine_db/engine_table/engine_id=2/year=2018/month=1/day=1/<file>
...
/.../engine_db/engine_table/engine_id=3/year=2018/month=1/day=1/<file>
...
```

This kind of temporal partitioning is very useful to retrieve the data of a specific range of dates. For example, the query `SELECT * FROM engine_db.engine_table WHERE engine_id=2 and year=2018 and month=1` will only scan the partitions in the corresponding subdirectories. The Hive syntax for creating a table with this partitioning is `PARTITIONED BY( engine_id INT, year INT, month TINYINT, day TINYINT)`. Note that the order of the columns determines the hierarchy of partitions, and highly depends on the data and use cases. More generally, choosing the partitioning is crucial, as it can make vary the performance of queries by up to several orders of magnitude. There is no universally best partitioning, but there are key principles:

- The partition sizes should not be too large, because on each query the whole table will need to be scanned.
- There should not be too many small partitions, as the overhead of MapReduce will take over the benefits of partitioning, scanning the directories will be slow and the metadata will take a lot of space in the metastore.

In other words, the optimal partitioning should give a reasonable number of partitions with a reasonable size. The meaning of "reasonable" depends on the configuration of the cluster and benchmarks can help to find the right trade-off. As a rule-of-the-thumb, a good order of magnitude for the partition size is the HDFS block size (128MB by default). The number of partitions is determined by the cardinality of the column used for partitioning. If we often need to retrieve all measurements for a given engine, the current partitioning is a reasonable choice. If the per-day partitioning is too finegrained (i.e. results in very small partitions), it can be removed to only partition on the engine, year and month, or only on the engine. Hive leaves a choice between two types of partitioning:

- *Static* partitioning: the user provides explicitly the list of partitions to write the data into.
- *Dynamic* partitioning: the user provides a column and Hive creates as many partitions as unique values in the column provided.

The previous example assumed dynamic partitioning.

D.1.6 Bucketing

Another way to cut the data into smaller chunks is *bucketing*. A bucket is equivalent to a partition. Bucketing enables to create a fixed number of data chunks with roughly equal sizes. The user provides the number of *buckets* and a column, and the buckets are defined by applying a hashing function on the provided column, using the formula $\text{bucketID} = \text{hash(column)} \bmod N_{\text{buckets}}$ (akin to Spark's HashPartitioner, presented in the next section 7.3). This feature is useful when partition sizes vary a lot (e.g. some very large partitions and some very small partitions), or if no column is well suited for partitioning.

D.1.7 Views

Hive also allows to create *views*, a traditional concept in RDBMS, but with some particularities. Hive views are logical copies of tables. They can be used to create "simplified" version of tables, for example to reduce the number of columns, and simplifying the subsequent queries. Note that the view's schema is frozen at its creation, thus, changing the schema of the underlying table or dropping it will not affect the view (subsequent queries on this view may then fail).

D.2 Hadoop storage formats

This section aims at presenting the different data formats that can be used to store files in HDFS. First, a good format for data warehousing in Hadoop must be *splittable*, because HDFS files are split into blocks. A file is splittable if we can start reading it at any line in the file; this excludes formats like XML or JSON documents (JSON records, where each line is a JSON object, are splittable). Then, data storage formats can be compared on several criteria:

- Read performances (with a difference between partial reads and full reads)
- Write performances
- Compression
- Schema flexibility

Each of these properties directly depends on the way the data is organized by the format and how it is compressed (compression will be the focus of the last paragraph of this section). The choice of the data format is driven by the use case:

1. What tools and engines will process the data?
2. What are the performance requirements?
3. Is storage volume an issue?
4. Will the schema evolve?

Choosing a storage format is making a trade-off between the four aforementioned criteria. The choice may also depend on the environment (Hadoop distributions may support or recommend different formats).

If storage volume is an issue (question 3), the files can be compressed using a compression library. Compression algorithms use patterns int the data to encode files efficiently. Several libraries may be used in Hadoop, such as ZLIB, Snappy, LZO, LZ4 or ZSTD. For this, the file format must support *block compression*, meaning that the blocks can be independently compressed. If it is not the case, the compressed file will be non-splittable. Depending on the format and the compression library and settings, a lot of storage space can be saved using compression, but of course this comes at the cost of read and write performances, as data will need to be decompressed and compressed each time (question 2).

Examples of file formats are plain text files or CSV files, JSON records, Sequence files, and more advanced formats such as Avro, RCFile, ORC and Parquet.

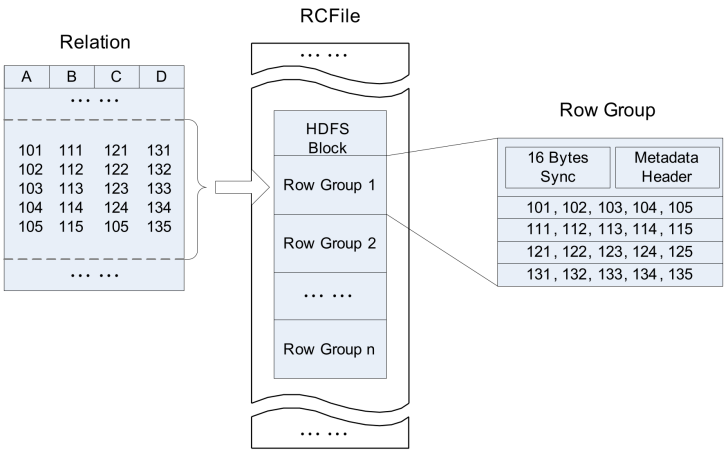


Fig. D.5.: RCFile data layout in a HDFS block.

D.2.1 RCFile

RCFile (Record Columnar File) [He et al., 2011a] was the standard binary format for storing structured tabular data in Hive. RCFile's data layout combines advantages of row-store and column-store: first, the file is partitioned horizontally, creating *row groups* of a specific size (the default row group size is 4MB); then, within each row group, the data is partitioned vertically and stored by column. This layout is illustrated in figure D.5. *Row-store* formats are efficient for reading entire rows, because the values belonging to the same row are stored adjacently on disk. *Column-store* formats are more efficient for reading a subset of columns since this time, the data belonging to the same column are stored adjacently. RCFile combines

As a row-store, RCFile guarantees that data belonging to the same row is located on the same node; as a column-store, it can leverage column-compression, as the data of a column within a row group is stored adjacently. To go into more detail, is composed of:

1. A file header, containing the magic bytes, a byte for the format version number, a Boolean for indicating if compression is used, the compression codec, SequenceFile metadata and a sync marker denoting the end of the header.
2. Each row group is composed of:
   a) A sync marker.
   b) A row group metadata header.
   c) The data record which is a key-value pair, where the key contains metrics (record length, key length, number of rows, length of all columns...) and the value contains the values of the row-group organized column-wise.

D.2.2 ORC

ORC (Optimized Row Columnar file) is the successor of RCFile as the standard Hive format for storing tabular data, and offers several improvements over its predecessor (see specification of ORC v1[5]). First, ORC uses the type information of the table to improve compression thanks to type-specific compression techniques, resulting in smaller files compared to RCFile (in RCFile where values as stored as key-value pairs, each column is simply treated as a binary blob). On top of this, the generic compression libraries zlib and Snappy can be applied. ORC is also optimized for reading only a subset of columns, includes index data with the minimum and maximum value of each column every 10000 rows, and is able to skip sets of rows using Hive filters. An ORC file is composed of:

1. Row groups called *stripes*, with a default size of 250MB to enable efficient reads from HDFS. Each stripe is itself composed of:
   a) Index data containing column-level metrics (minimum and maximum values) and an index with the row positions within each column for row-skipping, by default every 10000 rows.
   b) Row data.
   c) A stripe footer.
2. A *file footer* that contains the list of stripes, number of rows per stripe, column types, and file-level column metrics (count, minimum, maximum and sum).
3. A *postscript* indicating the file format version, type of compression used and footer length.

This structure is illustrated in Figure D.6. An ORC file is read backwards by first parsing the postscript (that contains the footer length) and then decompressing and parsing the footer. ORC metadata are stored using Protocol Buffers, a serialization format that allows to add or remove fields (schema evolution).

D.2.3 Parquet

Parquet[6] is a columnar storage format for the Hadoop ecosystem compatible with almost all processing engines. It is similar to RCFile on various aspects: the data is partitioned horizontally into row groups containing *column chunks* whose data are stored contiguously for sequential reading. Parquet recommend large row group sizes of 512MB to 1GB, with optimally one row group per HDFS block. Additionally,

[5] `https://orc.apache.org/specification/ORCv1/`
[6] `https://parquet.apache.org/`

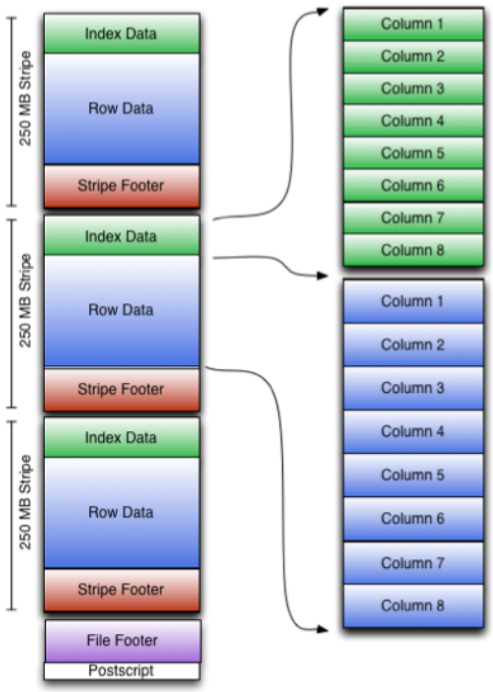


Fig. D.6.: Structure of ORC file format.

column chunks are divided up into *pages*, the smallest and indivisible units of a Parquet file (recommended size: 8KB). A footer contains the file metadata (version of the format, schema) and the metadata of each column (type, encoding codec, position (offset) of the first page, column size, etc.). Parquet metadata are stored using the Thrift serialization system (but it is also possible to use Avro or Protocol Buffers). The structure of a Parquet file is illustrated on Figure D.7. Parquet is designed for handling nested data structures (i.e. columns with several levels) and uses the *record shredding and assembly* algorithm from Google's Dremel system [Melnik et al., 2010]. It supports efficient compression schemes that can be specified on column-level.

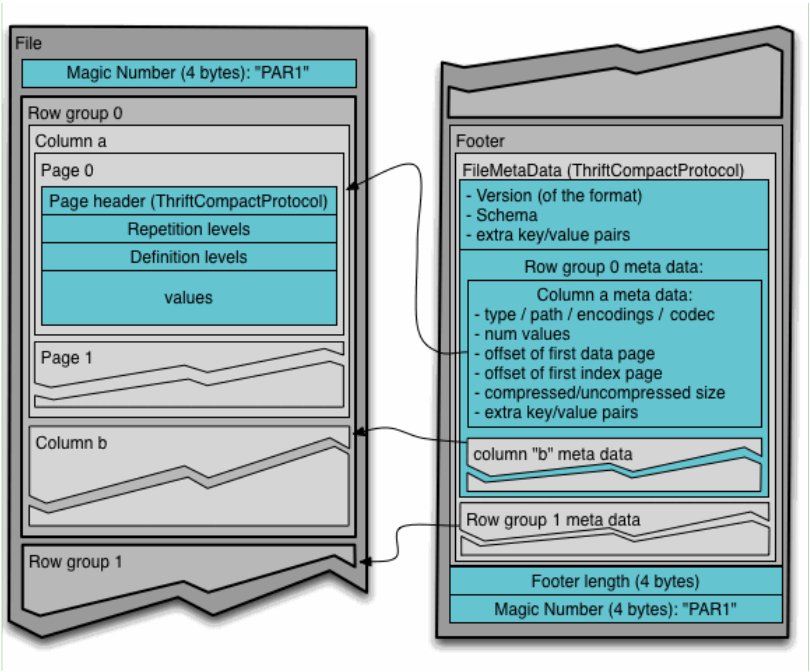
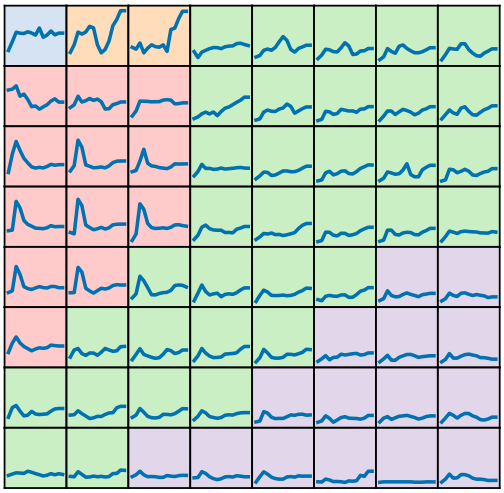


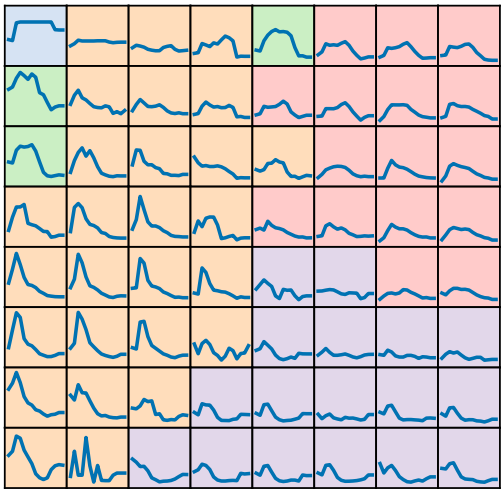
Fig. D.7.: Structure of Parquet file format.

Appendix to chapter 8 — Vibration profiles SOM maps

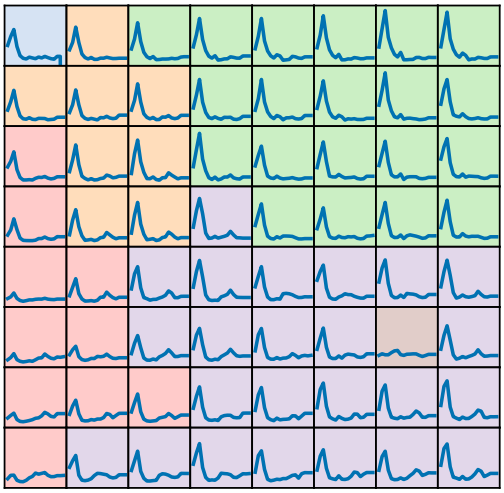
E



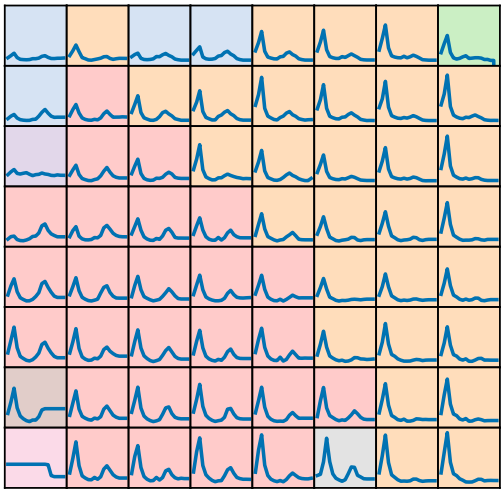
(a) Signature 1 (LP-ACC1 vs N1).



(b) Signature 2 (LP-ACC2 vs N1).



(c) Signature 3 (HP-ACC1 vs N2).



(d) Signature 4 (HP-ACC2 vs N2).

Fig. E.1.: SOM maps of signature 1, 2, 3 and 4. Each cell represents a vibration signature prototype. The background colors are higher-level profiles obtained by Ward hierarchical clustering (here with respectively 5, 5, 6 and 8 clusters).

List of Figures

List of Tables

List of Algorithms and Program Code

Glossary

ACARS	Aircraft Communication Addressing and Reporting System
AE	Autoencoder
API	Application Programming Interface
ARI	Adjusted Rand Index
BIC	Bayesian Information Criterion
BMU	Best-Matching Unit
CBM	Condition-Based Maintenance
CEOD	Continuous Engine Operational Data
CM	Condition Monitoring
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CVI	Clustering Validity Index
DAG	Directed Acyclic Graph
(R)DBMS	(Relational) DataBase Management System
DESOM	Deep Embedded SOM
DR	Dimensionality Reduction
EHM	Engine Health Monitoring
EM	Expectation-Maximization
GMM	Gaussian Mixture Model
GPU	Graphical Processing Unit
HC	Hierarchical Clustering
HDFS	Hadoop Distributed FileSystem
JVM	Java Virtual Machine
KL-divergence	Kullback-Leibler divergence
k-NN	k-Nearest Neighbors
ML	Machine Learning
MLE	Maximum Likelihood Estimation
MSE	Mean Squared Error
NMI	Normalized Mutual Information
PCA	Principal Component Analysis
PHM	Prognostics & Health Monitoring
QE/TE/CE	Quantization/Topographic/Combined Error
RDD	Resilient Distributed Dataset
RNN	Recurrent Neural Network
Safran A.E.	Safran Aircraft Engines
SDAE	Stacked Denoising Autoencoder
SGD	Stochastic Gradient Descent
SOM	Self-Organizing Map
SQL	Structured Query Language
UDF	User-Defined Function
UMAP	Uniform Manifold Approximation and Projection
VAE	Variational Autoencoder
YARN	Yet Another Resource Negotiator

Bibliography

[Abdel-Sayed, 2016] Abdel-Sayed, M. (2016). *Étude de représentations pour la détection d'anomalies - Application aux données vibratoires des moteurs d'avions*. PhD thesis, Université Paris-Saclay.

[Abdel-Sayed et al., 2015] Abdel-Sayed, M., Duclos, D., Faÿ, G., Lacaille, J., and Mougeot, M. (2015). NMF-based decomposition for anomaly detection applied to vibration analysis. In *International Conference on Condition Monitoring and Machinery Failure Prevention Technologies*, pages 73–81.

[Ackerman and Ben-David, 2009] Ackerman, M. and Ben-David, S. (2009). Measures of clustering quality: A working set of axioms for clustering. In *NIPS*.

[Affeldt et al., 2020] Affeldt, S., Labiod, L., and Nadif, M. (2020). Spectral clustering via ensemble deep autoencoder learning (SC-EDAE). *Pattern Recognition*.

[Aggarwal and Reddy, 2013] Aggarwal, C. C. and Reddy, C. K. (2013). *Data Clustering: Algorithms and Applications*.

[Aghabozorgi et al., 2015] Aghabozorgi, S., Seyed Shirkhorshidi, A., and Ying Wah, T. (2015). Time-series clustering - A decade review. *Information Systems*, 53:16–38. http://dx.doi.org/10.1016/j.is.2015.04.007.

[Agrawal et al., 1998] Agrawal, R., Gehrke, J., Gunopulos, D., and Raghavan, P. (1998). Automatic Subspace Clustering Mining. *ACM SIGMOD*, pages 94–105.

[Agrawal et al., 1995] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, a. I. (1995). Fast discovery of association rules. *Advances in knowledge discovery and data mining*.

[Airbus, 2018] Airbus (2018). Airbus' open aviation data platform Skywise continues to gain market traction. http://www.airbus.com/newsroom/press-releases/en/2018/02/airbus--open-aviation-data-platform-skywise-continues-to-gain-ma.html.

[Akaike, 1973] Akaike, H. (1973). Information Theory and an Extension of the Maximum Likelihood Principle. In *International Symposium on Information Theory*.

[Akerkar, 2014] Akerkar, R. (2014). Analytics on big aviation data: Turning data into insights. *International Journal of Computer Science and Applications*, 11(3):116–127.

[Akpinar and Karabacak, 2017] Akpinar, M. T. and Karabacak, M. E. (2017). Data mining applications in civil aviation sector: State-of-art review. *CEUR Workshop Proceedings*, 1852:18–25.

[Alahakoon et al., 2000] Alahakoon, D., Halgamuge, S. K., and Srinivasan, B. (2000). Dynamic Self-Organizing Maps with Controlled Growth for Knowledge Discovery. *IEEE Transactions on Neural Networks*, 11(3):601–614.

[Alain and Bengio, 2014] Alain, G. and Bengio, Y. (2014). What Regularized Auto-Encoders Learn from the Data-Generating Distribution. *Journal of Machine Learning Research*, 15:3743–3773.

[Alhoniemi et al., 1999] Alhoniemi, E., Himberg, J., Parviainen, J., and Vesanto, J. (1999). SOM Toolbox. `https://github.com/ilarinieminen/SOM-Toolbox`.

[Aljalbout et al., 2018] Aljalbout, E., Golkov, V., Siddiqui, Y., and Cremers, D. (2018). Clustering with Deep Learning: Taxonomy and New Methods. `http://arxiv.org/abs/1801.07648`.

[Ambroise et al., 2000] Ambroise, C., Sèze, G., Badran, F., and Thiria, S. (2000). Hierarchical clustering of self-organizing maps for cloud classification. *Neurocomputing*, 30(1-4):47–52.

[Ankerst et al., 1999] Ankerst, M., Breunig, M. M., Kriegel, H.-p., and Sander, J. (1999). OPTICS: Ordering Points To Identify the Clustering Structure. In *ACM SIGMOD*.

[Anouar et al., 1998] Anouar, F., Badran, F., and Thiria, S. (1998). Probabilistic self-organizing map and radial basis function networks. *Neurocomputing*, 20(1-3):83–96.

[Apache Hive, 2010] Apache Hive (2010). Hive Project. `http://hive.apache.org/`.

[Apache Spark, 2014] Apache Spark (2014). Spark Project. `https://spark.apache.org/`.

[Arbelaitz et al., 2013] Arbelaitz, O., Gurrutxaga, I., Muguerza, J., Pérez, J. M., and Perona, I. (2013). An extensive comparative study of cluster validity indices. *Pattern Recognition*, 46(1):243–256.

[Arpit et al., 2016] Arpit, D., Zhou, Y., Ngo, H. Q., and Govindaraju, V. (2016). Why regularized auto-encoders learn sparse representation? *International Conference on Machine Learning (ICML)*, 1:211–223.

[Arthur and Vassilvitskii, 2007] Arthur, D. and Vassilvitskii, S. (2007). k-means++: The Advantages of Careful Seeding. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035.

[Attaoui et al., 2020] Attaoui, M. O., Azzag, H., Lebbah, M., and Keskes, N. (2020). Subspace data stream clustering with global and local weighting models. *Neural Computing and Applications*, 0123456789. `https://doi.org/10.1007/s00521-020-05184-z`.

[Awasthi et al., 2012] Awasthi, P., Blum, A., and Sheffet, O. (2012). Center-based clustering under perturbation stability. *Information Processing Letters*, 112(1-2):49–54.

[Ayhan et al., 2013] Ayhan, S., Pesce, J., Comitz, P., Sweet, D., Bliesner, S., and Gerberick, G. (2013). Predictive analytics with aviation big data. In *Integrated Communications, Navigation and Surveillance Conference (ICNS)*.

[Aytekin et al., 2018] Aytekin, C., Ni, X., Cricri, F., and Aksu, E. (2018). Clustering and Unsupervised Anomaly Detection with l2 Normalized Deep Auto-Encoder Representations. In *International Joint Conference on Neural Networks (IJCNN)*.

[Azzag and Lebbah, 2008] Azzag, H. and Lebbah, M. (2008). Clustering of Self-Organizing Map. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*.

[Azzag et al., 2003] Azzag, H., Monmarche, N., Slimane, M., Venturini, G., and Guinot, C. (2003). AntTree: A new model for clustering with artificial ants. In *Congress on Evolutionary Computation (CEC)*.

[Bagnall et al., 2018] Bagnall, A., Dau, H. A., Lines, J., Flynn, M., Large, J., Bostrom, A., Southam, P., and Keogh, E. (2018). The UEA multivariate time series classification archive, 2018. `http://arxiv.org/abs/1811.00075`.

[Balcan et al., 2020] Balcan, M.-F., Haghtalab, N., and White, C. (2020). k-center Clustering under Perturbation Resilience. *ACM Transactions on Algorithms*, 16(2).

[Balcan and Liang, 2016] Balcan, M.-F. and Liang, Y. (2016). Clustering under perturbation resilience. *SIAM Journal on Computing*, 45(1):102–155.

[Banfield and Raftery, 1993] Banfield, J. D. and Raftery, A. E. (1993). Model-based Gaussian and non-Gaussian clustering. *Biometrics*, 49(3):803–821.

[Baragona, 2001] Baragona, R. (2001). A simulation study on clustering time series with metaheuristic methods. *Quaderni di Statistica*, 3.

[Bastard et al., 2016] Bastard, G., Lacaille, J., Coupard, J., and Stouky, Y. (2016). Engine Health Management in Safran Aircraft Engines. In *Annual Conference of the PHM Society*.

[Batista et al., 2011] Batista, G. E., Wang, X., and Keogh, E. (2011). A Complexity-Invariant Distance Measure for Time Series. In *SIAM International Conference on Data Mining (SDM)*, pages 699–710. `http://epubs.siam.org/doi/abs/10.1137/1.9781611972818.60`.

[Bauer et al., 1992] Bauer, H.-U., Pawelzik, K., and Geisel, T. (1992). A Topographic Product for the Optimization of Self-Organizing Feature Maps. *NIPS*, 4:1141–1147.

[Beck, 2019] Beck, G. (2019). *Scalable Clustering Applying Local Accretions (Accrétions Locales appliquées au Clustering Scalable et Distribué)*. PhD thesis, Université Paris 13.

[Becker, 1991] Becker, S. (1991). Unsupervised Learning Procedures for Neural Networks. *The International Journal of Neural Systems*, 1:17–33.

[Bell et al., 2009] Bell, G., Hey, T., and Szalay, A. (2009). Beyond the Data Deluge. *Science*, 323(5919):1297–1298. `http://www.cloudinnovation.com.au/Bell_Hey_Szalay_Science_March_2009.pdf`.

[Bellas et al., 2014] Bellas, A., Bouveyron, C., Cottrell, M., and Lacaille, J. (2014). Anomaly Detection Based on Confidence Intervals Using SOM with an Application to Health Monitoring. *International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)*, pages 145–155.

[Bellet et al., 2015] Bellet, A., Habrard, A., and Sebban, M. (2015). *Metric Learning*. Morgan & Claypool. `http://www.cs.cmu.edu/$\sim$liuy/distlearn.htm`.

[Bellman, 1961] Bellman, R. E. (1961). *Adaptive control processes: a guided tour*. Princeton university press.

[Ben-David, 2018] Ben-David, S. (2018). Clustering - What both theoreticians and practitioners are doing wrong. *AAAI Conference on Artificial Intelligence*, pages 7962–7964.

[Ben-David et al., 2007] Ben-David, S., Pal, D., and Simon, H. U. (2007). Stability of k-Means Clustering. In *Conference on Learning Theory (COLT)*, pages 20–34.

[Ben-David and Von Luxburg, 2008] Ben-David, S. and Von Luxburg, U. (2008). Relating clustering stability to properties of cluster boundaries. *Conference on Learning Theory (COLT)*, pages 379–390.

[Ben-David et al., 2006] Ben-David, S., Von Luxburg, U., and Pál, D. (2006). A sober look at clustering stability. *Lecture Notes in Computer Science*, 4005(2002):5–19.

[Ben-Hur et al., 2002] Ben-Hur, A., Elisseeff, A., and Guyon, I. (2002). A stability based method for discovering structure in clustered data. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, 17:6–17.

[Benabdeslem and Lebbah, 2007] Benabdeslem, K. and Lebbah, M. (2007). Feature selection for self-organizing map. In *International Conference on Information Technology Interfaces (ITI)*, pages 45–58.

[Benavoli et al., 2016] Benavoli, A., Corani, G., and Mangili, F. (2016). Should we really use post-hoc tests based on mean-ranks? *Journal of Machine Learning Research*, 17.

[Bengio, 2009] Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1).

[Bengio, 2012] Bengio, Y. (2012). Deep Learning of Representations for Unsupervised and Transfer Learning. *JMLR: Workshop and Conference Proceedings*, 27:17–37.

[Bengio et al., 2013] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

[Bengio et al., 2007] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. *NIPS*.

[Bezdek et al., 1988] Bezdek, J. C., Ehrlich, R., and Full, W. (1988). A generalisation of the Fuzzy c-Means clustering algorithm. *Computers & Geosciences*, 10(2):1783–1784.

[Biernacki et al., 2000] Biernacki, C., Celeux, G., and Govaert, G. (2000). Assessing a Mixture Model for Clustering with Integrated Completed likelihood. *IEEE Transactions on Pattern Analysis and Machine Learning*, 22(7):1899–1906.

[Bilu and Linial, 2012] Bilu, Y. and Linial, N. (2012). Are stable instances easy? *Combinatorics Probability and Computing*, 21(5):643–660.

[Bishop et al., 1998] Bishop, C. M., Svensen, M., and Williams, C. K. I. (1998). Developments of the generative topographic mapping. *Neurocomputing*, 21(1-3):203–224.

[Blanchard et al., 2009] Blanchard, S., Cottrell, M., and Lacaille, J. (2009). Health monitoring des moteurs d'avions. In *Les entretients de Toulouse*.

[Bo and Wang, 2020] Bo, D. and Wang, X. (2020). Structural Deep Clustering Network. In *International World Wide Web Conference (WWW)*.

[Bock, 1996] Bock, H. H. (1996). Probabilistic models in cluster analysis. *Computational Statistics and Data Analysis*, 23(1):5–28.

[Boelaert et al., 2014] Boelaert, J., Bendhaiba, L., Olteanu, M., and Villa-Vialaneix, N. (2014). SOMbrero: An R Package for Numeric and Non-numeric Self-Organizing Maps. In *International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)*, pages 219–228. `https://github.com/tuxette/SOMbrero`.

[Bourlard and Kamp, 1988] Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4-5):291–294.

[Bouveyron and Brunet-Saumard, 2014] Bouveyron, C. and Brunet-Saumard, C. (2014). Model-based clustering of high-dimensional data: A review. *Computational Statistics and Data Analysis*, 71:52–78. `http://dx.doi.org/10.1016/j.csda.2012.12.008`.

[Bouveyron et al., 2015] Bouveyron, C., Côme, E., and Jacques, J. (2015). The discriminative functional mixture model for a comparative analysis of bike sharing systems. *Annals of Applied Statistics*, 9(4):1726–1760.

[Bouveyron et al., 2007] Bouveyron, C., Girard, S., and Schmid, C. (2007). High-dimensional data clustering. *Computational Statistics and Data Analysis*, 52(1):502–519.

[Bowman et al., 2016] Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., and Bengio, S. (2016). Generating sentences from a continuous space. *SIGNLL Conference on Computational Natural Language Learning (CoNLL)*, pages 10–21.

[Broderick, 2016] Broderick, S. (2016). GE Ramping Up Results-Driven Big Data Analytics. `http://www.mro-network.com/maintenance-repair-overhaul/ge-ramping-results-driven-big-data-analytics`.

[Bubeck et al., 2012] Bubeck, S., Meilă, M., and Luxburg, U. V. (2012). How the initialization affects the stability of the k-means algorithm. *ESAIM - Probability and Statistics*, 16:436–452.

[Bullen et al., 2003] Bullen, R. J., Cornford, D., and Nabney, I. (2003). Outlier detection in scatterometer data: Neural network approaches. *Neural Networks*, 16(3-4):419–426.

[Cabanes and Bennani, 2007] Cabanes, G. and Bennani, Y. (2007). A simultaneous two-level clustering algorithm for automatic model selection. In *International Conference on Machine Learning and Applications (ICMLA)*, pages 316–321.

[Cabanes and Bennani, 2010] Cabanes, G. and Bennani, Y. (2010). Learning the number of clusters in SOM. In *Self-Organizing Maps*, pages 15–29. IntechOpen.

[Caliński and Harabasz, 1974] Caliński, T. and Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics*, 3(1). `https://www.tandfonline.com/doi/abs/10.1080/03610927408827101`.

[Carlsson, 2009] Carlsson, G. (2009). Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308.

[Carlsson and Mémoli, 2010] Carlsson, G. and Mémoli, F. (2010). Characterization, stability and convergence of hierarchical clustering methods. *Journal of Machine Learning Research*, 11:1425–1470.

[Caruana, 1997] Caruana, R. (1997). Multitask Learning. *Machine Learning*, 28:41–75.

[Chamroukhi and Nguyen, 2018] Chamroukhi, F. and Nguyen, H. D. (2018). Model-Based Clustering and Classification of Functional Data. `http://arxiv.org/abs/1803.00276`.

[Chavent et al., 2020] Chavent, M., Lacaille, J., Mourer, A., and Olteanu, M. (2020). Sparse k -means for mixed data via group-sparse clustering. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*.

[Chazan et al., 2019] Chazan, S. E., Gannot, S., and Goldberger, J. (2019). Deep Clustering based on a Mixture of Autoencoders. In *International Workshop on Machine Learning for Signal Processing*.

[Chen et al., 2017] Chen, D., Lv, J., and Yi, Z. (2017). Unsupervised Multi-Manifold Clustering by Learning Deep Representation. In *AAAI Conference on Artificial Intelligence*, pages 385–391.

[Chen, 2015] Chen, G. (2015). Deep Learning with Nonparametric Clustering.

[Chen et al., 2018] Chen, T. Q., Li, X., Grosse, R., and Duvenaud, D. (2018). Isolating sources of disentanglement in variational autoencoders. In *NeurIPS*.

[Chen et al., 2016] Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016). InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. `http://arxiv.org/abs/1606.03657`.

[Cheng et al., 1999] Cheng, C.-H., Fu, A. W., and Zhang, Y. (1999). Entropy-based subspace clustering for mining numerical data. In *KDD*, pages 84–93.

[Côme et al., 2010] Côme, E., Cottrell, M., Verleysen, M., and Lacaille, J. (2010). Aircraft engine health monitoring using Self-Organizing Maps. In *Industrial Conference on Data Mining*.

[Côme et al., 2011] Côme, E., Cottrell, M., Verleysen, M., and Lacaille, J. (2011). Aircraft engine fleet monitoring using Self-Organizing Maps and Edit Distance. In *International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)*, pages 298–307.

[Costa and Netto, 1999] Costa, J. A. F. and Netto, M. L. d. A. (1999). Estimating the Number of Clusters in Multivariate Data by Self-Organizing Maps. *International Journal of Neural Systems*, 9(3):195–202.

[Cottrell and De Bodt, 1996] Cottrell, M. and De Bodt, E. (1996). A Kohonen map representation to avoid misleading interpretations. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pages 103–110.

[Cottrell et al., 2009] Cottrell, M., Gaubert, P., Eloy, C., François, D., Hallaux, G., Lacaille, J., and Verleysen, M. (2009). Fault prediction in aircraft engines using Self-Organizing Maps. In *International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)*.

[Cottrell et al., 2018] Cottrell, M., Olteanu, A., Rossi, F., and Villa-vialaneix, N. (2018). Self-Organizing Maps, theory and applications. *Revista de Investigacion Operacional*, 39(1).

[Coupard et al., 2018] Coupard, J., Garnier, A., and Lacaille, J. (2018). Aircraft engines possession costs reduction with structural health monitoring. In *European Workshop on Structural Health Monitoring (EWSHM)*.

[Csáji, 2001] Csáji, B. (2001). *Approximation with artificial neural networks*. Msc. thesis, Eötvös Loránd University.

[Dahal, 2018] Dahal, P. (2018). Learning Embedding Space for Clustering From Deep Representations. In *IEEE International Conference on Big Data*.

[Davies and Bouldin, 1979] Davies, D. L. and Bouldin, D. W. (1979). A Cluster Separation Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227.

[De La Torre and Kanade, 2006] De La Torre, F. and Kanade, T. (2006). Discriminative cluster analysis. In *International Conference on Machine Learning (ICML)*.

[De Soete and Carroll, 1994] De Soete, G. and Carroll, J. D. (1994). K-means clustering in a low-dimensional Euclidean space. *New Approaches in Classification and Data Analysis*, pages 212–219.

[Dean and Ghemawat, 2008] Dean, J. and Ghemawat, S. (2008). MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*.

[Demartines and Blayo, 1992] Demartines, P. and Blayo, F. (1992). Kohonen Self-Organizing Maps: Is the Normalization Necessary ? *Complex Systems*, 6:105–123.

[Dempster et al., 1977] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society Series B*, 39(1).

[Demšar, 2006] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7.

[Deng et al., 2016] Deng, Z., Choi, K. S., Jiang, Y., Wang, J., and Wang, S. (2016). A survey on soft subspace clustering. *Information Sciences*, 348:84–106. http://dx.doi.org/10.1016/j.ins.2016.01.101.

[Desgraupes, 2013] Desgraupes, B. (2013). ClusterCrit: Clustering Indices. cran.r-project.org/web/packages/clusterCrit.

[Devries and Taylor, 2017] Devries, T. and Taylor, G. W. (2017). Dataset Augmentation in Feature Space. In *ICLR Workshop*.

[Deza and Deza, 2009] Deza, M. M. and Deza, E. (2009). *Encyclopedia of distances*. Springer Berlin Heidelberg.

[Diday and Simon, 1976] Diday, E. and Simon, J. C. (1976). *Clustering Analysis*, pages 47–94. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-96303-2_3.

[Diebold, 2000] Diebold, F. X. (2000). 'Big data' dynamic factor models for macroeconomic measuring and forecasting. *Advances in Economics and Econometrics, Eighth World Congress of the Econometric Society*, pages 115–122.

[Dilokthanakul et al., 2017] Dilokthanakul, N., Mediano, P. A. M., Garnelo, M., Lee, M. C. H., Salimbeni, H., Arulkumaran, K., and Shanahan, M. (2017). Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders.

[Ding and Li, 2007] Ding, C. and Li, T. (2007). Adaptive dimension reduction using discriminant analysis and K-means clustering. In *International Conference on Machine Learning (ICML)*.

[Dittenbach et al., 2000] Dittenbach, M., Merkl, D., and Rauber, A. (2000). The growing hierarchical self-organizing map. In *International Joint Conference on Neural Networks (IJCNN)*, pages 15–19.

[Dizaji et al., 2017] Dizaji, K. G., Herandi, A., Deng, C., Cai, W., and Huang, H. (2017). Deep Clustering via Joint Convolutional Autoencoder Embedding and Relative Entropy Minimization. In *ICCV*, pages 5747–5756.

[Doersch et al., 2015] Doersch, C., Gupta, A., and Efros, A. A. (2015). Unsupervised visual representation learning by context prediction. In *ICCV*, pages 1422–1430.

[Dudoit and Fridlyand, 2002] Dudoit, S. and Fridlyand, J. (2002). A prediction-based resampling method for estimating the number of clusters in a dataset. *Genome biology*, 3(7).

[Dunn, 1973] Dunn, J. C. (1973). A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3):32–57.

[Dunn, 1974] Dunn, J. C. (1974). Well-Separated Clusters and Optimal Fuzzy Partitions. *Journal of Cybernetics*, 1(4):95–104.

[Dutta et al., 2017] Dutta, A., Vijayaraghavan, A., and Wang, A. (2017). Clustering stable instances of euclidean k-means. In *NIPS*, pages 6501–6510.

[Elend and Kramer, 2019] Elend, L. and Kramer, O. (2019). Self-Organizing Maps with Convolutional Layers. In *International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)*.

[Elhamifar and Vidal, 2013] Elhamifar, E. and Vidal, R. (2013). Sparse subspace clustering: Algorithm, theory, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(11):2765–2781.

[Erhan et al., 2010] Erhan, D., Courville, A., and Vincent, P. (2010). Why Does Unsupervised Pre-training Help Deep Learning ? *Journal of Machine Learning Research*, 11:625–660.

[Ester et al., 1996] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*.

[Falasconi et al., 2010] Falasconi, M., Gutierrez, A., Pardo, M., Sberveglieri, G., and Marco, S. (2010). A stability based validity method for fuzzy clustering. *Pattern Recognition*, 43(4):1292–1305. `http://dx.doi.org/10.1016/j.patcog.2009.10.001`.

[Fang and Wang, 2012] Fang, Y. and Wang, J. (2012). Selection of the number of clusters via the bootstrap method. *Computational Statistics and Data Analysis*, 56(3):468–477. `http://dx.doi.org/10.1016/j.csda.2011.09.003`.

[Färber et al., 2010] Färber, I., Günnemann, S., Kriegel, H.-P., Kröger, P., Müller, E., Schubert, E., Seidl, T., and Zimek, A. (2010). On Using Class-Labels in Evaluation of Clusterings. *KDD International Workshop on Discovering, Summarizing and Using Multiple Clusterings (MultiClust)*, page 9.

[Fard et al., 2018] Fard, M. M., Thonet, T., and Gaussier, E. (2018). Deep k-Means: Jointly Clustering with k-Means and Learning Representations. `http://arxiv.org/abs/1806.10069`.

[Faure et al., 2017] Faure, C., Olteanu, M., Bardet, J.-M., and Lacaille, J. (2017). Using self-organizing maps for clustering and labelling aircraft engine data phases. In *International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)*.

[Fawaz et al., 2019] Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., and Muller, P. A. (2019). Adversarial Attacks on Deep Neural Networks for Time Series Classification. In *International Joint Conference on Neural Networks (IJCNN)*.

[Fawzi et al., 2016] Fawzi, A., Samulowitz, H., Turaga, D., and Frossard, P. (2016). Adaptive data augmentation for image classification. *International Conference on Image Processing (ICIP)*.

[Feng and Hamerly, 2007] Feng, Y. and Hamerly, G. (2007). PG-means: Learning the number of clusters in data. *NIPS*, pages 393–400.

[Ferguson, 1973] Ferguson, T. S. (1973). A Bayesian analysis of some nonparametric problems. *The Annals of statistics*, 1(2):209–230.

[Ferles et al., 2018] Ferles, C., Papanikolaou, Y., and Naidoo, K. J. (2018). Denoising Autoencoder Self-Organizing Map (DASOM). *Neural Networks*, 105:112–131. `https://doi.org/10.1016/j.neunet.2018.04.016`.

[Fink et al., 2020] Fink, O., Wang, Q., Svensén, M., Dersin, P., Lee, W. J., and Ducoffe, M. (2020). Potential, challenges and future directions for deep learning in prognostics and health management applications. *Engineering Applications of Artificial Intelligence*, 92(April):103678. `https://doi.org/10.1016/j.engappai.2020.103678`.

[Forest, 2019] Forest, F. (2019). Spark ML SOM: Spark ML implementation of SOM algorithm. `https://github.com/FlorentF9/sparkml-som`.

[Forest, 2020] Forest, F. (2020). SOMperf: Self-organizing maps performance metrics and quality indices. `https://github.com/FlorentF9/SOMperf`.

[Forest et al., 2020a] Forest, F., Cochard, Q., Noyer, C., Cabut, A., Joncour, M., Lacaille, J., Lebbah, M., and Azzag, H. (2020a). Large-scale Vibration Monitoring of Aircraft Engines from Operational Data using Self-organized Models. In *Annual Conference of the PHM Society*.

[Forest et al., 2018] Forest, F., Lacaille, J., Lebbah, M., and Azzag, H. (2018). A Generic and Scalable Pipeline for Large-Scale Analytics of Continuous Aircraft Engine Data. In *IEEE International Conference on Big Data*.

[Forest et al., 2019a] Forest, F., Lebbah, M., Azzag, H., and Lacaille, J. (2019a). Deep Architectures for Joint Clustering and Visualization with Self-Organizing Maps. In *PAKDD Workshop on Learning Data Representations for Clustering (LDRC)*.

[Forest et al., 2019b] Forest, F., Lebbah, M., Azzag, H., and Lacaille, J. (2019b). Deep Embedded SOM: Joint Representation Learning and Self-Organization. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*.

[Forest et al., 2020b] Forest, F., Lebbah, M., Azzag, H., and Lacaille, J. (2020b). A Survey and Implementation of Performance Metrics for Self-Organized Maps. `https://arxiv.org/abs/2011.05847`.

[Forest et al., 2020c] Forest, F., Lebbah, M., Azzag, H., and Lacaille, J. (2020c). Carte SOM profonde : Apprentissage joint de représentations et auto-organisation. In *CAp: Conférence d'Apprentissage*. `https://hal.archives-ouvertes.fr/hal-02859997`.

[Forest and Mourer, 2020] Forest, F. and Mourer, A. (2020). skstab: Clustering stability analysis in Python with a scikit-learn compatible API. `https://github.com/FlorentF9/skstab`.

[Forest et al., 2021] Forest, F., Mourer, A., Lebbah, M., Azzag, H., and Lacaille, J. (2021). An Invariance-guided Stability Criterion for Time Series Clustering Validation. In *International Conference on Pattern Recognition (ICPR)*.

[Fortuin et al., 2019] Fortuin, V., Hüser, M., Locatello, F., Strathmann, H., and Rätsch, G. (2019). SOM-VAE: Interpretable Discrete Representation Learning on Time Series. In *International Conference on Learning Representations (ICLR)*.

[Fowlkes and Mallows, 1983] Fowlkes, E. B. and Mallows, C. L. (1983). A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78(383):553–569.

[Franceschi et al., 2019] Franceschi, J.-Y., Dieuleveut, A., and Jaggi, M. (2019). Unsupervised Scalable Representation Learning for Multivariate Time Series. In *NeurIPS*. `http://arxiv.org/abs/1901.10738`.

[Frey and Dueck, 2007] Frey, B. J. and Dueck, D. (2007). Clustering by passing messages between data points. *Science*, 315(5814):972–976.

[Frey and Jojic, 2000] Frey, B. J. and Jojic, N. (2000). Transformation-Invariant Clustering and Dimensionality Reduction Using EM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(1).

[Fridlyand and Dudoit, 2001] Fridlyand, J. and Dudoit, S. (2001). Applications of resampling methods to estimate the number of clusters and to improve the accuracy of a clustering method.

[Fritzke, 1994] Fritzke, B. (1994). Growing cell structures-A self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460.

[Fritzke, 1995] Fritzke, B. (1995). A Growing Neural Gas Learns Topologies. In *NIPS*, volume 7, pages 625–632.

[Fu et al., 2020] Fu, B., Kirchbuchner, F., and Kuijper, A. (2020). Data Augmentation for Time Series : Traditional vs Generative Models on Capacitive Proximity Time Series. In *ACM International Conference on PErvasive Technologies Related to Assistive Environment (PETRA)*, pages 107–116.

[Fujita et al., 2014] Fujita, A., Takahashi, D. Y., and Patriota, A. G. (2014). A non-parametric method to estimate the number of clusters. *Computational Statistics and Data Analysis*, 73:27–39. `http://dx.doi.org/10.1016/j.csda.2013.11.012`.

[Gan et al., 2006] Gan, G., Wu, J., and Yang, Z. (2006). A Fuzzy Subspace Algorithm for Clustering High Dimensional Data. *Advanced Data Mining and Applications*, pages 271–278.

[García and Herrera, 2008] García, S. and Herrera, F. (2008). An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677–2694.

[Ge et al., 2019] Ge, P., Ren, C.-x., Dai, D.-q., Feng, J., and Yan, S. (2019). Dual Adversarial Autoencoders for Clustering. *IEEE Transactions on Neural Networks and Learning Systems*.

[Gidaris et al., 2018] Gidaris, S., Singh, P., and Komodakis, N. (2018). Unsupervised representation learning by predicting image rotations. In *ICLR*.

[Giusti and Batista, 2013] Giusti, R. and Batista, G. E. (2013). An empirical comparison of dissimilarity measures for time series classification. In *Brazilian Conference on Intelligent Systems (BRACIS)*, pages 82–88.

[Goffinet et al., 2020a] Goffinet, É., Lebbah, M., Azzag, H., and Giraldi, L. (2020a). Autonomous Driving Validation With Model-Based Dictionary Clustering. In *ECML-PKDD*.

[Goffinet et al., 2020b] Goffinet, É., Lebbah, M., Azzag, H., and Giraldi, L. (2020b). Clustering de séries temporelles par construction de dictionnaire. In *EGC*, pages 181–192.

[Goil et al., 1999] Goil, S., Nagesh, H., and Choudhary, A. (1999). MAFIA: Efficient and scalable subspace clustering for very large data sets. `citeseer.ist.psu.edu/goil99mafia.html`.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. In *NIPS*.

[Goodhill and Sejnowski, 1996] Goodhill, G. J. and Sejnowski, T. J. (1996). Quantifying neighbourhood preservation in topographic mappings. In *Joint Symposium on Neural Computation*, pages 61–82.

[Gorban and Zinovyev, 2008] Gorban, A. N. and Zinovyev, A. Y. (2008). Principal Graphs and Manifolds.

[Guo et al., 2017a] Guo, X., Gao, L., Liu, X., and Yin, J. (2017a). Improved deep embedded clustering with local structure preservation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1753–1759.

[Guo et al., 2017b] Guo, X., Liu, X., Zhu, E., and Yin, J. (2017b). Deep Clustering with Convolutional Autoencoders. In *ICONIP*.

[Guyon and Elisseeff, 2003] Guyon, I. and Elisseeff, A. (2003). An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, pages 1157–1182.

[Hämäläinen et al., 2017] Hämäläinen, J., Jauhiainen, S., and Kärkkäinen, T. (2017). Comparison of internal clustering validation indices for prototype-based clustering. *Algorithms*, 10(3).

[Hamel, 2016] Hamel, L. (2016). SOM quality measures: An efficient statistical approach. *Advances in Intelligent Systems and Computing*, 428:49–59.

[Hamerly and Elkan, 2004] Hamerly, G. and Elkan, C. (2004). Learning the K in K-means. *NIPS*.

[Harchaoui et al., 2019] Harchaoui, W., Mattei, P.-A., Alamansa, A., and Bouveyron, C. (2019). Wasserstein Adversarial Mixture for Deep Generative Modeling and Clustering. In *AISTATS*.

[Harris, 1993] Harris, T. (1993). A Kohonen S.O.M. based, machine health monitoring system which enables diagnosis of faults not seen in the training set. In *International Joint Conference on Neural Networks (IJCNN)*, pages 947–950.

[Hartigan, 1975] Hartigan, J. A. (1975). *Clustering Algorithms*. John Wiley & Sons.

[Hartigan and Hartigan, 1985] Hartigan, J. A. and Hartigan, P. M. (1985). The dip test of unimodality. *Annals of Statistics*, 13(1):70–84.

[Hartigan and Wong, 1979] Hartigan, J. A. and Wong, M. A. (1979). Algorithm AS 146: A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society Series C*, 28(1):100–108.

[Hastie et al., 2008] Hastie, T., Tibshirani, R., and Friedman, J. (2008). *The Elements of Statistical Learning*. Springer.

[Hazan et al., 2010] Hazan, A., Verleysen, M., Cottrell, M., and Lacaille, J. (2010). Trajectory Clustering for Vibration Detection in Aircraft Engines. In *Industrial Conference on Data Mining*.

[He et al., 2011a] He, Y., Lee, R., Huai, Y., Shao, Z., Jain, N., Zhang, X., and Xu, Z. (2011a). RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems. In *International Conference on Data Engineering*, pages 1199–1208.

[He et al., 2011b] He, Y., Tan, H., Luo, W., Mao, H., Ma, D., Feng, S., and Fan, J. (2011b). MR-DBSCAN : An Efficient Parallel Density-based Clustering Algorithm using MapReduce. In *IEEE International Conference on Parallel and Distributed Systems*.

[Hebb, 1949] Hebb, D. O. (1949). *The organization of behavior: A neuropsychological theory*. Wiley.

[Heskes, 2001] Heskes, T. (2001). Self-Organizing Maps , Vector Quantization , and Mixture Modeling. *IEEE Transactions on Neural Networks*, 12(6):1299–1305.

[Hess and Duivesteijn, 2019] Hess, S. and Duivesteijn, W. (2019). K Is the Magic Number — Inferring the Number of Clusters Through Nonparametric Concentration Inequalities. In *EMCL-PKDD*.

[Higgins et al., 2017] Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2017). beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *International Conference on Learning Representations (ICLR)*.

[Hinton and Roweis, 2002] Hinton, G. and Roweis, S. (2002). Stochastic Neighbor Embedding. In *NIPS*.

[Hinton and Salakhutdinov, 2006] Hinton, G. E. and Salakhutdinov, R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(July):504–507.

[Hinton and Zemel, 1993] Hinton, G. E. and Zemel, R. S. (1993). Autoencoders, Minimum Description Length and Helmholtz Free Energy. In *NIPS*.

[Holm, 1979] Holm, S. (1979). A Simple Sequentially Rejective Multiple Test Procedure. *Scandinavian Journal of Statistics*, 6(2):65–70.

[Hsu et al., 2017] Hsu, C.-c., Lin, C.-w., and Member, S. (2017). CNN-Based Joint Clustering and Representation Learning with Feature Drift Compensation for Large - Scale Image Data. *IEEE Transactions on Multimedia*.

[Hu et al., 2017] Hu, W., Miyato, T., Tokui, S., Matsumoto, E., and Sugiyama, M. (2017). Learning discrete representations via information maximizing self-augmented training. In *International Conference on Machine Learning (ICML)*, volume 4, pages 2467–2481.

[Hu and Xu, 2003] Hu, X. and Xu, L. (2003). A Comparative Study of Several Cluster Number Selection Criteria. In *IDEAL*.

[Huang et al., 2005] Huang, J. Z., Ng, M. K., Rong, H., and Li, Z. (2005). Automated variable weighting in k-means type clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(5):657–668.

[Huang et al., 2014] Huang, P., Huang, Y., Wang, W., and Wang, L. (2014). Deep embedding network for clustering. *International Conference on Pattern Recognition (ICPR)*, pages 1532–1537.

[Hubert and Arabie, 1985] Hubert, L. and Arabie, P. (1985). Comparing partitions. *Journal of Classification*, 2(1):193–218.

[Hutter et al., 2014] Hutter, F., Hoos, H., and Leyton-Brown, K. (2014). An efficient approach for assessing hyperparameter importance. *International Conference on Machine Learning (ICML)*, 2:1130–1144.

[IBM Global Business Services, 2015] IBM Global Business Services (2015). Commercial Aviation and Aerospace: Big Data Analytics for Advantage, Differentiation and Dollars. Technical report. `https://fr.slideshare.net/`

SedaESKILER/commercial-aviation-and-aerospace-big-data-analytics-for-advantage-differentiation-and-dollars.

[Idan and Chevalier, 1991] Idan, Y. and Chevalier, R. C. (1991). Handwritten Digits Recognition by a Supervised Kohonen - Like Learning Algorithm. *Neural Networks*, pages 15–17.

[Inoubli et al., 2018] Inoubli, W., Aridhi, S., Mezni, H., Maddouri, M., and Mephu Nguifo, E. (2018). An experimental survey on big data frameworks. *Future Generation Computer Systems*, 86:546–564.

[Jain, 2010] Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8):651–666. http://dx.doi.org/10.1016/j.patrec.2009.09.011.

[Januzaj et al., 2004] Januzaj, E., Kriegel, H.-P., and Pfeifle, M. (2004). DBDC : Density Based Distributed Clustering. In *International Conference on Extending Database Technology (EDBT)*.

[Jawed et al., 2020] Jawed, S., Grabocka, J., and Schmidt-Thieme, L. (2020). Self-supervised Learning for Semi-supervised Time Series Classification. In *PAKDD*, pages 499–511.

[Jaziri et al., 2011] Jaziri, R., Lebbah, M., Rogovschi, N., and Bennani, Y. (2011). Probabilistic self-organizing maps for multivariate sequences. In *International Joint Conference on Neural Networks (IJCNN)*, pages 851–858.

[Jiang et al., 2017] Jiang, Z., Zheng, Y., Tan, H., Tang, B., and Zhou, H. (2017). Variational Deep Embedding : An Unsupervised and Generative Approach to Clustering. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1965–1972.

[Jing et al., 2005] Jing, L., Ng, M. K., Xu, J., and Huang, J. Z. (2005). Subspace Clustering of Text Documents with Feature Weighting K -Means Algorithm. In *PAKDD*, pages 802–812.

[Jing et al., 2017] Jing, L., Zhao, M., Li, P., and Xu, X. (2017). A convolutional neural network based feature learning and fault diagnosis method for the condition monitoring of gearbox. *Measurement: Journal of the International Measurement Confederation*, 111(July). http://dx.doi.org/10.1016/j.measurement.2017.07.017.

[Kaczynska et al., 2020] Kaczynska, S., Marion, R., and von Sachs, R. (2020). Comparison of Cluster Validity Indices and Decision Rules for Different Degrees of Cluster Separation. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*.

[Kailing and Kriegel, 2004] Kailing, K. and Kriegel, H.-p. (2004). Density-Connected Subspace Clustering for High-Dimensional Data. In *SIAM International Conference on Data Mining (SDM)*, pages 246–256.

[Kalogeratos and Likas, 2012] Kalogeratos, A. and Likas, A. (2012). Dip-means: An incremental clustering method for estimating the number of clusters. *NIPS*, 3:2393–2401.

[Kaly et al., 2004] Kaly, F., Niang, N., Ouattara, M., Niang, A., Thiria, S., Marticorena, B., and Janicot, S. (2004). Two step soft subspace SOM : une méthode de classification multi-bloc avec sélection de variables.

[Kaski and Lagus, 1996] Kaski, S. and Lagus, K. (1996). Comparing Self-Organizing Maps. In *International Conference on Artificial Neural Networks (ICANN)*.

[Kaski et al., 1998] Kaski, S., Nikkilä, J., and Kohonen, T. (1998). Methods for Interpreting a Self-Organized Map in Data Analysis. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pages 185–190.

[Kasturi et al., 2016] Kasturi, E., Prasanna Devi, S., Vinu Kiran, S., and Manivannan, S. (2016). Airline Route Profitability Analysis and Optimization Using BIG DATA Analytics on Aviation Data Sets under Heuristic Techniques. *Procedia Computer Science*, 87:86–92. `http://dx.doi.org/10.1016/j.procs.2016.05.131`.

[Kaufman and Rousseeuw, 1990] Kaufman, L. and Rousseeuw, P. J. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons.

[Kerr and Churchill, 2001] Kerr, M. K. and Churchill, G. A. (2001). Experimental design for gene expression microarrays. *Biostatistics*, 2(2):183–201.

[Kharyton, 2009] Kharyton, V. (2009). *Faults Detection In Blades Of An Aviation Engine In Operation*. PhD thesis, École Centrale Lyon.

[Kilinc and Uysal, 2017] Kilinc, O. and Uysal, I. (2017). Auto-clustering Output Layer : Automatic Learning of Latent Annotations in Neural Networks.

[Kilinc and Uysal, 2018] Kilinc, O. and Uysal, I. (2018). Learning latent representations in neural networks for clustering through pseudo supervision and graph-based activity regularization. In *International Conference on Learning Representations (ICLR)*.

[Kim and Mnih, 2018] Kim, H. and Mnih, A. (2018). Disentangling by factorising. In *International Conference on Machine Learning (ICML)*.

[Kingma and Ba, 2015] Kingma, D. P. and Ba, J. L. (2015). Adam: A Method For Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*. `http://arxiv.org/abs/1412.6980`.

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*.

[Kingma and Welling, 2019] Kingma, D. P. and Welling, M. (2019). An Introduction to Variational Autoencoders. *Foundations and Trends in Machine Learning*.

[Kiviluoto, 1996] Kiviluoto, K. (1996). Topology preservation in self-organizing maps. In *IEEE International Conference on Neural Networks (ICNN)*, volume 1, pages 294–299.

[Klassen et al., 2020] Klassen, G., Tatusch, M., Himmelspach, L., and Conrad, S. (2020). Fuzzy Clustering Stability Evaluation of Time Series. In *Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU)*. `http://dx.doi.org/10.1007/978-3-030-50146-4_50`.

[Kleinberg, 2003] Kleinberg, J. (2003). An impossibility theorem for clustering. *Advances in Neural Information Processing Systems*.

[Kohonen, 1982] Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69.

[Kohonen, 1990] Kohonen, T. (1990). The Self-Organizing Map. In *Proceedings of the IEEE*, volume 78, pages 1464–1480.

[Kohonen, 1995] Kohonen, T. (1995). The Adaptive-Subspace SOM (ASSOM) and its Use for the Implementation of Invariant Feature Detection. In *ICANN*.

[Kruskal, 1964] Kruskal, J. B. (1964). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1).

[Kuhn, 1955] Kuhn, H. W. (1955). The Hungarian Method for the assignment problem. *Naval Research Logistics Quarterly*.

[Kulis, 2012] Kulis, B. (2012). Metric learning: A survey. *Foundations and Trends in Machine Learning*, 5(4):287–364.

[Kushagra et al., 2018] Kushagra, S., Ben-David, S., and Ilyas, I. (2018). Semi-supervised clustering for de-duplication. `http://arxiv.org/abs/1810.04361`.

[Laaksonen et al., 2001] Laaksonen, J., Koskela, M., Laakso, S., and Oja, E. (2001). Self-organising maps as a relevance feedback technique in content-based image retrieval. *Pattern Analysis and Applications*, 4(2-3):140–152.

[Lacaille, 2013] Lacaille, J. (2013). Searching similar vibration patterns on turbofan engines. In *International Conference on Condition Monitoring and Machinery Failure Prevention Technologies*, pages 338–349.

[Lacaille et al., 2014] Lacaille, J., Bellas, A., and Bou (2014). Online normalization algorithm for engine turbofan monitoring. In *Annual Conference of the PHM Society*, pages 415–422.

[Lacaille and Côme, 2011] Lacaille, J. and Côme, E. (2011). Visual mining and statistics for a turbofan engine fleet. In *IEEE Aerospace Conference*.

[Laney, 2001] Laney, D. (2001). 3D Data Management: Controlling Data Volume, Velocity, and Variety. *Application Delivery Strategies*, 949(February 2001):4.

[Lange et al., 2004] Lange, T., Roth, V., Braun, M. L., and Buhmann, J. M. (2004). Stability-based validation of clustering solutions. *Neural Computation*, 16(6):1299–1323.

[Lara and González, 2020] Lara, J. S. and González, F. A. (2020). Dissimilarity Mixture Autoencoder for Deep Clustering.

[Lebbah et al., 1999] Lebbah, M., Badran, F., and Thiria, S. (1999). Topological Map for Binary Data. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*.

[Lebbah and Chazottes, 2005] Lebbah, M. and Chazottes, A. (2005). Mixed Topological Map. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*.

[Lebbah et al., 2015] Lebbah, M., Jaziri, R., Bennani, Y., and Chenot, J.-H. (2015). Probabilistic Self-Organizing Map for Clustering and Visualizing non-i.i.d Data. *International Journal of Computational Intelligence and Applications*, 14(02):1550007. `http://www.worldscientific.com/doi/abs/10.1142/S1469026815500078`.

[Lebbah et al., 2007] Lebbah, M., Rogovschi, N., and Bennani, Y. (2007). BeSOM: Bernoulli on self-organizing map. In *IEEE International Conference on Neural Networks*, pages 631–636.

[Lecun et al., 2015] Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based Learning Applied to Document Recognition. In *Proceedings of the IEEE*.

[Lee et al., 2015] Lee, J. A., Peluffo-ordóñez, D. H., and Verleysen, M. (2015). Multi-scale similarities in stochastic neighbour embedding : Reducing dimensionality while preserving both local and global structure. *Neurocomputing*, 169:246–261. http://dx.doi.org/10.1016/j.neucom.2014.12.095.

[Lee and Verleysen, 2007] Lee, J. A. and Verleysen, M. (2007). *Nonlinear Dimensionality Reduction*. Springer.

[Lee and Verleysen, 2009] Lee, J. A. and Verleysen, M. (2009). Quality assessment of dimensionality reduction : Rank-based criteria. *Neurocomputing*, 72:1431–1443.

[Lee et al., 2020] Lee, N., Azarian, M. H., and Pecht, M. (2020). An Explainable Deep Learning-based Prognostic Model for Rotating Machinery.

[Levine and Domany, 2001] Levine, E. and Domany, E. (2001). Resampling method for unsupervised estimation of cluster validity. *Neural Computation*, 13(11):2573–2593.

[Lewis et al., 2004] Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. (2004). RCV1: A New Benchmark Collection for Text Categorization Research. *Journal of Machine Learning Research*, 5:361–397. http://dl.acm.org/citation.cfm?id=1005332.1005345.

[Li et al., 2017] Li, S., Yang, Y., Yang, L., Su, H., Zhang, G., and Wang, J. (2017). Civil Aircraft Big Data Platform. *International Conference on Semantic Computing (ICSC)*, pages 328–333. http://ieeexplore.ieee.org/document/7889557/.

[Lim and van der Schaar, 2018] Lim, B. and van der Schaar, M. (2018). DiseaseAtlas: Navigating Disease Trajectories with Deep Learning. http://arxiv.org/abs/1803.10254.

[Lin et al., 2007] Lin, J., Keogh, E., Wei, L., and Lonardi, S. (2007). Experiencing SAX: a Novel Symbolic Representation of Time Series. *Cs.Gmu.Edu*, 15:107–144.

[Lin et al., 2018] Lin, W.-a., Carlos, J.-c. C., and Rama, D. C. (2018). Deep Density Clustering of Unconstrained Faces. In *CVPR*.

[LIPN, 2018] LIPN, U. S. P. N. (2018). C4E Project. `https://github.com/Clustering4Ever/Clustering4Ever`.

[Liu et al., 2015] Liu, N., Wang, J., and Gong, Y. (2015). Deep Self-Organizing Map for visual classification. In *International Joint Conference on Neural Networks (IJCNN)*.

[Liu et al., 2020] Liu, Z., Cao, J., Chen, S., Lu, Y., and Tan, F. (2020). Visualization Analysis of Seismic Facies Based on Deep Embedded SOM. *IEEE Geoscience and Remote Sensing Letters*.

[Lloyd, 1982] Lloyd, S. P. (1982). Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137.

[Locatello et al., 2020] Locatello, F., Bauer, S., Lucic, M., Rätsch, G., Gelly, S., Schölkopf, B., and Bachem, O. (2020). A sober look at the unsupervised learning of disentangled representations and their evaluation. *Journal of Machine Learning Research*, 21.

[López-Rubio et al., 2004] López-Rubio, E., Muñoz-Pérez, J., and Gómez-Ruiz, J. A. (2004). A principal components analysis self-organizing map. *Neural Networks*, 17(2):261–270.

[Lv et al., 2016] Lv, F., Wen, C., Bao, Z., and Liu, M. (2016). Fault diagnosis based on deep learning. In *American Control Conference*, pages 6851–6856.

[Ma et al., 2019] Ma, Q., Zheng, J., Li, S., and Cottrell, G. W. (2019). Learning Representations for Time Series Clustering. In *NeurIPS*.

[MacQueen, 1967] MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. *Berkeley symposium on mathematical statistics and probability*, 1(14):281–297.

[Madaan and Maiti, 2019] Madaan, P. and Maiti, A. (2019). *Deep Mean Shift Clustering*. PhD thesis, Indraprastha Institute of Information Technology.

[Madiraju et al., 2018] Madiraju, N. S., Sadat, S. M., Fisher, D., and Karimabadi, H. (2018). Deep Temporal Clustering: Fully Unsupervised Learning of Time-Domain Features. `http://arxiv.org/abs/1802.01059`.

[Maharaj, 2000] Maharaj, E. A. (2000). Clusters of time series. *Journal of Classification*, 17(2):297–314.

[Makhzani and Frey, 2013] Makhzani, A. and Frey, B. (2013). k-Sparse Autoencoders. In *ICLR*. http://arxiv.org/abs/1312.5663.

[Makhzani et al., 2014] Makhzani, A., Frey, B., and Goodfellow, I. (2014). Adversarial Autoencoders.

[Manduchi et al., 2020] Manduchi, L., Hüser, M., Rätsch, G., and Fortuin, V. (2020). DPSOM: Deep Probabilistic Clustering with Self-Organizing Maps. http://arxiv.org/abs/1910.01590.

[Marr, 2015] Marr, B. (2015). That's Data Science: Airbus Puts 10,000 Sensors in Every Single Wing! https://www.datasciencecentral.com/profiles/blogs/that-s-data-science-airbus-puts-10-000-sensors-in-every-single.

[Marti et al., 2016] Marti, G., Very, P., Donnat, P., and Nielsen, F. (2016). A proposal of a methodological framework with experimental guidelines to investigate clustering stability on financial time series. In *International Conference on Machine Learning and Applications (ICMLA)*, pages 32–37.

[Martinetz and Schulten, 1991] Martinetz, T. and Schulten, K. (1991). A "Neural-Gas" Network Learns Topologies. http://web.cs.swarthmore.edu/$\sim$meeden/DevelopmentalRobotics/fritzke95.pdf.

[Martinetz and Schulten, 1994] Martinetz, T. and Schulten, K. (1994). Topology representing networks. *Neural Networks*, 7(3):507–522.

[Massoni et al., 2009] Massoni, S., Olteanu, M., and Rousset, P. (2009). Career-path analysis using optimal matching and self-organizing maps. *International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)*.

[Mathew and Sahely, 2019] Mathew, A. and Sahely, D. P. (2019). Warping Resilient Time Series Embeddings. In *ICML Time Series Workshop*.

[Maurus and Plant, 2016] Maurus, S. and Plant, C. (2016). Skinny-dip: Clustering in a Sea of Noise. In *KDD*.

[McConville et al., 2021] McConville, R., Santos-Rodriguez, R., Piechocki, R. J., and Craddock, I. (2021). N2D: (Not Too) Deep Clustering via Clustering the Local Manifold of an Autoencoded Embedding. In *International Conference on Pattern Recognition (ICPR)*. http://arxiv.org/abs/1908.05968.

[McInnes et al., 2018] McInnes, L., Healy, J., and Melville, J. (2018). UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. http://arxiv.org/abs/1802.03426.

[McLachlan and Peel, 2000] McLachlan, G. and Peel, D. (2000). Finite Mixture Models. http://doi.wiley.com/10.1002/0471721182.

[Medeiros et al., 2020] Medeiros, H. R., Braga, P. H. M., and Bassani, H. F. (2020). Deep Clustering Self-Organizing Maps with Relevance Learning. In *ICML LatinX in AI Research Workshop*.

[Meilă, 2018] Meilă, M. (2018). How to tell when a clustering is (approximately) correct using convex relaxations. In *NeurIPS*, pages 7407–7418.

[Melnik et al., 2010] Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M., and Vassilakis, T. (2010). Dremel: Interactive Analysis of Web-Scale Datasets. In *VLDB*, pages 330–339.

[Merkl and Rauber, 1997] Merkl, D. and Rauber, A. (1997). Alternative Ways for Cluster Visualization in Self-Organizing Maps. In *International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)*, pages 106–111.

[Midenet and Grumbach, 1990] Midenet, S. and Grumbach, A. (1990). Supervised Learning Based on Kohonen's Self-Organising Feature Maps. In *International Neural Network Conference*, pages 773–776, Dordrecht. Springer Netherlands. https://doi.org/10.1007/978-94-009-0643-3_72.

[Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. In *International Conference on Learning Representations (ICLR) Workshop*.

[Milligan and Cooper, 1986] Milligan, G. W. and Cooper, M. C. (1986). A Study of the Comparability of External Criteria for Hierarchical Cluster Analysis. *Multivariate Behavioral Research*, 21(4):441–458.

[Min et al., 2018] Min, E., Guo, X., Liu, Q., Zhang, G., Cui, J., and Long, J. (2018). A Survey of Clustering With Deep Learning : From the Perspective of Network Architecture. *IEEE Access*, 6:39501–39514.

[Mirza and Osindero, 2014] Mirza, M. and Osindero, S. (2014). Conditional Generative Adversarial Nets. http://arxiv.org/abs/1411.1784.

[Mishra et al., 2020] Mishra, S., Flaxman, S., and Bhatt, S. (2020). πVAE: Encoding stochastic process priors with variational autoencoders.

[Misra et al., 2016] Misra, I., Lawrence Zitnick, C., and Hebert, M. (2016). Shuffle and learn: Unsupervised learning using temporal order verification. In *ECCV*, pages 527–544.

[Modha and Spangler, 2002] Modha, D. S. and Spangler, W. S. (2002). Feature Weighting in k-Means Clustering. *Machine Learning*, 47.

[Möller and Radke, 2006] Möller, U. and Radke, D. (2006). A cluster validity approach based on nearest-neighbor resampling. *International Conference on Pattern Recognition (ICPR)*, pages 892–895.

[Monnier et al., 2020] Monnier, T., Groueix, T., and Aubry, M. (2020). Deep Transformation-Invariant Clustering. `http://arxiv.org/abs/2006.11132`.

[Moosavi et al., 2014] Moosavi, V., Packmann, S., and Vallés, I. (2014). SOMPY: A Python Library for Self Organizing Map (SOM). `https://github.com/sevamoo/SOMPY`.

[Morey and Agresti, 1984] Morey, L. C. and Agresti, A. (1984). The Measurement of Classification Agreement: An Adjustment of the Rand Statistic for Chance Agreement. *Educational and Psychological Measurement*, 44:33–37.

[Mourer et al., 2020] Mourer, A., Forest, F., Lebbah, M., Azzag, H., and Lacaille, J. (2020). Selecting the Number of Clusters K with a Stability Trade-off: an Internal Validation Criterion. `https://arxiv.org/abs/2006.08530`.

[Mrabah et al., 2020] Mrabah, N., Khan, N. M., Ksantini, R., and Lachiri, Z. (2020). Deep clustering with a Dynamic Autoencoder: From reconstruction towards centroids construction. *Neural Networks*, 130:206–228. `https://doi.org/10.1016/j.neunet.2020.07.005`.

[Mukherjee et al., 2019] Mukherjee, S., Asnani, H., Lin, E., and Kannan, S. (2019). ClusterGAN: Latent Space Clustering in Generative Adversarial Networks. *AAAI Conference on Artificial Intelligence*, 33:4610–4617. `https://aaai.org/ojs/index.php/AAAI/article/view/4385`.

[Munkres, 1957] Munkres, J. (1957). Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38.

[Murtagh, 1995] Murtagh, F. (1995). Interpreting the Kohonen self-organizing feature map using contiguity-constrained clustering. *Pattern Recognition Letters*, 16(4):399–408.

[Murugan et al., 2014] Murugan, A., Mylaraswamy, D., Xu, B., and Dietrich, P. (2014). Big Data Infrastructure for Aviation Data Analytics. *IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*.

[Neel, 2005] Neel, J. (2005). *Cluster analysis methods for speech recognition*. PhD thesis, KTH.

[Ng, 2011] Ng, A. (2011). Sparse autoencoder. Technical report, Stanford University. `https://web.stanford.edu/class/cs294a/sparseAutoencoder.pdf`.

[Ng and Han, 1994] Ng, R. T. and Han, J. (1994). Efficient and Effective Clustering Data Mining Methods for Spatial Data Mining. In *VLDB*, pages 144–155.

[Noroozi and Favaro, 2017] Noroozi, M. and Favaro, P. (2017). Unsupervised learning of visual representations by solving jigsaw puzzles.

[Oh, 2017] Oh, C.-G. (2017). Application of Big Data Systems To Aviation and Aerospace Fields ; Pertinent Human Factors Considerati .... In *International Symposium on Aviation Psychology*.

[Oja, 1982] Oja, E. (1982). A Simplified Neuron Model as a Principal Component Analyzer. *Journal of Mathematical Biology*, 15(3):267–273.

[Oja, 1992] Oja, E. (1992). Principal Components, Minor Components, and Linear Neural Networks. `https://users.ics.aalto.fi/oja/Oja92.pdf`.

[Olier et al., 2010] Olier, I., Vellido, A., and Giraldo, J. (2010). Kernel generative topographic mapping. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pages 481–486.

[Olshausen and Fieldt, 1997] Olshausen, B. A. and Fieldt, D. J. (1997). Sparse Coding with an Overcomplete Basis Set: A Strategy Employed by V1 ? *Vision Res*, 37(23):3311–3325.

[Olteanu et al., 2013] Olteanu, M., Villa-Vialaneix, N., and Cottrell, M. (2013). Online relational SOM for dissimilarity data. *Advances in Intelligent Systems and Computing*, 198:13–22.

[Opochinsky et al., 2020] Opochinsky, Y., Chazan, S. E., Gannot, S., and Goldberger, J. (2020). K-Autoencoders Deep Clustering. In *ICASSP*.

[Orsagh et al., 2003] Orsagh, R. F., Sheldon, J., and Klenke, C. J. (2003). Prognostics/diagnostics for Gas Turbine Engine Bearings. In *ASME Turbo Expo*.

[Pan et al., 2020] Pan, Q., Li, X., and Fang, L. (2020). Data Augmentation for Deep Learning-Based ECG Analysis. *Feature Engineering and Computational Intelligence in ECG Monitoring*.

[Paparrizos and Gravano, 2015] Paparrizos, J. and Gravano, L. (2015). k-Shape: Efficient and Accurate Clustering of Time Series. *ACM SIGMOD*, pages 1855–1870. `http://dl.acm.org/citation.cfm?id=2723372.2737793`.

[Parsons et al., 2004] Parsons, L., Haque, E., and Liu, H. (2004). Subspace clustering of high dimensional data. *SIGKDD Explorations*, 6(1):517–521.

[Patel et al., 2003] Patel, P., Keogh, E., Lin, J., and Lonardi, S. (2003). Mining motifs in massive time series databases.

[Patel et al., 2013] Patel, V. M., Nguyen, H. V., and Vidal, R. (2013). Latent space sparse subspace clustering. *ICCV*, pages 225–232.

[Pearson, 1901] Pearson, K. (1901). 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.

[Pelleg and Moore, 2000] Pelleg, D. and Moore, A. (2000). X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *International Conference on Machine Learning (ICML)*.

[Peng et al., 2005] Peng, Z. K., Chu, F. L., and Tse, P. W. (2005). Detection of the rubbing-caused impacts for rotor-stator fault diagnosis using reassigned scalogram. *Mechanical Systems and Signal Processing*, 19(2):391–409.

[Pesteie et al., 2018] Pesteie, M., Abolmaesumi, P., and Rohling, R. (2018). Deep Neural Maps. In *ICML workshop*. `http://arxiv.org/abs/1810.07291`.

[Petersohn, 1998] Petersohn, H. (1998). Assessment of cluster analysis and Self-Organizing Maps. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(2):139–149.

[Petitjean et al., 2011] Petitjean, F., Ketterlin, A., and Gançarski, P. (2011). A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition*, 44(3):678–693.

[Piccolo, 1990] Piccolo, D. (1990). A Distance Measure for Classifying ARIMA Models. *Journal of Time Series Analysis*, 11(2):153–164.

[Plaut, 2018] Plaut, E. (2018). From Principal Subspaces to Principal Components with Linear Autoencoders.

[Polzlbauer, 2004] Polzlbauer, G. (2004). Survey and comparison of quality measures for self-organizing maps. *Workshop on Data Analysis (WDA)*, pages 67–82.

[Qiu and Joe, 2006] Qiu, W. and Joe, H. (2006). Generation of random clusters with specified degree of separation. *Journal of Classification*, 23(2):315–334.

[Qu et al., 2019] Qu, Y., Zhang, Y., He, D., He, M., and Zhou, Z. (2019). A regularized deep clustering method for fault trend analysis. *Annual Conference of the PHM Society*, 11(1).

[Rand, 1971] Rand, W. M. (1971). Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association*, 66(336):846–850.

[Randall, 2004] Randall, R. B. (2004). State of the art in monitoring rotating machinery - Part 1. *Sound and Vibration*, pages 14–20.

[Randall, 2011] Randall, R. B. (2011). *Vibration-based condition monitoring*. Wiley.

[Ray and Turi, 1999] Ray, S. and Turi, R. (1999). Determination of number of clusters in k-means clustering and application in colour image segmentation. *International conference on advances in pattern recognition and digital techniques*, pages 137–143.

[Ressom et al., 2003] Ressom, H., Wang, D., and Natarajan, P. (2003). Adaptive double self-organizing maps for clustering gene expression profiles. *Neural Networks*, 16(5-6):633–640.

[Rezende et al., 2014] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *International Conference on Machine Learning (ICML)*, 4:3057–3070.

[Rifai et al., 2011] Rifai, S., Vincent, P., Muller, X., Glorot, X., and Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. *International Conference on Machine Learning (ICML)*, pages 833–840.

[Romano and Bailey, 2016] Romano, S. and Bailey, J. (2016). Adjusting for Chance Clustering Comparison Measures. *Journal of Machine Learning Research*, 17.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review*, pages 65–386.

[Roth et al., 2002] Roth, V., Lange, T., Braun, M., and Buhmann, J. (2002). A Resampling Approach to Cluster Validation. *Compstat*, pages 123–128.

[Rousseeuw, 1987] Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20(C):53–65.

[Safran, 2018] Safran (2018). Cassiopée. `https://www.cassiopee.aero/`.

[Sakoe and Chiba, 1978] Sakoe, H. and Chiba, S. (1978). Dynamic Programming Algorithm Optimization for Spoken Word Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49.

[Sarazin, 2018] Sarazin, T. (2018). *Massively distributed learning in a Big Data environment (Apprentissage massivement distribué dans un environnement Big Data)*. PhD thesis, Université Paris 13.

[Sarazin et al., 2014a] Sarazin, T., Azzag, H., and Lebbah, M. (2014a). SOM clustering using spark-MapReduce. In *International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*.

[Sarazin et al., 2014b] Sarazin, T., Lebbah, M., and Azzag, H. (2014b). Biclustering using Spark-MapReduce. *IEEE International Conference on Big Data*, pages 58–60.

[Schäfer, 2015] Schäfer, P. (2015). The BOSS is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29(6):1505–1530.

[Schäfer and Leser, 2016] Schäfer, P. and Leser, U. (2016). Multivariate Time Series Classification with WEASEL + MUSE. In *ACM*.

[Schulam and Arora, 2016] Schulam, P. and Arora, R. (2016). Disease Trajectory Maps. In *NIPS*. `http://arxiv.org/abs/1606.09184`.

[Schwartz, 1978] Schwartz, G. (1978). Estimating the dimension of a model. *The Annals of statistics*, 6(2):461–464.

[Scrucca and Raftery, 2015] Scrucca, L. and Raftery, A. E. (2015). Improved initialisation of model-based clustering using Gaussian hierarchical partitions. *Advances in Data Analysis and Classification*, 9(4):447–460.

[Sembiring et al., 2010] Sembiring, R. W., Mohamad Zain, J., and Embong, A. (2010). Clustering High Dimensional Data Using Subspace and Projected Clustering Algorithms. *International Journal of Computer Science and Information Technology*, 2(4):162–170.

[Shah and Koltun, 2018] Shah, S. A. and Koltun, V. (2018). Deep Continuous Clustering.

[Shaham et al., 2018] Shaham, U., Stanton, K., Li, H., Nadler, B., Basri, R., and Kluger, Y. (2018). SpectralNet: Spectral clustering using deep neural networks. In *International Conference on Learning Representations (ICLR)*.

[Shalev-Shwartz and Ben-David, 2013] Shalev-Shwartz, S. and Ben-David, S. (2013). *Understanding machine learning: From theory to algorithms*, volume 9781107057.

[Shlens, 2014] Shlens, J. (2014). A Tutorial on Principal Component Analysis. `http://arxiv.org/abs/1404.1100`.

[Shorten and Khoshgoftaar, 2019] Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1). `https://doi.org/10.1186/s40537-019-0197-0`.

[Smolkin and Ghosh, 2003] Smolkin, M. and Ghosh, D. (2003). Cluster Stability Scores for Microarray Data in Cancer Studies. *BMC bioinformatics*.

[Sønderby et al., 2016] Sønderby, C. K., Raiko, T., Maaløe, L., Sønderby, S. K., and Winther, O. (2016). Ladder variational autoencoders. In *NIPS*, pages 3745–3753.

[Song et al., 2014] Song, C., Huang, Y., Liu, F., Wang, Z., and Wang, L. (2014). Deep auto-encoder based clustering. *Intelligent Data Analysis*, 18(6).

[Strauss et al., 1973] Strauss, J. S., Bartko, J. J., and Carpenter, W. T. (1973). The use of clustering techniques for the classification of psychiatric patients. *British Journal of Psychiatry*, 122(570):531–540.

[Strehl and Ghosh, 2003] Strehl, A. and Ghosh, J. (2003). Cluster ensembles - A knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3(3):583–617.

[Su and Chang, 2001] Su, M. C. and Chang, H. T. (2001). A new model of self-organizing neural networks and its application in data projection. *IEEE Transactions on Neural Networks*, 12(1):153–158.

[Sun et al., 2012] Sun, W., Wang, J., and Fang, Y. (2012). Regularized k-means clustering of high-dimensional data and its asymptotic consistency. *Electronic Journal of Statistics*, 6(April 2011):148–167.

[Sutskever et al., 2013] Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. *International Conference on Machine Learning (ICML)*, pages 2176–2184.

[Svensen et al., 1997] Svensen, M., Bishop, C. M., and Williams, C. K. I. (1997). GTM: The Generative Topographic Mapping. *Neural Computation*.

[Tavenard, 2017] Tavenard, R. (2017). tslearn: A machine learning toolkit dedicated to time-series data. `https://github.com/rtavenar/tslearn`.

[Taşdemir, 2011] Taşdemir, K. (2011). Spectral clustering as an automated SOM segmentation tool. In *International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)*, pages 71–78.

[Taşdemir and Merényi, 2009] Taşdemir, K. and Merényi, E. (2009). Exploiting Data Topology in Visualization and Clustering of Self-Organizing Maps. *IEEE Transactions on Neural Networks*, 20(4):549–562.

[Taşdemir et al., 2011] Taşdemir, K., Milenov, P., and Tapsall, B. (2011). Topology-based hierarchical clustering of self-organizing maps. *IEEE Transactions on Neural Networks*, 22(3):474–485.

[TechAmerica Foundation, 2012] TechAmerica Foundation (2012). A Practical Guide To Transforming The Business of Government. Technical report, TechAmerica Foundation: Federal Big Data Commission.

[Tian et al., 2014] Tian, F., Gao, B., Cui, Q., Chen, E., and Liu, T. Y. (2014). Learning deep representations for graph clustering. In *AAAI Conference on Artificial Intelligence*, volume 2, pages 1293–1299.

[Tian et al., 2017] Tian, K., Zhou, S., and Guan, J. (2017). DeepCluster: A General Clustering Framework Based on Deep Learning. In *ECML-PKDD*.

[Tibshirani, 1996] Tibshirani, R. (1996). Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society Series B*, pages 267–288.

[Tibshirani and Walther, 2005] Tibshirani, R. and Walther, G. (2005). Cluster validation by prediction strength. *Journal of Computational and Graphical Statistics*, 14(3):511–528.

[Tibshirani et al., 2001] Tibshirani, R., Walther, G., and Hastie, T. (2001). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society Series B*, 63:411–423.

[Tino and Nabney, 2002] Tino, P. and Nabney, I. (2002). Hierarchical GTM: constructiong localized nonlinear projection manifolds in a principled way. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):639–656.

[Ullah et al., 2020] Ullah, A., Haydarov, K., Haq, I. U., Muhammad, K., Rho, S., Lee, M., and Baik, S. W. (2020). Deep learning assisted buildings energy consumption profiling using smart meter data. *Sensors*, 20(3).

[Ultsch, 1999] Ultsch, A. (1999). Data Mining and Knowledge Discovery with Emergent Self-Organizing Feature Maps for Multivariate Time Series.

[Ultsch, 2003a] Ultsch, A. (2003a). Maps for the visualization of high-dimensional data spaces. In *International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)*.

[Ultsch, 2003b] Ultsch, A. (2003b). U*-Matrix: a tool to visualize clusters in high dimensional data. Technical report, University of Marburg, Department of Computer Science.

[Ultsch, 2005] Ultsch, A. (2005). Clustering Wih SOM: U*C. In *International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)*, pages 75–82, Paris.

[van den Oord et al., 2017] van den Oord, A., Vinyals, O., and Kavukcuoglu, K. (2017). Neural Discrete Representation Learning. In *NIPS*. `http://arxiv.org/abs/1711.00937`.

[Van Der Maaten, 2009] Van Der Maaten, L. (2009). Learning a Parametric Embedding by Preserving Local Structure. In *AISTATS*, pages 384–391.

[Van Der Maaten and Hinton, 2008] Van Der Maaten, L. and Hinton, G. E. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605. `https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf`.

[Vellido et al., 1999] Vellido, A., Lisboa, P. J., and Meehan, K. (1999). Segmentation of the on-line shopping market using neural networks. *Expert Systems with Applications*, 17(4):303–314.

[Venna and Kaski, 2001] Venna, J. and Kaski, S. (2001). Neighborhood preservation in nonlinear projection methods: An experimental study. *Lecture Notes in Computer Science*, 2130.

[Verbeek et al., 2005] Verbeek, J., Vlassis, N., and Krose, B. (2005). Self-organizing mixture models. *Neurocomputing*, 63:99–123.

[Verleysen and François, 2005] Verleysen, M. and François, D. (2005). The Curse of Dimensionality in Data Mining. In *IWANN*. `http://www.springerlink.com/index/n65tna6vwt3b1pw6.pdf`.

[Vesanto, 1999] Vesanto, J. (1999). SOM-based data visualization methods. *Intelligent Data Analysis*, 3(2):111–126.

[Vesanto and Alhoniemi, 2000] Vesanto, J. and Alhoniemi, E. (2000). Clustering of the Self-Organizing Map. *IEEE Transactions on Neural Networks*, 11(3):586–600.

[Vesanto and Sulkava, 2002] Vesanto, J. and Sulkava, M. (2002). Distance matrix based clustering of the Self-Organizing Map. In *ICANN*, pages 951–956.

[Villmann et al., 1994] Villmann, T., Der, R., and Martinetz, T. (1994). A New Quantitative Measure of Topology Preservation in Kohonen's Feature Maps. In *IEEE International Conference on Neural Networks (ICNN)*, pages 645–648.

[Vincent et al., 2010] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research*, 11:3371–3408.

[Vinh et al., 2010] Vinh, N. X., Epps, J., and Bailey, J. (2010). Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11:2837–2854.

[Von Luxburg, 2007] Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416.

[Von Luxburg, 2009] Von Luxburg, U. (2009). Clustering stability: An overview. *Foundations and Trends in Machine Learning*, 2(3):129–168.

[Von Luxburg and Ben-David, 2005] Von Luxburg, U. and Ben-David, S. (2005). Towards a statistical theory of clustering. *Pascal workshop on statistics and optimization of clustering*, pages 20–26.

[von Luxburg et al., 2012] von Luxburg, U., Williamson, R. C., and Guyon, I. (2012). Clustering: Science or Art? *JMLR: Workshop and Conference Proceedings*, 27:6579.

[Wang et al., 2019] Wang, F., Wang, Q., Nie, F., Li, Z., Yu, W., and Wang, R. (2019). Unsupervised Linear Discriminant Analysis for Jointly Clustering and Subspace Learning. *IEEE Transactions on Knowledge and Data Engineering*, 4347(c).

[Wang, 2010] Wang, J. (2010). Consistent selection of the number of clusters via crossvalidation. *Biometrika*, 97(4):893–904.

[Wang and Jiang, 2018] Wang, J. and Jiang, J. (2018). An unsupervised deep learning framework via integrated optimization of representation learning and GMM-based modeling. In *Asian Conference of Computer Vision*. http://arxiv.org/abs/2009.05234.

[Wang et al., 2001] Wang, W. Q., Ismail, F., and Farid Golnaraghi, M. (2001). Assessment of gear damage monitoring techniques using vibration measurements. *Mechanical Systems and Signal Processing*, 15(5):905–922.

[Wang and Gupta, 2015] Wang, X. and Gupta, A. (2015). Unsupervised Learning of Visual Representations using Videos. In *ICCV*.

[Ward, 1963] Ward, J. H. (1963). Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 58(301):236–244. https://www.jstor.org/stable/2282967.

[Warren Liao, 2005] Warren Liao, T. (2005). Clustering of time series data - A survey. *Pattern Recognition*, 38(11):1857–1874.

[Witten and Tibshirani, 2010] Witten, D. M. and Tibshirani, R. (2010). A framework for feature selection in clustering. *Journal of the American Statistical Association*, 105(490):713–726.

[Xiao et al., 2017] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. http://arxiv.org/abs/1708.07747.

[Xie et al., 2016] Xie, J., Girshick, R., and Farhadi, A. (2016). Unsupervised Deep Embedding for Clustering Analysis. In *International Conference on Machine Learning (ICML)*, volume 48. http://arxiv.org/abs/1511.06335.

[Xie and Beni, 1991] Xie, X. L. and Beni, G. (1991). A validity measure for fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(8):841–847.

[Yan and Yu, 2019] Yan, W. and Yu, L. (2019). On accurate and reliable anomaly detection for gas turbine combustors: A deep learning approach. In *Annual Conference of the PHM Society*.

[Yang et al., 2017a] Yang, B., Fu, X., and Sidiropoulos, N. D. (2017a). Learning from Hidden Traits: Joint Factor Analysis and Latent Clustering. *IEEE Transactions on Signal Processing*, 65(1):256–269.

[Yang et al., 2017b] Yang, B., Fu, X., Sidiropoulos, N. D., and Hong, M. (2017b). Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering. In *International Conference on Machine Learning (ICML)*. `http://arxiv.org/abs/1610.04794`.

[Yang et al., 2003] Yang, H., Mathew, J., and Ma, L. (2003). Vibration Feature Extraction Techniques for Fault Diagnosis of Rotating Machinery : A Literature Survey. In *Asia- Pacific Vibration Conference*.

[Yang and Leskovec, 2011] Yang, J. and Leskovec, J. (2011). Patterns of Temporal Variation in Online Media. In *WSDM*.

[Yang et al., 2016] Yang, J., Parikh, D., and Batra, D. (2016). Joint Unsupervised Learning of Deep Representations and Image Clusters. `http://arxiv.org/abs/1604.03628`.

[Yang et al., 2020] Yang, X., Deng, C., Wei, K., Yan, J., and Liu, W. (2020). Adversarial Learning for Robust Deep Clustering. In *NeurIPS*.

[Yang et al., 2019] Yang, X., Deng, C., Zheng, F., Yan, J., and Liu, W. (2019). Deep spectral clustering using dual autoencoder network. In *CVPR*, pages 4061–4070.

[Ye, 2007] Ye, J. (2007). Discriminative K-means for Clustering. In *NIPS*.

[Yeung et al., 2001] Yeung, K. Y., Haynor, D. R., and Ruzzo, W. L. (2001). Validating clustering for gene expression data. *Bioinformatics*, 17(4):309–318.

[Yuan and Lin, 2006] Yuan, M. and Lin, Y. (2006). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society Series B*, 68(1):49–67.

[Zakaria et al., 2012] Zakaria, J., Mueen, A., and Keogh, E. (2012). Clustering time series using unsupervised-shapelets. In *International Conference on Data Mining (ICDM)*, pages 785–794.

[Zhang et al., 2017a] Zhang, D., Sun, Y., Eriksson, B., and Balzano, L. (2017a). Deep Unsupervised Clustering Using Mixture of Autoencoders. `http://arxiv.org/abs/1712.07788`.

[Zhang et al., 2018] Zhang, Q., Wu, J., Zhang, P., Long, G., and Zhang, C. (2018). Salient Subsequence Learning for Time Series Clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[Zhang et al., 2016] Zhang, R., Isola, P., and Efros, A. A. (2016). Colorful image colorization. In *ECCV*, pages 649–666.

[Zhang et al., 2017b] Zhang, W., Peng, G., Li, C., Chen, Y., and Zhang, Z. (2017b). A new deep learning model for fault diagnosis with good anti-noise and domain adaptation ability on raw vibration signals. *Sensors*, 17(2).

[Zhao et al., 2011] Zhao, Q., Xu, M., and Fränti, P. (2011). Extending external validity measures for determining the number of clusters. *International Conference on Intelligent Systems Design and Applications, ISDA*, pages 931–936.

[Zhao et al., 2019] Zhao, R., Yan, R., Chen, Z., Mao, K., Wang, P., and Gao, R. X. (2019). Deep learning and its applications to machine health monitoring. *Mechanical Systems and Signal Processing*, 115:213–237. `https://doi.org/10.1016/j.ymssp.2018.05.050`.

[Zhao et al., 2017] Zhao, S., Song, J., and Ermon, S. (2017). InfoVAE: Information Maximizing Variational Autoencoders. `http://arxiv.org/abs/1706.02262`.

[Zhao et al., 2009] Zhao, W., Ma, H., and He, Q. (2009). Parallel K-Means Clustering Based on MapReduce. In *CloudCom*.

[Zhou and Zhou, 2019] Zhou, W. A. and Zhou, Q. (2019). Deep Embedded Clustering With Adversarial Distribution Adaptation. *IEEE Access*, 7:113801–113809.

[Zhou et al., 2015] Zhou, Y., Arpit, D., Nwogu, I., and Govindaraju, V. (2015). Is Joint Training Better for Deep Auto-Encoders ?

[Zhu et al., 2019] Zhu, D., Han, T., Zhou, L., Yang, X., and Wu, Y. N. (2019). Deep unsupervised clustering with clustered generator model. `http://arxiv.org/abs/1911.08459`.