Online machine learning-based predictive maintenance for the railway industry

Minh Huong Le Nguyen

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2023IPPAT027

Thèse de doctorat

Online machine learning-based predictive maintenance for the railway industry

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (ED IP PARIS)
Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Palaiseau, le 30 juin 2023, par

MINH HUONG LE NGUYEN

Composition du Jury :

Slawomir NOWACZYK Professeur, Halmstad University	Président et Rapporteur
Vincent GUIGUE Professeur, AgroParisTech	Rapporteur
Ioana MANOLESCU Professeur, INRIA Saclay	Examinatrice
Rita Paula RIBEIRO Maître de conférence, University of Porto	Examinatrice
Mihaela MITICI Maître de conférence, Utrecht University	Examinatrice
Albert BIFET Professeur, Télécom Paris	Directeur de thèse
Fabien TURGIS Docteur, IKOS Consulting	Invité

I dedicate this thesis to my dear family - my mother, father, little sister, and to my beloved partner, for their warm, unceasing, and unconditional support that has helped me through the ups and downs of this journey.

Acknowledgements

First and foremost, I express my most sincere gratitude to my thesis director, **Albert Bifet** from Télécom Paris, and to my co-supervisors, **Fabien Turgis** and **Pierre-Emmanuel Fayemi** from IKOS Consulting, for their support and mentorship throughout the four years of this doctoral thesis. I had worked with Albert as a research intern prior to this thesis; his kindness, his vast understanding on the topic of data stream mining, and the freedom he gave me to do my research greatly motivated me to pursue a doctoral thesis under his guidance. Then, I had the chance to work with Fabien and Pierre-Emmanuel, from whom I have learned a lot about the sophisticated world of railway, and whose passion for their profession has taught me to put my heart and soul into what I love most. The pandemic COVID-19 undoubtedly complicated the matters at hand, with meetings scheduled at either 8 AM or 9 PM to accommodate the timezone of everyone, but finally we have pushed through it together. To me, Albert, Fabien, and Pierre-Emmanuel are an incredible source of inspiration and have shaped me into a researcher as I now am.

It is my honor to welcome five outstanding researchers to my defense committee. In particular, I very much appreciate **Vincent Guigue** from AgroParisTech and **Slawomir Nowaczyk** from Halmstad University, for their time and effort in reviewing this thesis. I thank **Ioana Manolescu** from INRIA Saclay, **Rita Ribeiro** from University of Porto, and **Mihaela Mitici** from Utrecht University, for being part of the defense committee and for sharing their insights and expertise.

I would like to thank **IKOS Consulting** and **IKOS Lab** for supporting this thesis and for providing the necessary equipment to conduct the research works - without which it would have been difficult for my works to come to fruition. I would also like to acknowledge the **Société nationale des chemins de fer français** (SNCF) for the data that they have generously shared with me to help validating my methods. Finally, I thank the **Association Nationale de la Recherche et de la Technologie** (ANRT) of France for the funding that allows this thesis to be a harmonious academic-industrial collaboration.

I have greatly enjoyed my time with the **DIG team** at Télécom Paris. Although the in-and-out COVID-19 lockdowns delayed my arrival to the team until March 2022, I am glad I was able to be there after all and to meet other DIG-ers. Thank you for having been a part of my Ph.D. life. I will always treasure the moments we shared together - the team building event in June 2022, the coffee breaks, and the seminar (free!) lunches. I wish you all the best to pursue the path that you have chosen.

The company of my dear friends has been an immense emotional support to me during this long, sometimes solitary, and often stressful journey. Thank you, **Dionysia**, **Nga**, **Armita**, and **Pawel**, for keeping me company, for the carefree banters, and for being my friends.

Last but not least, I am forever grateful to my loving family - my **mother**, **father**, **little sister**, and my dearly beloved **partner**, for having believed in me. Their encouragement and support have helped me through the highest ups and lowest downs of these four years. Thank you all for your love, care, and patience, without which I could not have gotten to the end of this journey.

TÉLÉCOM PARIS

Abstract

Online machine learning-based predictive maintenance for the railway industry

by Minh Huong Le Nguyen

Being an effective long-distance mass transit, the railway will continue to flourish for its limited carbon footprint in the environment. Ensuring the equipment's reliability and passenger safety brings forth the need for efficient maintenance. Apart from the prevalence of corrective and periodic maintenance, predictive maintenance has come into prominence lately. Recent advances in machine learning and the abundance of data drive practitioners to data-driven predictive maintenance. The common practice is to collect data to train a machine learning model, then deploy the model for production and keep it unchanged afterward. We argue that such practice is suboptimal on a data stream. The unboundedness of the stream makes the model prone to incomplete learning. Dynamic changes on the stream introduce novel concepts unseen by the model and decrease its accuracy. The velocity of the stream makes manual labeling infeasible and disables supervised learning algorithms. Therefore, switching from a static, offline learning paradigm to an adaptive, online one is necessary, especially when new generations of connected trains continuously generating sensor data have already been a reality. We investigate the applicability of online machine learning for predictive maintenance on typical complex systems in the railway. First, we develop InterCE as an active learning-based framework that extracts cycles from an unlabeled stream by interacting with a human expert. Then, we implement a long short-term memory autoencoder to transform the extracted cycles into feature vectors that are more compact yet remain representative. Finally, we design CheMoc as a framework that continuously monitors the condition of the systems using online adaptive clustering. Our methods are evaluated on the passenger access systems on two fleets of passenger trains managed by the national railway company SNCF of France.

TÉLÉCOM PARIS

Résumé

Maintenance prévisionnelle basée sur l'apprentissage automatique en ligne dans le secteur ferroviaire

par Minh Huong Le Nguyen

Le transport ferroviaire permet de transporter de nombreux passagers tout en laissant une faible empreinte carbone dans l'environnement, ce qui en fait un mode de transport en commun efficace et écologique. Donc, le chemin de fer continuera à se développer dans les décennies suivantes, d'où le besoin d'une maintenance efficace. La maintenance est une fonction support essentielle car elle aide à assurer la fiabilité des équipements, la disponibilité du service, et la sécurité des humains. Dans le ferroviaire, la maintenance corrective et la maintenance préventive périodique sont dominantes. Le premier vise à mettre en place les interventions urgentes ayant lieu après l'occurrence d'une panne dans le système, alors que le dernier a pour l'objectif de planifier des inspections par intervalles régulières pour prévenir toutes défaillances potentielles.

Récemment, émerge une nouvelle stratégie de maintenance qui est la maintenance prévisionnelle. Cette stratégie émet des ordre de maintenance en s'appuyant sur l'état actuel d'un système, révélé par la surveillance en continu de l'état du système, et sur la prévision de ses conditions futures. La maintenance prévisionnelle a attiré une attention croissante des praticiens du domaine, en particulier dans l'ère de l'Industrie 4.0 où les systèmes sont équipés de capteurs qui génèrent un flux de données en temps réel, facilitant la surveillance en continue de l'état des systèmes. Les techniques réalisant la maintenance prévisionnelle peuvent se diviser en deux catégories: l'approche basée sur les connaissances et l'approche basée sur les données. Pour cette thèse, nous nous concentrons sur le dernier, pour lequel l'apprentissage automatique (machine learning) a pris de l'importance.

Pourtant, l'apprentissage automtique se fait souvent de manière hors-ligne. C'est-à-dire, un modèle s'apprend sur un ensemble de données collectées préalablement en plusieurs itérations jusqu'à sa convergence, puis le modèle est deployé en ligne pour émettre ses prédictions sur les nouvelles données. En dépit des nouveautés sur un flux de données, le modèle déployé restant constant et ses paramètres ne changent pas, même si les caractéristiques des données ont dévié largement de celles utilisées pendant l'apprentissage du modèle. Ceci demande un ré-entraînement du modèle, ce qui est inefficace, considéran que les systèmes connectés produisent de nouvelles données rapidement.

Considérant ces désavantages de l'apprentissage automatique hors-ligne, nous portons l'attention sur l'apprentissage automatique en ligne, qui consiste à apprendre de façon continue, en mettant à jour le modèle incrémentalement sur chaque nouvel exemple de données, et à s'adapter automatiquement aux nouveautés sur le flux de données. Par résultat, l'apprentissage automatique en ligne permet aussi à un modèle d'interagir avec les humains en collectant leur retour pour ajuster ses paramètres si nécessaire. Cette thèse étudie l'applicabilité de l'apprentissage automatique en ligne pour la maintenance prévisionnelle dans le ferroviaire, utilisant les données des systèmes d'accès passager sur deux flottes de trains NAT et R2N, fournies par la SNCF de la France, comme cas d'études.

Donc, la question de recherche de cette thèse est comme suit : étant donné que l'apprentissage automatique en ligne peut surmonter des limites de l'apprentissage automatique hors ligne traditionnel,

pourrions-nous utiliser l'apprentissage automatique en ligne pour atteindre les résultats satisfaisant pour la maintenance prévisionnelle dans le ferroviaire ? Afin d'implémenter l'apprentissage automatique en ligne pour la maintenance prévisionnelle dans le ferroviaire, il faut considérer les contraintes opérationnelles et caractéristiques spécifiques aux ferroviaire. Nous les divisons en quatre blocs: granularité, cyclicité, indicateurs, et santé, pour lesquels nous proposons dix hypothèses. La validation de ces hypothèses est basée sur les résultats expérimentaux collectés sur les données des systèmes d'accès passager de deux flottes de trains NAT et R2N. À présent, nous supposons une analyse au niveau de la flotte pour répondre à l'hypothèse de la granularité. Nous avons conçu une méthode pour les blocs restant : cyclicité, indicateurs, santé.

Pour entamer les hypothèses de cyclicité, nous avons proposé InterCE (Interactive Cycle Extraction) pour automatiser l'extraction des cycles à partir d'un flux de donnés, en appliquant le principe de l'apprentissage actif pour apprendre à extraire des cycles au fur et à mesure. InterCE s'appuie sur la communication asynchronisée pour pouvoir traiter de nouveaux fichiers binaires tout en attendant les réponses des experts sans blocage. Les résultats montrent que InterCE atteint une précision supérieure à celle d'un système expert, ce qui valide le fait qu'un algorithme d'apprentissage est utilise pour améliorer la performance de base des humains. Pourtant, la réactivité de InterCE n'est pas supérieure à sa version hors-ligne en terme du temps à convergence, mais InterCE reste compétitif.

Pour entamer les hypothèses du bloc indicateurs, nous avons implémenté le modèle LSTM-AE (Long short-term memory autoencoder) pour apprendre des indicateurs de façon non-supervisée à partir des cycles détectés. Non seulement à reconstruire un cycle, nous cherchons aussi à apprendre au modèle à classer le contexte d'un cycle, pour but de rendre le modèle plus robuste contre les bruits contextuels. Cela a abouti à deux versions du LSTM-AE: l'une avec seulement le décodeur, et l'autre avec un classeur partageant le même encodeur avec le décodeur. Les résultats montrent que le LSTM-AE hors-ligne produit la meilleure reconstruction, tandis que la version en ligne aboutit à une reconstruction encore fautive et nécessite un temps d'entrainement plus long que la version hors-ligne. Toutefois, le LSTM-AE surpasse le système expert pour sa capacité de préserver les informations après l'encodage des cycles, ce qui confirme que le LSTM-AE apprend des indicateurs qui sont plus fidèles aux cycles d'origine que les indicateurs identifiés manuellement par le système expert.

Pour entamer les hypothèses de santé, nous avons développé CheMoc (Continuous Health Monitoring using Online Clustering) qui découvre les profils de santé sur le flux de données et qui calcule un score de santé adaptatif pour chaque système. Nous avons utilisé DenStream comme l'algorithme de clustering central de CheMoc pour détecter et maintenir les profils de santé sous forme de clusters évoluant. Nous avons apporté quelques ajustements à DenStream pour l'aligner avec les contraintes opérationnelles dans le ferroviaire. Nous avons aussi proposé des formules pour calculer, à n'importe quel moment donné, le degré d'anomalie d'un cluster, le score d'anomalie liée à un profil de santé spécifique d'un système, et le score de santé d'un système. Les résultats expérimentaux montrent que CheMoc est capable de capter les profils de santé pertinents de la flotte, qui sont vérifiés et confirmés par un expert du domaine. Pour mesure la réactivité du clustering en ligne contre le clustering hors-ligne, nous avons comparé la vitesse de convergence de CheMoc contre celle de DBSCAN. Un algorithme converge s'il produit des clusters de bonne qualité, mesurée par les indices de validité de clusters (Davies-Bouldin et Xie-Beni). Les résultats montrent que CheMoc et DBSCAN sont compétitifs en réactivité et ni l'un ni l'autre se montre gagnant décidément.

Contents

List of Figures

List of Tables

List of Abbreviations

PdM	**Pre**d**i**ctive **m**aintenance
ML	**M**achine **l**earning
OML	**O**nline **m**achine **l**earning
InterCE	**Inter**active **C**ycle **E**xtraction
LSTM-AE	**L**ong **s**hort-**t**erm **m**emory **a**uto**e**ncoder
CheMoc	**C**ontinuous **He**alth **M**onitoring using **O**nline **C**lustering
PAS	**P**assenger **a**ccess **s**ystem
RUL	**R**emaining **u**seful **l**ife

List of Symbols

	General notation
$D(T)$	A data stream recorded until the time T
$\mathcal{S}$	A fleet of systems
S_m	A system in the fleet $\mathcal{S}$
	Notation related to cycle extraction (InterCE)
$C_{S_m}^T$	A cycle produced by a system S_m at the time T
D	Number of variables in a cycle
K	Number of timesteps in a cycle
$\mathbf{a}^{(k)} \in \mathbb{R}^D$	A vector recording D real-valued variables in one timestep
	Notation related to feature learning (LSTM-AE)
$X_{S_m}^T$	A feature vector extracted from a cycle $C_{S_m}^T$
P	The number of variables in a feature vector
	Notation related to health detection (CheMoc)
$\mathcal{G}(T)$	The set of clusters maintained on the stream until the time T
$G_k(T)$	A cluster in $\mathcal{G}(T)$
$\omega_k(T)$	The anomaly degree of a cluster $G_k(T)$
$A_k^{S_m}(T)$	The anomaly score of an anomaly type G_k of a system S_m computed at T
$H_{S_m}(T)$	The health score of a system S_m computed at T

Chapter 1

Introduction

Contents

1 Motivation

Railway enables mass transit on long distance and alleviates the burden of traffic during rush hours. Being the most decarbonized terrestrial mass transportation, railway will undoubtedly continue to grow, bringing forth the need for efficient maintenance. Because rail transport interacts directly with humans (passengers, drivers, technicians), maintenance is crucial to ensure equipment reliability, service availability, and human safety. For railway maintenance, corrective maintenance and preventive maintenance are prevalent. The former fixes a failure after it has happened and incurs expensive repair costs, discontinued service, or even fatal consequences; the latter regularly performs systematic inspections to reduce the frequency of failures at the expense of higher inspection costs (Figure 1.1).

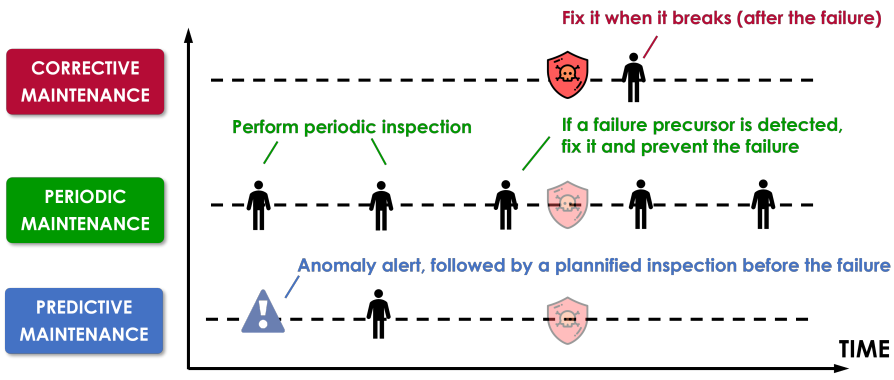


Figure 1.1: Maintenance strategies: corrective maintenance, time-based preventive maintenance, predictive maintenance

Recently, a novel strategy emerges: *predictive maintenance* [256]. Predictive maintenance monitors the target systems to predict critical failures, enabling near-corrective maintenance before a failure occurs. Predictive maintenance also monitors functional degradation and equipment maladjustment

to optimize preventive maintenance. Yet, it does not mean to replace the other two strategies; rather, predictive maintenance enhances them to optimize maintenance management as a whole.

We focus on data-driven predictive maintenance for which *machine learning* has become a major player [54]. Machine learning encompasses statistical algorithms that "teach", or "train", a model to perform an intended task given a set of relevant observations. Training a machine learning model is to find the most appropriate model parameters by optimizing an objective function on the observations to reach the desired goal, for instance, to classify whether an equipment is normal or faulty with a satisfactory accuracy.

To apply machine learning to predictive maintenance, the common practice is to collect data, such as sensor signals, maintenance logs, failure alerts, on which a suitable machine learning model is trained. After training, the model predicts failures on unseen data. Its parameters remain constant afterwards, until the model is retrained to learn new parameters from scratch. This is the *offline* approach, because the model is trained on static data and does not account for new samples during its training. Although this approach works for many predictive maintenance applications [48], we argue that it is suboptimal.

On the one hand, sensorized systems in operation never stop producing data, thus forming an infinite data stream. Using the offline approach, we must sample data from the stream, but it is unlikely that such sample represents the stream adequately. If the sample mainly contains data generated by systems operating in a normal[1] state, a model trained on these ideal data will not recognize those generated by degraded systems and will issue inaccurate predictions. This is known as *data drifting* [75]. Consequently, the model must be retrained once the incoming data deviate largely from the sample used for training. Model retraining can be done on only new data examples (old knowledge that was learned is forgotten as a result), or on a new training set that accumulates both old and new data. However, the training set will become excessively large over time, due to the unboundedness of the data stream.

On the other hand, streaming data challenge the creation of *labels*. For a machine learning task, a label is the value of a desired output and is assigned individually to each sample. Labeling a data stream is infeasible, as the human annotators cannot cope with the speed and volume of the stream. Also, in the railway, each occurrence of a failure is followed by the FRACAS[2] procedure to prevent it from re-occurring, and this failure label will not be seen again from the stream. It hinders the use of supervised learning techniques that require otherwise a vast amount of labeled data.

Because sensorized systems producing data infinitely have already become a reality on new generations of connected trains, the offline approach will not keep up with this new horizon. We turn our attention to *online machine learning* to enable incremental model update, adaptation to data novelties, querying for unknown phenomena and learning from feedback.

Therefore, we investigate the applicability of online machine learning for railway predictive maintenance in this thesis, using the passenger access systems on two fleets of passenger trains as study cases. The data sets used to design and evaluate our methods are supplied by the French national railway company SNCF[3].

Prior to our research work, an expert system for predictive maintenance was developed and deployed on the passenger access systems [227–229, 233]. An expert system is a program hard-coded by

[1] By normal, we mean the state in which a system functions correctly under expected conditions.
[2] Failure reporting, analysis and corrective action system.
[3] Société nationale des chemins de fer français.

a domain expert based on their extensive domain understanding to simulate and automate an expert's judgment, but there are several shortcomings.

- Crafting an expert systems demands time, varying from two to six months depending on the complexity of the systems.
- An expert system does not learn. Its coverage is limited by the human knowledge. It may miss behaviors unexpected to the humans yet observable from the data.
- An expert system uses static rules that do not detect nor adapt to changes from the data stream. For instance, a system might undergo a functional upgrade which modifies the behaviors encoded in the expert system. This requires modifying or rewriting the entire program.

Offline machine learning can improve an expert system because it can generalize on other systems and can detect hidden patterns from the data, but it is also non-adaptive: data drifting obliges a model to be retrained, which incurs computational cost, model validation effort, and interrupted usage during retraining. Meanwhile, online machine learning can amend these issues.

- Online machine learning is incrementally updatable on new data, enabling lifelong learning and overcoming the training bottleneck.
- Via incremental learning, online machine learning can adapt to novelties from the stream, which otherwise will fail an offline model that do not see these novelties during its training.
- Online machine learning implements lightweight models that perform on-the-fly update using moderate computational resources, adding a bonus on efficiency. Note that this may come with a trade-off of approximate instead of exact results.

Given the advantages of online machine learning, we propose novel methods that address the requirement of predictive maintenance on typical railway complex systems and evaluate them against offline models. The results obtained with this thesis will serve as a baseline and proof-of-concept of the potentials of online machine learning for railway predictive maintenance.

The research question we study in this thesis is the following:

> Given that online machine learning can overcome certain limits of traditional machine learning, could we use online machine learning to achieve satisfactory results for railway predictive maintenance?

Because the railway in itself is a complex system and entails many operational constraints, we formulate the hypotheses that address the research question while respecting such constraints.

2 Contributions

Given the research question, the works conducted during this thesis implement each stage in a predictive maintenance solution using online machine learning. We summarize and schematize our contributions in Figure 1.2. Given an input stream of raw sensor data $D(T)$ from a fleet of M systems, we craft four modules to process the stream to produce predictive maintenance alerts.

First, we propose **Interactive Cycle Extraction** (InterCE) as an active learning-based framework to automate the extraction of *cycles*, which are repeating patterns, from raw sensor data (Chapter 4).

- InterCE leverages active learning to query for human feedback on inputs from which it does not know how to extract cycles.

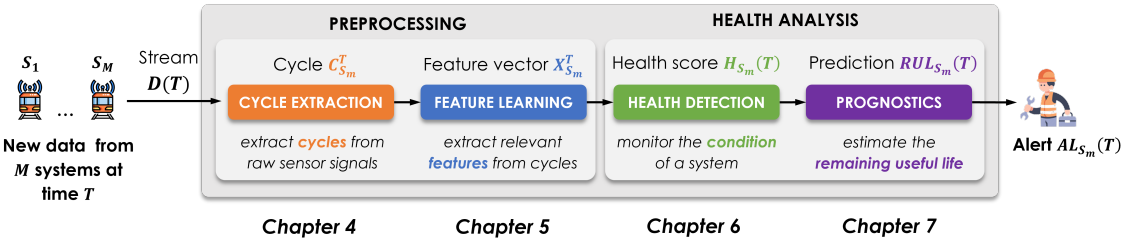


FIGURE 1.2: Our contributions in the form of a pipeline of four modules, each tackling one stage in a predictive maintenance solution

- The human feedback is used to update InterCE's understanding of the cycles and to improve its extraction capacity.
- We use an ensemble of extractors in InterCE, one based on the idleness/activeness of data signals and one based on a neural network.
- We implement a heuristical, motif matching-based mechanism for querying strategy.

Secondly, we propose a **long short-term memory autoencoder** (LSTM-AE), jointly trained with a classifier, to learn relevant features from the extracted cycles (Chapter 5).

- We implement, in the LSTM-AE, a decoder that learns to reconstruct a cycle, and a classifier that learns to map the cycle to its own contextual information, in order to learn features that are robust to contextual noises.
- We devise three training settings for the LSTM-AE: offline, online on each example, and online incremental on mini-batches.

Thirdly, we propose **Continuous Health Monitoring using Online Clustering** (CheMoc) as a framework that uses online clustering to estimate the evolution of the systems' condition (Chapter 6).

- We use DenStream [44] as the core clustering of CheMoc to capture evolving clusters that represent the set of possible conditions of the monitored systems.
- We make important modifications to DenStream to align it to the railway operational constraints: adaptive density threshold, cluster features per system, pruning and offline clustering omitted.
- We devise formula to assess the condition of a system, computable at any given moment.

3 Publications

The publications we have made in conferences include:

- Minh Huong Le Nguyen, Fabien Turgis, Pierre-Emmanuel Fayemi, and Albert Bifet. "Challenges of Stream Learning for Predictive Maintenance in the Railway Sector". In: *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning*. Communications in Computer and Information Science. Springer International Publishing, 2020, pp. 14–29. ISBN: 978-3-030-66770-2. DOI: 10.1007/978-3-030-66770-2_2,
- Minh Huong Le Nguyen, Fabien Turgis, Pierre-Emmanuel Fayemi, and Albert Bifet. "A Complete Streaming Pipeline for Real-time Monitoring and Predictive Maintenance". In: *Proceedings of the 31st European Safety and Reliability Conference*. 2021, p. 2119. DOI: 10.3850/978-981-18-2016-8_400-cd,

- Minh-Huong Le-Nguyen, Fabien Turgis, Pierre-Emmanuel Fayemi, and Albert Bifet. "Real-time learning for real-time data: online machine learning for predictive maintenance of railway systems". In: *Transport Research Arena (TRA)*. Lisbon, Portugal, Nov. 2022,
- Minh-Huong Le-Nguyen, Fabien Turgis, Pierre-Emmanuel Fayemi, and Albert Bifet. "Continuous Health Monitoring of Machinery using Online Clustering on Unlabeled Data Streams". In: *2022 IEEE International Conference on Big Data (Big Data)*. Dec. 2022, pp. 1866–1873. DOI: 10.1109/BigData55660.2022.10021002,
- *Minh-Huong Le-Nguyen, Fabien Turgis, Pierre-Emmanuel Fayemi, and Albert Bifet. "Exploring the potentials of online machine learning for predictive maintenance: A case study in the railway industry". In: Applied Intelligence (Springer) (under review).*

The following publications are not included in this thesis but are contributed to by the candidate:

- Heitor Murilo Gomes, Maciej Grzenda, Rodrigo Mello, Jesse Read, Minh Huong Le Nguyen, and Albert Bifet. "A Survey on Semi-supervised Learning for Delayed Partially Labelled Data Streams". In: *ACM Computing Surveys* 55.4 (Nov. 2022), 75:1–75:42. ISSN: 0360-0300. DOI: 10.1145/3523055. URL: https://dl.acm.org/doi/10.1145/3523055,
- Jacob Montiel, Hoang-Anh Ngo, Minh-Huong Le-Nguyen, and Albert Bifet. "Online Clustering: Algorithms, Evaluation, Metrics, Applications and Benchmarking". In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. KDD '22. New York, NY, USA: Association for Computing Machinery, Aug. 2022, pp. 4808–4809. ISBN: 978-1-4503-9385-0. DOI: 10.1145/3534678.3542600. URL: https://doi.org/10.1145/3534678.3542600.

4 Outline

This doctoral thesis is organized as follows.

Chapter 2 (Literature review) studies the related literature on predictive maintenance, online machine learning, the combination of both, from which we position our work on the current direction of research on online machine learning for predictive maintenance.

Chapter 3 (Hypotheses) formulates the hypotheses to approach the research question. It also provides a description of the passenger access systems as the main study case of this thesis.

Chapter 4 (Cycle extraction) presents the framework InterCE that implements a solution to detect and identify cycles automatically from a stream of raw sensor data.

Chapter 5 (Feature learning) presents the LSTM-AE that learns to determine relevant statistical features from the extracted cycles and produces feature vectors that serve as the ingredients to unveil the underlying condition of the systems.

Chapter 6 (Health detection) presents the framework CheMoc that assesses the condition of the monitored systems via a set of continuously evolving clusters discovered from the stream of feature vectors. Detecting a system's current condition is the baseline allowing us to project its condition in the future and to prepare the ground for failure prediction.

Chapter 7 (Prognostics) sketches the idea we have established to implement a framework for estimating the remaining useful life of a system to achieve failure prediction. The framework has not been finalized at the end of the thesis and will be developed in future works.

Chapter 8 (Conclusion and Perspectives) concludes the works carried out during this thesis, reviews the proposed methods and their shortcomings, and envisions future improvements of our methods.

Chapter 2

Literature review

Contents

Summary

We review the literature landscape of predictive maintenance in the railway, separated into four aspects: the fundamental notions related to maintenance research, the literature body on predictive maintenance, the industrial standards for maintenance operations, and the current progress on online machine learning. We discover that a majority of previous works on data-driven predictive maintenance focus on training an intricate model offline then deploying it for online failure prediction, bu very few works use online machine learning directly to train and test the models on a data stream.

In this chapter, we review works that are related to our research topic on online machine learning for railway predictive maintenance. Firstly, we introduce the foundational concepts underlying research on machinery maintenance of complex systems (Section 1). Secondly, we analyze the literature body on predictive maintenance to position our work with respect to the current progress of the research community (Section 2). Thirdly, we present several industrial standards related to maintenance operations and emphasize on OSA-CBM, a de-facto standard for condition-based/predictive maintenance (Section 3). Finally, we describe online machine learning on data streams, which is an emerging learning paradigm extending the traditional offline learning (Section 4). The literature study allows us to grasp the current progress of online machine learning applied to railway predictive maintenance, based on which we can propose methodologies that bridge the gap in the literature.

1 Foundational concepts of maintenance

This section introduces foundational concepts making up the basis of maintenance research in complex systems: the definition of *complex systems* (Section 1.1), *reliability theory* (Section 1.2), and *maintenance strategies* (Section 1.3).

1.1 Complex systems

A *system* is an inextricable whole made of interconnected parts that satisfy the following requirements: it is a collection of at least two parts, there are interactions between the parts, and the collection contains no independent subgroups of parts, that is, each part has an inseparable role in the collection. Systems are omnipresent in multiple domains. In computer science, a software, hardware, or platform is a system. In mechanics, a steam engine or a motor is a system designed to generate power. In astronomy, the Solar system is formed by planets that orbit the Sun via gravitational force and function in perfect harmony since billion years ago.

Complexity has various definitions depending on the discipline in which it is studied. Complexity can be measured by size, entropy, degree of hierarchy, computational capacity, or algorithmic information content [160]. The term "complexity" is used to characterize the behavior of a collection of parts that are entwined, and the collective behavior from individual parts is unpredictable.

A complex system is a *system* that exhibits *complexity* and is formally defined as [160]:

Definition 1.1 (Complex system)**.** *A complex system is a system in which large network of components with no central control and simple rules of operation give rise to complex collective behavior, sophisticated information processing, and adaptation via learning or evolution.*

Examples of complex systems include but not limited to physical systems studied in condensed matters, ecosystems and biological evolution, human societies, economics and markets, pattern formation and collective motion [166]. Transportation systems, such as railway, airlines, or highways, are also complex systems, although the system does not evolve by itself but via maintenance and/or upgrade carried out by human technicians.

In our work, the definition of complexity is based on the railway transportation network. The railway network is a complex system and is composed of other complex subsystems: signaling, infrastructure, energy, maintenance, and the rolling stock (Figure 2.1).

- Trains moving on fixed rails are prone to collision which may cause catastrophic consequences. Railway **signaling** controls and collaborates the trains to ensure the safety and fluidity of traffic.

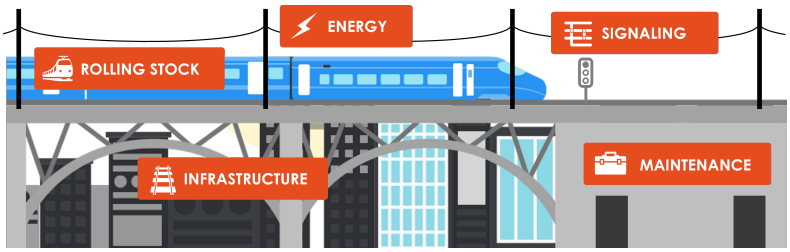


FIGURE 2.1: Components of a railway network

- The **infrastructure** is the skeleton that supports railway operation and comprises tracks (monorail, magnetic tracks, overhead tracks), rails (bolted rails, continuous welded rails), sleepers (wood, steel, concrete), ballast (stone, gravel), and switches. The switches are indispensable for signaling because they redirect and guide the trains from one track to another at junction points while maintaining continuous movement of the trains.
- A rolling stock draws power from a source of **energy** to operate. Nowadays, most locomotives use fossil fuel (diesel trains) or electricity (electric trains). The appropriate infrastructure is installed according to the type of the energy source. For example, electric trains draw from a third rail or from the caternary.
- **Maintenance** aims to prevent equipment failures and to maximize productivity. It plays a crucial role to ensure the availability, reliability, and safety of the transportation services.
- The **rolling stock** refers to any vehicles that run on the rails, such as locomotive, passenger cars, freight cars. A train is a sequence of connected rolling stocks. A train itself is a complex (sub)system of the rail network because it is composed of multiple interacting parts allowing it to move (wheels, bogies, pantographs), to stop (brakes), to ensure passenger safety (doors, windows). This thesis focuses on passenger trains. Because the passengers interact with a train directly, the rolling stock is a critical system and timely maintenance is required to guarantee its reliability. Figure 2.2 illustrates the components of an electric rolling stock.

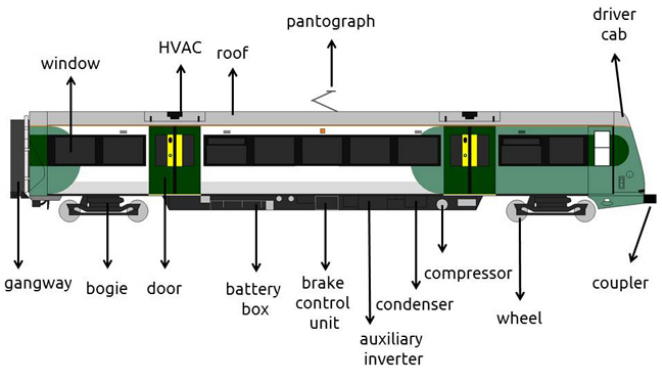


FIGURE 2.2: Typical components of an electric train vehicle: the pantograph transfers electricity from the catenary to the battery box; the energy stored in the battery powers the motor, which rotates the wheels and moves the train; the brake stops the train; the doors give access to the train; the HVAC, compressors, and condensers are part of the air ventilation system that enhances the passengers' comfort. Source: http://www.railsystem.net/rolling-stock-components

1.2 Reliability theory

Reliability theory lays the foundation of research in maintenance. We cover shortly the basic concept of reliability function, the failure rate function, and the mean time between failures.

1.2.1 Reliability theory

Reliability is the probability that an asset functions correctly over a specified period t [62]. Let T be the moment where a failure occurs, the *reliability function* $R(t)$ is $R(t) = P(T > t)$, such that $R(t) \geq 0$, $R(0) = 1$, $\lim_{t\to\infty} R(t) = 0$. Let $F(t) = 1 - R(t)$ be the probability that a failure occurs before t, such that $F(0) = 0$ and $\lim_{t\to\infty} F(t) = 1$. $F(t)$ is therefore the *cumulative function* of the failure distribution.

From $R(t)$ and $F(t)$, let $f(t)$ be a function that describes the shape of the failure distribution of the asset, $f(t)$ is therefore the *probability density function* of the failure distribution, such that $f(0) = 0$ and $\int_0^\infty f(t)\,dt = 1$ (2.1).

$$f(t) = \frac{dF(t)}{dt} = -\frac{dR(t)}{dt} \tag{2.1}$$

1.2.2 Failure rate function

If an asset has survived until time $t \leqslant T$, the *conditional probability of failure* of this asset within the interval $t + \Delta t$ is given in (2.2). Dividing (2.2) by $\Delta t \times R(t)$ yields the *failure rate*, that is, the conditional failure probability per unit of time, from which we define the *failure rate function* $\lambda(t)$ (2.3), also known as the hazard function in survival analysis. $\lambda(t)$ can be increasing, decreasing, or non-monotonic.

$$P(t \leq T \leq t + \Delta t \mid t \leq T) = R(t) - R(t + \Delta t) \tag{2.2}$$

$$\lambda(t) = \lim_{\Delta t\to 0} \frac{R(t) - R(t + \Delta t)}{\Delta t \cdot R(t)} = \lim_{\Delta t\to 0} -\frac{R(t + \Delta t) - R(t)}{\Delta t} \cdot \frac{1}{R(t)} = -\frac{dR(t)}{dt} \cdot \frac{1}{R(t)} \overset{(2.1)}{=} \frac{f(t)}{R(t)} \tag{2.3}$$

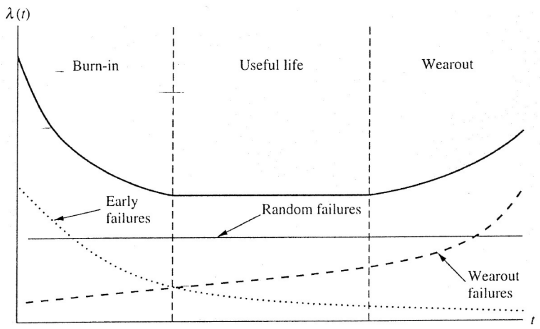


FIGURE 2.3: The bathtub curve, where $\lambda(t)$ is the failure rate function [62]

The famous bathtub curve (Figure 2.3) is one of the forms of the failure rate function [127]. The curve states that an equipment is prone to infant mortality and is more susceptible to failure at the starting point, then failures may occur randomly throughout the equipment life until it reaches the wear-out period, where the degradation continuously worsens. Besides the bathtub curve, there exists many other failure patterns that appear much more frequently. A study on civil aircraft showed that only 4% of the equipment parts conform to the bathtub shape [164].

1.2.3 Mean time between failures

The *mean time between failures* (MTBF) is the average amount of time between two consecutive failures of a *repairable* equipment. Commonly used with the MTBF is the *mean time to repair* (MTTR), estimating the average amount of time that elapses since a failure until the system recovers to a functional state. Another well-known incident metrics is the *mean time to failure* (MTTF) that is the amount of time since the start until the definitive end of the lifecycle of the equipment. Mathematically, the MTTF is the expected time T until a failure (2.4) and is characterized by the probability density function of the failure distribution f (2.1).

$$MTTF = \mathbb{E}(T) = \int_0^\infty t \times f(t)\, dt = \int_0^\infty R(t)\, dt \tag{2.4}$$

Figure 2.4 illustrates how the MTTF, MTTR, and MTBF are measured. An equipment starts its lifecyle at R_1 until F_3 where it encounters a complete failure and requires replacement. After R_1, the first failure F_1 occurs. The equipment is repaired and recovers to a functional state at R_2. The time that elapses between F_1 and R_2 is first time to repair (TTR_1). The equipment continues to function until a second failure F_2 and requires a second repair until its recovery at R_3. The interval between R_2 and F_2 is the first time between failures (TBF_1). The equipment is functional again at R_3 until a total failure at F_3 permanently ends its lifecycle. The time to failure (TTF) is the entire duration from R_1 to F_3. The MTBF is the sum of the TBFs during the equipment lifecycle divided by the number of failures, i.e., $MTBF = \frac{TBF_1+TBF_2}{2}$, and similarly for MTTR. The MTTF applies for the entire lifecycle of an equipment and the mean is computed over the number of equipment.

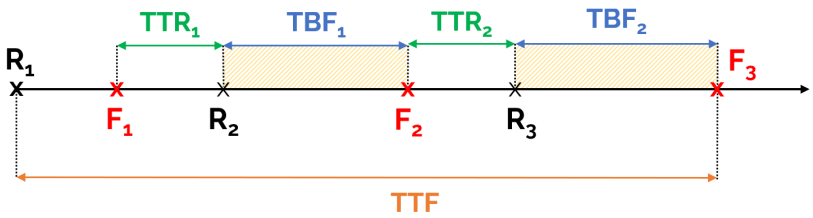


Figure 2.4: An example of computing the MTBF, MTTR, and MTTF on a system (TTR = time to repair, TBF = time between failures, TTF = time to failure

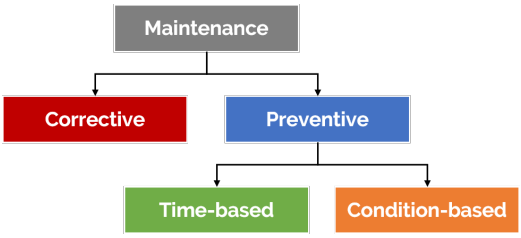
The equipment is therefore available during R_2–F_2 and R_3–F_3 (yellow regions). The available of an equipment is equal to $\frac{MTBF}{MTBF+MTTR}$. The goal of maintenance is to maximize the availability by reducing $MTTR$ (more efficient maintenance inspections) and/or increasing $MTBF$ (more resistant equipment).

1.3 Maintenance strategies

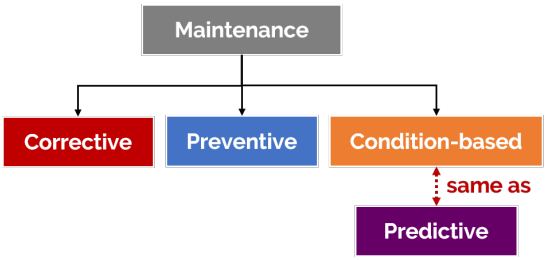
A system is designed to fulfill its functions within a minimum standard of performance. Its inability to fulfill the functions to the predefined standard is called a *functional failure*, or *failure* for short [164]. Maintenance prevents these failures from occurring or rids of those that already happened. Maintenance is a set of necessary operations, including checking, repairing, replacing parts, to prevent a functional failure of the system [40].

In the railway, the safety of the passengers is of utmost priority. Therefore, maintenance is a crucial support function to guarantee the reliability of equipment, which ensures human safety and improves the availability of services. Research on maintenance started to emerge in the 60's [164]. Recently, the

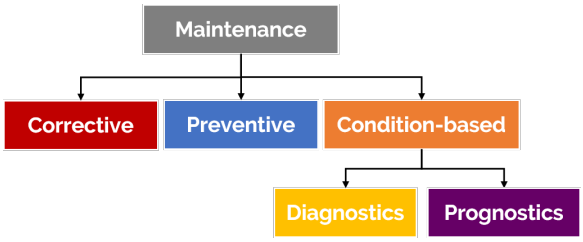
abundance of data and rapid development of computing hardwares are encouraging the experimental use of Internet of Things, big data technologies, and artificial intelligence in maintenance research [119, 135].



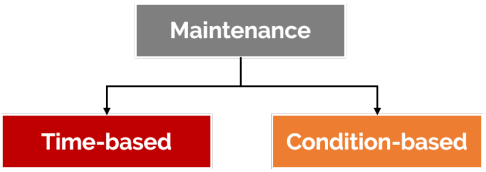
(A) Officially adopted by a maintenance standard [40], this categorization insists on the level of reactivity: maintenance is either corrective (completely reactive) or preventive (proactive) [1, 6, 8, 15, 56, 69, 170].



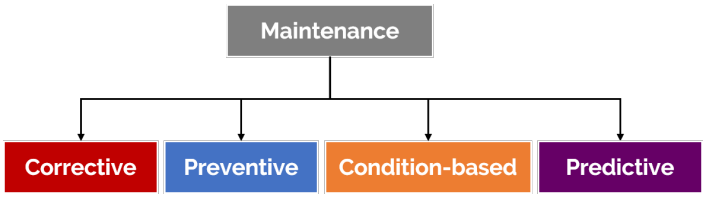
(B) This variant views condition-based maintenance the same as predictive maintenance [42, 48, 119, 179, 214]



(C) Condition-based maintenance may encompass condition monitoring, diagnostics, and prognostics, which is also viewed as predictive maintenance [113, 183].



(D) The corrective strategy may be omitted [10, 129]



(E) All four strategies considered independent with predictive maintenance seen as the most futuristic [95, 235]

FIGURE 2.5: Different categorizations of maintenance strategies

The literature of maintenance research proposes a variety of categorizations of maintenance strategies, listed in Figure 2.5. There are three primary maintenance strategies, namely *corrective maintenance* (CM), *time-based preventive maintenance* (PM), *condition-based maintenance* (CBM), sometimes used interchangeably with *predictive maintenance* (PdM). The variation arises from how researchers group one strategy under another. Overall, CM and PM are always separated, whereas the distinction between CBM and PdM diverges. In this work, we adopt the categorization of Figure 2.5b and consider PdM the same as CBM, since they both share the common requirements: maintenance orders are emitted when needs arise, based on an empirically defined threshold, on a set of rules given by domain experts, or on the predictions of potential failures. The sections that follow provide a brief overview on the key principles of CM, PM, and PdM.

1.3.1 Corrective maintenance

Corrective maintenance (CM) performs unscheduled reparations that restore the system after a fault already occurred [28]. It is also called *reactive maintenance*, *breakdown maintenance*, or *run-to-failure*.

Despite its simplicity, it may incur expensive costs, particularly on complex railway equipment. It leaves little to no time for planning and likely leads to delay of services, costly reparation, or even fatal consequences [225]. For instance, the unexpected malfunctioning of a train paralyzes the entire track section allocated to that train, resulting in the lateness of the next ones.

Nevertheless, the simplicity of CM makes it more practical in non-critical systems in which equipment breakdown results in small costs, slight consequences of failure, quick restoration, and without safety risk [212]. In the railway, CM is to be avoided to limit serious consequences and redundancy is implemented for critical systems.

1.3.2 Time-based preventive maintenance

To compensate the drawbacks of the corrective strategy, preventive maintenance (PM) is a schedule-based strategy, commonly known as time-based maintenance [6, 234]. By implementing PM, companies can plan maintenance inspections at regular intervals while the systems are still functioning to prevent failures. Such interval is estimated from the expected lifetime of the system, which is inferred from historical failures using reliability modeling tools such as the bathtub curve or the Weibull distribution [116]. Until today, PM remains a popular choice in railway management.

The inconvenience of PM is the unnecessary intervention and premature replacement of equipment. In reality, the time to failure of equipment is not totally predictable and varies by equipment despite identical working conditions [91]. Besides, PM cannot prevent random failures that may occur between two consecutive inspections, therefore the system is still prone to unexpected shutdown.

1.3.3 Predictive maintenance

Predictive maintenance (PdM) is gaining popularity because systems are becoming increasingly connected and generate data for training failure-predicting models. Maintenance under PdM is carried out only when needs arise. Hence, it allows to schedule inspections efficiently, maximize service availability, minimize costs, and reduce the stress from urgent interventions. An effective PdM solution must guarantee accurate predictions to avoid false positives and emit the alerts soon enough to leave sufficient time for maintenance preparation.

Condition-based maintenance (CBM) occupies a important place in the literature body of maintenance research. CBM deduces the equipment condition via *condition monitoring* to detect faults and predict failures when certain conditions are met before any failures occur [204]. Condition monitoring collects and interprets relevant equipment parameters such as vibrations, temperatures, humidity, to capture the evolution of the equipment health [240]. From the outputs of condition monitoring, decision-making of CBM includes *diagnostics* and *prognostics*. Diagnostics performs localization, detection, and identification of faults that are already present in the system. Prognostics predicts the chance of failure occurring in the system [113].

To the best of our knowledge, there is no clear distinction between CBM and PdM. PdM is a rather recent term for CBM, coupled with the application of machine learning in late times. Noman, Nasr, Al-Shayea, and Kaid [172] claimed that PdM is an early term for CBM, and both have been used interchangeably ever since. Yet, we reckon that a boundary exists: CBM relies on predefined thresholds to decide when a system crosses an alarming state and needs maintenance, whereas PdM learns such thresholds by itself from the data. In this work, we consider CBM and PdM the same and only use the term PdM. The categorization we adopt is the one shown in Figure 2.5b.

Summary

In this section, we introduce the fundamental concepts of maintenance research in complex systems. A *complex system* is a system with multiple components cooperating without a central control that gives rise to a collective behavior, information communication, and adaptation via evolution. A railway network falls within the definition of complex systems given its number of heavily intertwining components. Such complexity has an impact on the difficulty of implementing railway maintenance. For maintenance, reliability theory is the foundation brick. Reliability is the probability that an asset functions correctly over a specified period of time. The most basic notions of reliability theory are the reliability function, the failure rate function, and the mean time between failure. Finally, we present the maintenance strategies and mention different categorizations. Generally, the two most distinguished strategies are corrective maintenance and time-based preventive maintenance. We consider predictive maintenance the same as condition-based maintenance.

2 Review of predictive maintenance

This section reviews the methods used to implement PdM to identify the main approaches and to position our work in the literature body. Our literature study is by no means exhaustive and more comprehensive surveys on PdM [95, 113, 123, 137, 172, 183] have been done prior to this research.

Following is the relevant terminology to PdM.

- **Failure**, or *functional failure*, is a state in which the system is "unable to fulfill a function to a standard of performance which is acceptable to a user" [164].
- **Fault** is the "state of an item characterized by its inability to perform a required function" [40]. A fault is a state, whereas a failure is an event. A fault is therefore the result of a failure.
- **Diagnostics** is the process of handling faults that have already occurred or precursors of faults that deviate the system from its nominal state. Diagnostics include fault detection (detecting the presence of anomalies), fault isolation (locating the faulty components), and fault identification (identifying the nature of the faults) [113].
- **Prognostics** performs fault prediction and estimates how soon and how likely a fault will occur [113]. Fault prediction estimates the remaining useful life at a given moment t and/or the probability of occurrence of impending faults. For example, a prognostics result can be "there is 75% chance that fault A will happen in four days".
- **Remaining useful life (RUL)** is the amount of time that remains until a failure from a given moment t. Let $X(t)$ be the degradation of a system at t and w the threshold of satisfactory performance, a system starts anew at $t = 0$ ($X(0) = 0$) and gradually degrades over time. When $X(t) \geqslant w$, the system encounters a functional failure. The RUL T estimated at t is the time remaining during which the system can work properly until t_f (2.5). Any maintenance taken before t_f is preventive or predictive by nature, and becomes corrective when it approaches or passes t_f.

$$T = \inf\{t \,:\, X(t_{now}) + t \geqslant w \mid X(t_{now}) < w\} \tag{2.5}$$

We categorizes the techniques used to implement PdM in two classes, depending on the type of available resources (Figure 2.6). The *knowledge-based* class relies on the knowledge solicited from the domain experts. The *data-driven* class leverages the operational data to extract insights on the system condition with limited domain knowledge.

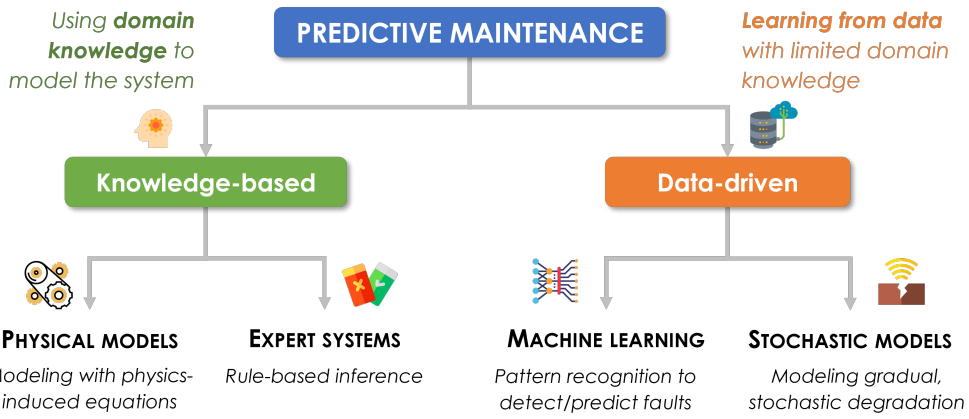


FIGURE 2.6: Categorization of PdM techniques.

2.1 Knowledge-based approach

The knowledge-based approach solicits domain experts to build mathematical models based on the underlying physics of the systems (*physical models*), or to formalize domain knowledge in form of rules that enable automatic reasoning and inference (*expert systems*).

2.1.1 Physical models

A physical model uses mathematical equations to describe explicitly the degradation mechanics in the monitored system, combining extensive mechanical knowledge and domain expertise [183]. The three most common types of degradation are *creep*, *fatigue*, and *wear* [53, 222].

CREEP Creep is the permanent deformation in a material under high temperature for a long duration of time [53]. It is a slow and time-dependent process, divided in three regions (Figure 2.7): the first region is the primary creep with an accelerating creep rate at the beginning, followed by the secondary phase with a stable creep rate, then the creep rate accelerates during the tertiary region that ultimately leads to a rupture of operation.

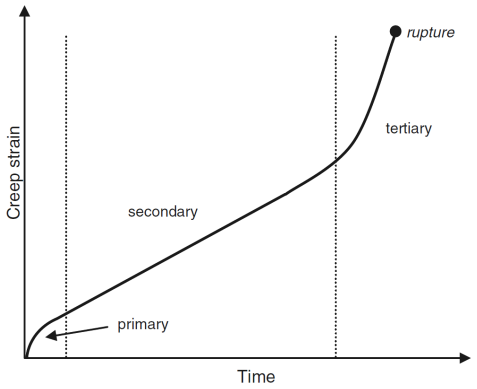


FIGURE 2.7: Three regions of the creep curve [222].

The lifetime of a component under creep degradation is the time until its rupture. Because the second region spans most of the component lifetime, it is highly important to capture the creep rate within this region, for which the Norton creep law (2.6) can be of use. The creep rate in the second

region is denoted $\frac{\Delta\epsilon}{\Delta t}$, where t is the time and ϵ the extent of deformation, and it is affected by the stress level σ, the temperature T, with A, n, m being material-dependent constants. Baraldi, Mangili, and Zio [24] estimate the RUL of turbine blades under creep using the Norton creep law. The degradation level measured by the Norton law at each time step form the training data of an ensemble model, coupled with a Kalman filter to aggregate the RUL estimates of individual models and to filter out noises.

$$\frac{\Delta\epsilon}{\Delta t} = AT^n\sigma^m \tag{2.6}$$

Another method to estimate the time to rupture is the Larson-Miller parameter (LMP) (2.7), where T is the temperature in Kelvin, C an experimental constant, and t the time until rupture. LMP is a parametric model that extrapolates the creep rate of engineering materials under steady-state conditions. It enables the use of short-term creep life data under high temperature or high load conditions for estimation, thus overcomes the long testing time [117]. Kandare, Feih, Lattimer, and Mouritz [117] model creep rupture using LMP to determine the failure time and failure temperature of aluminum exposed to fire. Vasudevan, Venkadesan, Sivaprasad, and Mannan [230] study the influence of thermal aging on the hardness of cold-worked stainless steel used in fast breeder reactions, focusing on the adequate choice of the constant C on different cold-work levels. Loghman and Moradi [147] investigate the time-dependent creep damage and remnant life of a thick-walled spherical reactions using the LMP correlation.

$$LMP = T(C + \log t) \tag{2.7}$$

Fatigue Fatigue occurs in components subjected to high cyclic loading (repetitive rotations or vibrations). The lifetime of a component under fatigue is the number of cycles until failure. Models for fatigue modeling include the S-N curve, the Basquin law [175], the Manson-Coffin law [211], the cumulative damage rule [39]. Qiu, Seth, Liang, and Zhang [191] use the cumulative damage rule to develop a prognostic model for bearing fatigue by correlating the natural frequencies and their amplitude to the stiffness of the system.

A common consequence of long-term fatigue damage is *crack*. The crack growth has three regions (Figure 2.8). The first region is when a crack is initiated in the component as micro-crack (crack initiation). Then, it is propagated due to the continuous cyclic loading of the component, resulting in a constant and significant growth rate (crack propagation). The final region is when the crack grows rapidly until a fracture occurs. Prognostics under crack degradation projects the crack propagation to the future degradation of the component to estimate the time to failure.

A famous formula for crack growth modeling is the Paris law [180]. It estimates the growth of the crack length a over the number of running cycles N, where ΔK is the intensity of the crack in one loading cycle, C_0 and n are material-depending constants (2.8). The Paris law only applies to the second region of the crack, but because this region occupies the majority of the lifetime of a component under crack, the Paris law is sufficient to model the degradation of a system under crack fatigue [222]. Oppenheimer and Loparo [178] perform diagnostics and prognostics in rotor shafts via crack growth modeling using a modified version of the Paris law to calculate the crack in both regions II and III.

$$\frac{da}{dN} = C_0(\Delta K)^n \tag{2.8}$$

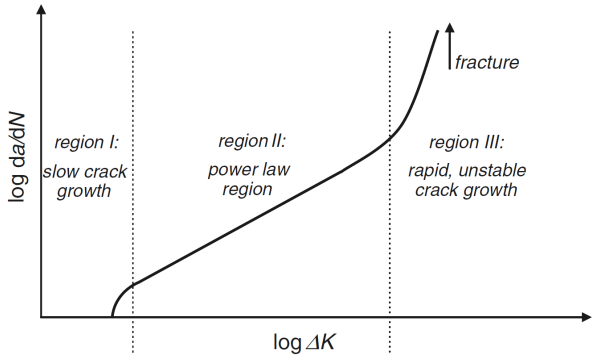


FIGURE 2.8: The crack growth divided in three regions [222].

Another equation for crack growth modeling is proposed by Hoeprich [97]. In place of the crack length, the severity of the crack defect is characterized by the growth of the crack surface D over the number of running cycles N (2.9), with C_0 and n being material-dependent constants. Li, Billington, Zhang, Kurfess, Danyluk, and Liang [139] propose a recursive adaptive algorithm to predict the defect propagation rate over time via vibration measurement following Hoeprich's formula.

$$\frac{\Delta D}{\Delta N} = C_0 D^n \tag{2.9}$$

WEAR Wear is a gradual degradation at the surface due to the friction between two components in sliding motions, resulting in a loss of material of at least one of the components. There are four wear mechanisms: adhesive wear, abrasive wear, corrosive wear, and surface fatigue [19]. Modeling component wear is challenging because external factors, such as environmental conditions, have an important impact on the contact of the surfaces [53].

The Archard's law is applicable to all types of wear [18]. It defines V as the total wear volume, proportional to the load P and the sliding distance Δs, K the wear coefficient depending on the material and H the material hardness (2.10). Wear is a dynamic process and wear prediction is formulated as an initial value problem solvable with the Archard's law [203, 208]. Silva and Pintaude [208] model the wear of surfaces in contact by formulating an initial value problem using the Archard model, using random variables and stochastic processes to capture the uncertainty in wear coefficient. Shen, Cao, and Li [203] use the Archard law to study sliding wear via numerical simulation on pin-on-disk tests between aluminum alloy plate and self-lubricating linear.

$$V = \frac{KP}{H}\Delta s \tag{2.10}$$

STRENGTHS AND LIMITS Physical models tackle lifetime prediction via physics-induced equations to describe the degradation affecting the system, with the help of domain expertise and mechanical knowledge. The parameters of the equations are fine-tuned by minimizing the *residuals* (difference between the predictions and true measurements collected during tests) [183]. A suitable model reflects accurately the physical behavior of the system, providing reliable insights into its condition and long-term behavior [16].

However, real-life systems are often too complex to be modeled correctly. A physical model is tailored to one specific system and cannot be generalized or easily adapted to another. Even if physical models require less data for validation, the parameter fine-tuning requires disruptive tests in the system. To avoid this issue, practitioners opt for recursive algorithms to update the models in real-time without re-processing at the arrival of new measurements [139, 191]. Still, these models require extensive expert knowledge, which may not be accessible nor cover all possible scenarios.

2.1.2 Expert systems

An expert system (ES) consists of a knowledge base containing facts, rules, heuristics, and an inference engine for automatic reasoning and query answering [182]. An ES can be rule-based, case-based, or model-based [113].

Rule-based reasoning formalizes human expertise as a set of machine-analyzable and -understandable rules for reasoning. Chande and Tokekar [49] develop a rule-based ES for fault anticipation, fault recovery, and components calibration in embedded systems, coupled with a Markovian chain to dictate the preferred course of actions when facing a fault. Tang and Wang [216] implement a framework of two rule-based ES to diagnose a cutting and delivering system: an offline fault analyzer and an an online fault diagnosis and prevention. Four artificial neural networks are trained to approximate the dynamic parameters and an explanation module is provided to convert the inference results from digital to linguistic form.

Case-based reasoning reuses the solutions in past scenarios and learns from the experience to handle new problems. Bengtsson, Olsson, Funk, and Jackson [27] performs sound analysis to diagnose audible faults on industrial robots using a case-based ES in combination with the nearest neighbor approach. The recorded sound is preprocessed for filtering and noise removal. Then, the ES identifies relevant features and classifies them based on previously classified measurements (case library).

Model-based reasoning builds a model of the physical system on which it performs inference. Turgis, Auder, Coutadeur, and Verdun [226] estimates the RUL of electric doors on passenger trains using rules deduced from the doors' mechanics. The preventive maintenance threshold is dynamically adjusted to issue alerts with the correct severity.

Strengths and limits An ES profits from domain knowledge, hardware computation power, and reasoning algorithms to generate solutions faster than human experts. It is one of the first successful forms of artificial intelligence capable to deduce new knowledge for reasoning and for problem solving on their own. ESs are traditionally used for fault diagnostics in CBM applications [182, 187].

Although domain knowledge is available and reliable, converting it to machine rules demands immense effort. Modeling a complex system requires a large set of rules, consequently it causes the "combinatorial explosion" in computation problems. In particular, some relationships between system variables cannot be expressed by a simple IF-ELSE rule [216], so more intricate modeling is needed to properly formulate such relationships. By nature, an ES cannot handle unexpected situations not covered by the rules.

Knowledge-based approach: Discussion

The knowledge-based approach consists of physical models and expert systems that are constructed with extensive understanding of the target systems. Physical models explicitly describe the physics of a system and of the fault mechanism via mathematical equations. Expert systems formalize domain

knowledge into a set of rules and perform inference to diagnose a system. These models prove their strengths if domain knowledge is available, reliable, and complete. The explainability of knowledge-based models is also a considerable advantage. Expert systems have been widely used in the maintenance of the railway industry as well [20, 58, 72, 228, 238, 247].

Regardless, converting domain expertise to a machine-comprehensible language remains challenging. Tailoring a solution that approximates the behavior of a real-world complex system requires a substantial amount of effort. The result is a system-specific model and is hardly generalizable from one system to another.

2.2 Data-driven approach

Besides domain knowledge, operational data such as historical failures, event logs, or condition monitoring measurements can provide rich information about the health of systems [206]. Data-driven approach exploits these data to learn failure patterns and issue prediction. Data-driven methods fall into either *machine learning* or *stochastic modeling* category.

2.2.1 Machine learning

Machine learning (ML) has become a major player in PdM applications given its versatile use. ML is the study of algorithms and statistical tools to produce a model that performs a specific task it is trained for. Overall, we can divide machine learning paradigms into three dominant groups, namely *supervised learning*, *unsupervised learning*, and *reinforcement learning*.

Supervised learning Supervised learning (SL) learns an arbitrary function that maps an input space $\mathcal{X}$ to an output space $\mathcal{Y}$. The learning process is guided by the output data. From a mathematical point of view, supervised learning requires a dataset $S = \{ x_i, y_i \}_{1 \leqslant i \leqslant N}$ with $x_i \in \mathbb{R}^D$, where N denotes the number of samples, D the data dimension, and $y_i \in \mathcal{Y}^L$ (if $L = 1$, it is a single-output task, else it is a multi-output prediction). Depending on the type of y_i, we distinguish two kinds of tasks: *classification* for discrete output values, and *regression* for continuous output values. For classification, if $|y_i| = 2$, it is a binary classification, else if $|y_i| > 2$ it is a multiclass classification.

Classification In PdM scenarios, the outputs are the states of the system as a function of the inputs. Decision trees, support vector machines, logistic regression, ensembling, neural networks, have been extensively studied and remarkably improved in recent years. In particular, deep neural networks have been widely applied to solving PdM problems in the last few years, to cite a few [84, 85, 93, 202, 250, 252].

Although classification is intuitive for revealing the system's state, it does not extrapolate the its state in the future, as future measurements cannot be obtained in current time. A possible workaround is to classify the system's state for a time window instead of for each individual measurement sample [214]. Phillips, Cripps, Lau, and Hodkiewicz [185] use binary logistic regression to classify the condition of oil engines as good or not good. Inturi, Shreyas, Chetti, and Sabareesh [104] frame the diagnostics of wind turbine gearbox as a multi-level classification problem using adaptive neural fuzzy inference system: the first level classifies the speed stage of the defected gearboxes, the second level locates the defect, the third level identifies the type of defect, and the fourth level quantifies the severity of the identified defect (25%, 50%, 75%, 100%).

Nevertheless, it may be difficult to train a classification model for critical systems because of class imbalance due to rare failure data. Failures in highly critical systems such as aircraft [129] are catastrophic and maintenance operations are often scheduled ahead of failure moments, making occurrences of failed state rarely appear in the data.

Regression For regression, the prediction target is a real-value number. It can be the probability that a sample falls in a certain class, or the target value of a continuous function. The most relevant application of regression in PdM is RUL estimation. Baptista, Sankararaman, Medeiros, Nascimento, Prendinger, and Henriques [23] combine sequential learning of autoregressive–moving-average model (ARMA) with an ML algorithm to train a regressor on life usage data. The predicted failure times and additional statistical features are used to train an RUL regressor that predicts the next failure time for unseen instances. Recurrent neural networks (RNN) are also a popular choice to deal with regression on temporal data. Heimes [94] customizes an RNN that combines the extended Kalman filter and an evolutionary algorithm to reduce the impact of noises for efficient training. Korvesis, Besseau, and Vazirgiannis [129] learn a function that quantifies the risk of an event of interest (e.g., failure) occurring in the near future via multiple learning instance from event logs in post flight reports.

Unsupervised learning Supervised learning is only feasible if labeled data exist, but real-life use cases are seldom so ideal. First, creating labeled data is a tedious, costly, and time-consuming process. In particular, for connected systems with sensors that may capture hundreds of readings in seconds, manual annotation on such high-speed data stream is almost infeasible. Secondly, highly critical systems are maintained regularly to avoid catastrophic failure. If the system is always restored before a failure could occur, there will be no data of the exact failure time, also known as *right-censoring data* in survival analysis.

When labeled data are scarce, unsupervised learning is a better fit to discover patterns without knowing the desired output, but it will require more effort to evaluate the results. Examples of unsupervised learning tasks include, but not limited to, *clustering*, *anomaly detection*, *association rules*, and *dimensionality reduction* (mostly for data preprocessing).

Clustering Clustering is the process of assigning objects into groups, such that the similarity is high between members in the same group (high cohesion) and low for those from different groups (high separation). Amruthnath and Gupta [15] conduct fault detection and fault prediction using density estimation via Gaussian mixture models and K-Means to detect three system states: healthy, faulty, and reset. Lima, Paredes Crovato, Goytia Mejia, Rosa Righi, Oliveira Ramos, André da Costa, and Pesenti [143] decomposes sensor signals to detect deterioration trends of the machines and uses K-Means to produce clusters of meta-trends. The condition of a machine is assessed via the Euclidean distances between clusters. This process is repeated periodically on new sensor measurements. Luo, Fong, Sun, and Leung [151] use K-means to discover failure patterns for fault detection and diagnosis of chilled water systems. The Davies-Bouldin index [55] is used to identify the optimal number of clusters from the sensor readings. To deal with high-dimensional input data, Gao, Kang, Tian, Wu, and Pecht [76] combines subspace clustering with locality-preserving latent low rank recovery for fault diagnostics.

Anomaly detection Anomaly detection identifies objects that are significantly different from the majority of the data. For PdM, anomaly detection separates odd behaviors from typical ones to enable fault detection. Zhao, Kurihara, Tanaka, Noda, Chikuma, and Suzuki [251] detects an occurrence of anomaly if the correlation coefficients from the sensor data deviate from the reference clusters,

obtained from normal sensor data and their correlation coefficients. Aydemir and Acar [21] use the Cumulative Sum (CUSUM) to detect anomalies from machinery. The sensor signals are continuously monitored and an anomaly alert is emitted when the signals exceed the limits of healthy operation, which also triggers a posterior failure prediction operation. CUSUM is simple to use, but such simplicity hinders the analysis on a finer granularity.

Reinforcement learning In a reinforcement learning (RL) setup, an agent interacts with an environment via a set of actions, such that each action A_t is associated to a reward R_t dictated by a reward function and may modify the state of the environment S_t, where t denotes the timestep (Figure 2.9). The complete specification of the environment constitutes a task [215]. The agent learns by taking sequences of actions that maximizes the total rewards within a number of timesteps. The reward function should ensure a balance of exploitation-exploration.

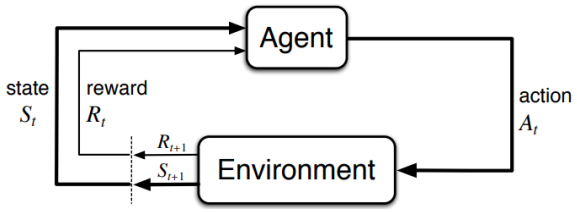


Figure 2.9: The setup of reinforcement learning [215].

Kozjek, Malus, and Vrabič [130] use RL to adjust the raw RUL estimations returned by random forest such that the final estimations satisfy multiple objectives: high utilization, effective maintenance planning, and safety. The reward function penalizes RUL alerts that are too early or late. RL being an optimization task, it can also recommend the appropriate course of maintenance actions. Hoong Ong, Niyato, and Yuen [99] develop a double deep Q-learning network on sensor data to self-learn optimal maintenance policies and to issue actionable recommendations. The agent is given a limited credit of repair frequency and repair cost, then learns how to choose the effective maintenance policy among repair, replace, or hold. To push the efficiency of RL further, Yao, Lu, and Zhang [242] integrate deep Q-learning network (DQN) in a recurrent neural network combined with transfer learning to adapt an agent trained to estimate RUL in one tool to another. Zhang, Gupta, Farahat, Ristovski, and Ghosh [248] formulate equipment monitoring as a credit assignment problem, using a model-based approach with Markov decision process to derive the value function and a model-free approach with a bootstrapping algorithm to learn the value function without implicit modeling. The system states are the outputs of the learned value function. The outputted states are then used to train a regressor for RUL estimation on run-to-failure data.

Strengths and limits ML is able to search for patterns that intrinsically reflect the states of an equipment and to perform failure prediction based on past data. There are different paradigms of ML: learning a function that maps the input space to the output space (supervised learning), separating individuals into groups or detecting anomalous individuals (unsupervised learning), or finding the optimal policy to solve a problem (reinforcement learning).

So far, the most robust class of ML algorithms are still the supervised ones. This implies that a significant volume of labels is needed to produce an accurate model. Moderate to long training time is to be expected. Adequate hyperparameters are crucial to the model's accuracy. For example, artificial neural networks must be set up with the optimal network architecture, number of layers, number of neurons, activation functions, weight initializers, and so on. Hyperparameter tuning is usually the

most time-consuming and critical process for ML. Explainability is an open issue. Simple models such as logistic regression or decision trees can produce human-understandable explanation of a prediction, but it is not the case for any ML models, such as random forests or deep neural networks, even if the latter has been widely applied to PdM lately.

2.2.2 Stochastic modeling

We distinguish two types of failures. A *hard failure* is a random event that abruptly interrupts the system and can only be remedied by corrective maintenance. On the contrary, a *soft failure* is a gradual deterioration in the system performance until the outcome is unsatisfactory and the system fails to fulfill its purpose. The latter can be handled by stochastic modeling.

System degradation is a stochastic process because it contains small, random changes over time. Let $\{X(t) : t \geqslant 0\}$ be a stochastic process. To model $X(t)$, Markovian methods apply when the degradation is studied in a finite state space. Otherwise, Lévy processes such as Wiener and Gamma processes are commonly used for continuous stochastic processes. For Lévy processes, an important notion is the *first hitting time* (FHT) that points to the moment where the degradation level exceeds a stress threshold, over which the system performance is no longer satisfactory. Then, RUL estimation is equivalent to finding the FHT from the current time to project the future degradation of the system.

Markov models Markov models are suitable if the degradation is represented by a finite set of discrete states $\Theta = \{0, 1, \ldots, N\}$, with 0 being a perfectly normal state and N the failure state. For example, Liang and Parlikad [141] build two Markovian state transition models to tackle maintenance of insulation paper in a power transformer: one model describes the transition between the equipment states (healthy, aged, defective, faulty, failure) according to the degradation modes (normal, accelerated chemical aging, accelerated mechanical aging), and the other describes the transition between the equipment states according to the maintenance methods (no maintenance needed, minor maintenance, major maintenance, minor preventive maintenance). The authors conclude that Markov models help to optimize the maintenance of power transformers by minimizing the cost of periodic inspections and maximizing the availability of the equipment.

Wiener processes The Wiener process supposes independent increments $\Delta X(t)$ of alternate increases and decreases. It is also known as the *Brownian motion with drifts*. A system that experiences non-monotonic degradation over time - due to increased or reduced intensity of use, can be modeled by the Wiener process. However, it is not suitable for systems under monotonous, irreversible deterioration. Si, Wang, Hu, Chen, and Zhou [205] address RUL estimation using the Wiener process. They argue that the conventional Wiener process exhibits Markovian property and does not consider the entire history of the data. To circumvent this problem, they formulate the drift parameter λ as a hidden state space. A Kalman filter updates λ recursively to account for past data without requiring full data storage. Other hidden parameters are estimated using the expectation-maximization algorithm.

Gamma processes The Gamma processes are designed for stochastic processes with monotonic trends, that is, for systems with irreversible degradation. Since a degradation is likely irreversible in nature, the Gamma processes are more applicable. Liao, Elsayed, and Chan [142] propose a condition-based policy focusing on the asset availability, assuming perfect monitoring (the true state of the asset is immediately revealed when needed) and imperfect maintenance (the maintenance actions do not restore the asset to an as-good-as-new state). The asset degradation is modeled with a Gamma process. The optimization function finds the optimal preventive threshold D^*_{PM} above which a maintenance

action must take place, and the optimal number of maintenance actions N^* allowed in a cycle, after which a complete replacement of the asset is performed. The replacement must occur before a failure causes damage to the asset.

Strengths and limits Stochastic modeling tackles PdM via stochastic processes with small, random changes over time, which fits the nature of machinery degradation. This approach also allows to project the equipment degradation in the future, thus implementing both monitoring and failure prediction. Depending on the state space (discrete or continuous), the choice of models varies. Once a model is chosen, its parameters can be estimated from the available data. It is worth to note that the parameters of a stochastic model are different from the hyperparameters of a machine learning model. While some ML models are assumed model-free and the hyperparameters are not related to any distribution (e.g., the tree depth of a decision tree), stochastic models are more restrained and the parameters define a distribution.

The main challenge of stochastic models is its heavy use of mathematical modeling, contrary to the plug-and-play characteristic of some ML models. A sufficient amount of data stays necessary to estimate the parameters for the models. Stochastic modeling is dedicated to discrete or continuous data, for instance, from sensor readings. Meanwhile, ML can also be used for texts [128] and images [132], making it more versatile.

Data-driven approach: Discussion

The data-driven approach has two subcategories: machine learning and stochastic modeling. Machine learning employs statistical learning methodology such as tree-based techniques, neural networks, support vector machines, to discover patterns from the data (supervised/unsupervised learning) or to find the optimal policy for a problem (reinforcement learning). Stochastic models consider the system degradation a stochastic process with small temporal increments or decrements. Markov models are suitable for discrete-state degradation, whereas Gamma and Wiener processes are used for continuous degradation.

As machines are being increasingly sensorized, they generate more operational data, which encourages the use of data-driven approach. It does not require a comprehensive knowledge of the physics of the system and of the faults to implement PdM solutions. Nonetheless, a basic understanding of the system is preferable to facilitate (hyper)parameter tuning. The quality and amount of data are essential for the successful realization of data-driven PdM. A model is trained for a moderate or long time and must be updated regularly as new data become available, if machine learning is used. Then, many models are black-boxes: algorithms such as neural networks are not easily explained to humans because of their intricate learning process.

The evolution of the system state can be framed as a gradual degradation process. Such process can be effectively modeled with stochastic tools such as the Markov models and its variants (hidden Markov models or semi-hidden Markov models), Gamma processes, or Wiener processes. The choice of tools depends on the degradation state space. The available data aid parameter tuning, making the models more accurate and robust. However, stochastic modeling is more complicated than machine learning and requires a strong mathematical background to correctly apply and fine-tune the models.

Methodology for PdM: A summary

In this section, we review the methodologies for PdM and propose a categorization that divides the existing methods into two categories, namely the *knowledge-based* approach and the *data-driven* approach. This categorization insists on the type of resources used to implement PdM solutions. Using a knowledge-based approach, a practitioner solicits domain experts to craft a representation of the target equipment via physics-induced mathematical equations (*physical models*), or to formalize the domain knowledge in form of machine-processable rules for automatic inference and reasoning (*expert systems*). Nonetheless, humans are not perfect and their understanding may not cover all possible scenarios. Operational data and data-driven algorithms can compensate what the humans may miss. The data are fed to a statistical learning algorithm to discover the failure patterns of an equipment and/or to select the optimal course of actions (*machine learning*). The data can also be fitted to a stochastic model to monitor the equipment degradation as a random process with small increments or decrements over time (*stochastic modeling*).

Table 2.1 summarizes the knowledge-based and data-driven approaches to solve PdM.

TABLE 2.1: Predictive maintenance approaches

	Knowledge-based		**Data-driven**	
	Physical models	**Expert systems**	**Machine learning**	**Stochastic models**
Principles	Build an explicit mathematical model to simulate the physics of the faults	Build a knowledge base of rules and use an inference engine to generate diagnosis and prognosis	Train a model to discover failure patterns from the data	Construct a stochastic model to monitor the degradation process of the equipment
Examples	Paris law Archard law Crack growth modeling	Knowledge base	Tree-based methods Neural networks Support vector machines Clustering Anomaly detection	Markovian models Gamma process Wiener process
Strengths	High accuracy Physical meaning of health indicators Ability to predict the long-term behavior of the system	Automatic inference from a large set of rules More accurate inference than human experts	Robust and accurate models Many off-the-shelf libraries available Highly versatile	Accurate degradation models More realistic modeling of the degradation process
Limits	Tailored for a specific system Interruptive tests required to estimate the parameters Inability to model some degradation modes	Difficult to formalize human knowledge Inability to adapt to cases not covered by the rules Combinatorial explosion in case of large sets of rules	Time-consuming training and parameter tuning Limited labeled data to train robust models	Difficult parameter tuning Complicated mathematical modeling

We now shift our attention to the industrial standards that define the expected functionalities of a PdM framework. These standards give us a guideline to formulate the hypotheses that conform to both the academic and industrial expectations of a PdM solution.

3 Standards for predictive maintenance

In this section, we draw our attention to the standardization of PdM. Specifically, we will address the following questions.

Q1: What is a standard? A *standard* is a body of knowledge of a specific domain, compiled and condensed by domain experts [184]. A standard provides the definition of terminology, the guidelines for implementation, and common practices to undertake in order to systematically and effectively solve a problem. Organizations and associations such as ISO[1], IEC[2], and IEEE Standards Association (IEEE-SA)[3] have been working on standardization for almost a century.

Q2: Why is standardization beneficial for the application of PdM in industry? The motivation arises from the intrinsic complexity of maintenance management. A system designed for managing maintenance operations can grow extremely complex: a large number of measurement variables to be processed and aggregated, hidden precursors of impending faults, the uncertainty in identifying and estimating the current health state, the data exchange within the system, et cetera. Combined together, these factors result in a sophisticated system that should be able to perform computations smoothly, timely, and accurately. When building a maintenance management system, an enterprise may choose to integrate third-party solutions, also called Commercial Off-The-Shelf (COTS) products. However, COTS products from different suppliers likely have conflicting interfaces, which complicate software and hardware integration. That is why standardization becomes essential for the progress of PdM. Standardization ensures the mutual compatibility of products, services, components, software and hardware programs originated from different suppliers [119]. Standards development is especially relevant in the context of Industry 4.0, in which intelligent manufacturing systems become increasingly interconnected and the amount of generated data grow exponentially.

Q3: What are the existing standards related to PdM? At present, PdM has harvested significant research results and application cases in several areas, such as railway, wind turbines, aerospace, and military, etc. [41, 43, 129, 138, 226]. Organizations that develop PdM-related standards include ISO, IEEE-SA, MIMOSA[4], IEC, VDMA[5], SAE[6], and FAA[7].

The rest of this section discusses the existing standards dedicated to PdM. We will first describe MIMOSA OSA-CBM as the de facto open standard for CBM [119] (Section 3.1). The popularity of OSA-CBM has led to the development of other standards that extend, customize, or complement it (3.2). Finally, we point out the usefulness of each standard and propose how they can be of use to develop a highly compliant platform for PdM (Section 3.2.3).

3.1 OSA-CBM as the de facto standard

OSA-CBM [158], short for *Open System Architecture – Condition Based Maintenance*, is an open standard architecture for data processing and communication within a CBM system. It encompasses a full range of functionalities, from data collection to result visualization. Before discussing OSA-CBM, it is worth to mention ISO 13374 - the foundation of OSA-CBM.

3.1.1 ISO 13374

ISO 13374 offers guidance for the implementation of condition monitoring and diagnostics (CM&D) systems that facilitates the integration of various software packages for processing, communicating, and displaying CM&D-related data [108–111]. It comprises four parts:

[1]International Organization for Standardization. https://www.iso.org/home.html
[2]International Electrotechnical Commission. https://www.iec.ch/index.htm
[3]Institute of Electrical and Electronics Engineers. https://standards.ieee.org/
[4]Machinery Information Management Open System Alliance. https://www.mimosa.org/
[5]Mechanical Engineering Industry Association (Germany). https://www.vdma.org/
[6]Society of Automotive Engineers. https://www.sae.org/
[7]Federal Aviation Administration. https://www.faa.gov/

- ISO 13374-1:2003 - General guidelines
- ISO 13374-2:2007 - Data processing
- ISO 13374-3:2012 - Communication
- ISO 13374-4:2015 - Presentation

ISO 13374-2 introduces the notion of *functional blocks*. A functional block is a computation unit that processes the input data, produces the outputs as required, and flags the outputs with the appropriate assessment. For a CM&D system, ISO 13374-2 specifies six functional blocks numerated from 1 to 6, as the procedure follows a sequential order (Figure 2.10). The objectives, input, and output of each block are summarized in Table 2.2.

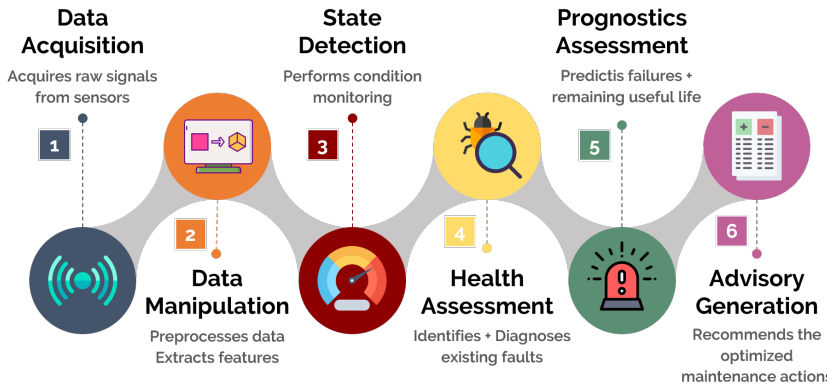


FIGURE 2.10: Six functional blocks defined by ISO 13374-2, numbered from 1 to 6.

It is possible to implement a functional block with more than one instance. For example, the PA block may have many instances that use different models and algorithms for prognostics to increase the richness and variety of functionalities in a CM&D system.

3.1.2 OSA-CBM

OSA-CBM implements the specifications of ISO 13374 by adding *data structures* and *interface methods* to the functional blocks. Therefore, we may say that OSA-CBM is the realization of ISO 13374. OSA-CBM uses the term *layers* rather than blocks. The full architecture of OSA-CBM is illustrated in Figure 2.11, showing an almost identical architecture to ISO 13374-2, except the additional Presentation layer that provides data visualization to the end-users.

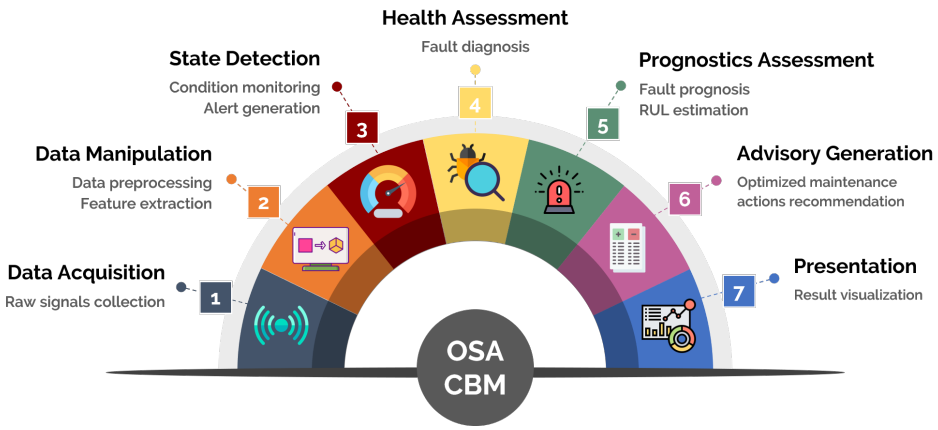


FIGURE 2.11: The standard architecture proposed by OSA-CBM.

TABLE 2.2: Objective, input, and output of each functional block

Block	**Objective**	**Input**	**Output**
Data Acquisition (DA)	Collect and digitize raw signals	Analog input Digital input Manual input	Digitized data Timestamps
Data Manipulation (DM)	Process the digitized data to extract relevant indicators	DA output	Indicators
State Detection (SD)	Perform *condition monitoring*: - Baseline comparison - Evaluate the current health state - Generate threshold-based alerts	DA outputs DM outputs Other SD outputs	State indicators (e.g. “good", “warning", “faulty")
Health Assessment (HA)	Perform *diagnostics*: - Determine the current health - Diagnose existing faults	DA outputs DM outputs SD outputs Other HA outputs	Current health state Diagnosis of faults
Prognostics Assessment (PA)	Perform *prognostics*: - Project the future health state - Project future failure modes - Estimate the RUL	DA outputs DM outputs SD outputs HA outputs History failure data & operational data	Future health state Future usage profile RUL estimation
Advisory Generation (AG)	Integrate the outputs from previous blocks and operational constraints to recommend optimized actions to applicable personnel and resources	Outputs from previous blocks Other AG outputs Constraints	Operation & Maintenance advisories

The main difference between OSA-CBM and ISO 13374 them is the products they offer. OSA-CBM provides UML schemas, XML schemas, binary reference data, a software development kit for .NET including a developer guide, C# .NET scaffolding for OSA-CBM layers, and web services to make connection to any OSA-CBM layer. Meanwhile, ISO 13374 are simply documents that elaborate on the expected functionalities of a CM&D system.

The second difference is the accessibility of each standard. ISO 13374 is proprietary, in contrary to OSA-CBM being open access. This accessibility issue may explain the popularity of OSA-CBM. The first version of OSA-CBM was released in 2001. The latest version of OSA-CBM was published in June 2010 and is publicly accessible on the website of MIMOSA[8].

OSA-CBM has been used by the research community on maintenance thematics. Amaya and Alvares [11] construct a rule-based expert system for intelligent maintenance of hydroelectric machinery by covering layer 1 to 6 of OSA-CBM. Niu and Yang [169] propose a data fusion-based system implementing Layer 2 to 5 of OSA-CBM to perform online condition monitoring and data-driven prognostics on methane compressors in petrochemical industry, but health assessment and diagnostics (Layer 4) are not mentioned in their work. Later, Niu, Yang, and Pecht [171] improve the work of Niu and Yang [169] by incorporating the philosophy of reliability-centered maintenance in the system to maximize the system reliability while reducing the implementation cost. Thurston, Lebold, and Box [220] prove

[8] https://www.mimosa.org/mimosa-osa-cbm/

the benefits gained from applying OSA-CBM in an Integrated Vehicle Health Management system[9], via benchmarking results that show an increased scalability and flexibility of hardware and software integration. Byington, Watson, Roemer, Galie, McGroarty, and Savage [41] discuss the design of a modular prognostic software as part of the Navy's Integrated Condition Assessment System (ICAS) to enable maintenance troubleshooting and planning for shipboard machinery systems. ICAS is not solely built on OSA-CBM, but it makes use of the OSA-CBM specifications to facilitate the interfacing with other programs that are OSA-CBM compliant. Thus, this demonstrated the remarkable benefit of standardization: it allows a flexible communication with other third-party programs without requiring the understanding of how they operate.

3.2 Other maintenance-related standards

OSA-CBM provides an architecture covering the core functionalities a proper CBM system, without details about algorithms or models. We will present in this section other standards that cover one or more maintenance functionalities defined by OSA-CBM or aim to extend, customize, or formulate OSA-CBM differently.

3.2.1 Standards for condition monitoring

In OSA-CBM, the SD block is responsible for *condition monitoring*. Condition monitoring is the process of observing and comparing new data to a baseline profile to detect abnormalities in the equipment and to generate warnings based on predefined thresholds. Condition monitoring is the first barrier that prevents unanticipated failures by alerting of faults or precursors of faults at an early stage.

VMDA 24582 Developed by the VDMA association, VDMA 24582 focuses on condition monitoring in automated systems [231, 237]. Similar to ISO 13374-2, VDMA 24582 uses functional blocks to encapsulate condition monitoring computations (Figure 2.12). A functional block processes the input to generate an enumerated status indicating the current state of the monitored equipment, for example, normal, warning, or faulty. External variables such as parameters, reference values, and predefined thresholds can be passed to the block to configure the processing algorithm. The outputs of one block may become the inputs of another (if any).

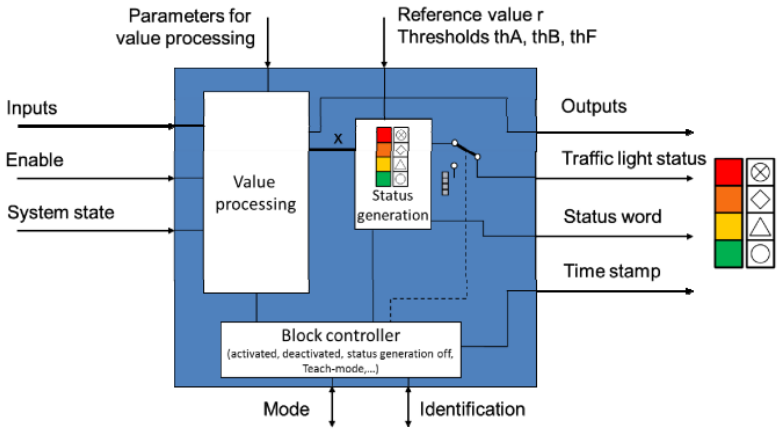


FIGURE 2.12: A functional block of VDMA 24582 [231].

[9]Integrated Vehicle Health Management is a system that provides the tools and assets to enable efficient, reliable, and autonomous vehicles. It makes use of large datasets collected from sensors and processes these data on-board and off-board to monitor the condition of the vehicle.

Additionally, VDMA 24582 decouples condition monitoring in three views, forming a hierarchy from the lowest to highest level:

(1) The **function view** defines the condition monitoring-related functions, formulated by functional blocks. This view addresses the question "what is to be monitored".
(2) The **application view** defines the relationships between functions. It dictates how the outputs of separate components of the equipment can be aggregated to produce an overall report.
(3) The **automation view** describes how and where the function blocks can be physically deployed within the automation system.

VDMA 24582 was published in 2014 and has since remained active.

MIMOSA OSA-EAI OSA-EAI, short for *Open System Architecture – Enterprise Asset Integration*, is another open standard developed by MIMOSA since 1998 and has been regularly updated, with the latest release in May 2014[10]. OSA-EAI defines the data structures for storing and exchanging information about all aspects of equipment, e.g., health report, monitoring values, operational records, into enterprise applications [159]. OSA-EAI complements OSA-CBM by providing reference schemas and data elements. OSA-CBM is responsible for the processing and analysis of equipment operational data, whereas OSA-EAI is in charge of data exchange and storage.

However, OSA-EAI suffers an excessive normalization that increases the complexity and reduces the flexibility of analysis in a CBM system [155]. Furthermore, the lack of documentation makes it difficult to fully grasp the concepts of this standard, as very few explanations are given on the official website of MIMOSA OSA-EAI.

ISO 13373 ISO 13373 draws close attention to the monitoring of *vibration signals*. Vibration signals are emitted by rotating machinery, one of the most common classes of machines [95]. Given the importance of vibration signals in detecting rotating machinery health, ISO develops ISO 13373 dedicated to the multifaceted vibration condition monitoring. It comprises nine parts, some of which are still under development (Table 2.3). The first three parts give general guidelines for analysis of vibration data and vibration diagnosis. The remaining parts focus on the diagnostic techniques for different types of rotating machinery.

Table 2.3: Nine parts of ISO 13373

Part	Name	Latest release
13373-1	General procedures	2002
13373-2	Processing, analysis, and presentation of vibration data	2016
13373-3	Guidelines for vibration diagnosis	2015
13373-4	Diagnostic techniques for gas and steam turbines with fluid-film bearings	2021
13373-5	Diagnostic techniques for fans and blowers	2020
13373-6	N/A	N/A
13373-7	Diagnostic techniques for machine sets in hydraulic power generating and pump-storage plants	2017
13373-8	N/A	N/A
13373-9	Diagnostic techniques for electric motors	2017

[10] https://www.mimosa.org/mimosa-osa-eai/

The guidelines of ISO 13373 are useful if the targeted equipment is a type of rotating machinery. Otherwise, ISO 13374 and OSA-CBM can be applied for any type of equipment.

3.2.2 Standards for diagnostics

The HA block of OSA-CBM covers *diagnostics*. Diagnostics is the process of examining the symptoms observed from the equipment to determine the nature of the faults or failures [106]. From the outputs of condition monitoring, a diagnostics determines the current health of the equipment and investigates the faults or failures that may already occur. Diagnostics indicates whether an abnormality is occurring in the system (fault detection), locate the faulty components (fault isolation), and determine the nature of the fault (fault identification) [113].

ISO 13379 ISO 13379 describes the diagnostic procedures and techniques with an emphasis on the data-driven approach. First, a preliminary study of the machine characteristics prepare the technical requirements and make clear of the major components and functions of the targeted equipment, its maintainability and criticality, the analysis of its failure modes, causes and symptoms, and a list of measurements from which relevant indicators can be derived for diagnostics. These tasks can be carried out using the Failure Mode Symptoms Analysis (FMSA) process.

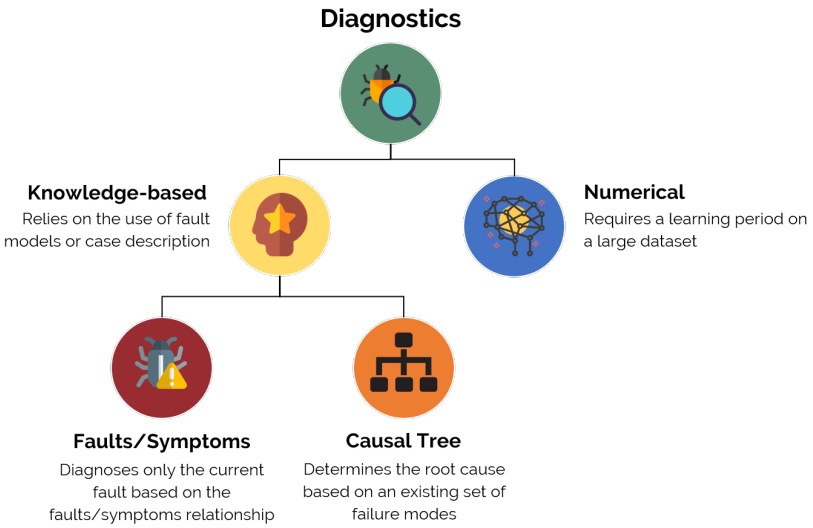


FIGURE 2.13: Two approaches of diagnostic techniques, according to ISO 13379.

After the study, a diagnostic approach must be selected. ISO 13379 categorizes diagnostic techniques into two approaches (Figure 2.13). The *knowledge-based* approach relies on fault models and requires sufficient understanding of the machine characteristics and of the fault propagation mechanism. Two possible techniques to establish a fault model are the Faults/Symptoms approach and the Causal Tree approach. On the other hand, the *numerical* approach requires a learning period on a large dataset using methods such as statistical learning, neural networks, pattern recognition, to name a few. This approach, also referred to as data-driven, is the main focus of ISO 13379-2 [107].

IEEE 1232 The full name of IEEE 1232 is the IEEE Standard for *Artificial Intelligence Exchange and Service Tie to All Test Environments*, or AI-ESTATE. AI-ESTATE standardizes the exchanging of diagnostic information in test environments and provides a framework that identifies the information required for diagnostics and how this information can be expressed in a machine-processable way [101].

Specifically, AI-ESTATE proposes a set of formal models of diagnostic information (Figure 2.14). Based on the object-oriented paradigm, these models are structured such that a Common Element Model (parent class) defines the common elements and attributes for all test and diagnostic information, and a set of derived models (child class) inherit and specialize the constructs of their parent class using different diagnostic reasoners for specific applications.

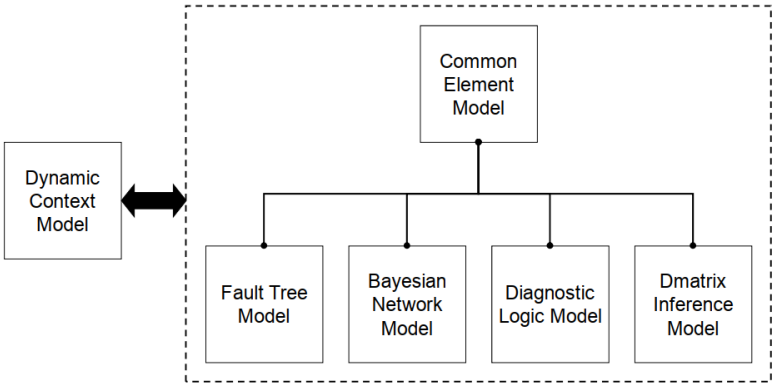


FIGURE 2.14: Structure of diagnostic models in IEEE AI-ESTATE [101]

AI-ESTATE was first published in 1995 and has been revised several times, with the latest version released in 2010. It is branched out to other subparts such as IEEE 1232.1 (Data and Knowledge Specification in 1997), IEEE 1232.2 (Service Specification in 1998), and IEEE 1232.3 (Use Guide in 2014).

3.2.3 Standards for prognostics

In OSA-CBM, the PA block takes charge of *prognostics*. Prognostics is the analysis of the symptoms of faults to predict future condition and residual life within design parameters [106]. In other words, prognostics determine how soon and how likely a fault will occur, given the current usage profile of the equipment. Prognostics is often linked to the estimation of the RUL, i.e., the amount of time from the current instant to the moment a fault is predicted to occur.

ISO 13381 ISO 13381 provides the guidance for the development and application of prognostics, determines the types of necessary data, and proposes appropriate approaches, namely performance changes approaches (13381-2), cyclic-driven life usage techniques (13381-3), and useful-life-remaining models (13381-4).

At present, only the first part (ISO 13381-1 [112]) is published, while the others are still under development. ISO 13381-1 defines prognostics as a sequential process of four steps (Figure 2.15).

FIGURE 2.15: Four steps of prognostics (FM = Failure Modes)

(1) Preprocessing prepares the data for prognostic analysis. The failure modes (FM) are identified along with their relations, symptoms, parameters, and relevant indicators.

(2) Existing FM prognostics studies the existing FM in the equipment and their severity. Then, it estimates the time to the next occurrences of the existing FM.

(3) Future FM prognostics predicts the future FM that may be the consequences of the existing ones in step **(2)**. An estimated time to failure is also given for each potential FM.

(4) Post-action prognostics recommends maintenance actions to undertake to avoid, reduce, or delay the FM effects. Additionally, it re-evaluates the prognosis result if these actions are taken. The confidence level of the estimated time to failure is calculated based on the outputs of the previous steps.

ISO 13381 suggests a systematic procedure to undertake when implementing the prognostic module of a CBM system. It assists practitioners to gain more understanding of the prognostic process, the details of which are not included in the description of the PA block in OSA-CBM.

IEEE 1856 IEEE 1856 is dedicated to Prognostics and Health Management (PHM) of electronic systems [102]. It is a recent standard published in 2017. According to IEEE 1856, PHM is "the approach to protect the integrity of equipment and avoid unanticipated operational problems leading to mission performance deficiencies, degradation, and adverse effect on mission safety". PHM is meant to "encompass both the monitoring and data processing functions [...] used to proactively manage and restore the system health". Hence, the goal of PHM coincides significantly with that of CBM/PdM. We may regard PHM as a systematic framework, while CBM/PdM is a philosophy of performing maintenance that can exist within PHM.

IEEE 1856 classifies the core principles of PHM in electronic systems via a normative framework. It helps practitioners to select the proper strategies for implementing PHM and the appropriate performance metrics to evaluate the PHM results. IEEE 1856 defines five core operational processes of a PHM system (Figure 2.16). Each process includes one or more functional blocks, some of which are already defined in ISO 13374 (DA, DM, SA, HA, PA, AG). Two new functional blocks added by IEEE 1856 are Sense (S) and Health Management (HM).

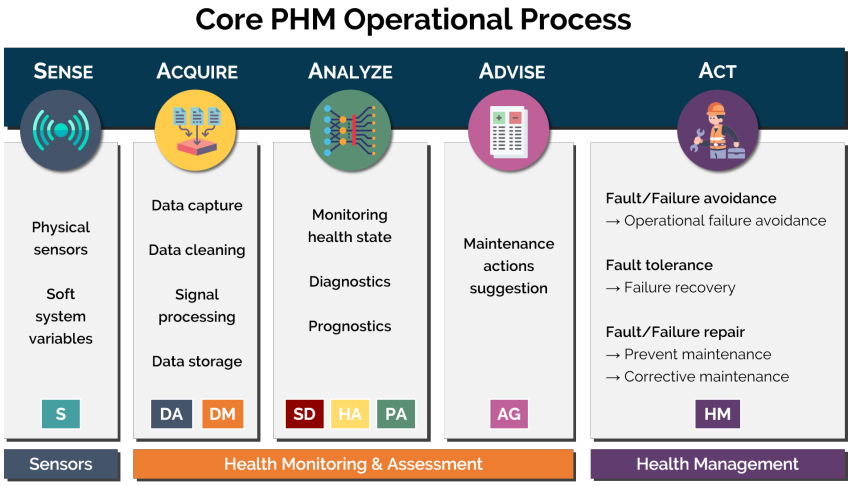


FIGURE 2.16: Overview of the operational processes of a PHM system

(1) Sense is a low-level process that manages the sensors. (Sensor functional block)

(2) Acquire collects and digitizes raw signals (DA). It applies signal processing algorithms and other statistical methods to derive indicators from the processed signals (DM).

(3) Analyze carries out condition monitoring (SD), performs diagnostics to detect faults (HA) and prognostics to predict future faults (PA).

(4) Advise generates optimal actions considering the result of the **Analyze** process (AG).

(5) **Act** transforms the suggestions from the **Advise** process to real-life actions, such as maintenance schedule, spare parts allocation, and worker assignment.

Overall, IEEE 1856 maintains the specification of OSA-CBM/ISO 13374 and adds more functionalities to construct a complete PHM framework. Since IEEE has developed a series for its standard 1451 on smart transducer interface for sensors and actuators, it may explain why IEEE 1856 extends the PHM framework to low-level processing like the **Sense** process. This process is normally not covered in other standards.

PdM standardization: A summary

In this section, we discuss the standardization of PdM and present some maintenance-related standards. MIMOSA OSA-CBM is the de facto, public standard for condition-based/predictive maintenance. It covers a full range of functionalities: data acquisition, data manipulation, state detection, health assessment, prognostics assessment, and advisory generation. We will apply the principle of OSA-CBM to the implementation of PdM in this thesis.

There are other standards developed by different organizations: ISO, IEEE, VDMA, IEC, SAE, to name a few. They aim at customizing, extending, or detailing one or more functionalities described in OSA-CBM. VDMA 24582, ISO 13373, and MIMOSA OSA-EAI focus on condition monitoring, emphasizing different aspects. ISO 13379 and IEEE 1232 AI-ESTATE dive into the detail of the diagnostic process. ISO 13381 and IEEE 1856 concentrate on the prognostic process. Table 2.4 summarizes the standards we have mentioned in this section and the year of their latest release.

Generally, there are very few standards for the advisory and maintenance action scheduling. It is an important functionality that valorizes the outputs of condition monitoring, diagnostics, and prognostics. It suggests a set of optimized actions according to the analysis results and guides practitioners towards a good maintenance of their equipment. The lack of standardization in these aspects can be complemented by the Computerized Maintenance Management Systems (CMMS). CMMS are software solutions that manage databases of information about an organization's maintenance operations. Examples of such information are work orders, worker scheduling, parts allocation, inventory control, purchase plans and budgets, et cetera.

Having reviewed the techniques for PdM and the PdM-related industrial standards, we discover that online learning has not had many applications in this domain. In the next section, we will discuss the principles of online learning, review several representative algorithms of the field, and explain how online learning can be beneficial for PdM in the railway industry.

4 Online learning on data streams

In most common scenarios, data are processed *offline* and by *batches*. Offline batch processing means the entirety of data are processed at once and the outputs are returned at the end. The same principle applies for offline batch machine learning: a model reads a complete batch of data and iterates over the data until the training finishes. However, the assumption that the data are bounded is not realistic. A company running a business that produces data today will continue producing data tomorrow if it maintains its operation, therefore the data are unbounded and arrive ceaselessly. This is referred to as a *stream*: the data that are continuously made available over time [126].

Streaming data are a derivation of big data: both are characterized by a huge amount of data that are too large to be fully loaded in memory for processing. The key difference is the *unboundedness* of

TABLE 2.4: Summary of the standards and the year of their latest release

Release	Standard	Title
2002	ISO 13373-1	Condition monitoring and diagnostics of machines – Vibration condition monitoring – Part 1: General procedures
2003	ISO 13374-1	Condition monitoring and diagnostics of machines – Data processing, communication and presentation – Part 1: General guidelines
2007	ISO 13374-2	Condition monitoring and diagnostics of machines – Data processing, communication and presentation – Part 2: Data processing
2010	IEEE 1232	IEEE Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE)
	MIMOSA OSA-CBM	Open System Architecture - Condition-Based Maintenance
2012	ISO 13379-1	Condition monitoring and diagnostics of machines – Data interpretation and diagnostics techniques – Part 1: General guidelines
	ISO 13374-3	Condition monitoring and diagnostics of machines – Data processing, communication and presentation – Part 3: Communication
2014	VDMA 24582	Fieldbus neutral reference architecture for Condition Monitoring in production automation
	MIMOSA OSA-EAI	Open System Architecture – Enterprise Asset Integration
2015	ISO 13379-2	Condition monitoring and diagnostics of machines – Data interpretation and diagnostics techniques – Part 2: Data-driven applications
	ISO 13374-4	Condition monitoring and diagnostics of machines – Data processing, communication and presentation – Part 4: Presentation
	ISO 13381-1	Condition monitoring and diagnostics of machines – Prognostics – Part 1: General guidelines
	ISO 13373-3	Condition monitoring and diagnostics of machines – Vibration condition monitoring – Part 3: Guidelines for vibration diagnosis
2016	ISO 13373-2	Condition monitoring and diagnostics of machines – Vibration condition monitoring – Part 2: Processing, analysis and presentation of vibration data
2017	IEEE 1856	IEEE Standard Framework for Prognostics and Health Management of Electronic Systems
	ISO 13373-7	Condition monitoring and diagnostics of machines – Vibration condition monitoring – Part 7: Diagnostic techniques for machine sets in hydraulic power generating and pump-storage plants
	ISO 13373-9	Condition monitoring and diagnostics of machines – Vibration condition monitoring – Part 9: Diagnostic techniques for electric motors

streaming data: a stream is generated continuously, infinitely, and possibly at high speed. Examples of data streams are web clicks of users on a website, sensor data generated by equipment in operation, medical ECG signals of a patient, and so on. Streaming is associated to real-time services, therefore, *reactivity* and *availability* are crucial factors. Dynamic changes may appear on the stream, making the distribution of the data shifts over time, a phenomenon coined as *concept drift*, thus *adaptivity* is also a requirement facing dynamic changes.

Five requirements of learning from data streams are identified to handle each of the aforementioned characteristics (Figure 2.17).

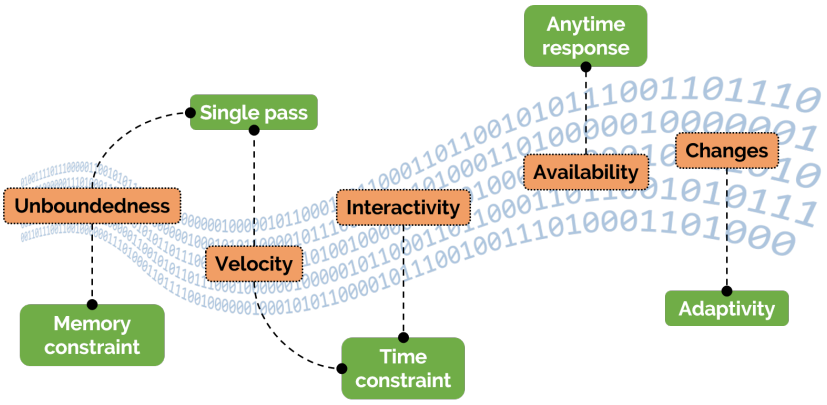


FIGURE 2.17: Mapping data streams' special characteristics to the requirements for learning.

- **Single pass processing.** The incoming data are processed once and discarded[11] to make place for newly arriving data.
- **Time constraint.** Unlike batch learning, a highly available online learning model must learn from new data quickly, depending on the arrival rate of the data.
- **Memory constraint.** Memory consumption may also be an issue if the learning algorithm is deployed on devices with low computing capacity, such as sensors for edge learning. Lightweight data structure and efficient algorithms are a necessity.
- **Anytime response.** An online learning model must enable anytime-response to handle real-time queries.
- **Adaptive mechanism.** To cope with concept drifts, an online learning algorithm must be aware of changes and react quickly to these novelties from the stream.

4.1 Terminology

We may encounter different terms in the literature of learning from data streams: incremental learning, stream learning, and online learning. Sometimes these are used interchangeably. Losing, Hammer, and Wersing [148] define an *incremental learning* algorithm as one that generates a sequence of models h_1 to h_t from a stream of t data instances from s_1 to s_t. They view online learning as incremental learning bounded by time and memory constraint, which enables endless learning on a device with restricted resources. Meanwhile, Gomes, Read, Bifet, Barddal, and Gama [81] associate all three terms to leaning algorithms that update the models immediately as new data become available without revising historical data. Gama [73] states that, to perform stream learning, incremental learning is necessary but not sufficient; although incremental algorithms can incorporate new data into the models, they do not cope very well with non-stationary data. Karp [118] notes that an online algorithm is one that performs an action immediately after receiving a new request. Hoi, Sahoo, Lu, and Zhao [98] consider online learning a subfield of machine learning that comprises learning techniques "devised to learn models incrementally from data in a sequential manner", while incremental learning is to learn a model from a data source with memory and time constraints. Incremental learning can work in both online and batch setting. Incremental learning in batch setting is relevant when the amount of data cannot fit in the memory all at once.

In the scope of our work, we consider incremental learning the foundation of learning on data streams, as all algorithms for learning on data streams are incremental in nature. We refer to learning

[11] By "discarded", it means the data are not kept in memory, but they can be persisted in hard disks.

on data streams as **online learning** (or *online machine learning*), which encompasses incremental techniques with time and memory constraints and concept drift awareness.

4.2 Paradigms of online learning

We review the techniques for online learning (OL) in different paradigms in online setting. As it is a vast research field, we do not intend to make a comprehensive survey and cover only the learning algorithms. More in-depth analyses on various aspect of mining from data streams, such as preprocessing or feature transformation, are found in the following surveys.

Hoi, Sahoo, Lu, and Zhao [98] comprehensively review OL on a large range of learning tasks, such as online transfer learning, bandit online learning, online active learning, and particularly emphasizes the theoretical guarantee and mathematical formulation of OL algorithms, backed by three major theory communities, namely learning theory, optimization theory, and game theory. Gomes, Read, Bifet, Barddal, and Gama [81] establish a wide landscape of techniques for data stream mining, ranging from stream processing (feature scaling, dimensionality reduction), to learning algorithms and handling drifts. Bahri, Bifet, Maniu, and Gomes [22] focus on dimensionality reduction techniques and investigate how they could be made incremental and online-compatible. Losing, Hammer, and Wersing [149] review and benchmark eight incremental learning algorithms. For specific tasks, Carnein and Trautmann [47] conduct an extensive study of 51 stream clustering algorithms, and Salehi and Rashidi [200] study algorithms for online anomaly detection.

Next, we will formulate the problem of online learning from data streams, show how an OL algorithm is usually evaluated in the online setting, and present the OL algorithms divided in different paradigms.

Let $D = ((x_1, y_1), (x_2, y_2), \dots, (x_t, y_t))_{t\to\infty}$ be a stream of infinite number of data instances[12]. Each instance (x_t, y_t) is a tuple of the instance features $x_t \in \mathbb{R}^D$, and (if available) the label(s) $y_t \in \mathcal{Y}^L$ classifying x_t. Given the label(s) $y_t \in \mathcal{Y}^L$, if $L = 1$, it is a single-output prediction task. If $L > 1$, it is a multi-output prediction task. If $\mathcal{Y} \subseteq \mathbb{N}$, it is a classification task. If $|\mathcal{Y}| = 2$, it is a binary classification, else a multiclass classification. If $\mathcal{Y} \subseteq \mathbb{R}$, it is a regression task.

On a continuous stream, traditional evaluation settings commonly used for batch learning such as k-fold cross validation or train-test splitting are not applicable. To evaluate OL algorithms, two main approaches arise [34]:

- **Holdout evaluation**. A single set of data instances is put aside for testing once the learner has been trained. This approach is useful when a train-test set has been well-defined for a given data set, thus allowing the comparisons of different OL methods [34].
- **Prequential evaluation**. For each incoming data instance x_t, the learner first issues a prediction $\hat{y}_t$, then $\hat{y}_t$ is compared to the real label y_t to compute the prediction loss $\mathcal{L}(y_t, \hat{y}_t)$. The learner is finally updated on both x_t and $\mathcal{L}(y_t, \hat{y}_t)$ (Figure 2.18). This approach allows the learner to monitor its own performance continuously [74].

We classify OL techniques based on the amount of labels available for learning. The three primary classes are *supervised learning* (fully labeled data), *semi-supervised learning* (partially labeled data), and *unsupervised learning* (no labeled data) (Figure 2.19). In addition, *active learning*, which queries for labels from an external agent, is a subclass of semi-supervised learning. Notable subclasses of unsupervised learning are *clustering* and *anomaly detection*. On a data stream $D = ((x_1, y_1), \dots (x_t, y_t)]_{t\to\infty}$,

[12] We use the terms "data instances", "data examples", and "data points" interchangeably.

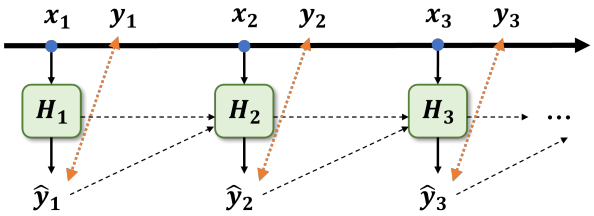


FIGURE 2.18: Prequential (test-then-train) evaluation: H_t denotes the model updated at the instant t.

a supervised task is applicable if y_t is available for each x_t. For a semi-supervised task, some x_t may not have y_t. Unsupervised tasks have no information about y_t.

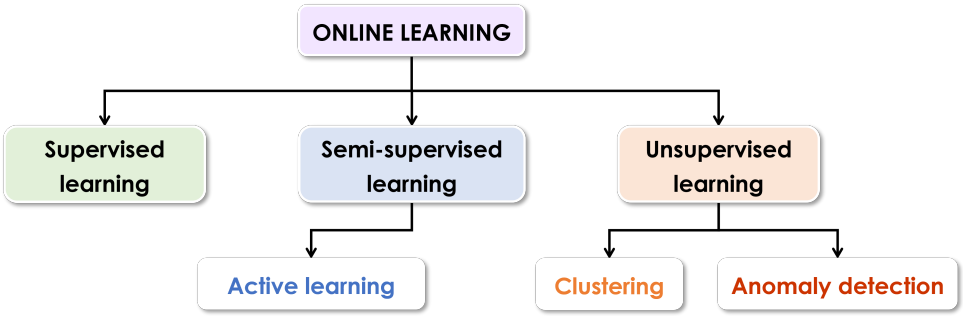


FIGURE 2.19: Classification of OL techniques.

Online learning on data streams may propose various ways to deal with concept drifts. Details about concept drift handling will be given in Section 4.3.

4.2.1 Supervised learning

In this section, we review supervised techniques for online classification and online regression, supposing that a label is available for each data example.

CLASSIFICATION Similar to traditional batch learning, OL classification techniques include tree-based methods, linear models, Naïve Bayes, nearest neighbors, support vector machines, and neural networks.

Tree-based methods The most well-known tree-based online classifier is the Very Fast Decision Tree (VFDT) [60], often referred to as the Hoeffding tree, dated back to 2000. VFDT is built incrementally by exploiting the Hoeffding's bound to select the best splitting attribute. Let X_a and X_b the best and second-best attributes at a node, $\bar{G}$ the average observed thus far of a predefined heuristic measure (e.g., information gain) on that node, and δ the error tolerance threshold, the Hoeffding bound guaranteed that $\Delta\bar{G} = \bar{G}(X_a) - \bar{G}(X_b) > \epsilon$ and X_a is thus the best splitting attribute with the probability $1-\delta$ (δ preferably very small), where $\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$ defines the confidence interval of the true mean $G = \bar{G} \pm \epsilon$, n the number of instances seen on that node thus far, R the value range of the data.

However, recent works found out that the Hoeffding bound was incorrectly used in [60] and proposed other bounds to correct it [156, 197]. To make the Hoeffding tree more adaptive to time-changing streams, Hulten, Spencer, and Domingos [100] propose an extension of the Hoeffding called CVFDT (Concept-adapting VFDT) by creating and replacing alternative subtrees when a change in the data

distribution is detected, thus maintaining the tree consistent with the current sliding window. Bifet and Gavaldà [32] further improve CVFDT by incorporating a concept drift detector in each node of the tree, eliminating the need for a fixed-size sliding window over the stream.

Linear models Perceptron [194] is the oldest algorithm for online learning [98]. A perceptron defines a linear hyperplane that separates the instances into two classes. For each incoming instance $\mathbf{x} = (x_1, ..., x_N) \in \mathbb{R}^N$, the perceptron predict its label by computing $f(\mathbf{x}) = \text{sign}(\langle w, \mathbf{x} \rangle)$ and classifies $\hat{y} = +1$ if $f(x) > 0$, and $\hat{y} = -1$ otherwise (Figure 2.20). The perceptron maintains a set of weight $\mathbf{w} = [\, w_1, \ldots, w_N \,]^T$ and updates it based on the Hinge loss of the prediction $\hat{y}$ and true label y (2.11). If the loss is 0, meaning the prediction $\hat{y} = f(\mathbf{x})$ is correct, no update is needed. Otherwise, the weights $\mathbf{w}$ are updated such that $w_{t+1} = w_t + \eta y \mathbf{x}$, with η being the learning rate.

$$\mathcal{L}(\mathbf{w}; (\mathbf{x}, y)) = \begin{cases} 0 & \text{if } y\langle w, x \rangle \geq 1 \\ 1 & \text{otherwise} \end{cases} \tag{2.11}$$

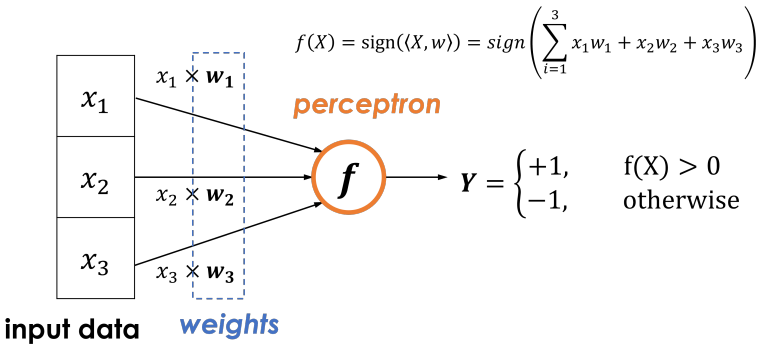


FIGURE 2.20: A simplified view on the perceptron model.

Based on perceptron learning, another margin-based linear technique is the Passive Aggressive (PA) algorithm [52]. Its goal is to maintain w_t as close as possible to w_{t-1} to retain information learned previously (passive), while striving to classify the example $\mathbf{x}_t$ with a sufficiently high margin to clearly separate the classes (aggressive). To achieve it, the PA algorithm modifies the weight update rule by framing it as a constrained optimization problem (2.12). For simplicity, we set $\mathcal{L}(\mathbf{w}; (\mathbf{x}_t, y_t)) = \mathcal{L}_t$. The optimization problem in (2.12) has a simple close-form solution as shown in (2.13). The update is passive when $\mathcal{L}_t = 0$, that is, $\mathbf{w}_{t+1} = \mathbf{w}_t$, and the weights are kept intact when the prediction is correct. Otherwise, it *aggressively* forces $\mathbf{w}_{t+1}$ to satisfy $\mathcal{L}_t = 0$ to teach the learner to correctly classify the example.

$$\mathbf{w}_{t+1} = \underset{w \in \mathbb{R}^N}{\text{argmin}} \frac{1}{2} \|\mathbf{w} - \mathbf{w}_t\|^2 \quad \text{such that} \quad \mathcal{L}_t = 0 \tag{2.12}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \tau_t y_t \mathbf{x}_t \quad \text{where} \quad \tau_t = \frac{\mathcal{L}_t}{\|\mathbf{x}_t\|^2} \tag{2.13}$$

Naïve Bayes Naïve Bayes assumes independence between the attributes and uses Bayes' rule to predict the most likely class for a given example [115]. Given an instance $\mathbf{x} = (x_1, \ldots x_N)$ and the classes $C = (c_1, \ldots c_L)$, the probability that $\mathbf{x}$ belongs to a class c can be updated using the Bayes' rule (2.14),

where $p(C = c)$ is the fraction of a class label c over the entire sample, $p(X = \mathbf{x})$ is normally not directly estimated, instead, a normalization makes the conditional probabilities of each class sum up to 1. By the independence assumption: $p(\mathbf{X} = \mathbf{x} \mid C = c) = p(\bigcap_{1 \leqslant i \leqslant N} X_i = x_i \mid C = c) = \prod_{i=1}^{N} p(X_i = x_i \mid C = c)$, which is easily computed by counting the value occurrence of an attribute X_i by the class c.

$$p(C = c|\mathbf{X} = \mathbf{x}) = \frac{p(C = c)\, p(\mathbf{X} = \mathbf{x} \mid C = c)}{p(\mathbf{X} = \mathbf{x})} \tag{2.14}$$

Naïve Bayes is online-compatible. It suffices to store the count of each attribute value by classes. It is time and memory efficient, as instances need not to be stored explicitly to make it work. Nonetheless, attribute independence is a strong assumption that does not always hold in practice. Naïve Bayes risk to wrongly predict instances of an emerging class, for example, a class c_{N+1} that appears recently. Because the count associated to c_{L+1} has not been sufficiently accumulated, Naïve Bayes does not return the correct class probability, in contrary to batch Naïve Bayes that has access to all the instances to collect the correct counts.

Nearest neighbors K-Nearest Neighbors (kNN) classifies an instance $\mathbf{x}$ by aggregating the class label of the k closest neighbors to $\mathbf{x}$. It is a *lazy* learning method because it only issues predictions when requested and no learning is involved in the process. Beringer and Hüllermeier [30] apply kNN to online classification with a sliding window to make the learner detect to concept drifts. A set of k nearest neighbors C is retrieved for each new instance $\mathbf{x}_t$. Half of the oldest examples in C are put to a statistical test to check for drifting. If a change is detected, these oldest neighbors are discarded.

The vital requirement and also the most important weakness of kNN is data storage to retrieve neighbors for any new instance. Yet, it is infeasible to store the entire stream. Usually, only a window of moderate size can be maintained and all the past points are forgotten, which is problematic because past data may reflect the stable characteristics of the stream. On the other hand, storing too many instances can slow down the search considerably, affecting the efficiency of kNN.

Support vector machines Support vector machines (SVM) is a margin-based classifier that separates the examples of two classes by finding the hyperplane that maximizes the distance from all training examples, also called the *margin* (Figure 2.21). Let x be a training example, w the normal vector of x to the hyperplane $wx - b = 0$, SVM aims to maximize the margin $\frac{2}{||w||}$, i.e., to minimize $||w||$. Thus, SVM solves the following optimization problem:

$$\min ||w|| \quad \text{subject to} \quad y_i(w^T x_i - b) \geqslant 1, \forall i = 1 \ldots n$$

The support vectors are those that lie near the hyperplane and whose position significantly affect the position of the hyperplane, denoted $\vec{x}_i$. So, SVM in online settings must incrementally update these support vectors. One of the issues is the exploding number of support vectors if the data keep coming over time. Agarwal, Vijaya Saradhi, and Karnick [3] propose a method that occasionally discard support vectors to fit in the memory limit. Kivinen, Smola, and Williamson [125] leverage stochastic gradient descent to make kernel-based learning, including SVM, more efficient in an online setting.

Neural networks Neural networks are compatible for incremental learning because they can learn from a small batch of data at a time. The weights can be updated incrementally using stochastic gradient descent (SGD). SGD is a simplification of the traditional gradient descent (2.15), which updates

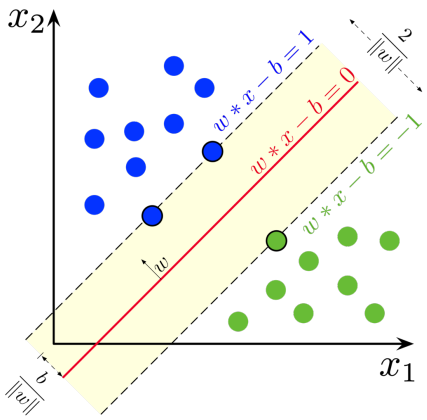


FIGURE 2.21: Support vector machine (source: https://commons.wikimedia.org/w/index.php?curid=73710028)

the weights w by adding the gradient of the loss $\mathcal{L}(f(X), Y)$ between the prediction $f(x_i)$ and the ground truth y_i computed over the entire dataset in one epoch t, attenuated by a learning rate η. But for large-scale training, it may be infeasible to iterate through the entire data set to compute gradients, therefore SGD simplifies the process by picking one random data example at a time to update the weight (2.16) [37]. However, SGD applied to non-convex loss functions does not benefit from convergence guarantee and is sensitive to the initial value of the weights [83].

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla_{\mathbf{w}} \mathcal{L}(f(x_i), y_i) \tag{2.15}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla_{\mathbf{w}} \mathcal{L}(f(x_t), y_t) \tag{2.16}$$

Powerful as they are, deep neural networks are data-hungry , which contradicts the memory constraint of OL. Besedin, Blanchart, Crucianu, and Ferecatu [31] implement a generative neural network (GAN) to generate new labeled samples from few real labeled examples to overcome the need for data storage. These synthetic samples are then inputted to a deep convolutional network for online learning and classification of real unlabeled samples. Sahoo, Pham, Lu, and Hoi [199] address online deep learning with hedge backpropagation (HBP). The key idea of HBP is to start with an overcomplete network (many layers) then adapt the network depth dynamically. Pratama, Ashfahani, Ong, Ramasamy, and Lughofer [188] propose a deep evolving denoising autoencoder that adds and discards hidden units on-the-fly using a novel network signification (NS) measure. The network is trained with SGD in a single-pass in one epoch. The network is evaluated in a prequential fashion and achieves competitive results with other benchmarking algorithms.

Overall, online deep neural networks remains an open challenge, since most of the research effort aims at training complex networks offline on large amount of data. Ongoing research consists of adaptively changing the network structure, efficiently update the networks on-the-fly, avoiding catastrophic forgetting via continual learning [87, 232].

REGRESSION Regression algorithms for data streams can be tree-based, rule-based, or adapted from support vector machines, called the support vector regression.

Tree-based Ikonomovska, Gama, and Džeroski [103] present the Fast Incremental Model Trees with Drift Detection (FIMT-DD) that has become a popular regression tree algorithm for data streams. It shares the same principle as the Hoeffding tree: the tree is grown continuously on new data and the best splitting attributes are chosen using the Hoeffding's bound. In each leaf, a simple linear model is updated when a new instance is routed there and performs regression for any unlabeled instance that falls in this leaf. Gomes, Barddal, Boiko Ferreira, and Bifet [79] develop an adaptive random forest using the FIMT-DD as the base learner and enhance the model with an external change detector to cope with drifting.

Rule-based Almeida, Ferreira, and Gama [9] propose the Adaptive Model Rules from High Speed Data Streams (AMRules), a one-pass algorithm for learning regression rules from time-evolving streams. It starts with an empty set and expands or removes rules based on new data. Each rule contains a linear model trained with incremental gradient descent on the examples covered by this rule. The regression estimates given by all the rules that cover an unlabeled example are averaged to make the final regression.

Support vector regression Ma, Theiler, and Perkins [152] is the first to propose online support vector regression (SVR) that updated a trained SVR on new data. SVR has three parameters: the kernel $\mathcal{K}$, the regularization term C to control the bias-variance tradeoff, and the slack variables ϵ_i for each data example $\mathbf{x}_i$ that allow an instance to be on the wrong side of the hyperplane, such that the sum of all ϵ_i is bounded by C. However, these parameters are constant and cannot cope with potential drifts from dynamic streams. Omitaomu, Jeong, and Badiru [177] amend this by devising a weight function to adaptively update C and ϵ_i when drifts are detected. Yu, Lu, and Zhang [246] develop a continuous SVR for non-stationary data streams by learning a series of regressors f_i from the sliding windows tw_i on the stream and keeping only one regression f_i in memory for prediction. Knowledge learned from the previous window tw_{i-1} was transmitted to the classifier in the current window tw_i instead of being dropped completely. They also devise a way to incrementally update the Lagrange multipliers on one data example per timestep.

4.2.2 Semi-supervised learning

Assuming that a data stream is fully labeled is unrealistic, because the data may arrive so fast that manual labeling is practically infeasible to cope with the stream speed. There is also the problem of mislabeling or delayed arrival of the labels. The latter occurs when a data example arrives first then its label only reaches the stream several instants later, possibly due to network delay. Therefore, a stream could hardly fit in a supervised scenario in practice.

One way to overcome this difficulty is to train a model solely on labeled examples and formulate the problem as supervised learning, but the amount of unlabeled data may greatly exceed that of labeled data. Another way is to use *semi-supervised learning* (SSL) to leverage both labeled and unlabeled data to produce better classifiers than those that rely solely on limited amount of labeled data. There are many approaches to tackle SSL, including co-training [35], self-training [243], or active learning [201].

Active learning Active learning is a human-in-the-loop learning paradigm that interacts with an oracle to annotate unlabeled data instances. Usually, the instances to be labeled are the most uncertain, for example, a point lying close to the discriminative hyperplane. The oracle's answer can then bring the highest value to the learning process. Under a budget constraint, the active learner aims to achieve

high accuracy using as few labeled instances as possible, thereby minimizing the cost of collecting labeled data [201].

There are several concerns regarding this approach. First, it assumes that any instance can be labeled, which may not be true (it depends on the domain). Secondly, it includes one or multiple humans experts in the learning process to manually label queried instances, but they may not cope well with the speed and the volume of a real-time data streams, not to mention that humans may sometimes give incorrect labels and not give any labels at all [218].

Two traditional sampling strategies that perform well in stationary settings are random (R) and fixed uncertainty (FU). Nonetheless, R is a naïve strategy that requested labels randomly with a predefined probability, whereas FU might issue labeling requests excessively and soon exhaust the labeling budget. Žliobaitė, Bifet, Pfahringer, and Holmes [255] propose two novel online active learning strategies - variable uncertainty (VU) and randomized uncertainty (RU), that explicitly handle concept drift while keeping the budget balanced. VU chooses the least certain instances according an adaptive threshold to align with the budget. RU randomizes the labeling threshold to select both the instances close to the boundary and distant instances to amend the common issue of uncertainty strategies that tend to select instances near the decision boundary while changes occurring remotely from the boundary are likely missed.

Zhu, Zhang, Lin, and Shi [254] propose an ensembling based active learning framework using the minimal variance principle. Instances that cause the highest expected predictive error are selected for labeling. When a new data chunk S arrived, the learner predicts the label for all unlabeled instances $I \in S$, then it calculates the variance of I over all the class labels to obtain the expected ensemble variance. The unlabeled instance with the largest variance is sent to the oracle for labeling.

Singh and Chandak [210] implement ActMiner and make use of incremental ensembling to handle data streams and concept drift. Concept evolution occurs when novel classes appear on the stream. Instead of requiring all instances of the emerging class to be labeled, ActMiner only picks the instances that cause the highest expected error for manual labeling.

Haque, Khan, and Baron [89] argue that dividing a stream into fixed-size chunks fails to capture concept drift quickly if the chunk size is too large, or suffers from unnecessary updates during stable period if the chunk size is too small. They propose SAND and use an explicit change detector to detect drifts and to determine the chunk size dynamically. When a change is detected, a drift signal is emitted and the chunk boundary is determined immediately. SAND forms the training data from the current chunk including available labeled data and requesting true labels from human experts for the examples on which the model achieves weak prediction confidence. However, SAND is time-consuming due to its exhaustive invocation of the change detection module. An extension of SAND [90] amends this problem by exploiting dynamic programming and selective execution of the change detection module.

4.2.3 Unsupervised learning

Unsupervised learning applies when no information about the labels is available. This scenario is particularly relevant for machinery monitoring. It is not feasible to assign a label indicating the machinery health to each data example in the stream. An equipment or a fleet of equipment that has just started its operation have not yet produced any labels regarding its condition or failure time. Hence, unsupervised learning can be the starting point of learning from data streams. In this section, we review the two most prominent unsupervised learning tasks, which are *clustering* and *anomaly detection*.

Clustering Online clustering has been extensively studied for the past few years [13, 46, 47, 78, 209, 257]. Based on these surveys, we divide the clustering techniques in three primary classes: *hierarchical*, *partition-based*, and *density-based*. Then, we review the most representative algorithms of each class.

Hierarchical techniques Hierarchical techniques store the information about the cluster in a hierarchy, most often in the form of a tree. BIRCH [249] and ClusTree [131] are two well-known algorithms of this class.

BIRCH [249] is the first to propose the *cluster feature* to summarize a cluster, which is a tuple of three statistics: N the number of points in this cluster, $LS \in \mathbb{R}^D$ the linear sum of all points, and $SS \in \mathbb{R}^D$ the squared sum (D being the dimension of the data). The cluster features are maintained in a tree that is built incrementally. Starting from the root, a new data example descends the tree by following the child of its closest cluster at each branch until it reaches a leaf. Then, either the example is merged to the cluster at this leaf, or a new cluster is created.

ClusTree [131] is a parameter-free algorithm that automatically adapts to the speed of the stream. It relies on the index structure R-Tree to store and maintain a compact view of the current cluster features. It addresses the issue of fast streams where there may not be enough time to descend to the leaf or to split the node when inserting a new example (a split occurred when a node is full). If a leaf is reached and there is not enough time for a split (a new example arrived), the two closest entries are merged. If a new example arrives when the leaf has not yet been reached, the current example is stored in a temporary buffer at its current node and waits hitchhike with another example that descends the same branch.

Partition-based techniques Partition-based clustering splits the data into a predefined number of clusters (partitions) based on the distance between the examples [257]. These techniques are sometimes called *distance-based* techniques. Representative algorithms of this class are CluStream [5] and SWClustering [253].

CluStream [5] is the first to employ a two-phase clustering process for clustering data streams. The online phase efficiently captures and maintains summary statistics of the stream via micro-clusters. The offline phase applies a traditional clustering algorithm, such as k-means, on the micro-clusters to return official clusters if requested. CluStream extends the cluster features from BIRCH by adding the pyramidal timeframe that allowed the clustering over different time horizons.

SWClustering [253] uses the cluster features of BIRCH and the pyramidal timeframe of CluStream but it maintains these statistics in an exponential histogram to store the data efficiently in different levels of granularity, such that recent data are stored in greater details while older data are grouped and summarized.

Density-based techniques Density-based techniques examine the density of a region to group the examples together. Techniques of this class do not require a predefined number of clusters and can find clusters of arbitrary shape, in contrary to partition-based techniques that usually find spherical

clusters. Two most popular density-based online clustering algorithms are DenStream [45] and D-Stream [50][13].

DenStream [45] is an extension of its offline counterpart DBSCAN [64]. DenStream maintains a set of potential micro-clusters (PMC) and outlier micro-clusters (OMC) to detect real clusters and isolate outliers. Pruning is performed periodically to demote a PMC to an OMC if the PMC does not receive any new points recently, to promote an OMC to a PMC if the OMC has become dense, and to discard OMC that are noises.

D-Stream [50] divides the data space into small grids of fixed size and distinguishes dense, sparse, and transitional grids depending on their density. Dense grids that share adjacent facets constitute a cluster. Periodic pruning is performed to remove infrequent or empty grids.

DBSTREAM [88] studies closely low-density areas between two high-density areas to avoid mistakenly merging two separate clusters together. It introduces the shared density graph to explicitly capture the density area between micro-clusters. The shared density is updated via competitive learning: each time a new data example appears on the stream, all clusters that are in the vicinity of that example slightly inch closer to that example. Eventually, areas of high density may collapse and make DBSTREAM able to gradually adapt to concept drifts.

Anomaly detection Anomaly detection findings anomalous points with respect to the dominant data distribution of the data stream. Detecting anomalies from data streams directly relates to real-time monitoring, such as network intrusion detection and fault detection in machinery. Extensive studies on online anomaly detection have been conducted by Duraj and Szczepaniak [61], Gupta, Gao, Aggarwal, and Han [86], and Salehi and Rashidi [200].

Ahmad, Lavin, Purdy, and Agha [7] make use of the Hierarchical Temporal Memory (HTM) algorithm to detect outliers from noisy and drifting streams. By design, HTM cannot directly model anomalous objects from the input. The authors adapt HTM by stacking two additional components to the output of the HTM. The first component, "Prediction errors", measures the deviation of the HTM predictions to the real input, producing a prediction error. A distribution of prediction errors is modeled by the "Anomaly likelihood" component to assess whether an input is indeed an anomaly.

Manzoor, Lamba, and Akoglu [153] propose xStream to perform density-based anomaly detection in a feature-evolving data stream, that is, a stream where an instance does not arrive entirely with its full set of attributes, but the attributes arrive bit by bit over time. xStream uses a hash algorithm to project the instances to a low-dimensional space, then updates the half-space chains to compute the outlier score of an instance.

4.3 Concept drift

Concept drift occurs in non-stationary environments where the relation between the data and the target variable changes over time. This variability is often encountered in real-world applications, for example, in a recommendation system where the users may change their preferences over time. Concept drift may manifest in four forms [75] (Figure 2.22):

- **Sudden drifts:** the change occurs abruptly.

[13] Other authors may classify D-Stream as a grid-based technique. We consider density-based and grid-based techniques in the same category because they both cluster the data based on the density of local regions. While density-based techniques rely on distance metrics to estimate the density, grid-based techniques divide the data space into small grids, such that adjacent dense grids constitute a cluster.

- **Incremental drifts:** the relation changes little by little over time.
- **Gradual drifts:** the drift alternates between the new and old concepts.
- **Reoccurring/Recurrent drifts:** the new concept occurs temporarily, then recedes.

Sometimes, a sudden change in the data may not be a real drift but is a fleeting perturbation caused by noisy examples from the streams.

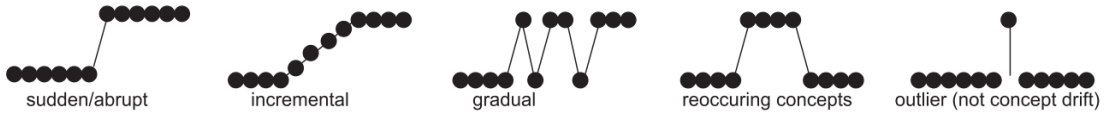


FIGURE 2.22: Concept drift in four forms: sudden, incremental, gradual, and reoccurring [75]

Concept drift is not the only type of dynamic changes on data streams. Other types of drift are *feature drift* (a subset of features becomes or ceases to be relevant to the learning task), *feature evolution* (new features become available or old features disappear), *concept evolution* (new class labels appear or existing class labels disappear) [81].

Naturally, an incremental algorithm can adapt to incremental drifts by continuously incorporating new data to the learning process. However, it may struggle to handle other forms of drifts. A sudden drift can decrease the model accuracy significantly until it receives enough data to finally adapt to the new concept. The same thing happens with recurrent drifts until the previous concept returns. Gradual drifts are most difficult to deal with - the worst scenario is when the model can never adapt because the concepts varies constantly [75].

Adapting to the arising of new concepts is realized via two mechanisms: learning new concepts and forgetting outdated ones. By design, learning new concepts from the stream is the default behavior of online algorithms. Within a limited memory space, incorporating new knowledge implies discarding the old one. Incremental learning and forgetting can be realized by the windowing techniques or via explicit drift detectors incorporated in a learner.

4.3.1 Time window model

Using a time window, a number of instances within a timeframe is retained to update the model at a time (*learning*). Any instances that fall out of this timeframe are discarded and forgotten by the model (*forgetting*). Three main types of windowing[14] are sliding window, landmark window, and damped window [47] (Figure 2.23).

SLIDING WINDOW MODEL A number of most recent examples from the stream is accumulated in a window of fixed or variable length. The model is kept up to date with the data stored in this window. Old instances are removed from the window to make space for more recent ones. The window size is a crucial parameter: a small window increases the model sensitivity to changes from the stream but it also causes frequent model updates, thus violating the time and memory constraints. Meanwhile, a large window implies slower adaptation to drifts, but the model performance is more stable when the stream is stationary.

[14]There is a fourth type of time window model which is the pyramidal timeframe, proposed by Aggarwal, Han, Wang, and Yu [5], but it is not commonly used.

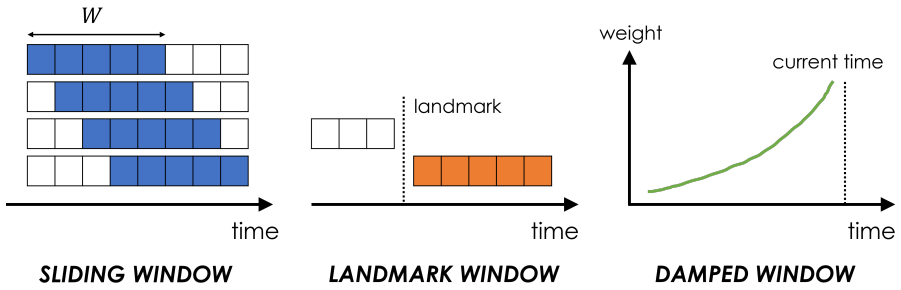


FIGURE 2.23: From left to right: sliding window, landmark window, damped window.

LANDMARK WINDOW MODEL A landmark window is a window that stores all the examples from a landmark point. The landmark can be an event that signals a change in the stream. When a landmark occurs, all the examples prior to it are forgotten and the window start accumulating the data anew. A landmark window may contain all the data from the beginning of the stream if no other landmark point is specified during the learning process, but this is infeasible for infinite streams.

DAMPED WINDOW MODEL Instead of fixing a limited budget to store the examples, the damped window model assigns to each example a relevance weight that decreases over time using exponential decay [45]. After each interval Δt, the weight w is decreased by an amount of $2^{-\lambda\Delta t}$, i.e., $w_{t+\Delta t} = w_t \times 2^{-\lambda\Delta t}$, where $\lambda > 0$ is the decay factor. A high value of λ means the data are quickly forgotten to place more importance to recent data.

4.3.2 Drift detectors

Adapting to drifts can be done implicitly via the windowing technique to let a model slowly forget old concepts, but the drift is not explicitly detected. Indicating where a drift occurs may bring insightful discoveries on the stream characteristics. This section presents explicit change detectors that identify and quantify change points.

CUMULATIVE SUM (CUSUM) CUSUM is a simple, memoryless change detection test that measures the changes accumulated over time. In (2.17), g_t measures the cumulative changes, x_t the prediction error produced by a streaming model, δ the allowed magnitude of change. Then, $x_t - \delta$ expresses how much the new data instance deviates from an acceptable value. If the new data x_t is constantly superior to δ ($x_t - \delta > 0$), it will increase the cumulative changes g_t until g_t exceeds the user-defined threshold λ. A change detection alarm will be triggered, after which g_t is reset to 0.

$$g_t = \max(0, g_{t-1} + (x_t - \delta)) \tag{2.17}$$

CUSUM only takes into account positive change and neutralizes negative changes. Despite its simplicity, CUSUM requires the change threshold λ which is domain-specific and difficult to determine in case there are multiple change criteria to measure.

PAGE-HINKLEY (PH) The PH test is a variant of the CUSUM test. It measures the cumulative differences between the deviation of the new values x_t from the mean $\overline{x_t} = \frac{1}{T}\sum_{t=1}^{T} x_t$ with respect to an acceptable magnitude δ (2.18). The measurement to monitor is $m_t - M_T$, where M_T is the minimum

value of all the values of m_t up to time T ($M_T = \min_{1...T} m_t$). A change detection alarm is triggered when $m_T - M_T > \lambda$, with λ being the user-defined change threshold.

$$m_t = \sum_{t=1}^{T} (x_t - \overline{x_t} - \delta) \tag{2.18}$$

Similar to CUSUM, the PH test also needs a correctly defined threshold δ in order to detect changes.

Adaptive Window (ADWIN) Different from CUSUM and PH, ADWIN is a window-based change detection algorithm [33]. The inputs of ADWIN is a data stream in which the values are bounded within the interval $[0, 1]$, and a confidence value $\delta \in (0, 1)$. Algorithm 2.1 describes how ADWIN works.

Algorithm 2.1: ADWIN Change Detection Algorithm

```
1 Initialize W
2 foreach x_t from the stream do
3     W ← W ∪ {x_t} # add x_t to the head of W
4     repeat
5         drop elements from W
6     until |μ̂_{W_0} − μ̂_{W_0}| < ε_cut;
```

ADWIN slides a window W over the last n instances from the streams and further splits this window into two subsequent windows W_0 of size n_0 and W_1 of size n_1 such that $n_0 + n_1 = n$. ADWIN detects drift if W_0 and W_1 exhibit "distinctive enough" averages. By "distinctive enough", it means the difference of the means of these two subsequent windows $\hat{\mu}_{W_0}$ and $\hat{\mu}_{W_1}$ is superior than the threshold $\epsilon_{cut} = \sqrt{\frac{1}{2m} \times \log \frac{4}{\delta'}}$, where $m = \frac{1}{\frac{1}{n_0} + \frac{1}{n_0}}$ is the harmonic mean of n_0 and n_1, and $\delta' = \frac{\delta}{n}$. When a change is detected, the old subwindow within W is dropped, thus shrinking the size of W. In the contrary, if no change is detected, the data are continuously appended to W. This is how ADWIN dynamically adapts the window size: W shrinks when changes are detected to discard stale data, but W may grow infinitely when the stream is static.

4.4 Online machine learning for predictive maintenance

Online learning can be applied to predictive maintenance to enable lifelong learning of the machinery health without expensive model retraining. The learner is constantly updated on new data collected from the equipment to monitor its condition over time, while being able to detect drifts. A drift in machinery data may signal the precursors of failure. Works that fully use online learning algorithms are still scarce at the moment.

Sahal, Breslin, and Ali [198] do a comprehensive review of open-source big data solutions and their applicability to predictive maintenance (Figure 2.24), illustrated by two case studies on railway and wind turbines maintenance.

Canizo, Onieva, Conde, Charramendieta, and Trujillo [43] implement a predictive maintenance application for wind turbines using big data technologies: Apache Spark for data processing and offline model training and online monitoring using the trained model, HDFS for data storage, Apache Mesos for cluster management and resource sharing, Apache ZooKeeper for load balancing in the cluster. However, the models are trained offline and remain unchanged on incoming data.

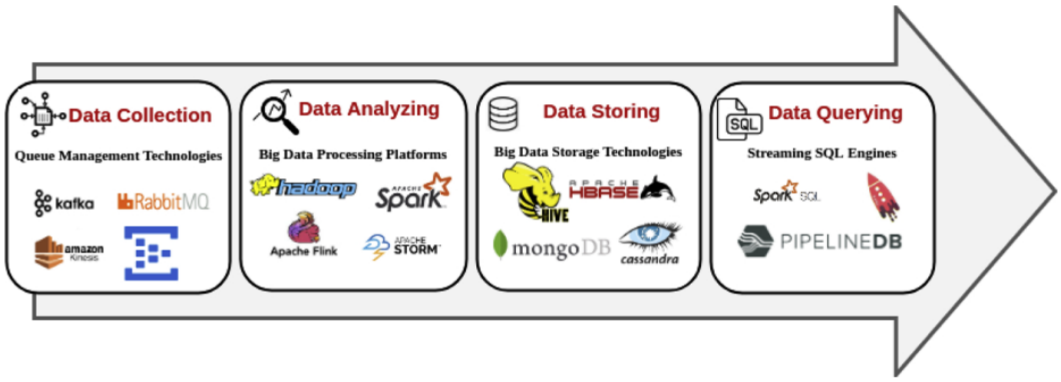


FIGURE 2.24: Open source big data technologies [198]

Su and Huang [213] predict hard disk drives failures in data centers, using Apache Spark for real-time data analytics and Hadoop for batch processing. The models are trained offline using random forest. Apache Spark preprocesses new data in real-time to extract features, to remove out-of-bound values, to treat missing values, then fetches the processed data to the model for prediction. The system deployment on cluster is managed by Hadoop YARN.

Ribeiro, Pereira, and Gama [193] addresses failure prediction of automatic train doors via sequential anomaly detection. Sensor signals are transformed into feature vectors via binning. These vectors are classified into normal or abnormal classes from a semi-supervised or unsupervised anomaly detection. Finally, a low-pass filter runs through the classification outputs to produce the final anomaly alert, considering the temporality of sequential anomalies.

In general, the common approach for online predictive maintenance is to train a model offline on a static batch of data, then to deploy it online for real-time prediction. The model stays unchanged, or is made adaptable on new instances.

Online learning: A summary

In this section, we make an overview on the different facets of online learning. We identify the requirements for an online learning algorithm to handle learning from data streams, namely single-pass processing, time and memory constraint, anytime response, and adaptive mechanism. We distinguish various terms related to learning on data streams (incremental learning, online learning, stream learning) and decide to adopt the term "online learning" in our work. We divide online learning in three classes (supervised learning, semi-supervised learning, unsupervised learning), and we present the most representative examples of each category. We discuss the concept drift phenomenon on dynamic streams and how to handle it, via time window models or explicit drift detectors. Finally, we look for related works that use online machine learning for predictive maintenance to position our research in the literature body.

In the next section, we will conclude the literature review and discuss how the domains we have covered in this chapter can be combined in order to leverage online learning to enhance predictive maintenance in the railway industry.

5 Conclusion

In this chapter, we review four relevant aspects to online machine learning for predictive maintenance in the railway.

Firstly, we discuss some foundational concepts that revolve around the context of this thesis, which are complex systems, reliability theory, and maintenance strategies. Because a railway system is a complex system, its complexity casts an impact on the maintenance methods to be implemented. Reliability is the probability that an asset functions correctly over a specified period of time, and reliability theory is the foundation brick of maintenance methodology. We also present the primary maintenance strategies: corrective, preventive, and condition-based/predictive maintenance. The last one is the most recent form of maintenance that only interferes when needs arise via monitoring and fault prediction.

Secondly, we study the techniques to implement predictive maintenance and divide them into two categories: knowledge-based and data-driven. The knowledge-based approach relies on expert knowledge to create a model simulating the fault mechanism (physical models) or to create a set of IF-THEN rules that simulate the reasoning of a human expert (*expert systems*). On the contrary, data-driven approach leverages the data collected from the equipment to extract insightful information without domain knowledge. Statistical techniques such as random forest, neural networks are used to train a predictive model (*machine learning*), or the machinery degradation is formulated as a stochastic process to be modeled with the appropriate tools (*stochastic models*).

Thirdly, we review a number of standards designed for predictive maintenance. For the development of a maintenance management system, standardization is essential to facilitate the integration of hardware and software products from third-party suppliers. MIMOSA OSA-CBM is the de factor standard for condition-based/predictive maintenance. It defines an architecture that encompasses the full range of maintenance functionalities, from data collection to advisory generation. Besides OSA-CBM, we also mention other standards that specialize one or more functionalities defined by OSA-CBM.

Lastly, we tackle **online (machine) learning** on data streams which differs significantly from the traditional offline, batch learning. Generally, online learning methods are adapted from their offline counterpart to be able to learn incrementally. Change detectors play a crucial role to detect concept drifts from continuous streams.

Predictive maintenance has come into prominence lately, as industrial equipment is being increasingly sensorized and consequently produce data abundantly. Coupled with the rapid advancement of artificial intelligence and big data technologies, predictive maintenance will continue to be the well-sought research topic and will see its applications in many domains. Although machine learning has achieved significant performance for predictive maintenance, there are issues to be addressed. Being data-hungry, machine learning algorithms need a large volume of labeled data to train robust models. In practice, such amount of data are not always available in advance (for example, a system has just been sensorized), neither are the labels. This is the first and foremost obstacle of applying machine learning to predictive maintenance. In addition, complex systems such as those used in the railway network evolve differently. Two systems of the same nature do not always degrade the same way due to various factors. Degradation causes a drift in the data of a system, because a system of good health generates data that different than those in a degraded state. Machine learning models that have not seen such drift during their training phase will fail to capture it when deployed in production. Moreover, systems may undergo functional upgrade, which modifies their behavior. Using static machine learning algorithms, the models remain unchanged facing these changes. As a result, we must wait until sufficient data of the new behaviors have been collected to launch a retraining process.

These issues can be addressed by online learning, which consists of incremental algorithms that update the models continuously on new data and are able to adapt to drifts without forcing model retraining. In this thesis, we study the applicability of online learning to predictive maintenance in the railway industry. Chapter 3 discusses in detail the challenges of predictive maintenance in the railway and formulates the hypotheses that address these challenges.

Chapter 3

Aim of the study, working hypotheses and issues

Contents

Summary

This doctoral thesis studies the applicability of online machine learning to railway predictive maintenance and investigates to which extent online machine learning can improve traditional batch learning. Because new generations of connected rolling stocks produce sensor data infinitely, we focus on the scenario of having a data stream as inputs. Given a fleet of sensorized systems (systems that are equipped with sensors), the goal is to develop methods that learn continuously from a sensor data stream to monitor the condition of the systems in order to achieve predictive maintenance. We expose the challenges associated to handling railway data and discuss how to tackle these challenges via online machine learning. Having studied the literature body of online machine learning and predictive maintenance, we state the research question and formulate the hypotheses that address the research question.

1 Introduction

The motivation backing this thesis is the enhancement of predictive maintenance in the railway industry by means of online learning models that work on connected systems that regularly produce data. The majority of works on data-driven predictive maintenance relies on creating an intricate model offline on static data and fixing the model's parameters afterwards. However, railway being the scope of this thesis, such approach is not fitting.

First of all, we may not have access to the data produced by a system that has just been commissioned or recently sensorized. For a newly commissioned system, some data may have been generated during the testing phase, but these data do not fully represent the real working condition of the system. Therefore, it is not guaranteed that a batch of high-quality static data exist beforehand for offline model training. Meanwhile, the systems are already in operation and may have started degrading.

Secondly, sensor data are unlabeled. For maintenance purpose, labels are any annotation that indicates the working condition of a system, such as functional, degraded, or faulty. Although the labels may become available over time, the amount of labeled data will be heavily dominated by that of unlabeled data. In the railway, only few labels are given by the human experts to sufficiently identify a type of faults in a system. In the contrary, traditional machine learning expects one label for each data instance.

Thirdly, a system changing from one state to another (e.g., from a functional to a degraded state) causes a drift in the data, because the data it produces in different states exhibit different properties. These dynamic changes will affect the accuracy of the model trained on the data without these changes. Traditional machine learning that does not modify a model after training cannot adapt to novelties arriving from the stream.

As a result, using traditional machine learning (TML) to handle railway predictive maintenance is lacking in several aspects.

- TML is data-hungry and requires data to be available in advance to train a model.
- TML can deal with scenarios where few labels are available, but not when labels arrive gradually over time.
- Learning an accurate TML model demands that the training data contain exhaustive information about all possible faults and/or precursors of failure. This is not always guaranteed.
- TML produces non-adaptive models, yet novelties from the data stream require the models to be updated continuously.
- Over the course of its lifecycle, a system may undergo multiple software updates or functionality upgrades that modify its behavior. A trained TML model thus becomes obsolete and must be retrained, for which a sufficient amount of new data must be collected (again).

In other words, TML is not proactive, for it stops the learning at the end of the training process, whereas a system in operation never stops evolving and never ceases to produce data that possibly carry new properties. To address predictive maintenance, we argue that online machine learning is a more suitable approach, because its core principle is to learn and update a model continuously from an infinite stream of data. Online machine learning can incorporate novelties (new data and new labels) from the stream in the models and make the models adapt to changes.

Although our primary focus is on data-driven predictive maintenance, expert systems are also a well-established approach in the railway industry. Expert systems are noteworthy for their explainability via the explicit rules defined by domain experts, and can perform anomaly detection and fault

identification, but an expert system also suffers the same weakness as TML: an expert system is not adaptive. The set of rules is fixed and does not change. If a software update or functional upgrade change the behaviors of the systems, an expert system will fail to recognize those changes and must be modified manually.

Hence, we investigate the following research question:

> Given that online machine learning can overcome certain limits of traditional machine learning, could we use online machine learning to achieve satisfactory results for railway predictive maintenance?

By *satisfactory*, we expect the results obtained with online machine learning will outperform, or at least are competitive to, those yielded by offline machine learning and the expert system (if any) hand-crafted for the target systems.

This chapter is organized as follows. Section 2 describes the passenger access systems on passenger trains as our study cases and explains how the data are generated. Then, Section 3 formulates the hypotheses to tackle the research question while respecting the railway operational constraints. Finally, Section 4 summarizes the hypotheses to be validated in the form of a schema.

2 The passenger access systems

To study the applicability of online machine learning for railway predictive maintenance, we focus on the electric passenger access systems (PAS)[1] on two fleets of passenger trains, namely the NAT and R2N fleets. A PAS is a complex system and is representative of many other systems in the railway, such as the batteries, the HVAC, or the compressors. Therefore, if we reach a viable predictive maintenance solution for the PASs using online machine learning, we can generalize the approach and apply it on other types of systems.

In this section, we describe the PASs, explain how the data are generated, and highlight the challenges associated to such data sets. Since a PAS works similarly on both fleets, we describe the PAS of the R2N trains specifically as an illustrative example.

2.1 Operating mechanism

An R2N train has 16 doors in total, with eight on each side (Figure 3.1). Opening the doors on the left, on the right, or on both side depends on the configuration of a train station.

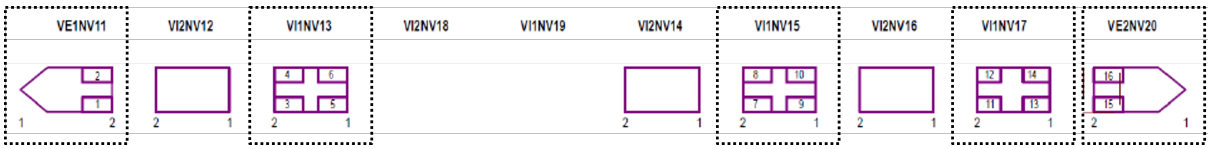


FIGURE 3.1: Position of the PASs in an R2N train. Dotted boxes indicate the cars that have PASs.

The PAS has one primary function, multiple security functions, and several secondary functions, described as follows.

[1] We will use the terms "PAS" and "doors" interchangeably.

Primary function. The primary function of the PAS is to allow passengers to board and to leave the train at any stop, which makes it the system that interacts with the passengers the most on a train. To perform its primary function, the PAS opens and closes when the train completely stops at a station. The duration of the opening is 3 ± 0.25 seconds, and of the closing is 3.25 ± 0.25 seconds. The width of a fully opened door is 1600 millimeters. Without electrical power, the maximal force required to manually open the door is 150 N.

Security functions. The PAS is a critical system on a train because it ensures the safety of the passengers. A PAS has several security functions.

(1) The doors must be closed and locked when the train is running to safely enclose the passengers it is carrying.
(2) A train is not authorized to leave a station if any of its doors is not closed and locked.
(3) In the case of an emergency, the PAS must allow a manual opening for evacuation even if the train is not connected to power.
(4) The PAS must not cut into the gauge while the train is running.

Secondary functions. The PAS has several secondary functions for the comfort of the passengers.

(1) The PAS must be waterproof to be resilient against meteorological phenomena such as heavy rains or flood.
(2) The PAS must enable access to people of reduced mobility (e.g., wheelchair users) as part of the normative obligation. To this end, a PAS may have a mobile footstep that deploys automatically for reduced-mobility users. The doors and the footsteps are two separate subsystems that are part of the PAS. The data collected from the PAS include the signals from both the doors and the footsteps, stored in separate sets of variables.

Figure 3.2 depicts the components of a PAS. The opening of the door is triggered if the button (3) is pressed or if the train stops at a station on a high platform. If there is a large gap between the train an the platform, the footstep is deployed before the door opens. The closing of the door is commanded by the train driver or automatically triggered when the internal timer of the PAS expires. The lights (11) and (19) flash when the door is closing and the footstep is retreating. In the case of an emergency, the doors can be manually opened by inserting a special key to the lock (4).

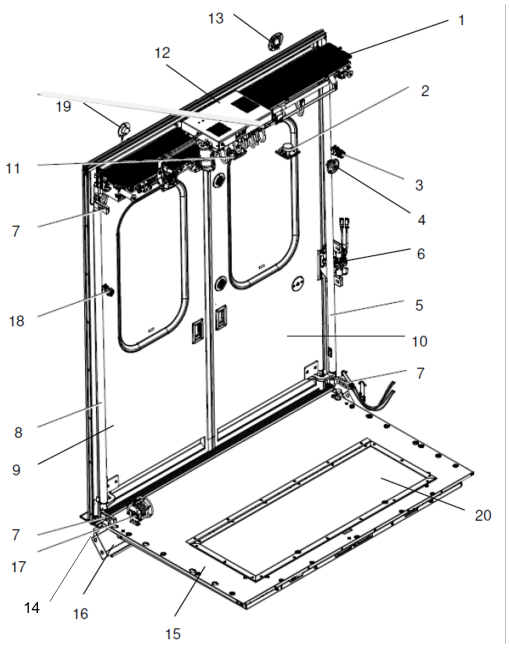
The PAS has two symmetric panels, on the left and on the right, that are swaying and sliding. During the opening, the panels first move out of their position with an angle of 45° for 77.4 millimeters, then they slide linearly along the body of the car for 722.6 millimeters. Figure 3.3) illustrates the movement of one panel when the door opens, from a top-down view.

To open and close the PAS, the components in each door panel follows a fixed order of mechanical movement. Figure 3.4 shows the order of movement when the door is opening, as it moves from "mechanical stop (closing)" to "mechanical stop (opening)". The reverse order applies for the closing.

Obstacles may obstruct the closing or opening of the doors. The PAS has a mechanism to detect obstacles before closing or opening for safety reason.

- Detecting obstacles during the **closing** is realized by measuring the over-current of the motor.
  - In case of a centralized closing commanded by the train driver, if an obstacle is detected, the doors stop in their track for two seconds before attempting to close again. If an obstacle is detected three times, a fault code[2] is sent to the control network in the train as the door

[2] A fault code is a unique identifier assigned to one type of anomaly of the systems (unrelated to machinery degradation), for example, when system fails to detect if a platform is accessible for wheelchair users, or when the door cannot be closed.



1	Command mechanism	**11**	Closing door light
2	Internal speaker	**12**	Control board
3	Push-button	**13**	External speaker
4	Square lock	**14**	Threshold
5	Right coordination bar	**15**	Mobile footstep
6	Condamnation lock	**16**	Footstep hood
7	Fixing bars	**17**	Square lock
8	Left coordination bar	**18**	Forcing lock
9	Left door panel	**19**	Mobile footstep light
10	Right door panel	**20**	Glazing bead

FIGURE 3.2: Components of a PAS with a mobile footstep

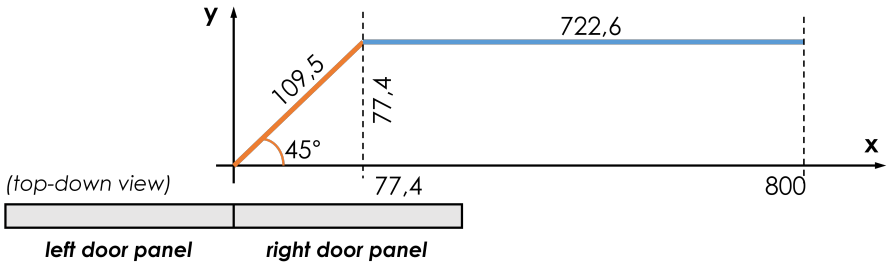


FIGURE 3.3: The movement of the door panels during the opening

attempts to close at slow speed, given that the closing command is always maintained.

- In case of a slow or timed closing, if an obstacle is detected, the door reopens completely and triggers a timer for the next closing attempt. If the number of obstacles is odd, the timer is set to 90 seconds for a PAS without the mobile footstep and 180 seconds for one with the footstep. Otherwise, the timer is 10 minutes for all PAS.

- Detecting obstacles during the **opening** is similar to the mechanism used for the centralized closing. If an obstacle is detected, the door halts its opening, remains idle for 2 seconds, then attempts to open again. The door renews its attempt until the third obstacle detection and stays idle afterwards. It is possible to trigger the opening or closing again, but a fault code will be sent to the control network in the train.

2.2 Data acquisition process

Each PAS has a Door Control Unit (DCU) that processes the commands sent by the central train network and records the sensor signals in the form of binary files. In addition, each train has a computer made specifically for condition-based maintenance purpose to collect the files generated by the DCU of each PAS and transmit them to a processing server. The computer is powered by the train and cannot function when the train is not connected to power.

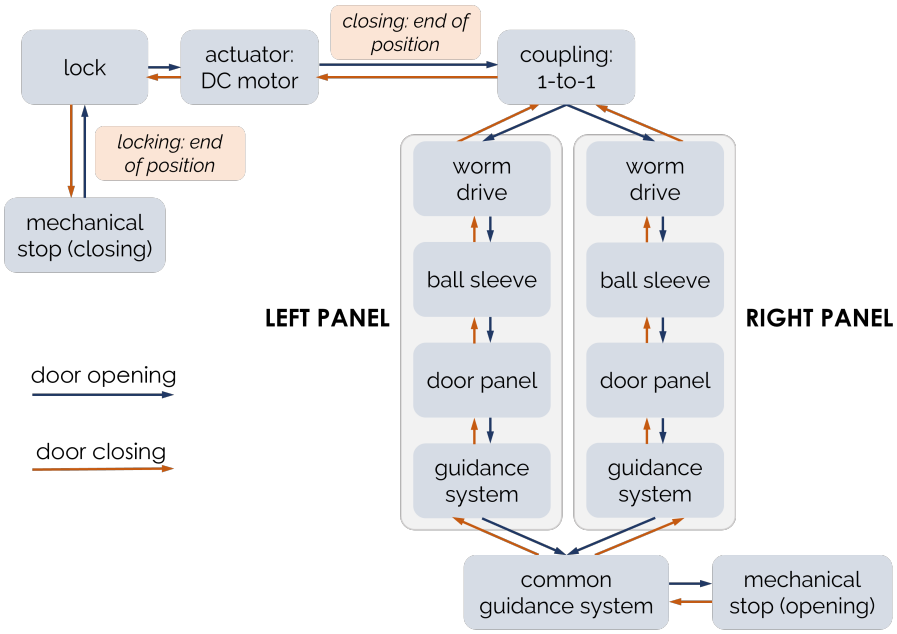


FIGURE 3.4: The mechanical movement of the door panels during an opening (blue arrows) and a closing (orange arrows)

For transmission efficiency, the data are first sent to an intermediary server on the edge, then sent to the final processing server (Figure 3.5). The transmission is scheduled to send a batch of all files collected every two hours. In other words, there is a delay of at least two hours between the creation time and the processing time of the data.

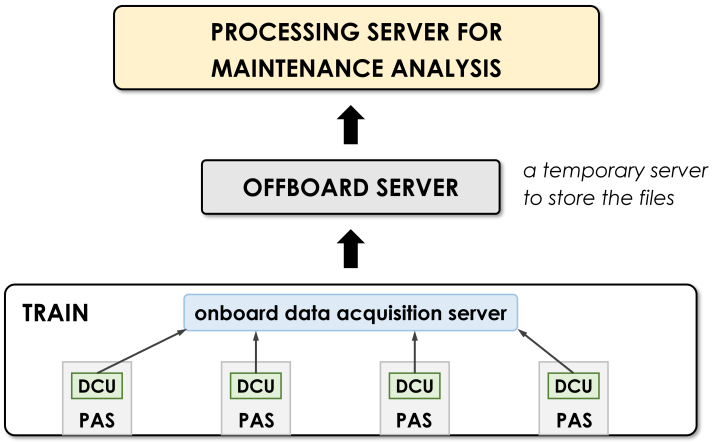


FIGURE 3.5: The files generated by the DCU on each PAS are sent to the onboard server in the train, then transmitted to an offboard server for temporary storage, before being sent to the final processing server for maintenance-related analysis.

The data acquisition starts when the train enters a station and all the PASs receive the opening authorization command. Then, all the signals produced by the PASs are recorded until the doors are completed closed and the train prepares to depart. The signals of one PAS are written and encoded into one binary file, waiting for transmission. One file is generated by each PAS at each station (Figure 3.6).

Each file contains a time series with both analog and boolean variables. The most interesting boolean variables are:

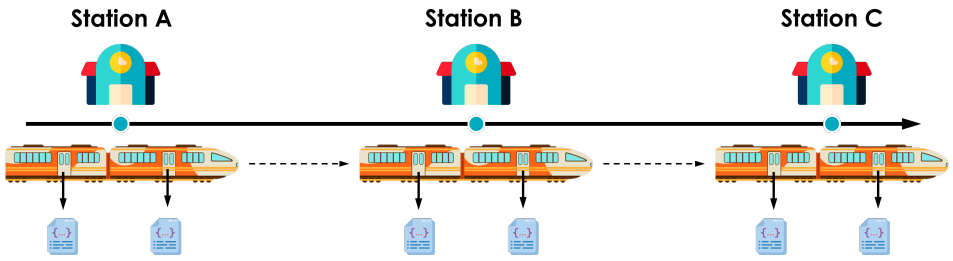


FIGURE 3.6: Every time a train enters and leaves a station, each door generates one new data file.

- whether an opening (or closing) command is sent to the PAS, which records the state of the command throughout the entire opening (or closing),
- whether the limit switch has been activated,
- the state of the locking (whether the PAS is correctly locked and the train is allowed to leave).

There are over 60 boolean variables in one data file. Analog variables come from the electric motor of the PAS and of the mobile footstep, including the position of the panels (Figure 3.3), the intensity in the motor, and the voltage in the motor. There are six analog variables in total, among which three record the signals from the door and three from the footstep. The analog variables are of type integer.

As long as a train remains in operation, the data are generated continuously by its PASs. Over time, an entire fleet creates a stream of data in near real-time, in which each data unit is a binary file produced by one PAS on one train at a given moment. On the stream, the files are organized in chronological order by their creation time (the moment the file is created by the DCU). Therefore, online machine learning appears suitable in this scenario.

3 Hypotheses

Given the data acquisition process of the PASs, we expect to receive as input a data stream of raw sensor signals and to return as output maintenance alerts. The input stream contains the signals from multiple PASs mixed together. The output should return one maintenance alert for one system.

At the time of writing, an expert system was developed and deployed in production for both the PASs of R2N and NAT trains (one expert system for each type of train) [227–229, 233]. One fleet of PASs forms one data set. We consider the expert system of each fleet the performance baseline for each data set. The solution using online machine learning is expected to perform superior to, or at least on-par with, the existing expert system on both fleets. In addition, online machine learning must be competitive to its offline counterpart. Therefore, we will compare the performance of the expert system (ES), offline learning methods (OF), and online learning methods (OL) on the data of the PASs on the R2N and NAT fleets. Please bear in mind that when we compare OF to OL, the online and offline algorithms share the same principle, but each is run differently if it is put in a batch setting (OF) or in an online setting (OL). This is to enable a fair comparison between the two paradigms.

In the following subsections, we elaborate on the hypotheses on applying online machine learning for implementing predictive maintenance on the PASs.

3.1 Granularity

Since we have multiple PASs to monitor, the input stream mixes the signals from all the PASs, only ordered by their creation time. Thus, it poses the question on the *granularity* of analysis.

3.1.1 Definition

The granularity of analysis is the level on which we analyze the data: on a *fleet* level or on an *individual* level. The former analyzes the data from all the systems in the same pipeline, whereas the latter analyzes the data of each system independently in its own pipeline.

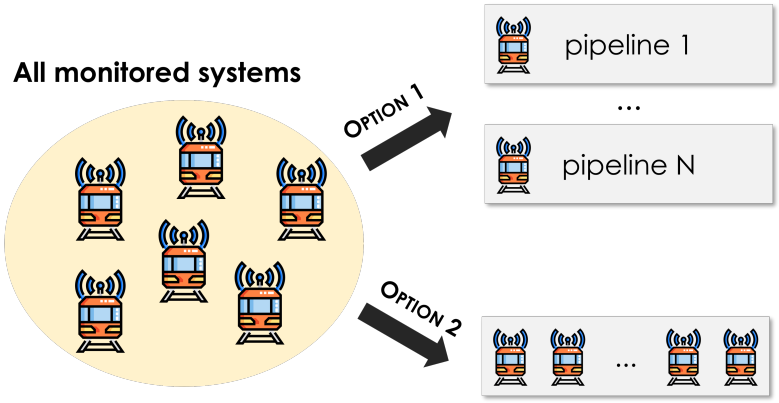


Figure 3.7: Granularity of analysis on an individual level (Option 1) or on a fleet level (Option 2).

We can also think of a third hybrid approach that performs the analysis on *groups* of systems. It is possible that in the fleet, some systems are more closely related between them than to others. For example, systems that are commissioned at approximately the same time, or systems that operate in similar conditions (weather, terrains, and so on), should degrade similarly. The analysis results of the groups can be aggregated to provide a global understanding of the fleet. To implement this approach, a similarity metrics on the systems must be crafted and requires further investigation. We save this for post-thesis future works.

3.1.2 Challenges

All PASs are manufactured identically but degrade differently, depending on their operational environment. There can be a large number of systems in a fleet. The fleet approach and individual approach have their own strengths and weaknesses.

Fleet-level analysis On a fleet level, the individuality of a system is erased. Training a single model on the data of multiple systems implies that we consider the fleet as an inseparable whole and that there exists a set of common behaviors shared by all the PASs. Even if each PAS evolves differently, it does not deviate largely from the shared behaviors identified from the fleet. As a consequence, the particular evolution of one system can be shrouded by the patterns of the mass.

Individual-level analysis The individual approach creates one model for each system and learns from the data of this system exclusively. Therefore, it regards each system on a finer granularity and captures patterns that are specific to one system. However, it misses the information that has been learned from other systems and must learn it again. For instance, a newly commissioned PAS

encounters a problem, which is novel to it but was already identified from other PASs in operation, but the new PAS does not have access to such information.

3.1.3 Hypothesis

Between the fleet-level approach and individual-level approach, we see that the former is a more viable choice. A large number of systems makes the individual approach infeasible to scale. There can be thousands of systems in a fleet, but maintaining thousands of models is inefficient. Furthermore, a model learned on the data of uniquely one system risks to overfit and/or to learn incoherent patterns. Consider a PAS that is defective from its commission and does not produce any "normal" data, the model learned on this system's data mistakenly views the patterns of defective state as normal and does not issue any maintenance alerts, which only worsens the situation.

($\mathbf{H_g}$) Fleet-level analysis produces more consistent results on the condition of systems and is more efficient than individual-level analysis.

3.2 Cyclicity

Given the characteristics of the PAS, we expect the data to arrive in the form of files containing raw sensor signals (Section 2.2). One file contains the signals from only one system. In each file, the signals generated when the system performs different functions are mixed together. For example, the signals when a PAS is opening and when it is closing are saved in the same file. Yet, a problem that occurs during an opening phase differs from that during a closing phase. That is why we must separate the signals of different functions prior to analysis. Thereafter, we refer to the duration during which the system performs one function as a *cycle*. Informally, a cycle can be seen as the lowest common denominator, or the smallest analysis unit, in the data of the systems.

3.2.1 Definition: cycles

Semantically, a cycle is a *realization of a function* of a system. Technically, a cycle is a segment of signals created when the system is performing one function, which is also a *time series*. If multiple variables are collected simultaneously, a cycle is a multivariate time series; otherwise, it is a univariate time series. For the PASs, we are dealing with multivariate time series. Railway systems have cyclic behavior because a system in the railway is designed to repeatedly perform a set of intended functions. Consequently, a cycle is a time series whose motif appears frequently in the input data stream. Figure 3.8 illustrates an example of one input file generated by a PAS. Two cycles (colored lines) are extracted from the raw signals (gray, dotted lines). The first cycle is an opening cycle of the PAS and is labeled "op"[3]. The second one is a closing cycle and is labeled "fp"[4].

Because the cycles record the behaviors of a system in operation, they carry the information of the system's condition and are the ingredient of analysis. The foremost tasks are (i) to extract from the data stream any segment that forms a cycle, and (ii) to associate an extracted cycle to a corresponding function (closing or opening in the case of the PASs). We refer to (i) as *cycle detection* and to (ii) as *cycle identification*. Together, the two tasks constitute *cycle extraction*.

[3] "op" is short for"ouverture porte", translated to "door opening" in English.
[4] "fp" is short for "fermeture porte", translated to "door closing" in English.

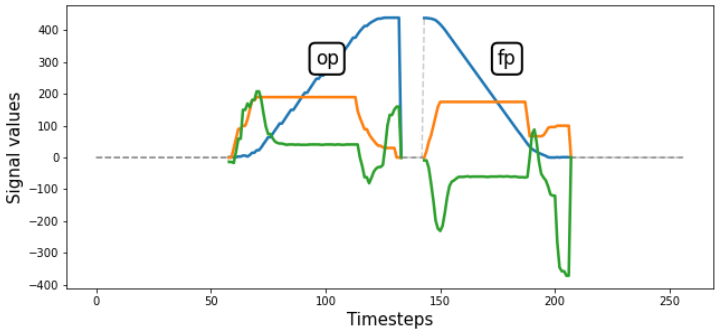


FIGURE 3.8: Example of two cycles (colored lines) extracted from an input file

3.2.2 Challenges

To extract cycles from a data stream using expert systems or offline learning, the common practice is to collect a batch of data from the stream, then train a model on this batch. This process must be repeated if the model is to be modified. The challenges associated to cycle extraction on a data stream are threefold.

CHALLENGE C.1 Collecting data followed by model training incurs a delay, starting from the arrival of the first cycle until model validation, during which no cycles are returned.

CHALLENGE C.2 There is no labels about cycles from the stream: we do not know in advance the shapes of expected cycles, nor the list of functions of a system, nor how to map a cycle to a function. Without labels, it is difficult to use supervised learning methods.

CHALLENGE C.3 Changes can occur and modify the cycles. Cycles of rare shape that are not included in the training set may appear only after a while. Existing cycles may have their shape changed: if the systems are functionally modified, the generated cycles are also impacted and no longer correspond to those that have been learned by the model.

3.2.3 Hypotheses

Considering the aforementioned challenges, we formulate the hypotheses to address cycle extraction on the data stream generated by the fleet of PASs.

REACTIVITY To address the challenge C.1, we make use of online learning algorithms. These algorithms learn continuously and incrementally. They are able to adapt their parameters on-the-fly on incoming data examples, and their learning process never stops. As a result, an online algorithm can start extracting cycles from the first input signals on the stream. Nevertheless, it will have unstable performance until it reaches convergence, that is, it has received enough data to perform well. Meanwhile, offline learning does not return anything until the training process finishes.

($\mathbf{H}_c^1$) Using online learning to learn to extract cycles has a higher reactivity than using offline learning.

We measure the reactivity of an algorithm by the time from the reception of the first data example until the time the algorithm reaches a stable performance. We do not include the expert systems in

this hypothesis because an expert system performs modeling instead of learning. An expert system can take more than two months to be validated, but it is readily usable after its validation.

Accuracy To address the challenges C.2 and C.3, we implement human-in-the-loop online learning. In other words, a human participates in the learning process and returns feedback to any query issued by the online algorithm. Such feedback becomes the labels that enrich the knowledge of the algorithm about the cycle shapes and types. Any novel cycle types that were not previously seen by the model are sent to the human as query. Therefore, the online model does not run into the risk of failing to recognize unknown cycles, making it robust against dynamic changes on the stream. In contrast, offline learning and expert systems are static and cannot change unless manual modification is made.

($\mathbf{H}_{\mathrm{C}}^{2}$) Online learning that enables model update via human feedback performs superior to, or on-par with, a static expert system in accuracy.

To evaluate the accuracy of online learning, we compare it to the baseline performance of the expert system. To measure accuracy, we consider both the correctness of cycle detection and cycle identification by counting the number of correctly detected and identified cycles for each approach (online learning versus expert system).

3.3 Features

The cycles are the main ingredient of the analysis, because they reflect the underlying condition of a system when it performs its functions. By nature, a cycle is a time series. Although time series analysis is an established research field, element-wise comparison between time series remains computationally expensive, because we must compare the data across both the timesteps and variables. The cycles can be of different length as well, since the duration of the same function may vary. Processing variable-length series induces additional complexity.

Let us consider an example of computing the Euclidean distance. The time complexity of computing the Euclidean distance between two vectors of K elements is $O(K)$. Meanwhile, the Euclidean distance between two time series of K timesteps and D variables is the average of the Euclidean distances between each univariate series, and the complexity is thus $O(KD)$ that increases with the number of variables D collected in each cycle. This applies to space complexity as well.

Meanwhile, we can collapse a time series to a vector of indicators. Such transformation has several advantages: it compresses the series to a more compact form to accelerate the computation, it benefits from vectorial optimization offered by many programming libraries, it maintains sufficient information of the original cycle by means of summary indicators. Hence, transforming a cycle to a vector of indicators, also called *feature vector*, is beneficial to the ensuing analysis.

3.3.1 Definition: features and feature vectors

A feature is a statistic extracted from a cycle, for instance, the cycle length, the mean value, the number of peaks, and so on. A feature vector comprises a number of features and corresponds to uniquely one cycle. In other words, a feature vector is another representation of a cycle. All feature vectors have the same number of elements.

To extract features from a cycle, an expert system relies on domain expertise to identify features relevant to the working condition of the systems, examples of which are the area under the curve, the duration of various phases in the cycle, the mean values in each phase, et cetera. This is the basis of

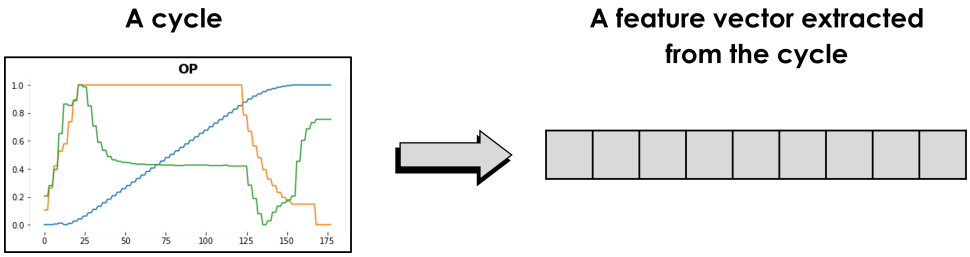


FIGURE 3.9: Transforming a cycle to a feature vector ("op" = door opening)

feature engineering. Different from feature engineering, feature learning, which is applicable to both online learning and offline learning, learns the features from the cycles, either by determining a direct mapping from the cycle space to the feature space, or by estimating the latent distribution that best describes the original cycle space. Feature learning is especially useful if one does not have access to domain expertise to identify important features.

3.3.2 Challenges

We identify four challenges for extracting features from cycles.

CHALLENGE F.1 We do not know which features are important to extract. The set of extracted features may be incomplete and lacks important features, or it can be overcomplete and contains redundant features. Moreover, existing features may lose or gain relevance over time, known as *feature evolution* [81].

CHALLENGE F.2 Contextual noises affect the quality of the features. A system always operates within a context, which encompasses every factor that surrounds the system, such as environmental conditions, current workload, or curvature of the station. A context may make a normal cycle appear abnormal, or vice-versa.

CHALLENGE F.3 If offline learning or expert systems are used, sufficient data must be collected to train a model. Data collection and model validation both incur delay.

CHALLENGE F.4 Information loss is inevitable, as we embed data from higher dimension (time series) to a lower dimension (vectors). It is crucial to minimize information loss.

3.3.3 Hypotheses

Considering the aforementioned challenges, we formulate the hypotheses to address feature learning on the stream of cycles generated by the cycle extraction task.

ACCURACY The challenges F.1 and F.2 concern the learning capacity of the model. The priority is thus to prove that a feature learning algorithm is able to learn features that are representative and robust against noises. Because feature learning is an unsupervised task and human feedback is not straightforward to collect and use in this scenario, we will employ the same algorithm to learn features offline and online, with some adaptation to make the algorithm offline-compatible. The following hypothesis only concerns the learning algorithms and exclude the expert system.

($\mathbf{H}_\mathbf{f}^1$) Online feature learning performs superior to, or at least on part with, offline feature learning in terms of accuracy.

Accurate features are those that summarize well the original cycles. The question is to quantify the extent of how well the features represent the cycles. Usually, the performance of feature learning is assessed via a downstream task. For example, given a classification task, we compare the predictions issued by a model trained with the original data to those by a model trained with the learned features. If the latter outperforms the former, it means the features are well learned. But because we do not have labeled data, we evaluate the feature quality by performing a cycle-feature ranking. Intuitively, if the features accurately summarize the cycles, anomalous cycles should yield anomalous features. If the top k anomalous features correspond to the top k anomalous cycles, it means the feature learning algorithm attains a high accuracy.

Reactivity The challenge F.3 is linked to the reactivity of a model: when the model is ready to issue usable features. Offline learning and expert systems must wait for data collection and model validation before deployment, while online learning is ready to return features from the first few examples but its performance fluctuates until it has reached convergence. Even if that is the case, we want to prove that an online model converges to a stable performance faster than an offline model. We exclude the expert system from this hypothesis for the same reason as explained in Section 3.2.3.

($\mathbf{H}_\mathbf{f}^2$) Online feature learning is more reactive than offline feature learning.

The reactivity of a model is measured by the time from the arrival of the first cycle to the time the model can return features accurately (convergence).

Information loss To tackle the challenge F.4, we must show that feature learning results in less information loss than feature engineering. Because feature learning leverages all information it can mine from the data, it has a lower risk to miss out features. Once we have compared online feature learning to offline feature learning ($\mathbf{H}_\mathbf{f}^1$), we compare it to an expert system.

($\mathbf{H}_\mathbf{f}^3$) Feature learning results in better information preservation than feature engineering.

The extent of information loss can be assessed by the accuracy of the features, as described previously, or by reconstructing the original cycles if the feature learning algorithm allows so.

3.4 Health

The feature vectors extracted from the stream of cycles form a stream of feature vectors. This stream is the input of the analysis to unveil the *health* of the monitored systems.

3.4.1 Definition: system health

Inspired by the literal definition of "health"[189], we define the health of a system *its extent of being free from anomaly*. Originally, the health is understood as a state [189], but we stress on the term "extent" because we want to quantify the system health: instead of simply being normal or degraded, the system health should be more nuanced to be located on the spectrum from normal to faulty, e.g., normal, somewhat normal, somewhat degraded, severely degraded, et cetera.

Then, we define anomaly as any deviation from what is normal. From the hypothesis of fleet-level granularity (Section 3.1), the normal health is observed from the majority of the fleet, because in the railway, we strive to maintain the systems in such a way that a large part of the fleet is functional to ensure minimum operational service. Only a small fraction of the systems is tolerated to be anomalous at any give time. Furthermore, because the PASs are designed to work in a unique nominal state, the normal health is uniquely defined.

Beside a health score that is attributed to one system at a time, we seek to discover the *health profiles* of the fleet. A health profile is an envelope of characteristics of the systems in the same condition, such that the data in one health profile are more similar to each other than to those in different profiles. Intuitively, we can regard one health profile as one cluster that groups the data generated by the systems impacted by a particular anomaly. Because the normal health is uniquely defined, there exists one unique normal health profile that is the *reference profile*. A health profile is defined by the data from multiple systems in the fleet, and it changes if the systems undergo a functional modification. One fleet has a finite number of health profiles. Figure 3.10 illustrates an example, where the data that are around the mean of the feature distribution (in green) constitute the reference cluster, and those that fall out of the normal range form anomalous health profiles.

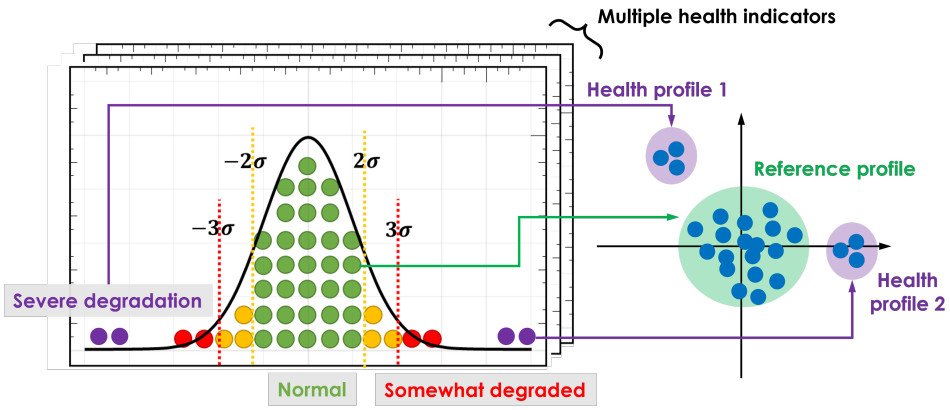


FIGURE 3.10: Example of health profiles captured from the data [229]

Intuitively, the health of a system is computed from the health profiles into which the system generates data. As a result, accurate discovery of health profiles is crucial as it is the basis to compute the system health.

3.4.2 Challenges

We distinguish two challenges related to estimating the health of a system, one linked to discovering the health profiles of the fleet and the other linked to computing the system health.

CHALLENGE H.1 The health profiles of the fleet must be discovered automatically without domain knowledge. Any evolution of the health profiles on the stream must be detected in a timely manner to ensure an accurate assessment of the system health.

CHALLENGE H.2 Given the set of discovered health profiles, the health of a system must be computable at any given moment while taking into account all the profiles, under the influence of which the system currently is. Also, the system health must be computed efficiently to avoid expensive recomputation and to enable real-time health monitoring.

3.4.3 Hypotheses

We formulate the hypotheses to address the aforementioned challenges.

Accuracy First and foremost, we focus on the capacity of the model to accurately discover the health profiles and identify the health of a system at any give moment.

Accuracy of the health profiles The health profiles are the basis on which we compute the health score of any system at any time. Due to the lack of labels and a priori knowledge, we use clustering as an unsupervised method to distinguish different profiles of the fleet. This hypothesis addresses partly the challenge H.1.

($\mathbf{H}_\mathbf{d}^1$) The clusters represent the health profiles of the fleet.

As the expert system does not find the health profiles but creates clusters on the systems (one data point is one PAS), we do not compare our clusters to those of the expert system. Instead, the accuracy of the health profiles in the form of clusters will be validated by a domain expert.

Accuracy of the health score Given the set of health profiles learned from all systems in the fleet, the health of an individual system is computed from the amount of data it creates in each health profile. It implies that the health of a system may be impacted by multiple health profiles simultaneously. This is consistent with the reality that a system may be impacted by multiple anomalies at the same time. This hypothesis addresses partly the challenge H.2.

($\mathbf{H}_\mathbf{d}^2$) Computing the health score by considering the data a system creates in the health profiles results in an accurate health detection, within an observable perimeter from the data.

Currently, we do not have a ground-truth against which we can assess the precision of the resulting health scores, therefore we rely on the quality of the health profiles as clusters to deduce whether the health scores are correctly estimated.

Reactivity Clustering can be done online or offline. As the input is a stream of feature vectors, we employ online clustering to build the health profiles incrementally from a continuous stream of data to address the challenge H.1. In contrast, offline clustering does not work on incoming data. It must wait for a sufficient volume of data before building the health profiles.

($\mathbf{H}_\mathbf{d}^3$) Online clustering reaches convergence earlier than offline clustering, if no drift occurs.

In this context, convergence is the moment when the clusters become stable (not undergoing drastic changes in the absence of drifts) and reflect the underlying physics of the PASs. Reactivity is measured via the time elapsed from the reception of the first feature vector to the moment of convergence.

Continuity An advantage of online clustering is that it maintains the continuity of the cluster evolution, such that a cluster at a time t is preceded by a cluster at a time $t-1$, unless it is a newly created cluster. Online clustering allows us to monitor the evolution of the clusters, and thus of the health profiles. As a result, the evolution of any system's health scores can also be continuously monitored, by tracking the amount of data that a system generates in the evolving profiles. Meanwhile,

offline clustering must be rerun on each new batch of data. The clusters between two offline runs are completely disconnected from each other: any connection between two "snapshots" of health profiles is lost and/or requires a complicated mapping to link them together.

($\mathbf{H}_{\mathbf{d}}^{4}$) Online clustering keeps track of the temporal evolution of the health profiles via cluster updates, while offline clustering cannot.

This hypothesis addresses both the challenges H.1 and H.2. To validate it, we will evaluate the quality of the clusters maintained online on the data stream and those obtained with an offline clustering algorithm on small batches of data.

3.5 An industrial solution for predictive maintenance on data streams

We have introduced *cyclicity*, *features*, and *health* as the building blocks of this research. The implementation of the hypotheses elaborated in the previous sections will result in a solution that enables a continuous monitoring of the health of the systems on a stream of sensor signals. Hence, a byproduct industrial outcome of this thesis is an end-to-end, modular pipeline for processing a sensor data stream for the purpose of machinery maintenance. Figure 3.11 shows the pipeline and its four modules.

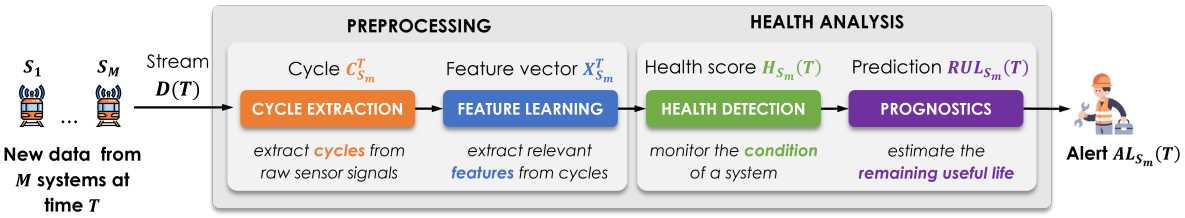


FIGURE 3.11: The end-to-end pipeline that functions on the basis of online machine learning

Our pipeline is inspired by the OSA-CBM standard that is the de facto standard for condition-based maintenance [158]. It proposes a modular architecture for implementing necessary functionalities in a condition-based maintenance framework: data acquisition, data manipulation, state detection, diagnostics, prognostics, advisory generation, and presentation (Section 3.1). Our solution covers four of the seven OSA-CBM modules: data manipulation, state detection, diagnostics, and prognostics. We omit data acquisition, advisory generation, and presentation, because of the following reasons.

- We focus solely on analyzing the data that are provided to us. Installation and sensor management are the responsibility of the data providers.
- Recommending optimal maintenance actions is a complex planning and optimization task, and goes beyond the scope of this thesis.
- Presenting and visualizing the analysis results are a matter of UX/UI design. We will only output maintenance alerts.

The modules Cycle extraction and Feature learning preprocess raw sensor data (Data manipulation). The module Health detection combines both State detection and Health assessment to detect and identify anomalies. The module Prognostics predicts failures by estimating the remaining useful life (Prognostics assessment). Table 3.1 explains how our modules map to the blocks of OSA-CBM.

Let $\mathcal{S} = \{S_1, ..., S_M\}$ be the fleet of M systems subject to maintenance. We denote $D(T)$ the input stream containing all the data from any system $S_m \in \mathcal{S}$ received thus far, from $t = 1$ to $t = T$, in

Our pipeline	**OSA-CBM**
Cycle extraction Feature learning	Data manipulation
Health detection	State detection Health assessment
Prognostics	Prognostics assessment

TABLE 3.1: Matching our modules to the blocks of OSA-CBM

chronological order. $D(T)$ can be seen as a snapshot of all the data that have been generated from the beginning of the stream until $t = T$. Beside the input stream $D(T)$ containing the data generated as is by the fleet, the input-output flow between the modules also constitutes a stream by itself. Therefore, we redefine $D(T)$ at each stage to fit it to the tasks encapsulated by the module.

The following sections describe the task of each module.

3.5.1 Cycle extraction

The module Cycle extraction performs cycle detection and cycle identification, as described in Section 3.2. It works on a stream of raw sensor signals in the form of data files[5]. Each file contains the signals generated by one PAS when it was performing its function.

We denote $C^T_{S_m}$ a cycle generated by a system $S_m \in \mathcal{S}$ at a time T. One or multiple cycles can be found in one data file from the stream $D(T)$. There may also be no cycle at all, due to data acquisition errors. Because multiple systems function simultaneously, there can be many cycles at the same timestamp T, but the identifier of any system S_m suffices to uniquely identify the cycles.

We define the input stream $D_{CE}(T)$ to the module Cycle extraction as a sequence of time series $F^t_{S_m}$ read from a file that has variable length and a fixed number of variables D. The length of $F^t_{S_m}$ depends on how long the system took to complete writing a file.

$$D_{CE}(T) = \left(F^t_{S_m} \mid S_m \in \mathcal{S}, 1 \leqslant t \leqslant T, F^t_{S_m} = [\mathbf{a}^{(1)}, \dots, \mathbf{a}^{(N)}] \;\; \textit{where} \;\; \mathbf{a}^{(n)} \in \mathbb{R}^D\right)$$

3.5.2 Feature learning

The module Feature learning receives a stream of cycles $D_{FL}(T)$ and transforms each cycle $C^T_{S_m}$ to a feature vector $X^T_{S_m}$ (Section 3.3). We define the input stream $D_{FL}(T)$ to the module Feature learning as a sequence of cycles created by the the fleet $\mathcal{S}$, ordered by their creation time, and each cycle is a small time series.

$$D_{FL}(T) = \left(C^t_{S_m} \mid S_m \in \mathcal{S}, 1 \leqslant t \leqslant T, C^t_{S_m} = [\mathbf{a}^{(1)}, \dots, \mathbf{a}^{(K)}] \;\; \textit{where} \;\; \mathbf{a}^{(k)} \in \mathbb{R}^D\right)$$

For each cycle $C^T_{S_m}$ of K timesteps[6] and D variables, its feature vector $X^T_{S_m}$ has P elements ($X^T_{S_m} \in \mathbb{R}^P$) such that $P \ll K \times D$.

[5]The files are usually encoded to a binary format to enable efficient transmission. The decoder must be supplied to us by the data provider to transform the files to a comprehensible content.

[6]The number of timesteps varies across cycles.

3.5.3 Health detection

The module Health detection monitors the *health* of the systems, keeps track of the evolution of any anomaly manifesting in the systems, and quantifies the deviation of the systems from an expected health profile. It learns and updates the health profiles from a stream of feature vectors $D_{HD}(T)$, and recomputes the health scores of the systems continuously.

We define the input stream $D_{HD}(T)$ to the module Health detection as an sequence of feature vectors produced by the systems in $\mathcal{S}$ in chronological order, that is:

$$D_{HD}(T) = \left(X^t_{S_m} \mid S_m \in \mathcal{S}, 1 \leqslant t \leqslant T, X^t_{S_m} \in \mathbb{R}^P\right)$$

We denote $H_{S_m}(T)$ the health score of the system S_m computed at the moment T.

3.5.4 Prognostics

The module Prognostics estimates the time until the next failure of a system, which is by definition its remaining useful life (RUL). However, it is not common to leave a system operate until it fails, thus we may not have the exact moment of a system's definite failure. Therefore, we loosen the goal and instead, aim to predict the next time the system encounters an anomaly that may lead to a failure.

This module receives a stream of health scores $D_{PG}(T)$ as input, defined in the following:

$$D_{PG}(T) = \left(H_{S_m}(t) \mid S_m \in \mathcal{S}, 1 \leqslant t \leqslant T, 0 \leqslant H_{S_m}(t) \leqslant 1\right)$$

We denote $RUL^T_{S_m}$ the remaining useful life of the system S_m at the moment T. The value $RUL^T_{S_m}$ is part of the maintenance alert $AL_{S_m}(T)$ sent to the maintenance technicians. An alert $AL_{S_m}(T)$ is created for each system S_m at a time T and contains additional information such as the health scores as a curve and the anomalies manifesting in S_m until T.

4 Conclusion

This thesis investigates the applicability of online learning for railway predictive maintenance. We perform a study case on the passenger access systems on the fleets of R2N and NAT trains, managed by SNCF. We schematize our following hypotheses by dividing them into four building blocks linked to the characteristics of the PASs: granularity, cyclicity, features, and health (Figure 3.12).

Granularity concerns the level at which we analyze the data from the systems in a fleet. This block has one hypothesis:

(H$_g$)	Fleet-level analysis produces more consistent results on the condition of systems and is more efficient than individual-level analysis.

Cyclicity is due to the cyclic behavior of railway complex systems that produce repeating patterns. These patterns describe the behavior of a system when it performs a function and as such reflect its underlying health. The cyclicity block has two hypotheses:

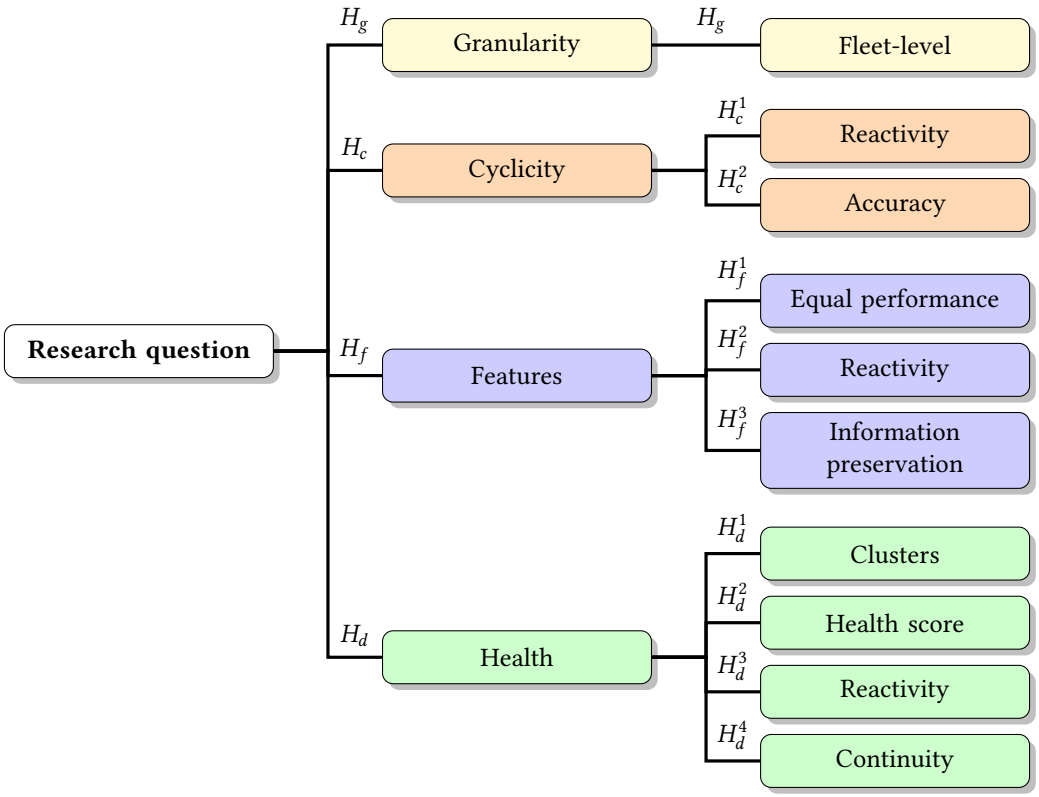


FIGURE 3.12: List of all hypotheses, organized in four main blocks: granularity, cyclicity, features, health

$(\mathbf{H_c^1})$	Using online learning to learn to extract cycles has a higher reactivity than using offline learning.
$(\mathbf{H_c^2})$	Online learning that enables model update via human feedback performs superior to, or on-par with, a static expert system in accuracy.

Once the cycles are extracted from the sensor data stream, a feature vector must be extracted from each cycle to compress the cycle while retaining relevant information. The features block has three hypotheses:

$(\mathbf{H_f^1})$	Online feature learning performs superior to, or at least on part with, offline feature learning in terms of accuracy.
$(\mathbf{H_f^2})$	Online feature learning is more reactive than offline feature learning.
$(\mathbf{H_f^3})$	Feature learning results in better information preservation than feature engineering.

Finally, the health of each system is estimated from the stream of feature vectors. To calculate the system health, a set of health profiles must first be discovered from the stream in the form of clusters. We formulate four hypotheses for this block.

($\mathbf{H}_\mathbf{d}^1$)	The clusters represent the health profiles of the fleet.
($\mathbf{H}_\mathbf{d}^2$)	Computing the health score by considering the data a system creates in the health profiles results in an accurate health detection, within an observable perimeter from the data.
($\mathbf{H}_\mathbf{d}^3$)	Online clustering reaches convergence earlier than offline clustering, if no drift occurs.
($\mathbf{H}_\mathbf{d}^4$)	Online clustering keeps track of the temporal evolution of the health profiles via cluster updates, while offline clustering cannot.

Given the industrial need accommodating this thesis, validating the hypotheses will result in an end-to-end processing pipeline of four modules: Cycle extraction, Feature learning, Health detection, and Prognostics.

Chapter 4

Cycle extraction

Contents

Summary

Systems with cyclic behavior generates repeating cycles on the stream of sensor signals. The cycles reflect the condition of a system and must be correctly extracted. Yet, the stream is unannotated: no information about the shape of cycles nor their type is available beforehand. We propose an active learning-based algorithm called *Interactive Cycle Extraction* (InterCE) that learns to extract cycles by querying a domain expert when necessary and uses such feedback for self-improvement. InterCE addresses the hypotheses H_c^1 and H_c^2.

The experimental results show that InterCE achieves an accuracy superior to that of the expert system. Its reactivity is competitive to its offline counterpart. As for efficiency, InterCE is able to process one file per second, and its querying strategy succeeds to limit the number of queries to less than 0.1% over the total number of files.

1 Introduction

In the railway, most systems have cyclic behavior. A system is designed to perform one or more functions repeatedly during its lifetime. The realization of one function constitutes a *cycle* that repeats over time as the system continues to perform such function. For instance, a PAS opens and closes, performing the opening cycles and closing cycles.

If a system is equipped with sensors, the onboard sensors record the signals measured on the system continuously or periodically, creating a stream of sensor data. Because a system has a cyclic behavior, repetitive patterns exist on the stream as cycles. As an example, Figure 4.1 shows the raw data from an input file. We see the same pattern repeating many times. The occurrences of this pattern are the cycles we want to extract.

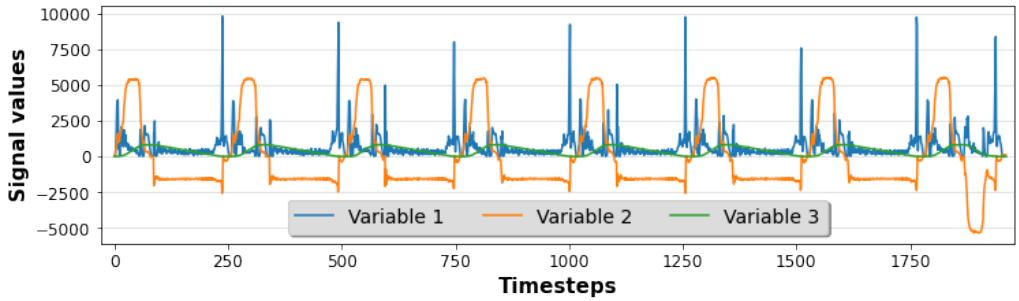


FIGURE 4.1: Many occurrences of the same pattern appear in this input. Each occurrence is a cycle to be extracted and labeled.

The behavior of a system in operation affects the shape of the cycles it creates. Therefore, the cycles shed a light on the condition of a system and must be extracted accurately because they are the primary ingredient of the analysis.

One solution is to define a set of fixed rules that identify the boundary of the cycles based on domain knowledge [226]. However, these rules do not adapt to changes in the data. For example, a system may have undergone an update that changes its functionality and alters its cycles. Consequently, the expert must revise the rules and/or redefine them. Furthermore, there may be cycles whose shape is unknown to the expert, possibly due to an unexpected event occurring in the systems. If the static rules cannot catch these unexpected cycles, a fault may be undetected, which leads to undesired problems and inefficient maintenance. Therefore, we devise a framework that learns to extract the cycles and to identify their function (e.g., assigning a label "opening" to an opening cycle), such that this algorithm starts learning from scratch and adapts itself to changes in the data.

Nonetheless, the model cannot learn to detect and label cycles by itself at the beginning, because the sensor stream is unannotated with cycle information. Although identifying the boundary of a cycle is possible via unsupervised techniques, labeling a cycle according to its function is impossible without prior knowledge. That is why the input from the domain experts is indispensable to give the framework some labeled examples to learn from.

As a result, we develop the framework *Interactive Cycle Extraction* (InterCE) to extract cycles from an unlabeled stream using an active learning approach [201]. When the first inputs arrive, InterCE does not have any information about the task, so it issues queries to a human (a domain expert) to ask for guidance. Once the human sends their feedback, InterCE updates its understanding of the task and learns to extract and label the cycles on its own.

The hypotheses to be validated for this task are:

($\mathbf{H}_c^1$) Using online learning to learn to extract cycles has a higher reactivity than using offline learning.

($\mathbf{H}_c^2$) Online learning that enables model update via human feedback performs superior to, or on-par with, a static expert system in accuracy.

Besides reactivity and accuracy, efficiency is a key criterion to handle the volume and speed of the stream. We assess the efficiency of InterCE via its processing time and the number of queries issued.

We evaluated InterCE on the data sets from two fleets of PASs described in 2. The results show that InterCE improves substantially the baseline performance of the expert system. The reactivity of InterCE is not superior to its offline counterpart but remains competitive. InterCE processes in average one input file per second, and the querying strategy allows InterCE to reduce the number of queries sent to the human: in both data sets, InterCE reaches a query ratio less than 1% over 100000 data files.

This chapter is organized as follows. Section 2 reviews the techniques usable for detecting repeating patterns in a time series. Section 3 describes InterCE. Section 4 dives into the experimental evaluation of InterCE on the sensor data from the PASs on the R2N and NAT fleets. Finally, Section 5 concludes the works done for the module Cycle extraction.

2 State of the art

Translated to data, a cycle is a subsequence of signals that reoccurs on the stream. The number of cycle types[1] depend on the number of functions a system performs, but a cycle has only one type. Cycles of the same type may vary slightly but should share a closely similar shape and value range.

Cycle detection consists of detecting the boundary of a cycle from raw data, while cycle identification maps an extracted cycle to the string describing the associated function. Cycle extraction consists of cycle detection followed by cycle identification. Applicable techniques for cycle extraction are related to time series analysis, because a cycle per se is a time series. The techniques we studied for this task are change point detection, motif discovery, and discretization.

2.1 Change point detection

Change points are sudden changes in time series data [14] and can be individual points or subsequences between two windows (Figure 4.2). Change point detection (CPD) is the task of finding such changes.

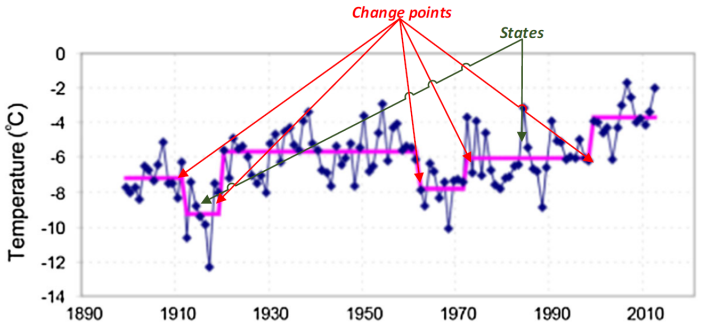


FIGURE 4.2: Pink lines indicate states whose boundaries are change points [14]

[1] We also refer to the type of a cycle as its *class* or *label*.

CPD can be online or offline. Online CPD detects changes on-the-fly on newly arriving data, thus being more reactive. Offline CPD receives the entire batch of data and identifies whether a change has occurred and at what position in the data [14].

Machine learning for CPD can be supervised or unsupervised. Supervised CPD learns a multiclass classifier that assigns a state label to each subsequence, or a binary classifier that gives a binary label to each subsequence: state-transition or within-state sequences [51, 57, 70]. Unsupervised CPD does not require prior training, and some examples are likelihood ratio [145], Gaussian and Bayesian processes [2], kernel-based methods [38], or comparison of the underlying distributions between consecutive subsequences [124].

A representative algorithm for CPD is Prune Exact Linear Time (PELT) [124]. PELT finds change points that optimize the following cost function:

$$\sum_{i=1}^{m+1} C\left(y_{(\tau_{i-1}+1:\tau_i)}\right) + \beta f(m) \tag{4.1}$$

where y is the time series of n points, m the number of change points, τ_i the i^{th} change point, $C(y_{(\tau_{i-1}+1:\tau_i)})$ the cost of one segment from $y_{\tau_{i-1}+1}$ to y_{τ_i}, and $\beta f(m)$ a penalty term to avoid overfitting. Killick, Fearnhead, and Eckley [124] use optimal partitioning to search for the optimal values of τ^* that minimizes (4.1) combined with a dynamic pruning scheme to eliminate candidate values of τ^* that can never be the optima. It gives an exact solution with a computational cost linear to n.

2.2 Motif discovery

Torkamani and Lohweg [223] define *motif discovery* as the task to "find frequent unknown patterns in a time series without any prior information about their locations or shapes". Figure 4.3 illustrates the case where two motifs are found from a telemetry stream. Motif discovery appears suitable for cycle extraction because it can detect repetitive patterns from an input time series.

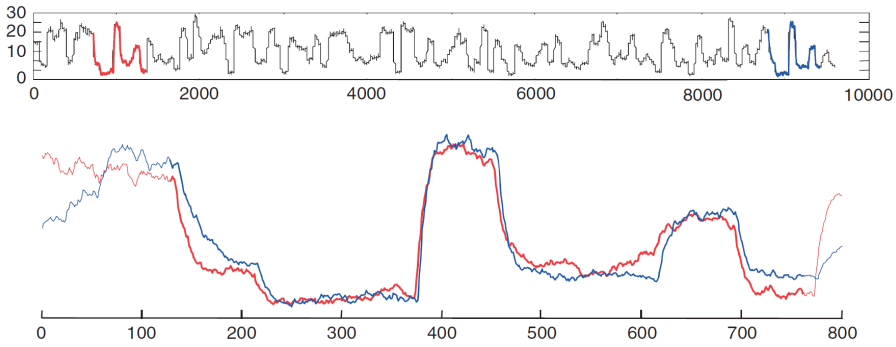


FIGURE 4.3: Two motifs in red and blue found in a steam flow telemetry [223].

Matrix profile [244] is a prominent research direction for motif discovery, with a growing research body since 2016 and implementations in different programming languages[2]. A matrix profile MP is a vector computed from a time series T such that each element MP_i stores the Euclidean distance between the subsequence $T_{i,m}$, starting at the timestamp i of length m, to its nearest neighbor $T_{j,m}$. To facilitate nearest neighbor retrieval, a matrix profile is accompanied by a matrix profile index I such that $I_i = j$ means that the nearest neighbor to the subsequence $T_{i,m}$ is is $T_{j,m}$.

[2] https://matrixprofile.org

Figure 4.4 shows how a matrix profile P is computed on a time series T for a query Q, via an intermediate computation of the distance profile D. The distance profile is a vector on a query Q of length m that stores, for each element, the Euclidean distance between Q to all the subsequences of length m in T. The motifs that are most similar to Q is the ones whose element in the matrix profile has the lowest value.

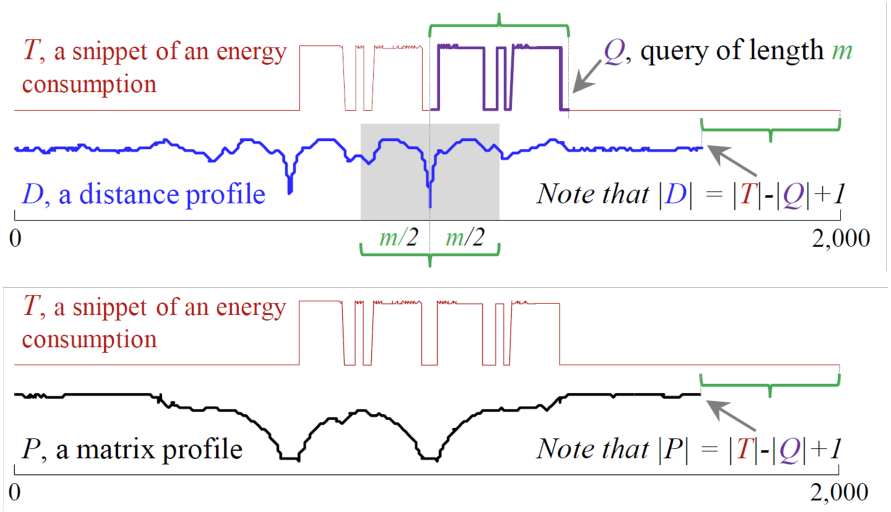


FIGURE 4.4: An example of the distance profile and the matrix profile [244]

To use matrix profile for cycle extraction, we need to determine the desired length of the cycles. However, one matrix profile is computed specifically for one query (one subsequence), but at the beginning we do not have any example of the cycles we wish to extract. Two solutions arise: (i) we ask an expert to extract one example cycle from an input and use it as the query to compute the matrix profile, or (ii) given a desired length, we compute the matrix profile for all subsequences of such length from the input data, then manually examine the resulting motifs to pick the correct cycles.

2.3 Discretization

Discretization is a task that finds a symbolic representation of a time series [144]. Discretization collapses continuous values in the input series and returns a simpler series with symbolic values that distinguish segments of different characteristics. Discretization is somewhat related to change point detection. For cycle extraction, discretization may return a representation that makes a cycle clearly stand out from the input data.

There are three approaches to time series discretization [121]:

- Sliding window: A segment is grown until it crosses an error bound. Algorithms of this type are simple, intuitive, and online-compatible.
- Top-down: The input series is recursively partitioned until the stopping criteria are met. Real data with noise may results in overly fragmented segments. These algorithms cannot run online.
- Bottom-up: Small segments are merged until the stopping criteria are met. Like top-down algorithms, bottom-up algorithms cannot run online.

SAX [144] is a classic algorithm for segmenting time series. SAX transforms a time series to a discrete sequence of letters such that the lower bound of the distance measure is guaranteed in both the original and symbolic spaces. SAX first uses Piecewise Aggregate Approximation (PAA) transformation [120] to reduce the dimensionality of the series, then assigns symbolic letters to the PAA

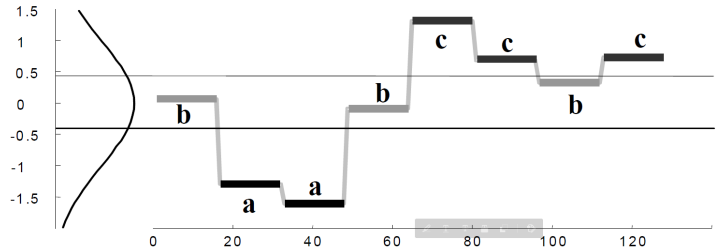


FIGURE 4.5: The PAA representation of a time series mapped into symbolic letters by the SAX algorithm [144]

representation based on breakpoints to ensure a high equiprobability of symbols within a SAX word (Figure 4.5).

2.4 Preliminary results

Having studied available approaches in the literature, we experiment with one representative algorithm of each category to see which approaches are the most suitable for InterCE. The algorithms chosen for the test are PELT [124] (change point detection), matrix profile [244] (motif discovery), and SAX [144] (discretization). We test each method on an example of data, displayed in Figure 4.6.

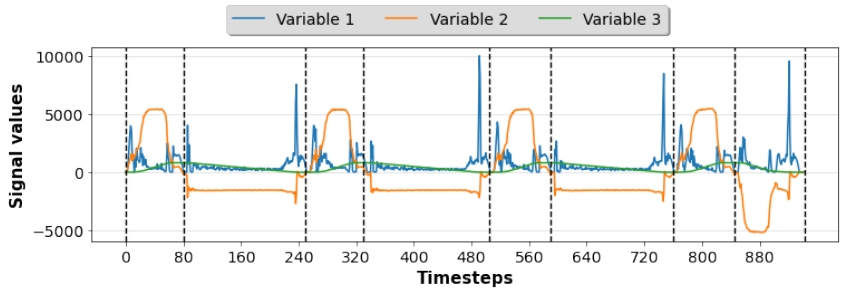


FIGURE 4.6: The signals in one input with multiple cycles (dotted lines = boundary of desired cycles)

The following subsections show the preliminary results obtained with PELT, matrix profile, and SAX on this example data.

2.4.1 Change point detection with PELT

We use the implementation of PELT from the `ruptures` library [224]. We test different values of penalty and finally obtain a good extraction on a penalty of 20 (Figure 4.7). Nevertheless, PELT misidentifies the last cycle by dividing it into two smaller cycles. Increasing the penalty only worsens the result. The penalty is thus a substantial hyperparameter to fine-tune.

Change point detection is intuitive for tackling cycle extraction because it is straightforward to apply and does not require to post-process the results to obtain the cycles.

2.4.2 Matrix profile

Matrix profile is a powerful tool to discover motifs or anomalies from time series data, but using it to detect multiple occurrences of the same motif is not so simple. Because the matrix profile stores only

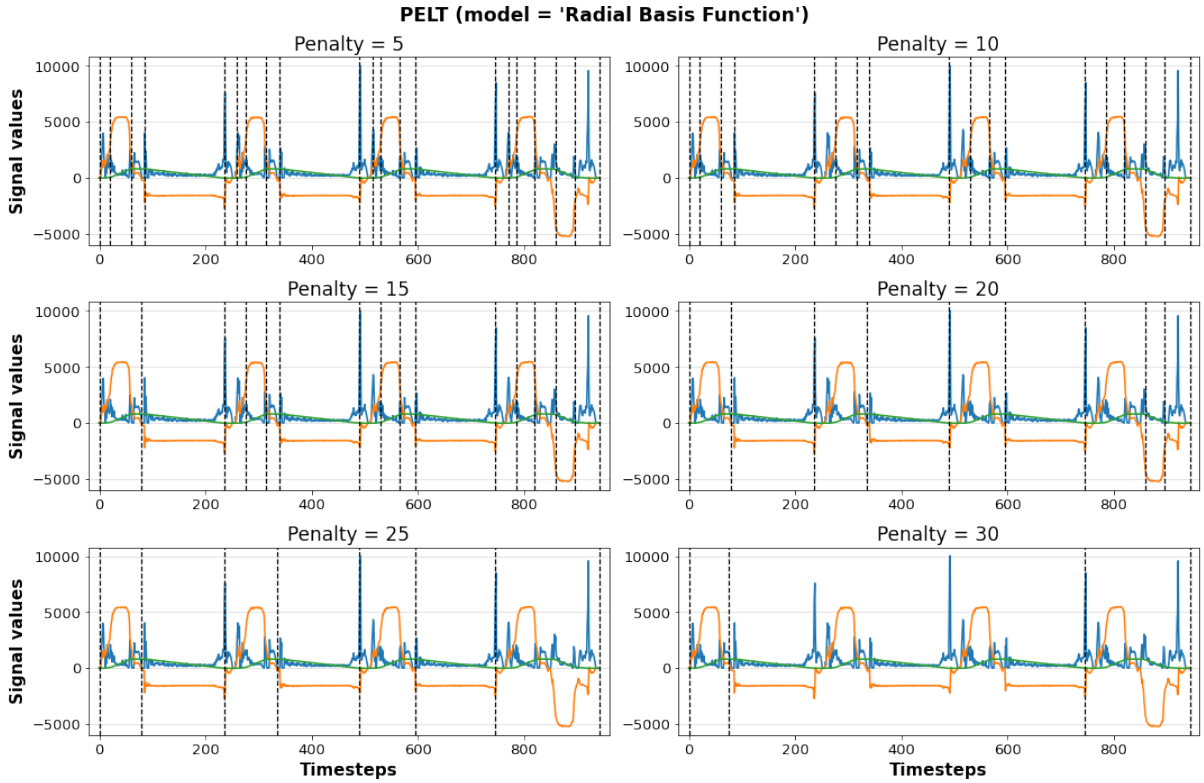


FIGURE 4.7: The change points detected by PELT on different penalties

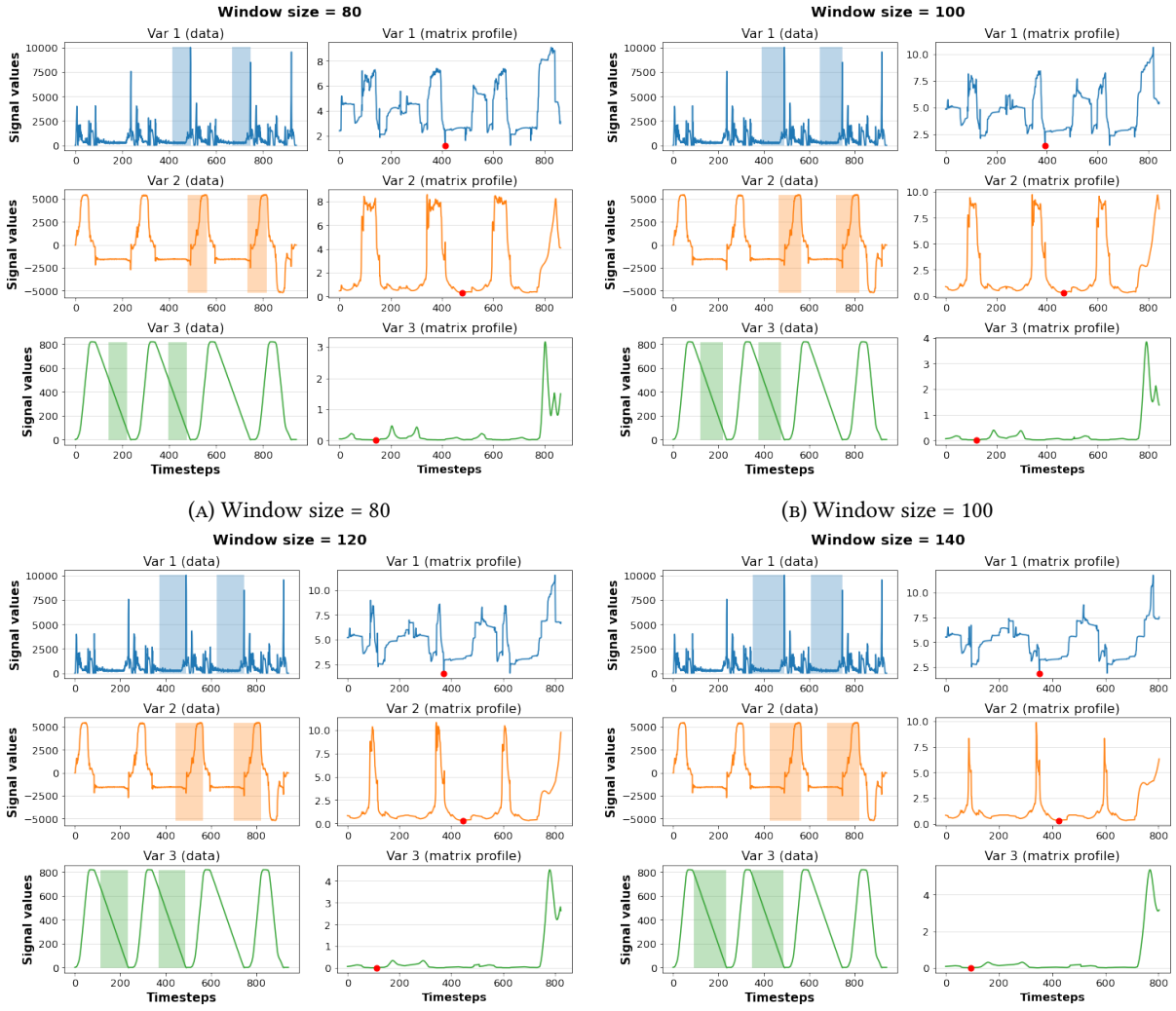
the distance between one subsequence and its nearest neighbor, we can only detect two occurrences of a cycle at most.

We use the matrix profile implemented in the Python package MPA [29]. The window size decides largely the shape of the cycles. Setting the window size requires a good understanding of the systems because different functions of the systems do not last for the same duration. Even if the window size is set to the exact duration, the matrix profile does not guarantee an accurate extraction: it may recognize that a smaller motif exists in a big motif instead of finding the correct cycle shape we want to identify (Figure 4.8). We use various window sizes to try to capture the cycles where the Var 3 has an ascending trend, but the matrix profile fails to find them.

Another drawback is that the matrix profile works only on univariate time series, while cycles are multivariate time series. One matrix profile produces one extraction result for one variable in the raw signals. Therefore, it requires a post-processing that aggregates individual extractions, which is not a simple task to automate.

2.4.3 Discretization with SAX

We use the implementation of SAX in the `pyts` package [66]. Figure 4.9 shows the SAX representation for each variable in the input data. The dots indicate the symbol chosen by SAX to represent a signal value at one timestep. The raw signals are the faded lines. SAX does not recognize any separate segments in Variable 1, as it is highly noisy. Meanwhile, SAX does recognize several segments in Variable 2 and Variable 3. A long segment indicates a potential cycle while a short one is a transition period. The long segments in Variable 2 correspond to the cycles we want to extract, whereas the ones



(A) Window size = 80

(B) Window size = 100

(C) Window size = 120

(D) Window size = 140

FIGURE 4.8: Applying matrix profile on different window sizes from 80 to 140. The red dot indicates the smallest value in the matrix profile, i.e., the smallest distance between one subsequence (the motif) and its nearest neighbor (its occurrence).

in Variable 3 group two consecutive cycles in the same segment. The results do not change even when we increase the bin size.

Similar to the matrix profile, SAX works only on univariate time series. Post-processing is necessary to aggregate individual discretization results to product the correct cycles. The same post-processing rules may not be easily generalized to other data sets.

2.4.4 Observations

From these preliminary results, we discover that change point detection is the most fitting approach because its results are intuitive: a detected change point marks the boundary of a cycle without further post-processing. PELT is a good candidate for InterCE, but its results are not always accurate, therefore we also include other extraction algorithms in InterCE to mitigate the mistakes of individual algorithms. In other words, we use an ensemble of extraction algorithms to detect cycles, as ensemble

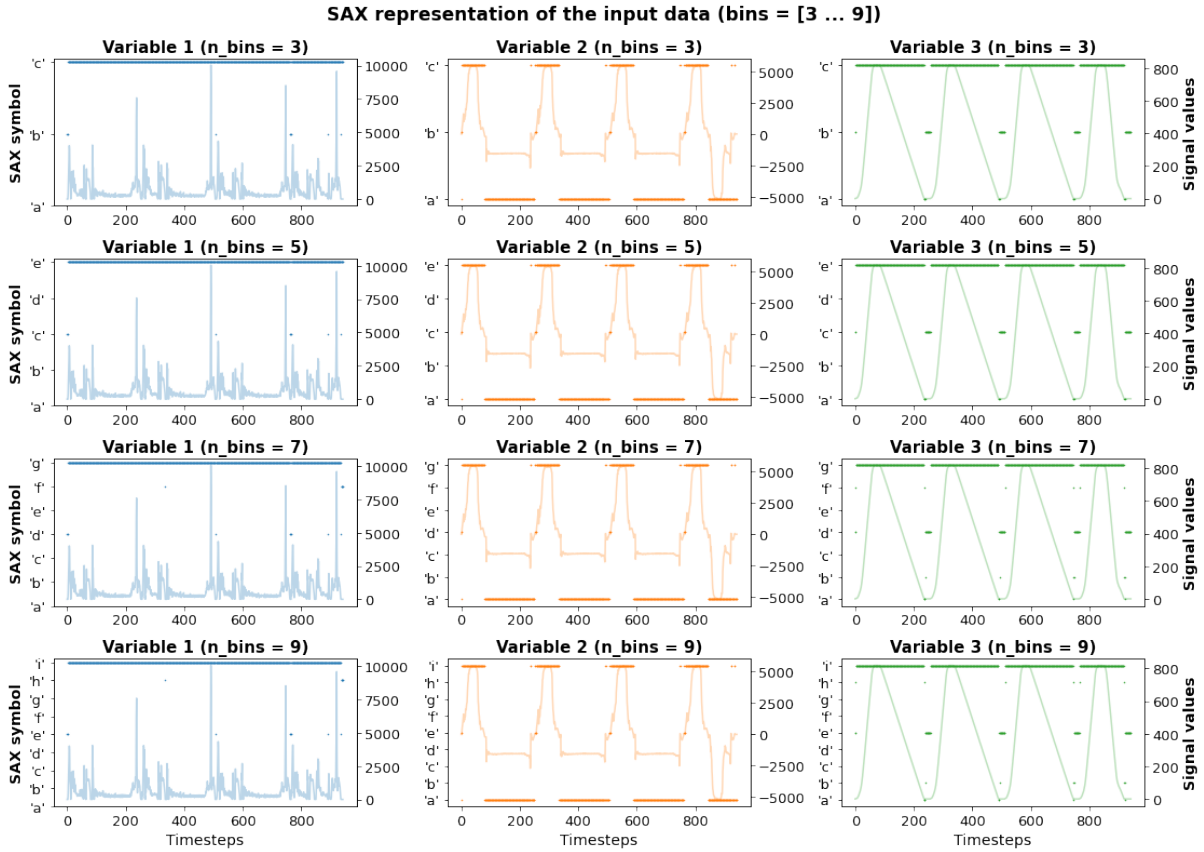


FIGURE 4.9: SAX representation (dots) of the raw signal (faded line)

learning is effective in reducing the variance of individual learners and increasing the overall accuracy, by aggregating the results from all learners [186]. The other extraction algorithms we develop also adopt the principle of change point detection.

3 InterCE: Interactive Cycle Extraction

In this section, we describe the framework InterCE that uses an active learning-based approach to query for human feedback on inputs it does not know how to process, and to detect and label cycles automatically otherwise.

First, we formulate the problem of cycle extraction on the data stream generated by a fleet of PASs (Section 3.1). Then, we detail the implementation of InterCE (Section 3.2). The nomenclature used in this section is given in Table 4.1.

3.1 Problem formulation

First, we formally define a cycle. Let $\mathcal{S} = \{S_1, \ldots, S_M\}$ be the fleet of M systems (the PASs) being monitored. Each PAS is equipped with a sensor that generates a data file every time the train, on which this PAS is located, enters a station. Each data file contains a multivariate time series of D variables and varying timesteps that records the cycles performed by this PAS. The number of timesteps in each file varies, depending on how long the train stays in a station. Over time, the continuous generation of these files forms a stream of data files, each of which contains a multivariate time series generated

TABLE 4.1: Nomenclature of InterCE

		CYCLE
$\mathcal{S} = \{S_1, \dots, S_M\}$		A fleet of M monitored systems
$C^T_{S_m}$		A cycle created by a system S_m at an instant T
$y^T_{S_m}$		A label indicating the type of a cycle $C^T_{S_m}$
$ctxt^T_{S_m}$		A set of context variable associated to a cycle $C^T_{S_m}$
		INPUT AND OUPUT
X		The input to InterCE (a data file, a time series, etc.)
$\hat{C}(X)$		The set of extracted cycles
		ENSEMBLE OF EXTRACTORS
$\mathcal{E} = \{E_1, \dots, E_J\}$		An ensemble of J extractors
$C_{E_j}(X)$		The cycles extracted from X by an extractor E_j
$\mathbb{C}(X)$ $\{C_{E_1}(X), \dots, C_{E_J}(X)\}$	=	The candidates proposed by all extractors in $\mathcal{E}$
		THE INNER MEMORY OF INTERCE
$\mathcal{M}$		The memory component
$Qr(X)$		A query to an input X
		THE KNOWLEDGE OF INTERCE
$\mathcal{K}$		The knowledge component
$Fb(Qr(X))$		A feedback answering the query $Qr(X)$

by a system $S_m \in \mathcal{S}$ at a time T. From such stream, we want to extract cycles representing different functions of S_m. A cycle $C^T_{S_m}$ is defined as follows.

Definition 3.1 (Cycle). *A* cycle *created by a system S_m at an instant T, denoted $C^T_{S_m}$, is a sequence of K D-dimension data points, that is, $C^T_{S_m} = [\boldsymbol{a}^{(1)}, \dots, \boldsymbol{a}^{(K)}]$ where $\boldsymbol{a}^{(k)} \in \mathbb{R}^D$. A cycle is always associated to a* label *$y^T_{S_m}$ indicating its type and to a* context *$ctxt^T_{S_m}$.*

A context is a set of variables that record the information surrounding the system in operation, such as the outside temperature, the mission code[3], or the train station. We assume that the context is explicitly known and can be easily extracted (for instance, from metadata accompanying the input data, or from a database). We do not handle hidden contexts in the scope of this work.

3.2 Implementation

We use an active learning-based approach to implement InterCE. An interface to interact with a human is essential. Figure 4.10 describes the components of InterCE. Given a new input X (a data file), InterCE feeds X to the ensemble of extractors $\mathcal{E}$, which produces a set of candidate cycles $\mathbb{C}(X)$. To check whether it has sufficient information to select the best cycles from $\mathbb{C}(X)$, InterCE verifies if its memory $\mathcal{M}$ has already processed inputs similar to X. If such is the case, it uses its knowledge $\mathcal{K}$ to select the best cycles among those proposed in $\mathbb{C}(X)$. Otherwise, it issues a query $Qr(X)$ to the human

[3] A mission of a train dictates the sequence of stations it stops by.

expert (*Human mode*). Once receiving the feedback $Fb(Qr(X))$, InterCE incorporates $Fb(Qr(X))$ to the knowledge $\mathcal{K}$ and updates its understanding of the task. The final cycles $\hat{C}(X)$ are returned and fed to the next module.

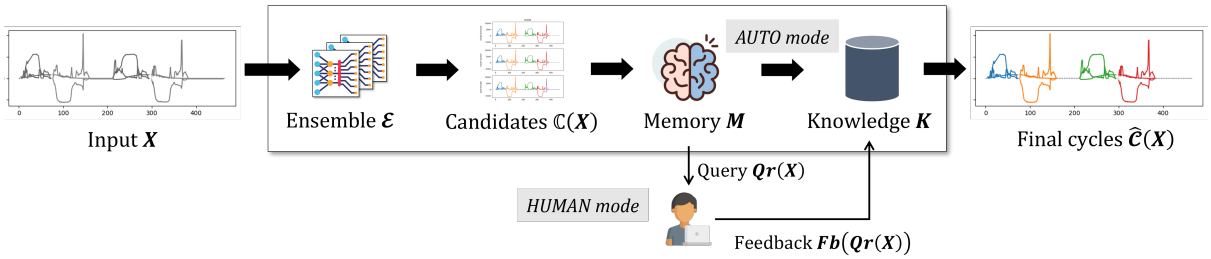


FIGURE 4.10: The components of InterCE

Further details of each component are given in the following sections.

3.2.1 Ensemble of extractors $\mathcal{E}$

We use an ensemble of J extractors $\mathcal{E} = [E_1, \ldots, E_J]$ to handle the variations in the cycles. Due to various factors, such as the operational context or data acquisition errors, the cycles of the same function are not always identical. Preliminary experiments show that a single extractor does not always succeed to capture all the correct cycles (Figure 4.11). "Expert system", "Auto Encoder", and "Simple extractor" are the names of three extractors we implement and will be explained shortly. We see that no extractor is able to capture the best cycles on all three examples. Therefore, we leverage ensemble learning to reduce the variance of individual extractors and to increase the overall accuracy. In InterCE, we use all three extractors and select the best cycles among their extraction results.

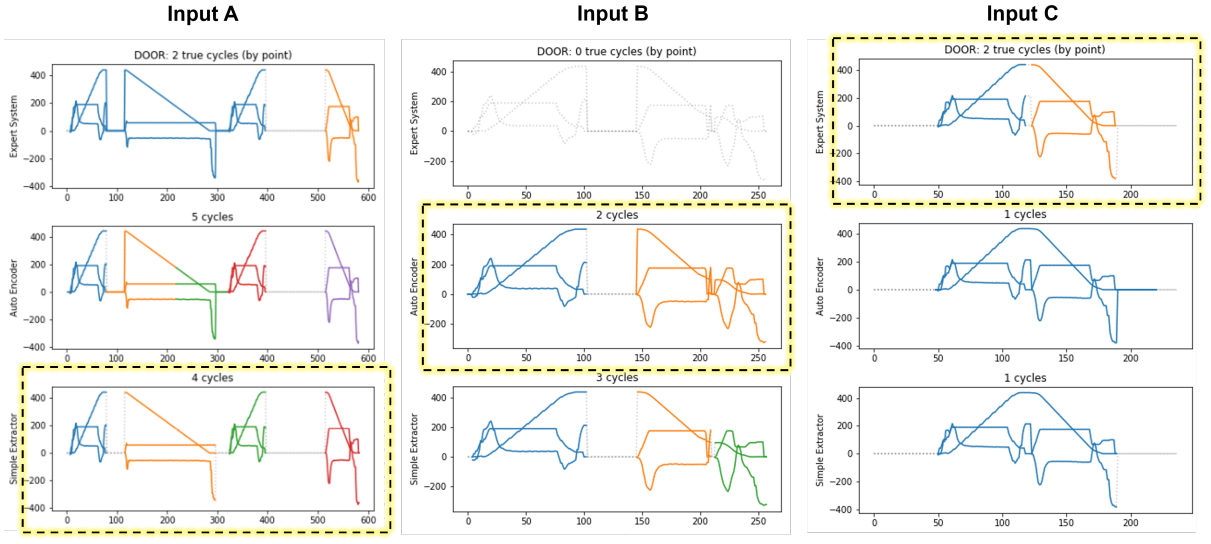


FIGURE 4.11: Each extractor (rows) yields a different extraction on three inputs (columns). The dotted box indicates the ground-truth extraction of each input.

The cycles extracted by the extractors in $\mathcal{E}$ constitutes the candidates $\mathbb{C}_X$. For example, in Figure 4.11, the ensemble $\mathcal{E}$ produces three candidates for the input A, with one candidate from each

extractor. We denote $C_{E_j}(X) = \{C_{S_m}^{(j,1)}(X), \dots, C_{S_m}^{(j,n_j)}(X)\}$ the set of cycles extracted from X by an extractor $E_j \in \mathcal{E}$, where $(j,1), \dots, (j,n_j)$ denotes the first to the n_j-th[4] cycle in X detected by E_j. The candidates $\mathbb{C}(X)$ is therefore:

$$\begin{aligned}\mathbb{C}_X &= \left\{C_{E_1}(X), \dots, C_{E_J}(X)\right\} \\ &= \left\{\left\{C_{S_m}^{(1,1)}(X), \dots, C_{S_m}^{(1,n_1)}\right\}, \dots, \left\{C_{S_m}^{(J,1)}(X), \dots, C_{S_m}^{(J,n_J)}\right\}\right\}\end{aligned} \tag{4.2}$$

We implement three extractors, therefore $|\mathcal{E}| = J = 3$, but more extractors can be easily integrated in $\mathcal{E}$ to diversify the learners in the ensemble.

Activity-based extractor This extractor relies on the data acquisition mechanism: the signals are only recorded by the sensors when the system is performing an activity. A system in idle state does not produce any signals (for example, a PAS stays open when the train stops at a station), during which a segment of null values is observed from the data. The activity-based extractor identifies segments of non-zero consecutive data points separated by a segment of null values as cycles (Figure 4.12).

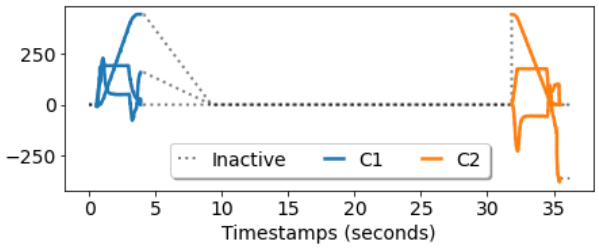


Figure 4.12: The activity-based extractor identifies two cycles in blue and orange, separated by a segment of null signals (dotted).

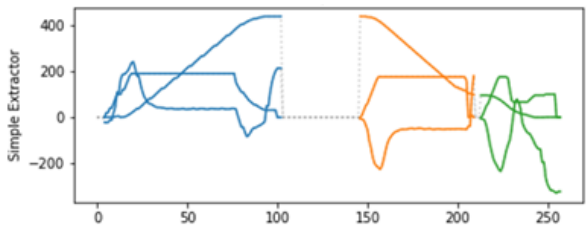


Figure 4.13: A perturbation in the data acquisition causes the loss of some data points, cutting the second cycle (orange) to two smaller cycles (orange and green).

This is a simple extractor and requires no learning at all (thus its alternative name "Simple extractor"). Despite its simplicity, it works well in some cases (NAT data), except when there are perturbations during data acquisition that remove some data points in a cycle, consequently separating this cycle into multiple non-consecutive segments (Figure 4.13).

Autoencoder-based extractor Lee, Ortiz, Ko, and Lee [136] propose to use an autoencoder to detect cycles[5] from time series by comparing their learned representation. First, a time series is cut into

[4] T indicates real-life time whereas $1, \dots, n_j \in \mathbb{N}^*$ are just the counts. This is to simplify the notation.

[5] They coined the task as "time series segmentation", but in nature it is the same as cycle detection, without cycle identification.

small subsequences. An autoencoder is trained on these subsequences and returns the representation of each subsequent. Then, the distances between the representation of pairs of consecutive subsequences is computed and forms a distance curve. Any peak in the distance curve indicates the start of a new cycle.

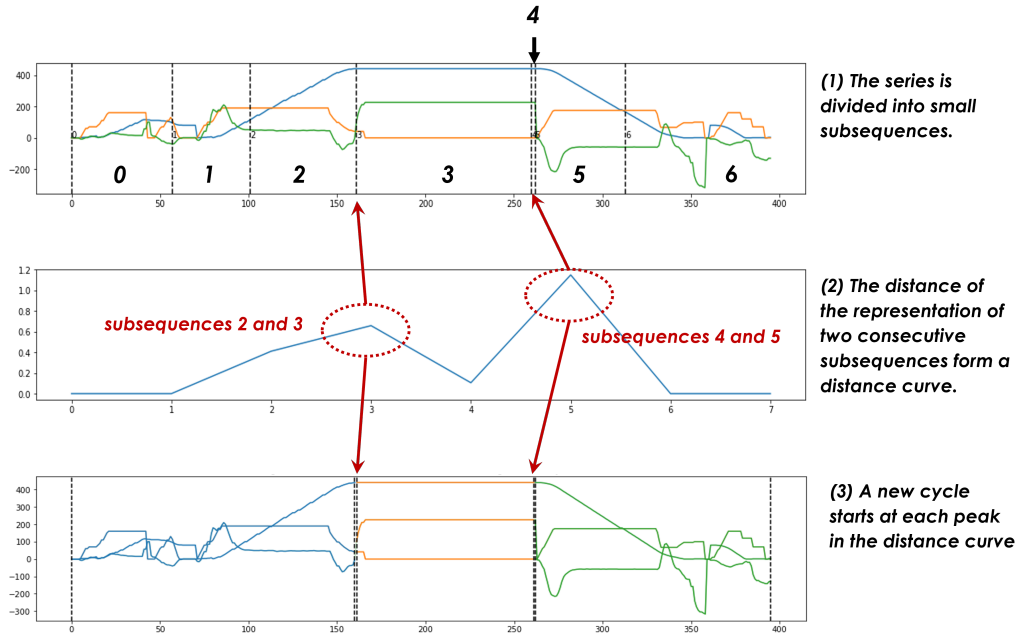


FIGURE 4.14: Three steps of the Autoencoder-based extractor

This extractor is learning-based, but the size of the subsequences must be correctly determined. Yet, the cycles of different types or even of the same type do not always have the same length. An inadequate choice of the subsequence size leads to an incorrect extraction. It is not easy to define this parameter, even with domain knowledge. An input may contain cycles of many types (and of various length), but the autoencoder-based extractor uses a unique value to attempt to find all the cycles, which may cause the errors in Figure 4.15: the blue and orange cycles are correctly identified, but the other two cycles are not. The green cycle has a large gap of null signals at the beginning, while the last cycle is cut into two smaller cycles, colored red and purple.

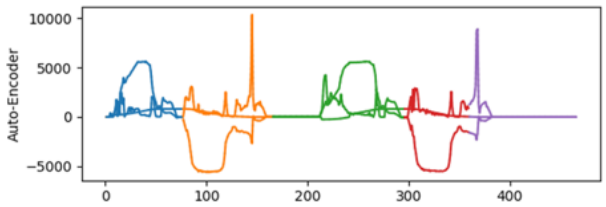


FIGURE 4.15: Setting the subsequence length incorrectly leads to an incorrect extraction of cycles (only the blue one and orange one are correct).

The primary drawback of this extractor is that we need to train it before using it to extract representations. Two options to tackle this issue are (i) to extract some subsequences from a training set and train the autoencoder on it before putting InterCE to use, or (ii) to train the autoencoder incrementally on new data, but it cannot be used until it has reached convergence. In our experiment, we pick the first option to avoid suboptimal performance of this extractor, which would add more uncertainty to the evaluation.

Expert system extractor The expert system is a set of rules hand-crafted by a domain expert. An expert system can capture most of the expected occurrences of cycles, but it may fail to recognize cycles that are unknown even to the human experts (Figure 4.16). Machine learning helps to catch these cycles by learning patterns from the data.

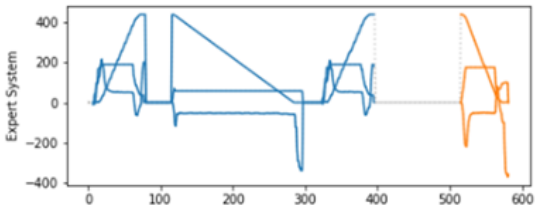


Figure 4.16: Unexpected cycles (those mixed in the first segment in blue) are not recognized by the expert system.

For any new use case (batteries, air-conditioning, traction, etc.), a new expert system must be made from scratch. Developing an expert system takes two to six months, depending on the complexity of the data. Machine learning helps to alleviate the workload of the human experts because the same algorithm can be applied on new data sets without major changes, only with different hyperparameters.

3.2.2 A memory $\mathcal{M}$ to keep track of previously processed data

Whether to process an input X in the Human mode or in the Auto mode relies on a memory $\mathcal{M}$ that stores the unique patterns of past inputs. We denote the set of stored patterns (or motifs) as $\mathcal{P}$. By design, a system performs a fixed number of functions, and repeating patterns of these functions frequently appear in the data. InterCE issues queries only when it does not know how to process an input. If the feedback to such inputs is provided, InterCE memorizes the patterns in its memory $\mathcal{M}$.

Searching for similar motifs The memory $\mathcal{M}$ retains one representative series as a motif of future inputs that may share the same pattern (Figure 4.17). When a new input X arrives, InterCE computes the distance from X to $\forall m \in \mathcal{P}$. If X is different from all the motifs in $\mathcal{P}$, X is a new pattern and is added in $\mathcal{P}$, and InterCE issues a new query for X. Otherwise, X is processed automatically.

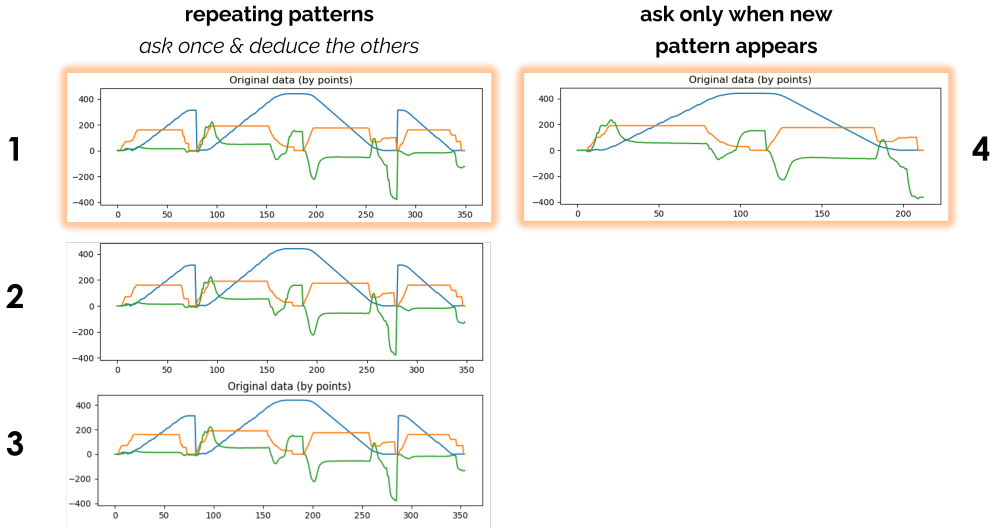


Figure 4.17: The inputs may have similar patterns, represented by a single query. Two patterns that will be stored in $\mathcal{M}$ are those in highlighted boxes.

Because a brute-force search for the closest motif $m \in \mathcal{P}$ to X is time-consuming, we devise a heuristic search based on the intra-distances between motifs. We maintain in $\mathcal{M}$ a set of motifs $\mathcal{P}$ and the intra-distances between each motif to all the other motifs in $\mathcal{P}$, that is, $intraDistances[m_i] = motifDistance(m_i, m_j), \forall m_i, m_j \in \mathcal{P}$. As illustrated in Figure 4.18, a new input X is a new motif if the distance from X to the motif m_1 closest to X is greater than the distance from m_1 to all the other motifs in $\mathcal{P}$. Otherwise, X might be an existing motif with noises.

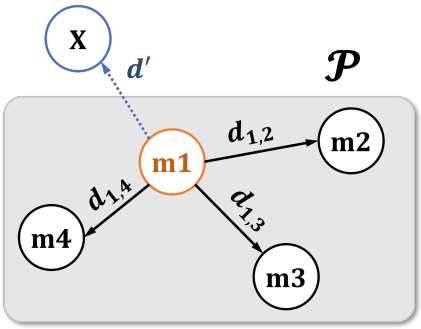


FIGURE 4.18: Heuristic decision on the novelty of a motif based on the intra-distances between motifs in $\mathcal{P}$

We also rely on the length ratio between an input X and a motif $m \in \mathcal{P}$ to decide whether a distance computation should be executed. We use Dynamic time warping [165] to compute the distance between two series because it is more accurate than the Euclidean distance, but unfortunately, it is also more computationally expensive. Therefore, we want to limit the call to the distance computation as much as possible. We only compute the distance if X and m are sufficiently different in length, gauged by a length threshold $\tau_L \in [0, 1]$. The length ratio between two series is the length of the shorter series divided by that of the longer one.

Algorithm 4.1 describes how an input X is recognized as a novel or existing motif, combining the heuristics of intra-distances and of length ratio.

- If there is no motif in $\mathcal{P}$, we add X as a new motif (line 2–4). Adding a new motif also updates the intra-distances between existing motifs (if any) and computes the intra-distances of the new motif to the existing ones (Algorithm 4.2).
- If there is only one motif $m \in \mathcal{P}$ and if the length ratio between X and m is smaller than τ_L, we add X as a new motif (line 5–10).
- Otherwise, we compute the distances between each motif $m \in \mathcal{P}$ to X and store them in an array *dist* initialized to NaN values. We only compute the distances if the length ratio between X and m is sufficiently large (line 13–15).
- After the distance computation, if all the values in *dist* are NaN, all motifs in $\mathcal{P}$ are different to X in terms of length, and we add X as a new motif (line 16–18).
- Otherwise, we retrieve the closest motif m_X to X and verify whether the distance between X and m_X are greater than all the intra-distances of m_X to other existing motifs. If it is the case, X is sufficiently different from all motifs in $\mathcal{P}$ and is added as a new motif; else, X is an existing motif (line 20–25).

QUERIES AND FEEDBACK If an input X is ruled as a new motif, InterCE creates a new query for X and sends it to the human annotator via an interactive interface. Upon receiving a new query, the human annotator selects the cycles they deem most correct from the candidates $\mathbb{C}_X$ and assign a label to each cycle. We define the query $Qr(X)$ and the feedback $Fb(Qr(X))$ in (4.3) and (4.4), respectively.

Algorithm 4.1: Verify whether an input is a known motif (`hasSeenData`)

Data: An input X
Input: A similarity threshold $\tau \in [0, 1]$, a length ratio threshold $\tau_L \in [0, 1]$
Output: True if X has been processed, False otherwise

$\mathcal{P} \leftarrow M$.`retrieveAllMotifs()`
if $|\mathcal{P}| = 0$ **then**
    M.addNewMotif(X)
    return False
else if $|\mathcal{P}| = 1$ **then**
    lenRatio ← computeLengthRatio(X, P[0])
    if *lenRatio* < τ_L **then**
        M.addNewMotif(X)
        return False
    else return True
else
    dist ← $[NaN \dots NaN]$ `/* array of all NaN values of length` $\mathcal{P}$ `*/`
    foreach m *in* P **do**
        lenRatio ← computeLengthRatio(X, m)
        if *lenRatio* ≥ τ_L **then** dist[m] ← motifDistance(X, m)
    if *all values in dist are NaN* **then**
        M.addNewMotif(X)
        return False
    else
        $m_X \leftarrow \arg\min_{min\mathcal{P}}$ dist
        intraDist ← intraDistances[idx]
        if *dist*[m_X] > *all values in intraDist* **then**
            M.addNewMotif(X)
            return False
        else return True

Algorithm 4.2: Add new motif and update the intra-distances (`addNewMotif`)

Data: A new motif m

$\mathcal{P} \leftarrow M$.retrieveAllMotifs()
$\mathcal{P} \leftarrow \mathcal{P} \cup \{m\}$
newIntraDist ← ∅
foreach $p \in \mathcal{P}$ $\{m\}$ **do**
    intraDist ← M.intraDistances[p] `/* retrieve the intra-distance of` p `*/`
    motifDist ← motifDistance(m, p) `/* distance between` m `and` p `*/`
    intraDist ← intraDist ∪ {motifDist} `/* new entry to the intra-distances of` p `*/`
    newIntraDist ← newIntraDist ∪ {motifDist} `/* new entry to the intra-distances of` m `*/`
M.intraDistances[m] ← newIntraDist

We denote $y_{S_m}^{(j,i)}$ the label of the cycle $C_{S_m}^{(j,i)}$ given by the human expert to the i^{th} cycle found in X by the extractor E_j.

$$Qr(X) = \langle\ X,\ \mathbb{C}(X)\ \rangle \tag{4.3}$$

$$Fb(Qr(X)) = \left\{ \left\langle C_{S_m}^{(j,i)}(X),\ y_{S_m}^{(j,i)} \right\rangle \right\}_{\substack{1 \leqslant j \leqslant J \\ 1 \leqslant i \leqslant n_j}} \tag{4.4}$$

In practice, a human is not always available to answer a query. InterCE must not wait for the answer while blocking incoming inputs. To circumvent this issue, we implement an asynchronous communication that allows InterCE to continue processing new inputs while queuing unanswered queries. InterCE will solve these queries once the feedback arrives.

Yet another issue arises. InterCE risks to create queries similar to those that have not been answered. Algorithm 4.1 switches InterCE to the Auto mode if it verifies that the input X is similar to another input Y that has been seen previously. But, if $Qr(Y)$ has not been answered, InterCE does not have sufficient information to solve $Qr(X)$ either, thus it creates a redundant query $Qr(X)$.

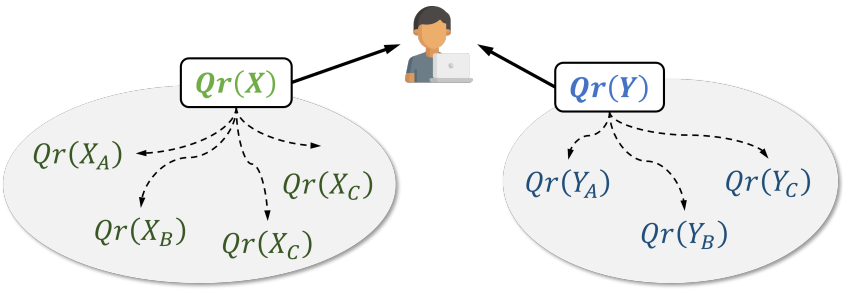


FIGURE 4.19: Only official queries (in **bold**) of two distinct patterns X and Y are sent to the humans. Buffered queries (in *italic*) are solved automatically based on the feedback to their official query.

To avoid sending repetitive queries, we distinguish *official* queries from *buffered* queries. Queries of similar inputs are grouped and represented by one official query, whereas the others are buffered (hence the name "buffered" queries). Only the official query is sent to the human. Once the feedback to an official query arrives, all of its buffered queries are solved automatically using this feedback (Figure 4.19).

3.2.3 Knowledge $\mathcal{K}$ to select the best cycles

The knowledge $\mathcal{K}$ has two roles: maintaining the feedback in the form of clusters and automatically selecting cycles.

To learn the typical cycle shape of a function, we use clusters to collect both the cycle signals and the associated labels. A feedback $Fb(Qr(X))$ contains the cycles selected by the human experts from a query $Qr(X)$. Once receiving a feedback, $\mathcal{K}$ updates the clusters by assigning the selected cycles in the corresponding clusters, such that a cluster is constituted of cycles of similar shape and thus is representative of a function.

We do not use a clustering algorithm to group the feedback cycles to avoid cascading unwanted inaccuracy in $\mathcal{K}$. Instead, we assume that the human always extracts and labels the cycles correctly, so we simply group the feedback cycles by the labels assigned to them. All the cycles having the same label are grouped into one cluster. Each cluster is represented by a centroid that is the average computed across the timesteps of the cycles in this cluster.

It is possible to update the clusters incrementally, by recomputing a cluster's centroid every time we add the cycles in a new feedback to this cluster. We store the clusters directly in memory to enable fast access when $\mathcal{K}$ automatically selects the cycles. The memory space dedicated to storing the clusters can potentially grow large, as InterCE works on an infinite data stream. Nonetheless, because the systems perform a limited number of functions, they produce a limited number of unique cycle types. We expect InterCE to run mostly in Auto mode and ask fewer queries after it has seen most of the common data motifs.

Algorithm 4.3 describes how $\mathcal{K}$ determines the best cycles from the candidates $\mathbb{C}_X$ of an input X. First, it computes the average distance between all the cycles C_{E_j} detected by an extractor $E_j \in \mathcal{E}$ to estimate how far C_{E_j} is from the feedback cycles learned by $\mathcal{K}$ (line 2–7). Then, the extractor producing cycles closest to the feedback cycles (i.e., having the smallest average distance) is picked as the best one (line 8). All of its cycles are retained and labeled according to the cluster closest to each of its cycle (line 9–13). Finally, the labeled cycles are returned as final extraction result.

Algorithm 4.3: Selecting cycles from $\mathbb{C}(X)$ in Auto mode (`selectCycles`)

Data: The candidates $\mathbb{C}(X)$
Output: The final cycles $\hat{C}(X)$

1 $distances \leftarrow \varnothing$
2 **foreach** *candidate* $C_{E_j}(X)$ **do**
3 $\quad distances[E_j] \leftarrow 0$
4 $\quad$ **foreach** *cycle* $C_{S_m}^{(j,i)}(X)$ *in* $C_{E_j}(X)$ **do**
5 $\quad\quad c \leftarrow K.\texttt{findClosestCluster}(C_{S_m}^{(j,i)}(X))$
6 $\quad\quad distances[E_j] \leftarrow distances[E_j] + \texttt{distance}(C_{S_m}^{(j,i)}, c)$
7 $\quad distances[E_j] \leftarrow \frac{distances[E_j]}{n_j}$ `/* n_j is the number of cycles detected by E_j */`
8 $\hat{C}(X) \leftarrow \arg\min_{C_{E_j} \in \mathbb{C}(X)} distances$
9 $R \leftarrow \varnothing$ `/* the set contains the detected cycles and its label */`
10 **foreach** *cycle* $C_{S_m}^T \in \hat{C}(X)$ **do**
11 $\quad c \leftarrow K.\texttt{findClosestCluster}(C_{S_m}^T(X))$
12 $\quad y_{S_m}^T \leftarrow c.\texttt{getLabel}()$
13 $\quad R \leftarrow R \cup \{(C_{S_m}^T, y_{S_m}^T)\}$
14 **return** R

For now, $\mathcal{K}$ selects the cycles from a single extractor. We have not yet enabled $\mathcal{K}$ to select cycles from different extractors in $\mathbb{C}(X)$. This will be addressed in future works.

3.3 Full algorithm of InterCE

The logic of InterCE is explained in Figure 4.20, accompanied by Algorithm 4.4 and 4.5. The query management and feedback management in InterCE communicate asynchronously.

The querying side of InterCE creates queries for new inputs and decides whether a query should be marked as official and sent to humans, or should be buffered to be solved in bulk once the feedback to the reference query is available (Algorithm 4.4 and the parts linked by solid lines in Figure 4.20).

The feedback side continuously waits for new feedback. Once a feedback is available, it updates the knowledge $\mathcal{K}$ and solves queued queries, both the official and buffered ones (Algorithm 4.5 and the parts linked by the dotted line in Figure 4.20). InterCE solves an official query $Qr(X)$ by simply taking all the cycles and labels indicated the human in $Fb(Qr(X))$ to update the clusters in $\mathcal{K}$. InterCE solves

the buffered queries $Qr(X')$ linked to $Qr(X)$ by running Algorithm 4.3 on $Qr(X')$ using the clusters that were updated on $Fb(Qr(X))$.

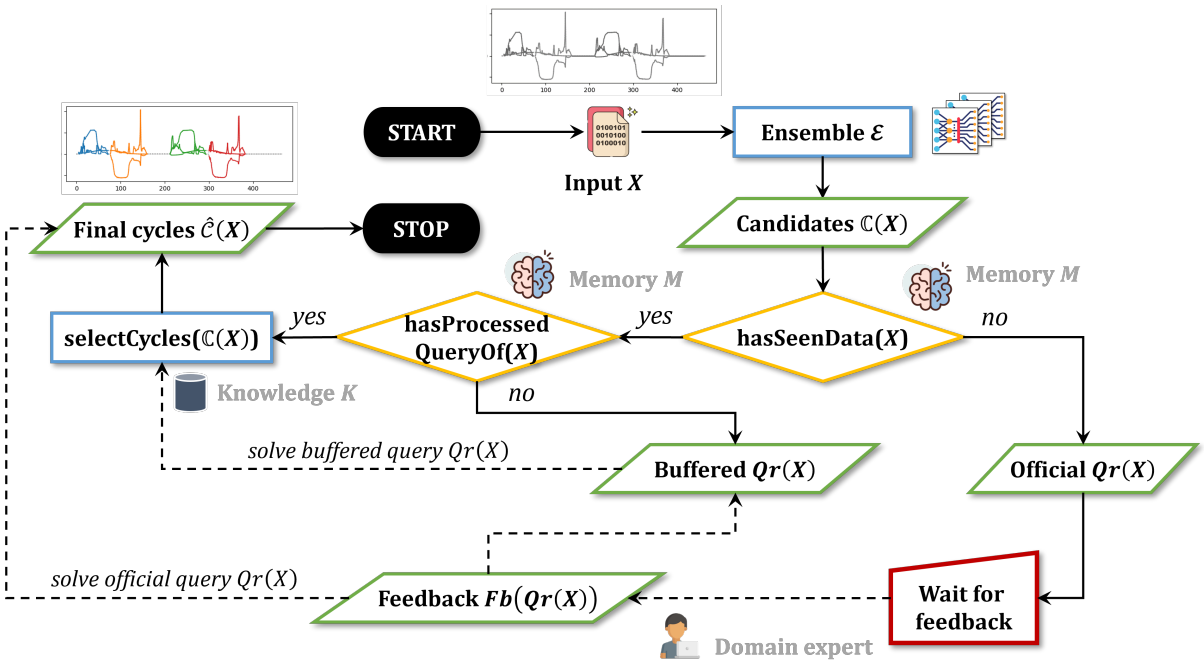


FIGURE 4.20: Flowchart of InterCE

Algorithm 4.4: Full algorithm of InterCE (managing queries)

Data: An input X
Output: The final cycles $\hat{C}(X)$ or a query $Qr(X)$

```
ℂ(X) ← ℰ.extract(X)
if M.hasSeenData(X) is True then
    if M.hasProcessedQueryOf(X) then Ĉ(X) ← K.selectCycles(ℂ(X))
    else
        Qr(X) ← M.createQuery(X, ℂ(X))
        markBuffered(Qr(X))
else
    Qr(X) ← M.createQuery(X, ℂ(X))
    markOfficial(Qr(X))
    sendToHuman(Qr(X))
```

Algorithm 4.5: Full algorithm of InterCE (processing feedback)

Output: The final cycles to an official query and to the linked buffered queries

```
/* run in loop to always wait for feedback */
while True do
    Fb(Qr(X)) ← detectNewFeedback()
    Ĉ(X) ← K.solveOfficialQuery(Qr(X))
    𝒬 ← K.getBufferedQueriesOf(Qr(X))
    foreach Qr(X') ∈ 𝒬 do Ĉ(X') ← K.selectCycle(Qr(X'))
```

4 Experimental results

To validate the hypotheses H_c^1 and H_c^2, InterCE must be able to reach convergence earlier than its offline counterpart (H_c^1), and to perform on-par or superior to the expert system in terms of accuracy (H_c^2). Efficiency being a crucial criterion, we test InterCE on two additional criteria that are the processing time per input and the query ratio. Therefore, we evaluated InterCE against four criteria: (i) the extraction accuracy, (ii) the convergence time, (iii) the execution time, and (iv) the ratio of queries over the total data.

To assess (i) and (ii), we involve a human in the experiments and run the experiments on 1000 data files. The accuracy is the number of cycles correctly detected and labeled. The expert system serves as the baseline performance. The convergence time is the time from the reception of the first input file until InterCE attains a stable extraction accuracy[6].

We evaluate (iii) and (iv) by running InterCE without human annotation on a large stream of 100000 files. The human annotators are not involved for the following reasons. First, it is tedious for a human to deal with such volume of data; even if the number of queries may not be high, it is still time-consuming to wait until the end of the experiments. Secondly, human annotation is stochastic: they may choose to annotate queries in any order randomly, which does not guarantee reproducible results.

We run the experiments on NAT and R2N data sets, using a Windows 10 machine with an Inter(R) Core(TM) i7-8850H CPU @ 2,60GHz 2,59GHz, with 16GB RAM. No GPU is used for the experiments.

4.1 Extraction accuracy

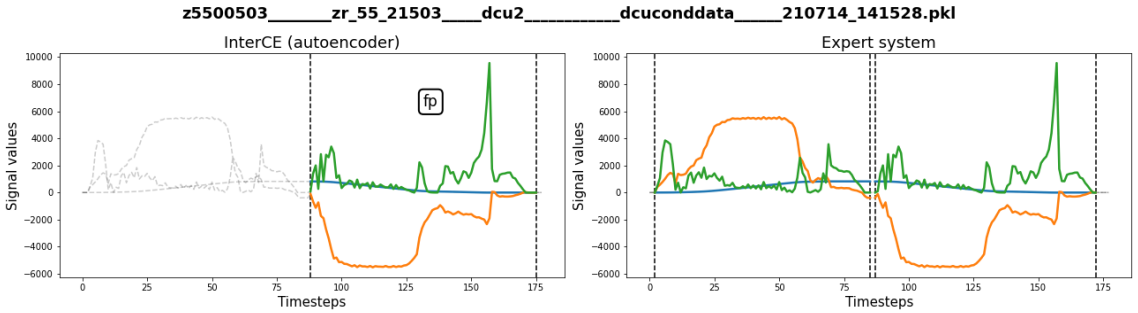
To assess the accuracy of InterCE, we count the number of cycles that are correctly extracted by InterCE and divide it by the total number of expected cycles. We examine each data file manually. From a file, we first find the expected cycles. Then, we check if InterCE succeeds to extract these cycles. The full experiment results are accessible on this site.

For both InterCE and the expert system, the score is attributed as follows.

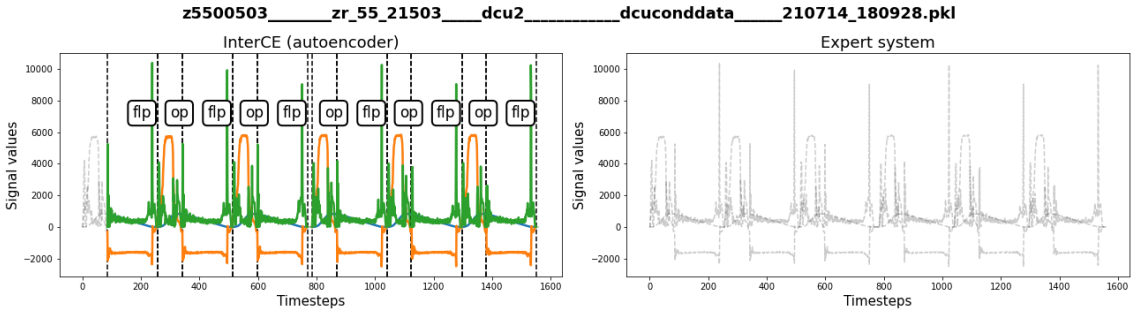
- We assign a count of one to one cycle that is correctly detected and labeled.
- If a cycle is undetected or detected but mislabeled, we assign a count of zero.
- If InterCE detects a cycle that includes other cycles (e.g., a big cycle with one closing and one opening cycle), we consider it wrong and assign a count of zero.
- If InterCE detects more cycles than there should be, we assign a count that is equal to the number of true cycles minus the number of excess cycles.

Figure 4.21 shows the extraction results of InterCE and of the expert systems on two data files, one with a simple motif (top row) and another with a complicated motif involving many opening and closing cycles (bottom row). In Figure 4.21a, the expert system succeeds to extract two cycles correctly, while InterCE fails to detect the first opening cycle. However, when the data file contains a complicated motif (Figure 4.21b), InterCE correctly detects and labels 11 out of 12 expected cycles, while the expert system does not find any. It shows that InterCE is not always correct, but it helps with difficult cases. Meanwhile, if only the expert system is used, there may have been a lot of undetected cycles.

[6]Another way to define convergence is when InterCE does not create more queries. However, because novel data motifs may arrive from the stream (which will prompt InterCE to create new queries) and we do not know in advance whether InterCE has learned all the motifs exhaustively, we cannot define convergence as such.



(A) A simple data file with one opening and one closing cycle (`fp` means door closing in French).



(B) A data file with complicated motif involving many opening-closing cycles. This may happen if an obstacle prevents the door from completely closing. (`flp` means slow closing in French, a special case of closing cycles.)

FIGURE 4.21: Examples of extraction results of InterCE and the expert system.

Following our scoring protocol, InterCE has a score of 1 and the expert system a score of 2 in the example in Figure 4.21a, and 11 and 0 respectively in Figure 4.21b. The accuracy of each method is the total score divided by the total number of correct cycles.

Figure 4.22 shows the accuracy of InterCE versus that of the expert system on the NAT and R2N data sets. We mark the number of correct cycles grouped by extractor chosen by InterCE. In many cases, all extractors are able to find several correct cycles, but ultimately InterCE only picks the extractor with the most number of correct cycles. It should not be mistaken that, for example, in the case of R2N data, the activity-based extractor does not find any cycles. It instead means that in many cases, this extractor does not find the best set of cycles among all the extractors.

Because InterCE includes the expert system in its ensemble, it achieves by definition the accuracy level of the expert system. The added accuracy comes from the other two extractors (activity-based and autoencoder-based), which significantly increases the number of correct cycles. Interestingly, the activity-based extractor and autoencoder-based extractor behave inversely on two use cases. A possible reason is that in NAT data, different cycles are almost always separated by a segment of inactivity that is easy to detect, which is not the case for the R2N data set. Meanwhile, the autoencoder appears to have learned better than on R2N data.

We also plot the number of correct cycles by each extractor, regardless which extractor is picked by InterCE as the best one for each data file. Figure 4.23 shows the number of correct cycles found by each extractor over 1000 data files. The difference in the performance of the activity-based extractor and the autoencoder-based extractor is shown starkly: in a data set, only one of them performs well. These extractors are possibly tailored specifically for one fleet and do not generalize well to the other. This issue can be overcome by adding more extractors to the ensemble of InterCE.

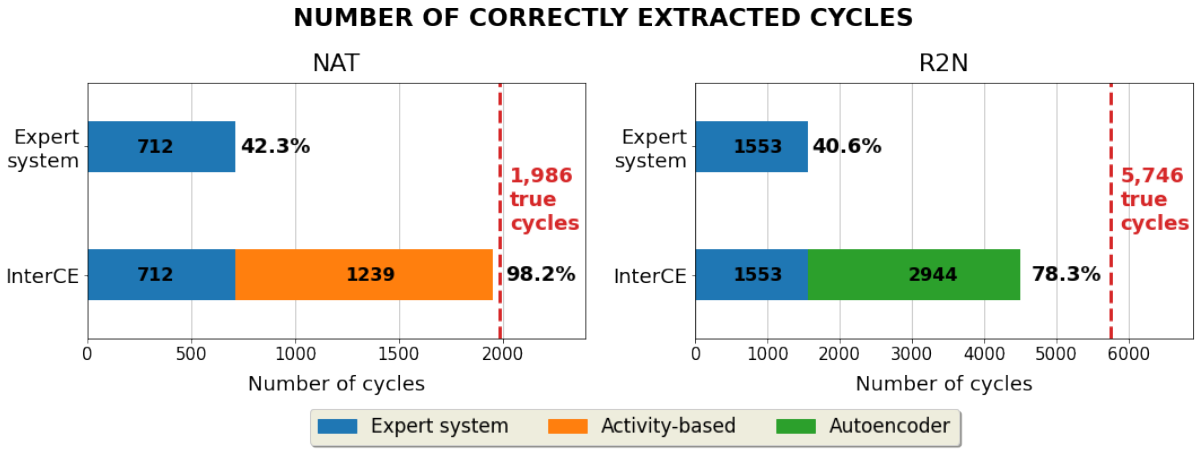


FIGURE 4.22: Number of cycles correctly extracted with only the expert system versus combining all three extractors in InterCE

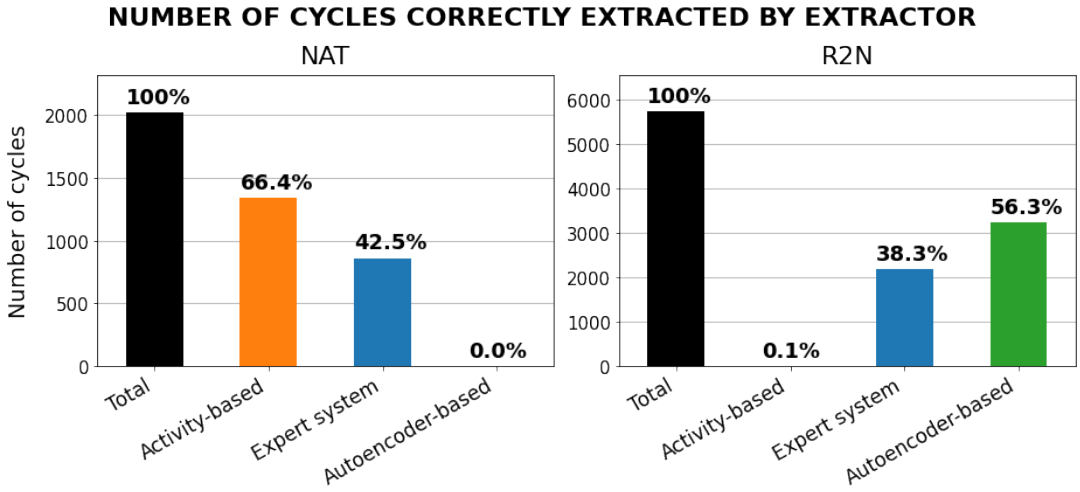


FIGURE 4.23: Number of correct cycles by each extractor in InterCE. The first column in each plot is the number of expected cycles (ground-truth).

Given that InterCE improves the baseline performance of the expert system substantially in both NAT and R2N data sets, the hypothesis H_c^2 is validated.

4.2 Reactivity

InterCE converges when it reaches a stable extraction accuracy. To reveal the convergence time of InterCE, we plot its accuracy over time. The accuracy of cycle extraction at a time T is equal to the number of cycles InterCE has correctly extracted thus far from $t = 1$ until $t = T$ divided by the total number of expected cycles thus far.

Beside the convergence of InterCE, we also evaluate its offline counterpart. The offline version of InterCE has a training and a testing phase. We take the first N input files as training data, and the rest of the $1000-N$ files for testing. During the training phase, InterCE only creates queries. All the queries are shown to the human annotator at the end of the training phase; their feedback is used to update the knowledge of InterCE all at once. Afterwards, InterCE extracts cycles from new inputs from the

stream and no longer processes feedback. As a consequence, InterCE will not create queries for novel motifs and will attempt to extract all the cycles by itself.

The offline version of InterCE is run with various number of training examples: $N = 100$, $N = 200$, and $N = 300$; we denote them `offline-100`, `offline-200`, and `offline-300` respectively. Because offline InterCE does not return any result before it finishes training, the accuracy only shows after the first testing examples. Since InterCE must see at least more files than the number of unique motifs, we select these numbers of training files (100, 200, 300) to ensure InterCE has seen most of the common motifs for a fair comparison. It is difficult to guarantee a definitive number of motifs, as we never know if (and when) novel motifs may eventually arrive from the stream.

The accuracy of InterCE and its offline counterpart are shown in Figure 4.24. The performance of InterCE on the two data sets is notably different. Because NAT data are simpler than R2N data as the former contains less perturbations, the overall accuracy of InterCE on NAT data is higher and more stable than that on R2N data. Generally, in both data sets, the online and offline versions of InterCE struggle at the beginning, but they eventually converge.

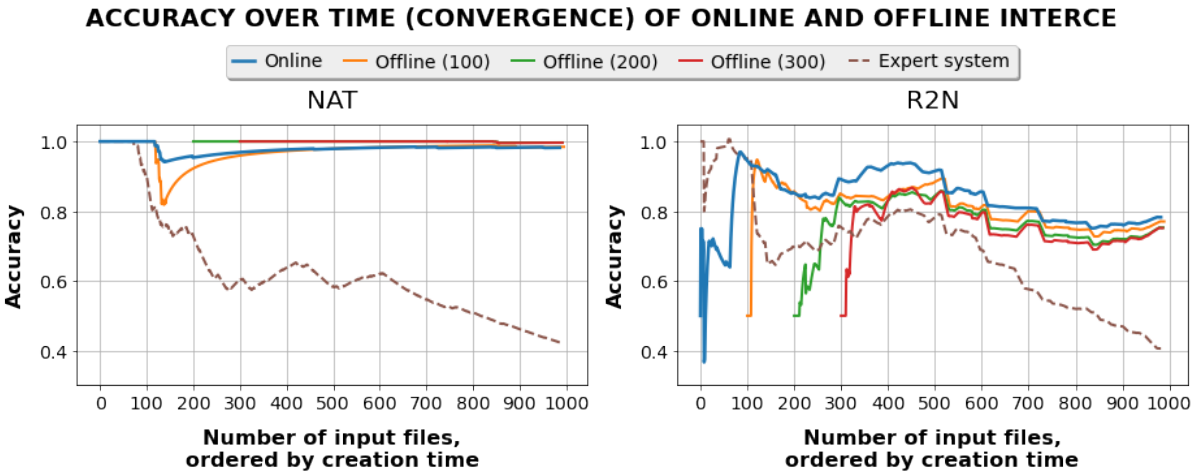


FIGURE 4.24: Accuracy of online InterCE, of offline InterCE, and of the expert system recorded over time

It is not evident to conclude on the superior reactivity of online InterCE. On NAT data, there is a fall in the accuracy of online InterCE shortly after the 100^{th} data example, but its accuracy rises afterwards. The same phenomenon also occurs to `offline-100`. It is possibly caused by some difficult data examples where the data motifs vary largely from the common cases. However, both `offline-200` and `offline-300` are able to reach a very high accuracy just after their training phase. On R2N data, the performance of all learners fluctuate. The overall accuracy of online InterCE is highest among all the learners starting from the 300^{th} input. Nevertheless, they all seem to converge toward an accuracy around 0.8. In both data sets, the accuracy of the expert system decreases over time, indicating that the expert system cannot cope with the variations of the cycles on the stream.

Because the results are not conclusive on the higher reactivity of online InterCE, the hypothesis H_c^1 is not validated. Despite not having a significant superior reactivity, online InterCE remains competitive to its offline counterpart.

4.3 Processing time

To assess the efficiency of InterCE, we run it on a stream of 100000 input files for each data set, without providing feedback. Figure 4.25 shows the total execution time of InterCE on NAT and R2N data sets. We also decompose it to the execution time of two submodules: the time for the memory $\mathcal{M}$ to find similar motifs, and the time for the ensemble $\mathcal{E}$ to extract candidate cycles. We observe that the latter is dominant in both data sets. The execution is much longer for R2N because R2N data contain more motifs than NAT data. We will analyze the motifs shortly.

FIGURE 4.25: Execution time of InterCE on NAT and R2N data sets, from 10^4 to 10^5 files

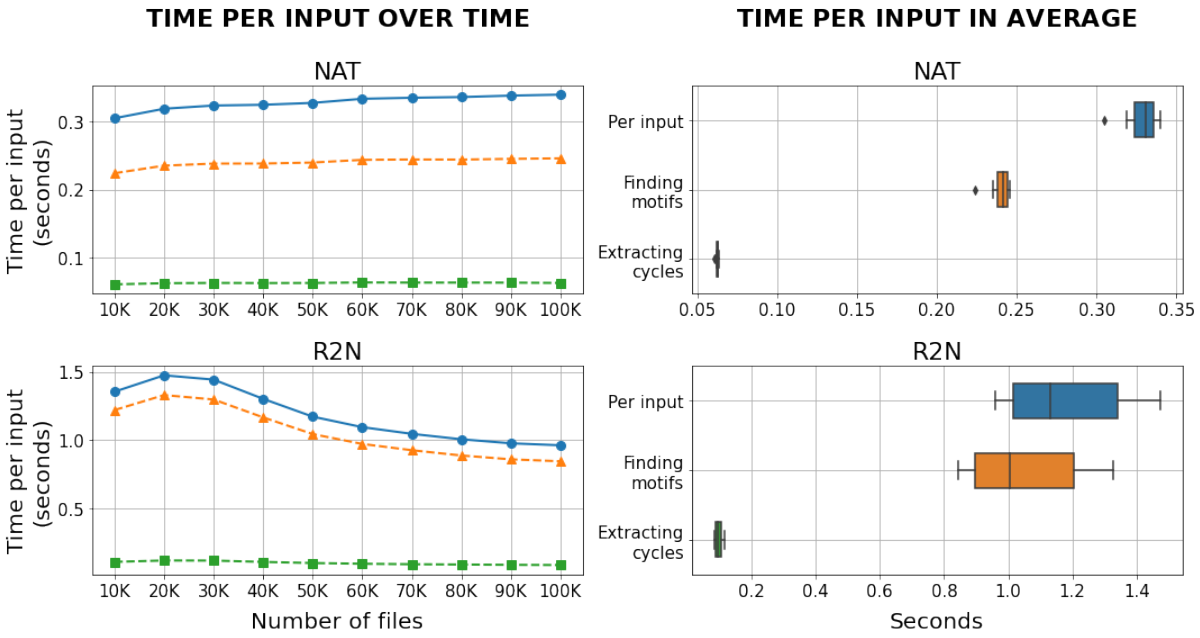


FIGURE 4.26: Execution time per input file

We break down the execution time per input file (Figure 4.26). The time occupied by the ensemble $\mathcal{E}$ is negligible. Meanwhile, finding a similar motif takes around 0.24 second per input file for NAT data and 1.0 second for R2N data (median values). The execution time is stable for NAT whereas it is much more variable for R2N, but the fluctuation remains relatively small. This can be improved with parallelization.

Time complexity To estimate the time complexity of InterCE on a new input X, let us denote N the number of timesteps in X, D the number of variables in X, P the number of motifs stored in $\mathcal{M}$, K the number of cycle clusters in $\mathcal{K}$, W the window size of the autoencoder-based extractor, J the size of $\mathcal{E}$, and N_j the worst-case number of cycles an extractor E_j may produce (we assume the same N_j for all extractors in $\mathcal{E}$). The time complexity of InterCE is the complexity of its primary operations:

- for the ensemble $\mathcal{E}$ to extract candidate cycles: $O(N+\frac{N}{W}+A)$, where A is the inference complexity of the autoencoder[7];
- for the memory $\mathcal{M}$ to search for a similar motif to the input: $O(PDN)$;
- for the knowledge $\mathcal{K}$ to automatically select cycles: $O(JN_jKDN)$.

Therefore, the total time complexity of InterCE is:

$$O(N + \frac{N}{W} + A + PDN + JN_jKDN)$$

where $O(DN)$ is due to the distance computation between two multivariate series (a cycle to a motif or a cycle to a feedback cluster). The most important factor in the complexity term is the number of timesteps N, because N can be up to more than a thousand in some files, albeit unusually. The second-most important factor is the number of motifs P, because on an infinite stream, P may keep growing as novel motifs arrive. An complex autoencoder in $\mathcal{E}$ will also prolong the execution time.

4.4 Querying efficiency

We strive to send as few queries as possible, by issuing only those that come from unique data motifs unseen by the memory $\mathcal{M}$. Figure 4.27 visualizes the number of official queries and the ratio of the number of official queries over the number of input files. For 100000 files of NAT data, 25 queries are issued in total. The number of queries are 40 in the case of R2N data, because there are more distinct motifs in R2N data files. In fact, the number of official queries is guaranteed to be equal to the number of motifs stored in $\mathcal{M}$, as we only create an official query if an input X has never been processed.

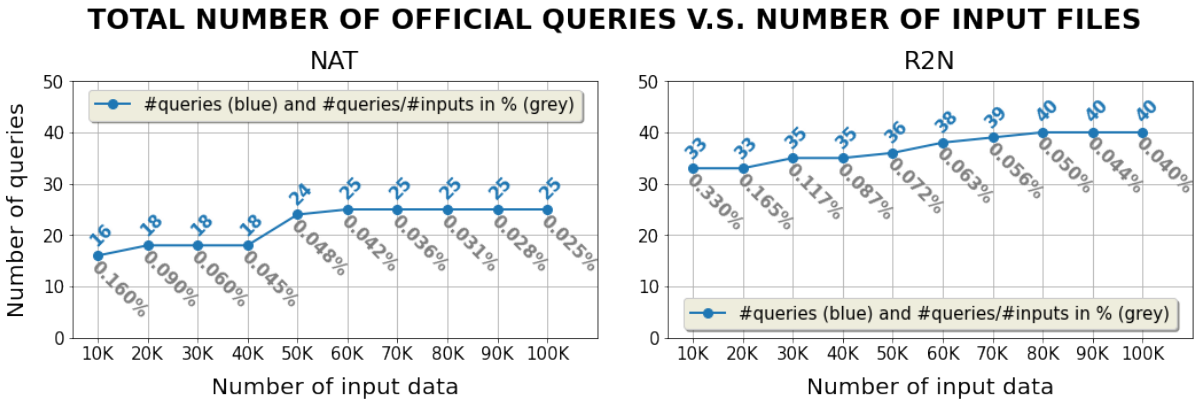


Figure 4.27: Number of official queries and the ratio of the number of such queries over the number of input files

Figure 4.28 shows the number of buffered queries associated to an official query, sorted in the descending order. A buffered query is create if an input X has a motif stored in the memory $\mathcal{M}$ but

[7]Since the inference complexity of a neural network comprises many factors such as the number of layers, number of hidden units in each layer, we do not explicit its full complexity.

the official query created for such motif has not been solved. Given the no-feedback setup, the number of buffered queries is also the number of input files sharing the same motif as the associated official query. We see that very few motifs are frequently seen in the data. For each data set, only one or two motifs are the most common. By buffering queries, we avoid issuing an exploding number of queries.

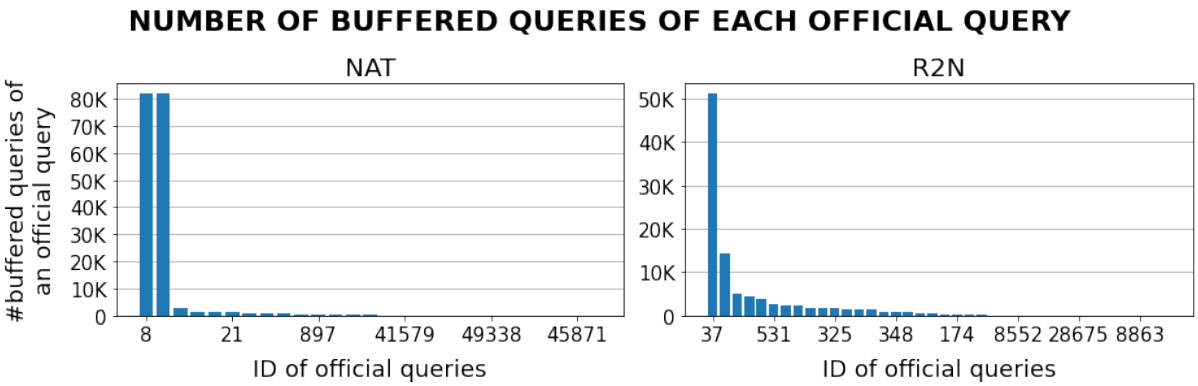


FIGURE 4.28: Number of buffered queries associated to an official query, sorted in the descending order

As for the number of motifs, there are 25 motifs found in 100000 files of NAT data and 40 motifs in 100000 files of R2N data (Figure 4.29 and 4.30). We also note the number of buffered queries associated to each motif. Note that some motifs appear only once as an official query and have no associated buffered query (motif 36 and 37 in R2N data). In both data sets, the most common motifs are ones with an opening cycle followed by a closing cycle. Otherwise, the motifs either include many opening-closing cycles, or are anomalous. Some motifs have only a closing or an opening, possibly due to data acquisition errors. We note a substantial number of redundant motifs in NAT data. This is due to the memory $\mathcal{M}$ failing to recognize known motifs with slight variations. For example, in NAT data, the motif 2 and motif 1 have almost the same shape, but differ in length (six seconds versus eight seconds). It requires that our method to search for similar motifs (Algorithm 4.1) needs improvement.

5 Conclusion

In this chapter, we address the challenge of cyclicity in railway complex systems by proposing InterCE (Interactive Cycle Extraction) as a solution to automatically detect and identify cycles from a stream of raw sensor signals. Using an active learning-based approach, we implement a feedback loop allowing InterCE to incorporate human feedback in its learning process. As a result, InterCE learns the shapes and types of expected cycles on-the-fly without requiring a priori knowledge about the task. We evaluate InterCE on the NAT and R2N data sets to validate the following hypotheses:

- ($\mathbf{H_c^1}$) Using online learning to learn to extract cycles has a higher reactivity than using offline learning.
- ($\mathbf{H_c^2}$) Online learning that enables model update via human feedback performs superior to, or on-par with, a static expert system in accuracy.

The experimental results show that InterCE achieves an accuracy superior to that of a static expert system, tested on 1000 data files: InterCE reaches an accuracy of 98.2% on NAT data set and 78.3% on R2N data set, in comparison to that of the expert system which is 42.3% and 40.6%, respectively. This result validates H_c^2. As for reactivity, InterCE does not decisively outperform its offline counterpart but remains competitive, thus H_c^1 cannot be validated.

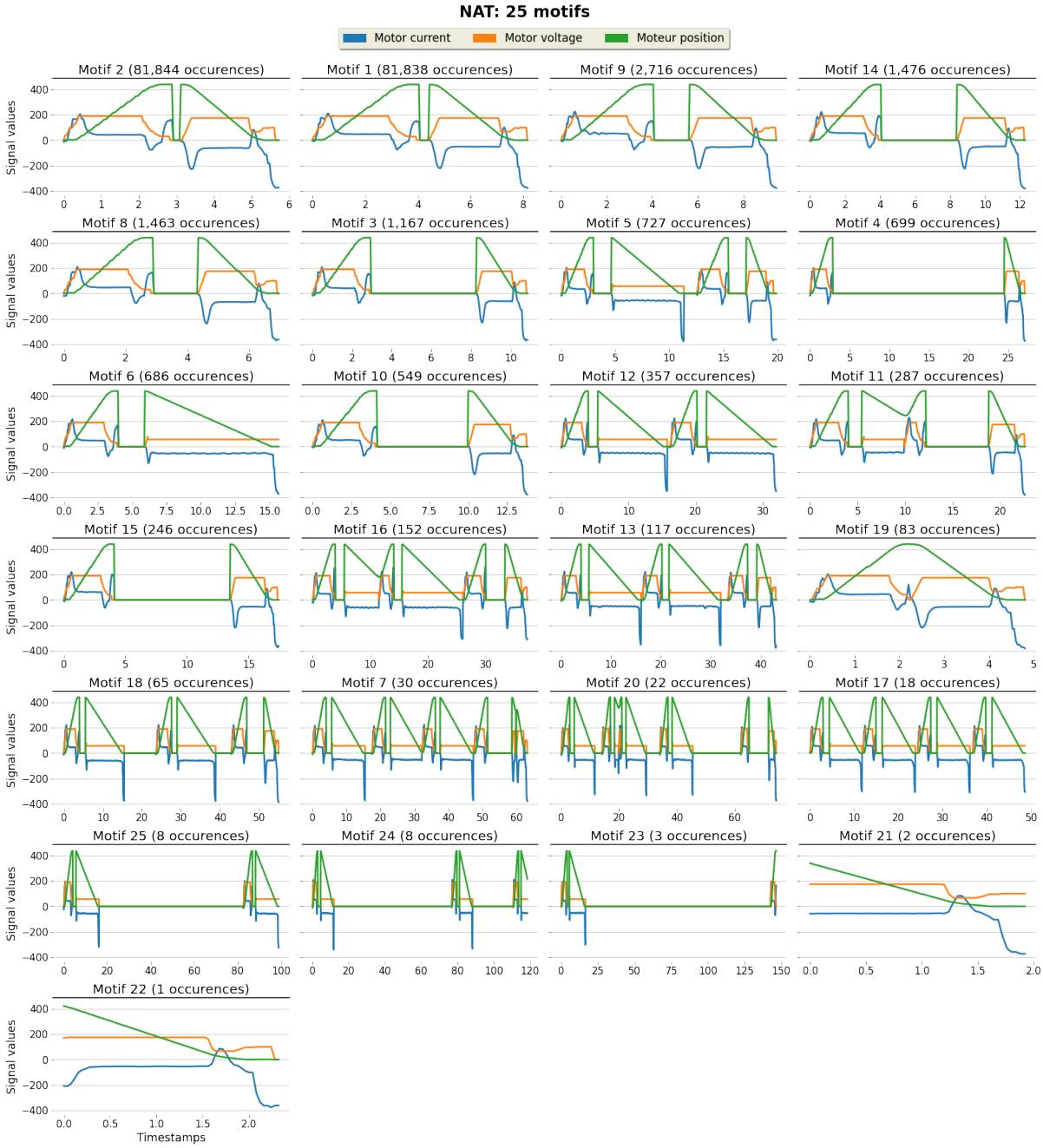


FIGURE 4.29: 25 motifs found in 100000 NAT data files

In addition to the reactivity and accuracy, we also evaluate the efficiency of InterCE via its processing time, tested on 100000 data files per data set. On NAT data, InterCE processes one input file within less than 0.5 seconds, whereas it takes about 1.5 seconds to process one R2N data file. The processing time per input is dominated by the time spent to find a motif most similar to a new input file. This is an important bottleneck that we will amend in future works.

Moreover, we aim for an efficient querying strategy: InterCE should learn the most possible from the fewest queries. The query ratio of InterCE on NAT data and on R2N data is 0.025% and 0.04%, respectively, which are modest with respect to the total number of files (100000 files per data set). It shows that the query strategy succeeds to reduce the number of queries and makes the manual

annotation less tedious for the human experts.

The primary shortcomings of InterCE are the following:

- Although InterCE updates its knowledge on human feedback, the update occurs at the labeling functionality, that is, which cycle shape maps to which function of the systems, but it does not update the learners in the ensemble, which is crucial to ameliorate the detection accuracy.
- Finding similar motifs is an important bottleneck in InterCE. The accuracy of motif matching is also a concern, as we want to avoid sending redundant queries to the human annotators.

Generally, InterCE functions by *memorizing* instead of truly *learning* from the data. It is preferably to have a proper learning principle backing InterCE's logic. For example, instead of memorizing the motifs of the input data, the memory of InterCE could learn the features of these motifs to quickly conclude whether a motif has been seen or not, thus re-framing motif matching as a classification task instead of a searching task. Then, instead of simply grouping cycles sharing the same label, InterCE could use a classifier, or a clustering algorithm, to perform a better mapping. And most notably, the learners in the ensemble should be updatable. This could be done in two ways: we may enable InterCE to update the hyperparameters of the learners on new feedback, or the learners themselves become adaptive to changes on the stream[8].

Finally, the cycles extracted by InterCE will become the input of the next task: feature learning. The aim is to learn relevant features from cycles and to represent a cycle in a form that is more compact while preserving the information of the original cycle.

[8]The two options can be confusing. Let us look at an example of a random forest. When we change the hyperparameters of a random forest, we modify the maximum number of trees, the maximum depth of each tree, et cetera. If the random forest itself is adaptable, it can select which trees to drop, create new trees, modify its own hyperparameters, or assign and re-calibrate the relevance weights of the trees.

FIGURE 4.30: 40 motifs found in 100000 R2N data files

Chapter 5

Feature learning

Contents

Summary

Cycles extracted from the stream are analyzed to unveil the condition of the systems, but using the cycles as time series is not efficient. Element-wise comparison between millions of cycles is time- and memory-consuming. Meanwhile, a learning algorithm can detect unique statistical features from the cycles to summarize them compactly. To this end, we implement a long short-term memory autoencoder (LSTM-AE) that learns features by reconstructing its own inputs in an unsupervised fashion, and is jointly trained with a classifier mapping the cycle to its own contextual information, making the LSTM-AE robust against noises. This chapter addresses the hypotheses H_f^1, H_f^2, and H_f^3.

The experimental results shows that the LSTM-AE features are better at preserving the cycle information than manually engineered features. After training the LSTM-AE in three settings - offline, online, and online incremental, we find out that the offline version of the LSTM-AE outperforms its online counterparts, both in terms of reconstruction capacity and convergence time, possibly because the training configuration unfairly favors the offline paradigm. We will address this in the future works.

1 Introduction

A cycle per se is a multivariate time series: it contains multiple measurements, each recorded over multiple timesteps. But an element-wise comparison between millions of time series is computationally costly. Meanwhile, a comparison on their *features*, such as the mean, the skewness, or the length, suffices to deduce the normality of a cycle. It also reduces the computational costs for time series mining. Therefore, we transform a cycle to a *feature vector* for the ensuing analysis. There are multiple approaches to extract features from a time series.

A straightforward solution is to let a domain expert define the relevant statistics using a set of fixed rules. Nonetheless, the domain experts may miss useful features of the cycles that can unveil the condition of the systems.

Another solution is time series decomposition. It decomposes a series into several individual components, such as the trend and seasonality. For instance, the Fourier transform decomposes a complex time series into different components in the time and frequency domains, which are recorded as features. However, this also requires domain expertise to identify which components are relevant among those decomposed.

Alternatively, the features can be *learned* from the data without extensive domain expertise using representation learning. Representation learning is a task that produces representations of the inputs that are useful for other downstream tasks [26]. For representation learning, autoencoders have recently become popular. An autoencoder is a neural network that learns to reconstruct its own inputs in an unsupervised manner. As our goal is to learn a compact yet representative form of the input cycles without labels, autoencoders are a promising method.

Furthermore, since cycles are time series which are sequential data, regular densely-connected layers may not fully capture the temporal information from the cycles. We use a recurrent architecture to emphasize their temporality. Among the existing recurrent architectures, we choose the long short-term memory architecture [96], because it overcomes the exploding gradient phenomenon often encountered in a regular recurrent network.

Therefore, we propose a *Long short-term memory autoencoder* (LSTM-AE) to extract feature vectors from cycles such that the resulting vectors are more compact than the original cycles but are equally representative. Precisely, we expect that the embedding feature space is coherent with the cycle space, such that an anomalous cycle produces an anomalous feature vector.

The LSTM-AE is expected to validate these hypotheses:

- ($\mathbf{H}_\mathbf{f}^1$) Online feature learning performs superior to, or at least on part with, offline feature learning in terms of accuracy.
- ($\mathbf{H}_\mathbf{f}^2$) Online feature learning is more reactive than offline feature learning.
- ($\mathbf{H}_\mathbf{f}^3$) Feature learning results in better information preservation than feature engineering.

We evaluate the LSTM-AE on the cycles extracted from the NAT and R2N data sets. We compare three versions of the LSTM-AE - offline, online, and online incremental, on their learning performance and reactivity. Then, we take the best model among those three and compare the features it produces to the indicators identified by an expert system to see which approach produces features that are more truthful to the original cycles. The results show that the offline version of the LSTM-AE achieves the best performance, both in terms of reconstruction capacity and convergence time. Also, the features

extracted by the LSTM-AE outperform the expert indicators in almost all cases, except on the opening cycles of the NAT data set, but with only a small gap in performance.

This chapter is organized as follows. Section 2 reviews the existing methods of representation learning, focusing on autoencoders. Section 3 describes the LSTM-AE in detail. Section 4 shows the experimental results. Finally, Section 5 concludes the works in this chapter.

2 State-of-the-art: Representation learning

In this section, we cover the state-of-the-art of representation learning. We focus on the unsupervised paradigm because our aim is to learn representations from cycles without guidance from domain expertise.

Representation learning, also known as *feature learning*, is the task of extracting discriminative features from a data set as an intermediary step for a downstream task, for instance, for classification or regression [26, 173]. Originally, representation learning was implicitly included in neural network training. During training, a neural network forwards the input data through multiple hidden layers and in the process transforms the inputs into complex features that improve the predictions. Recently, representation learning has become a relevant field on its own following the development of deep learning.

Representation learning is useful for dimensionality reduction, because it embeds high dimensional inputs to a lower dimensional space by retaining only the most relevant features. Principal Component Analysis is the most well-known technique for dimensionality reduction; it is also a method of representation learning, although it can only produce a linear mapping of the features. Deep learning, which consists of training a neural network with multiple layers, applies multiple non-linear transformations to yield more abstract representations. In fact, neural networks learn a *hierarchy* of abstract features, such that the features at one layer are combined from those learned at the previous layer.

For one input example (for instance, a time series, an image, an audio), a representation learning model returns one *feature vector* as a representation of this example. There are three paradigms to tackle representation learning: probabilistic modeling, neural networks via autoencoders, and manifold learning [26].

Probabilistic modeling terms learning representation as finding a set of latent variables h that best describes a distribution over the training data x, that is, to model $p(h, x)$. Learning features is to find the model parameters that maximize the likelihood of the training data. However, the feature vectors are not readily usable at the end of the learning process but must be derived from the estimated posterior distribution $p(h \mid x)$.

An autoencoder is a neural network that learns to reconstruct its own inputs. An autoencoder optimizes its parameters (network weights) by minimizing the difference between the inputs and their reconstruction. Different to probabilistic modeling, autoencoders learn a direct mapping from the inputs to their representation and return a feature vector for each input example.

Manifold learning is built on the manifold hypothesis, stating that real-world data in high-dimensional space are expected to concentrate in the vicinity of a manifold of much lower dimensionality [67]. The goal of manifold learning is to model the low-dimensional structure on which lie the data. A few examples of manifold learning methods include: multidimensional scaling (projecting samples on a low-dimensional space while translating as much information of pairwise distances as possible) ISOMAP

(extending multidimensional scaling by factoring in the distances among neighbors and geodesic distances) [219], local linear embedding (linearly constructing each point from its neighbors to maintain local structure) [195].

Given the complexity to make probabilistic modeling and manifold learning online-compatible and online-efficient, we turn our attention to autoencoders. The following subsections provide more details about autoencoders and some popular variants.

2.1 Autoencoders

An autoencoder is a neural network that reconstructs its own input. It is composed of an encoder f and a decoder g (Figure 5.1). We denote $\mathbf{W}_f$ and $\mathbf{W}_g$ the weights of the encoder and of the decoder, respectively. The encoder receives an input x and forwards it through the hidden layers, compressing x in the process via non-linear transformation. The last layer of the encoder produces the final features $h = f(x)$, also called the *representation*. The decoder passes the features h through its own hidden layers to reconstruct the original input at the final layer, i.e., $\hat{x} = g(h) = g(f(x))$.

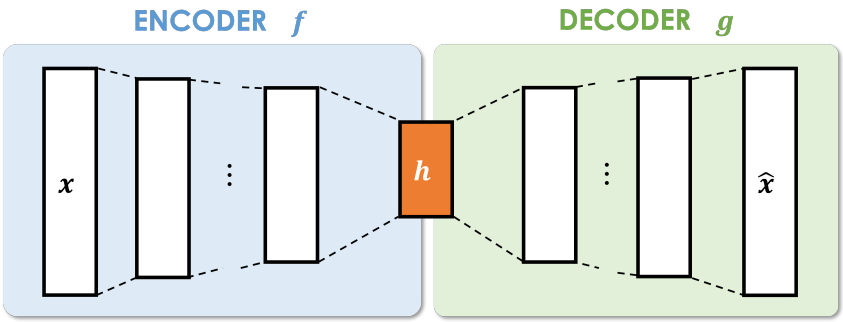


FIGURE 5.1: In an autoencoder, the encoder encodes an input x to a representation h, and the decoder reconstructs $\hat{x}$ from h

A reconstruction error is calculated on the difference between x and $\hat{x}$ and is backpropagated in the network to update the weights between the layers, such that the new weights reduce the reconstruction error in the next iteration. Training an autoencoder consists of adjusting $\mathbf{W}_f$ and $\mathbf{W}_g$ iteratively until the reconstruction errors are minimized over all the training examples. Once the autoencoder has found the optimal weight values and the reconstruction error cannot be reduced further, the autoencoder has converged and knows how to extract a set of features that allow it to reconstruct the inputs accurately. After convergence, we only use the encoder to extract features.

The reconstruction error is calculated by a loss function. Let $\mathcal{L}$ be the loss function that computes the difference between x and $\hat{x}$, the goal is to find the parameters $\hat{\theta} = [\mathbf{W}_f, \mathbf{W}_g]$ such that the reconstruction error averaged over all N training examples is minimized (5.1).

$$\hat{\theta} = \arg\min_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(x^{(i)}, g(f(x^{(i)}))\right) \tag{5.1}$$

(5.1) is sometimes written to (5.2), where $J(\theta; X, y)$ indicates the objective function with respect to the parameters θ and the data X. The objective function is a more generalized form of the loss function. For autoencoders, we only have the inputs X without accompanying labels y.

$$\hat{\theta} = \arg\min_{\theta \in \Theta} J(\theta; X) \tag{5.2}$$

The choice of the loss function depends on the types of the input data. For instance, mean squared error is commonly used for continuous inputs, while negative log-likelihood is often used for binary inputs and cross-entropy loss for discrete inputs.

Usually, an autoencoder has a bottleneck architecture, also called an undercomplete architecture, where the representation size is smaller than the input size. It forces the autoencoder to compress the original data to a more compact form while preserving the representation capacity. In addition, an autoencoder can be regularized to make it insensitive to small changes in the inputs, and also to prevent it from learning the identity function. This leads to several variants of regularized autoencoders.

2.1.1 Variants of regularized autoencoders

Several forms of regularized autoencoders exist, among which the three most popular variants are sparse autoencoders, denoising autoencoders, and variational autoencoders.

Sparse autoencoders Sparse autoencoders are autoencoders with sparsity constraint, in which most entries in the feature vectors are zero or close to zero. Such sparse representation contains few active units, but those units are learned from unique statistical features from the input data. Sparse autoencoders are obtained by adding a penalty on the representation h [26]:

$$\tilde{J}(\theta; X) = J(\theta; X) + \alpha\Omega(h) \tag{5.3}$$

where $\alpha > 0$ dictates the influence of the penalty. A large α means stronger penalization and adds more sparsity to the resulting representations.

Denoising autoencoders Denoising autoencoders attempt to learn the input distribution while undoing the effect of injected noises. The inputs are corrupted with noise and the autoencoder must try to reconstruct the original, non-corrupted inputs. Given a corruption process $q(\tilde{x} \mid x)$ that produces a noisy data sample $\tilde{x}$ from an original sample x, a denoising autoencoder is trained on pairs of $(x, \tilde{x})$ to optimize the objective function (5.4):

$$J(\theta; X) = \sum_{i=1}^{N} \mathbb{E}_{q(\tilde{x}^{(i)}|x^{(i)})}[\mathcal{L}(x^{(i)}, g(f(\tilde{x}^{(i)}))] \tag{5.4}$$

such that the expected loss on the errors between the inputs $x^{(i)}$ and the reconstruction from the corrupted version $\tilde{x}^{(i)}$ is minimized.

Variational autoencoders Variational autoencoders penalize the representations by using the Kullback-Leibler divergence to impose a distribution on the feature space. Different from sparse and denoising autoencoders, variational autoencoders are generative: instead of learning a direct parametric mapping from the input space to the encoding space, it learns the parameters of the distribution modeling the input data. In other words, a variational autoencoder learns a latent space model of the input space.

The encoder approximates the posterior distribution $p(z \mid x)$, where z is a data example sampled from the latent distribution modeling x. Given a data example x, the encoder outputs the parameters that define $p(z \mid x)$. For example, if we constraint the latent space to be Gaussian, the encoder will output the mean $\bar{z}$ and the variance σ^2, from which a sample z can be randomly drawn from $p(z \mid x)$. Then, the decoder maps the sample points z back to the original input space by approximating the conditional distribution $p(x \mid z)$.

$$D_{KL}(P \parallel Q) = \mathbb{E}_{x \sim P}\left[log\frac{P(x)}{Q(x)}\right] \tag{5.5}$$

A variational autoencoder is optimize on two loss functions: one is the reconstruction loss (e.g., mean squared error, cross-entropy), and the other is the Kullback-Leibler divergence between the latent space and the input space. The Kullback-Leibler divergence measures the distance between two distributions P and Q (5.5), which in this case is $p(z \mid x)$ and $p(x)$. The Kullback-Leibler divergence acts as a regularization term that encourages z to be sampled from many values close to x, instead of being collapsed to one single point that most likely generates x, in order to construct a well-formed latent space [82].

2.1.2 Evaluation of learned representations

A good representation is one that disentangles the underlying factors of variation in the input data [26]. However, because autoencoders work in unsupervised scenarios and are not interpretable (the resulting features are numerical values with no explicit meaning), it is difficult to conclude on the quality of the learned representations using common metrics such as accuracy or precision. Still, it is possible to indirectly assess the quality of the learned features. The reconstruction errors are the foremost metrics that reflect the goodness of the representations, but an autoencoder with a large capacity[1] can produce a small error on the test set even if it does not generalize well.

Another evaluation approach is to test the learned representations on a downstream supervised task, such as classification or regression. If the downstream model returns better results using the learned representations than using the original inputs, it means the autoencoder generalizes well and succeeds to learns useful latent features.

One can also visualize the learned features to spot the structure of the embedding and to verify whether it is consistent with the structure of the original inputs. However, because low-dimensional visualization often rely on dimensionality reduction techniques which require their own hyperparameters, selecting inappropriate parameters will impact the results.

There exist several setups for evaluating a representation learning model but there is no standard way to evaluate the representations learned by autoencoders [173]. In many cases, the evaluation is application-dependent.

3 Long short-term memory autoencoder

We now detail the implementation of the long short-term memory autoencoder (LSTM-AE). First, we define a feature vector as follows.

[1]The capacity of a model is its ability to approximate multiple functions mapping an input space to an output space. For neural networks, their capacity is defined by the number of layers and nodes in their architecture.

Definition 3.1 (Feature vector). *A* feature vector $X^T_{S_m}$ *obtained from a cycle* $C^T_{S_m}$ *by a system* $S_m \in \mathcal{S}$ *at the time* T *is a vector of* P *real-valued numbers, i.e.,* $X^T_{S_m} = [x^{(1)}, ..., x^{(P)}] \in \mathbb{R}^P$*, and is associated to one cycle type* $y^T_{S_m}$ *and one set of context variables* $ctxt^T_{S_m}$.

The architecture that combines the long short-term memory cells in each layer of the autoencoder is described in Section 3.1. Then, we extend it by including the contextual information to train the LSTM-AE to reconstruct its inputs and to classify the context simultaneously, making it aware of context-induced perturbations (Section 3.2).

3.1 A simple encoder-decoder architecture

In principle, a layer in an autoencoder can have any neural architecture. It can be a densely-connected layer, a convolutional layer, a recurrent layer, et cetera. As we work with cycles, we chose a recurrent architecture to handle sequential data adequately. Multiple variants of recurrent architectures exist, for instance, simple recurrent neural networks, long short-term memory (LSTM), and gated recurrent units. Through experimentation, we pick the LSTM architecture because it overcomes the exploding gradients phenomenon and produces the most satisfactory reconstruction. An LSTM architecture contains recurrent cells, connected one to another in chain (Figure 5.2). Each cell has four gates to drive the information flow in the LSTM network.

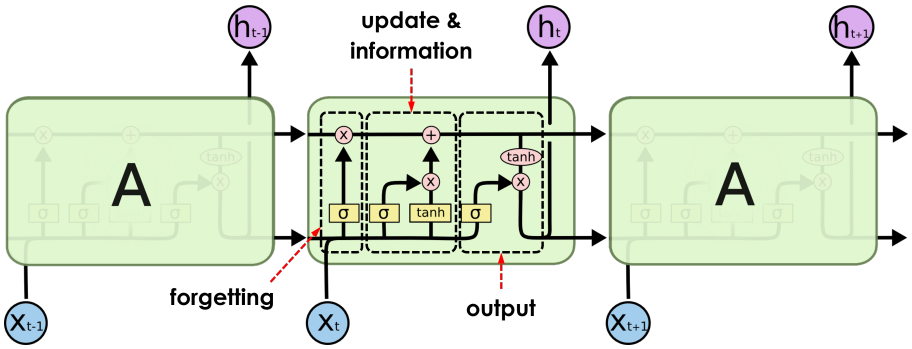


FIGURE 5.2: The inner structure of an LSTM cell [176]

- The forgetting gate regulates the amount of information allowed to enter this cell from the previous one via a sigmoid function (0 means that no information is allowed and is all forgotten, while 1 means that all the information is kept).
- The update gate and the information gate, respectively, decide which position in the input data is to be updated (via the sigmoid function) and by how much (via the tangent function).
- The output gate combines the information from the forgetting gate and the aggregation of the update and information gates to compute the output of this cell which is then fed to the next cell.

Each hidden layer in the LSTM-AE adopts the LSTM architecture, such that each layer is one block of a LSTM unit[2] (Figure 5.2) and the dimension of the output h_t can be customized. The output dimension of consecutive layers is decreasing to form a bottleneck architecture.

The full architecture of the LSTM-AE is shown in Figure 5.3. We build a network of seven layers for the encoder and seven layers for the decoder. The dimension of the output decreases from 160 to 40, with 40 being the size of the final feature vectors. We select this architecture empirically: we test

[2] According to the implementation of the LSTM layer on `keras` (link to source code)

different architectures with varying number of layers and dimensions and compare the reconstructed cycles of each candidate. Finally, we choose the architecture shown in Figure 5.3 because it is the one that retains most information from the input cycles.

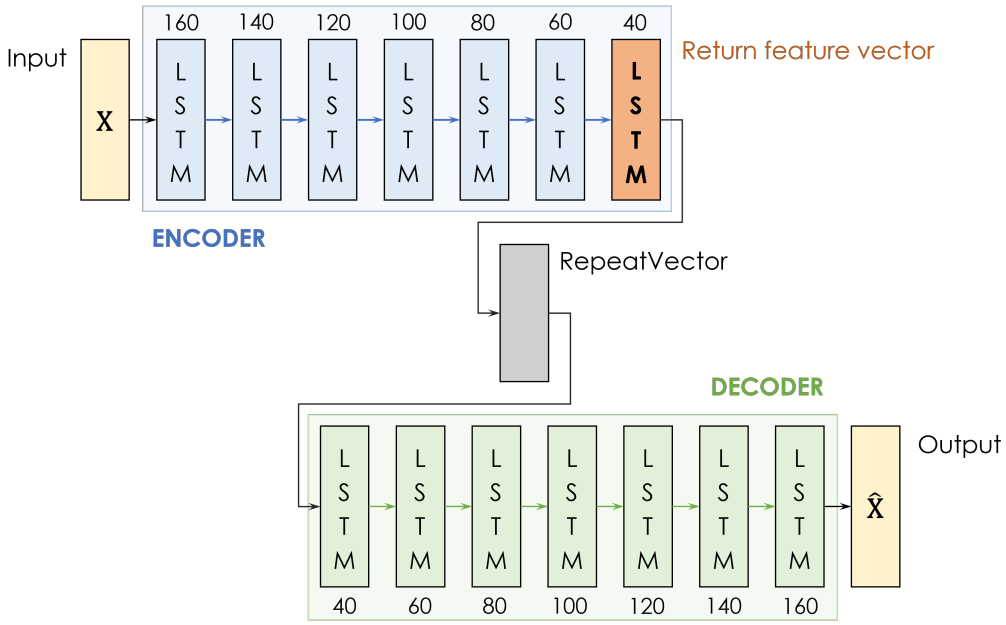


FIGURE 5.3: The LSTM-AE has an encoder (top row, blue) and a decoder (bottom row, green). The feature vector is obtained in the last layer of the encoder (orange). The input layer (yellow) receives a cycle X and the output layer (yellow) returns the reconstruction $\hat{X}$ of X. The numbers denote the dimension of each cell.

However, this architecture ignores contextual information of the cycles. Yet, the context plays an important role in distinguishing a false anomaly from a real anomaly. For example, the data generated by a PAS in two train stations may differ slightly due to the different configurations in these stations, but in both cases the PAS is in a normal condition. Excluding the context from the training of the LSTM-AE may incorrectly produce anomalous features for normal cycles. Therefore, we extend this architecture by incorporating the contextual information in a joint classifier.

3.2 LSTM-AE with self-supplied labels

To make the LSTM-AE aware of the contextual information, we connect to the encoder a classifier mapping a cycle to its own context. By doing so, the LSTM-AE simultaneously learns to represent a cycle and to classify the cycle's context. As a result, the weights of the encoder are adjusted by both the reconstruction accuracy and the context classification accuracy. The intuition is that, from a cycle, the autoencoder learns to extract features that are representative (good reconstruction) and map well to the context in which this cycle is generated (good context recognition).

There are several context variables associated to a cycle. We choose the train stations as the target label for classification, because the stations are an important factor that cause noises in the cycles, according to a railway expert. Because the station information is always available for each cycle, we always have the labels for the classifier of the LSTM-AE. Thus, no annotation effort is needed.

The joint architecture is shown in Figure 5.4. Beside the decoder, the LSTM-AE has an additional branch connected to the last layer of the encoder (pink). This branch has two layers: one dense layer

of 40 units, followed by another dense layer that returns the classification labels of the context. We denote g_c the function of the classifier, and $\mathbf{W}_c$ its weight matrix.

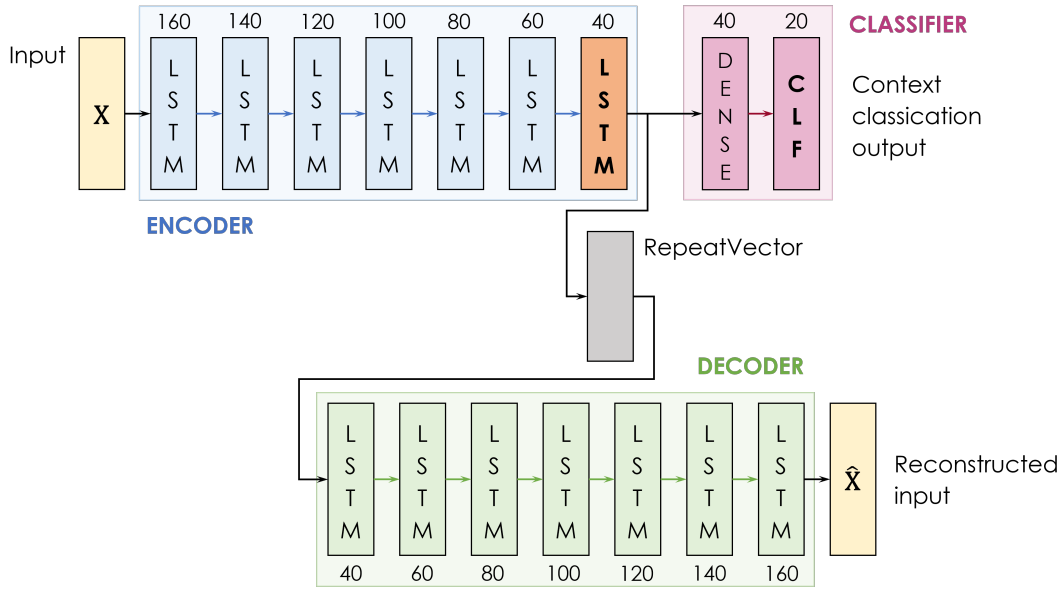


FIGURE 5.4: The joint architecture of the LSTM-AE that connects both the decoder and the context classifier to the last layer of the encoder

Let $\mathcal{L}_g$ be the loss function of the decoder, $\mathcal{L}_c$ the loss function of the classifier, N the number of training examples, P the size of the feature vector, and C the number of class labels (the number of train stations), we choose the mean squared error (MSE) for $\mathcal{L}_g$ (5.6) because the decoder outputs real-valued vectors, and categorical cross-entropy (CCE) for $\mathcal{L}_c$ (5.7) because the classifier deals with multiclass classification. We denote $x^{(i)}$ the i^{th} training cycle and $\hat{x}^{(i)}$ its reconstruction by the decoder, $y^{(i)}$ the true station of the i^{th} training cycle and $\hat{y}^{(i)}$ the prediction issued by the classifier.

The original formula of the MSE is:

$$MSE\left(x^{(i)}, \hat{x}^{(i)}\right) = \frac{1}{N}\sum_{i=1}^{N}\left(x^{(i)} - \hat{x}^{(i)}\right)^2$$

but because a cycle is a multivariate time series of D measurements, the MSE of a cycle and its reconstruction is the average of the MSE of each univariate series in the cycle, and the MSE of each univariate series is the averaged MSE over the K timesteps[3] in the cycle (5.6).

$$\mathcal{L}_g\left(x^{(i)}, \hat{x}^{(i)}\right) = \frac{1}{D}\sum_{j=1}^{D} MSE_j^{(i)} = \frac{1}{D}\sum_{j=1}^{D}\left(\frac{1}{K}\sum_{k=1}^{K}\left(x_j^{(i)}[k] - \hat{x}_j^{(i)}[k]\right)^2\right) \tag{5.6}$$

For the CCE loss (5.7), the metrics is not computed directly on the value of $y^{(i)}$ and $\hat{y}^{(i)}$, but on the probability $p_k^{(i)}$ associated to each k^{th} train station of the i^{th} example ($1 \leqslant k \leqslant C$).

[3] As cycles may differ slightly in length, we pad them with 0.0 to have equal-length cycles.

$$\mathcal{L}_c\left(y^{(i)}, \hat{y}^{(i)}\right) = -\sum_{k=1}^{C} p_k^{(i)} \log \hat{p}_k^{(i)} \tag{5.7}$$

The objective function to find the optimal parameters $\hat{\theta} = [\mathbf{W}_f, \mathbf{W}_g, \mathbf{W}_c]$ of the joint LSTM-AE is thus the sum of the MSE and CCE losses over all the training examples (5.8). To update the weights θ at each iteration, the gradient used for backpropagation is the gradient of the sum of the two losses, and by linearity, is also the sum of the gradient of each loss.

$$J(\theta; X, y) = \frac{1}{N}\sum_{i=1}^{N}\left[\mathcal{L}_g\left(x^{(i)}, \hat{x}^{(i)}\right) + \mathcal{L}_c\left(y^{(i)}, \hat{y}^{(i)}\right)\right] \tag{5.8}$$

Once the model converges, we only use the encoder to extract features.

4 Experimental results

To validate the hypotheses of the *features* block, the LSTM-AE is expected to yield similar reconstructions of cycles in both offline and online settings (H_f^1), be ready to issue usable features earlier than an LSTM-AE operating in offline mode (H_f^2), and produce features that maintain their characteristics with respect to the original cycles (H_f^3).

We test the LSTM-AE, with and without a classifier, on the NAT and R2N data sets. We refer to the version without a classifier as a plain model, and the other the joint model. We run the experiments on a Windows 10 machine with an Inter(R) Core(TM) i7-8850H CPU @ 2,60GHz 2,59GHz, 16GB RAM, and an NVIDIA GPU Quadro P3200. The models are implemented with `tensorflow` [154].

We train one LSTM-AE for each cycle type of each data set. We form the training set by retaining the least noisy cycles from a random sample of tens of thousands of cycles:

1. We filter out cycles whose length lie outside of $[\, Q_1 - 1.5\times IQR, Q_3 + 1.5\times IQR \,]$ of the sample's cycle lengths, where Q_1 and Q_3 denote the first and third quartile, $IQR = Q_3 - Q_1$ is the inter-quartile.
2. We compute the area under the curve[4] of the remaining cycles and remove the cycles whose area lie outside of the interval $[\, Q_1 - 1.5 \times IQR, Q_3 + 1.5 \times IQR \,]$ of the sample's areas.

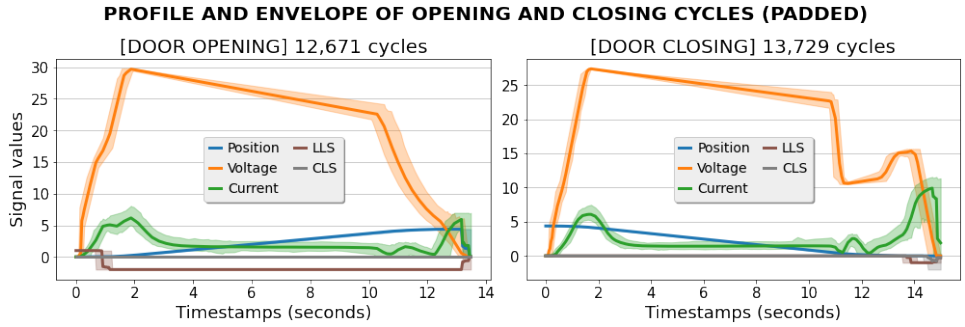
The cycles remaining after these preprocessing steps form the training set. The testing set does not include any of the training cycles. Table 5.1 shows the number of cycles of the training and testing sets of each type of cycles on each data set.

Table 5.1: Number of training and testing cycles of each data set

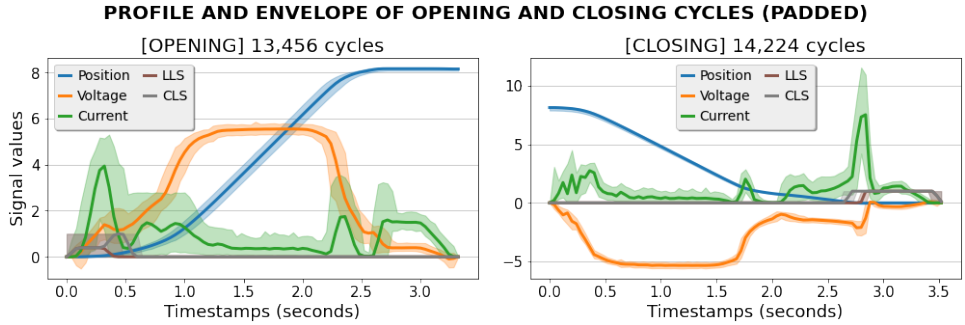
		NAT	**R2N**
Door opening	Training set	12,671	13,456
	Testing set	23,316	21,181
Door closing	Training set	13,729	14,224
	Testing set	23,123	11,851

[4] A curve in this context means a univariate time series in a cycle. A cycle being a multivariate time series, the area under the curve of one cycle is the average of the areas of the univariate series.

Because the cycles may slightly differ in duration, we pad them such that all cycles of the same type have the same number of timesteps[5]. We also scale the data to make the signals from all the variables close in value range. The final profiles and envelopes of each variable of each cycle type are shown in Figure 5.5a and 5.5b, for NAT and R2N data respectively. The profile of a cycle type is the average at each timestep for each variable of all the cycles of this type. The envelope of a cycle type shows the maximum and minimum value a cycle may have at each timestep for each variable. The profile and the envelope are computed independently for each variable. In the cycles, we keep only analog variables from which features are learned: the position of the electric motor, the voltage of the electric motor, the current of the electric motor, locked limit switch (LLS), and closed limit switch (CLS).



(A) NAT training cycles



(B) R2N training cycles

FIGURE 5.5: Profile and envelope of the opening and closing cycles (padded) from NAT and R2N data sets.

In the following sections, we describe the experiments carried to validate the hypotheses H_f^1, H_f^2, and H_f^3.

4.1 Reconstruction capacity

In this experiment, we verify whether an LSTM-AE trained online converges to the same performance as that of an LSTM-AE trained offline in batch mode. We show the results on the joint model only, because the observations obtained with the joint model also apply to the plain model. We train the online and offline versions of the LSTM-AE in three ways, respectively noted OFF, ONL, and OLI.

The *offline* version (OFF) is trained in a traditional batch setting: all the training cycles are fed to the LSTM-AE at once and the network is trained on 300 epochs. A training cycle is processed 300 times and the weights θ are adjusted 300 times on the gradients of the losses from tens of thousands cycles. During training, the LSTM-AE does not return any features.

[5]We pad the opening and closing cycles separately.

The *online* version (ONL) is trained sequentially on each incoming cycle. The LSTM-AE is updated once on each new cycle. There is no training phase for the online version and the LSTM-AE returns the feature vector for each incoming cycle. The ONL model is updated much more frequently than the OFF model, but each update of the former includes far fewer training examples.

We train another model in an *online incremental* fashion (OLI) so that for each incoming cycle, the network is updated on the new cycle and 299 previous cycles, i.e., the network is updated on a mini-batch of 300 cycles at a time. Therefore, each training cycle is processed 300 times as well. Our intention is to align as much as possible the online version to the offline version for fairness.

Figure 5.6 and 5.7 show the training losses of these three versions on the opening and closing cycles of NAT and R2N data, as well as the reconstruction capacity of each version on an example cycle. Note the difference in the number of iterations between OFF versus ONL and OLI: the number of iterations of OFF is the number of training epochs, while ONL and OLI are trained on-the-fly on incoming instances, thus the number of iterations of these two models is the number of training cycles.

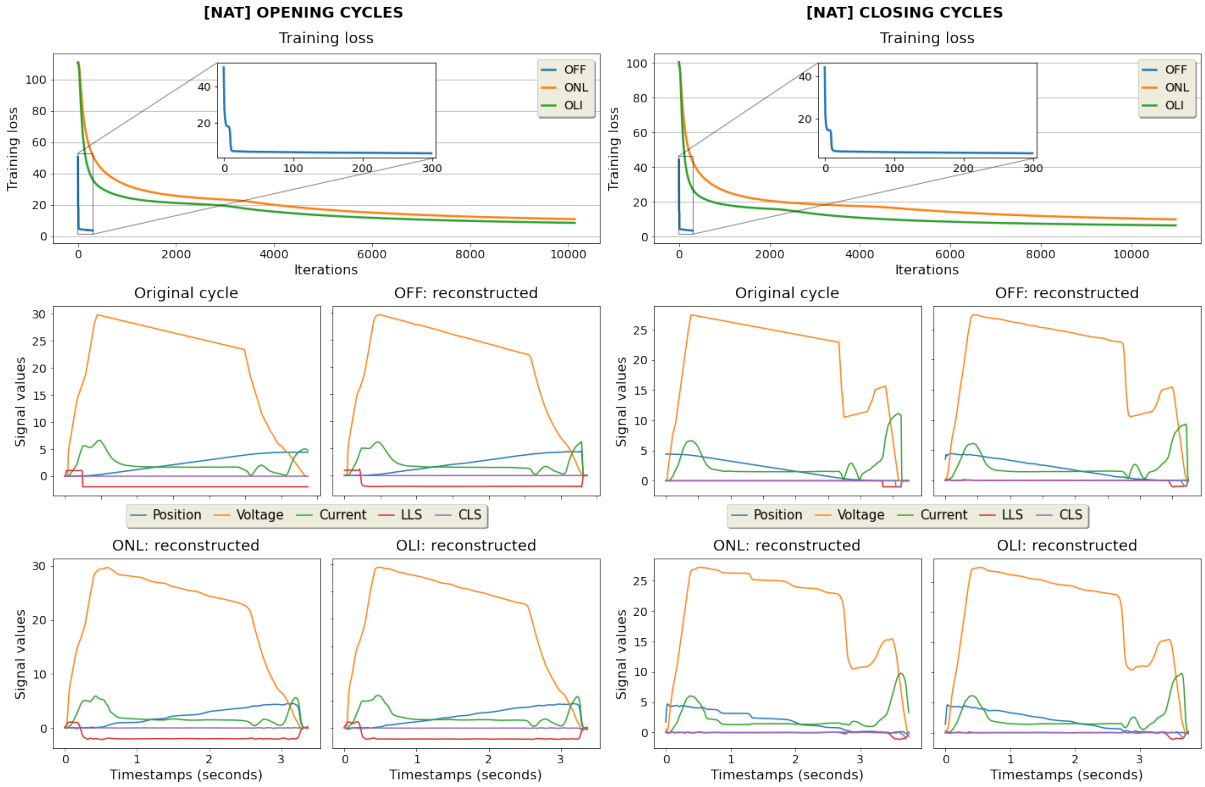


FIGURE 5.6: Training losses and reconstruction of one example cycle of OFF, ONL, and OLI models on NAT cycles

In all cases, OFF returns the closest reconstruction to the original cycle and achieves the lowest losses among the three models. It implies that it is preferable to update the weight θ on the entire training data in each iteration than to update it partially. On the side of online models, OLI is slightly better than ONL, but its superiority is not significant.

In this setup, the hypothesis H_f^1 is not validated, as the training configuration seems to favor the offline model (OFF) over the online ones (ONL, OLI).

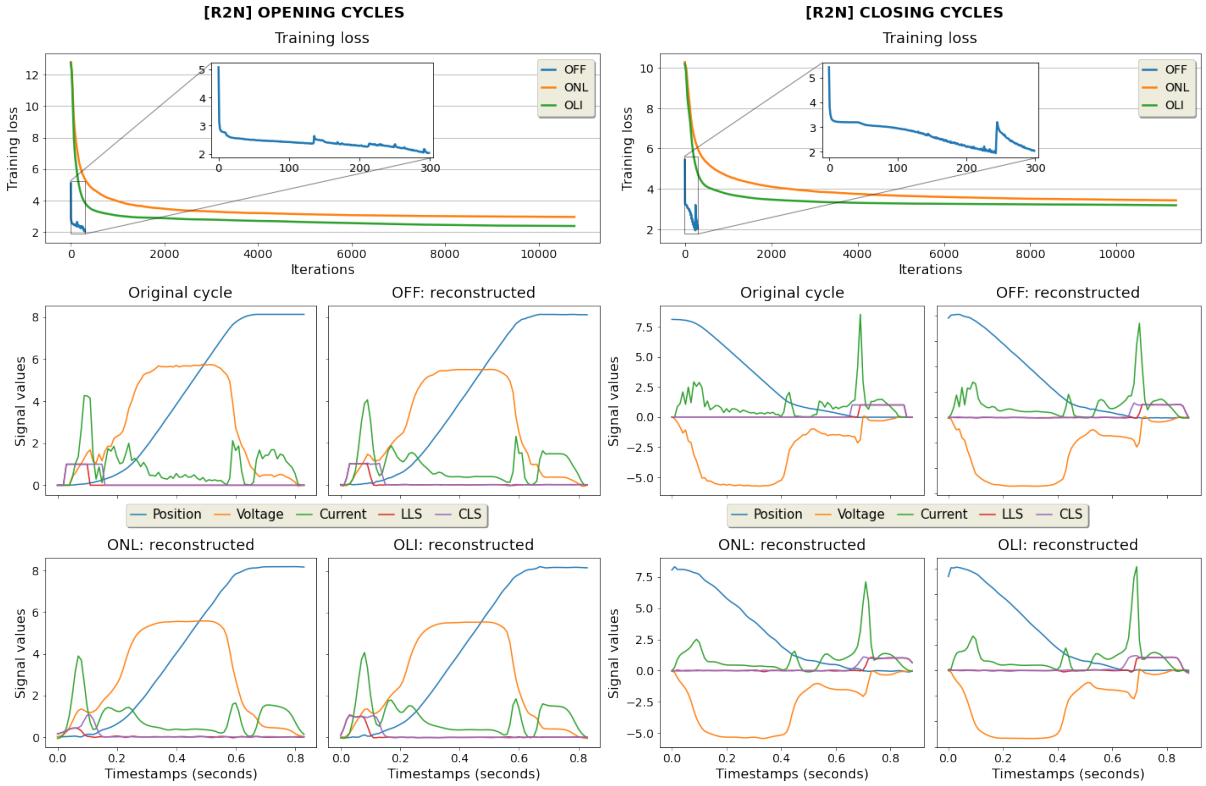


FIGURE 5.7: Training losses and reconstruction of one example cycle of OFF, ONL, and OLI models on R2N cycles

4.2 Reactivity of offline and online feature learning

In Chapter 4, we evaluate the reactivity of InterCE by observing its temporal accuracy, and a model that reaches a stable accuracy earlier is the one with a superior reactivity. However, we cannot apply it to this experiment. The three models are trained in starkly different settings: while the offline model is trained on all training examples over multiple epochs, the online models are trained continuously on a single small batch of examples for one epoch. Also, the performance of the offline model is notably superior to that of the two online models (Section 4.1). Even if an online model converges faster than the offline one, the features of the former are not truly usable due to its weak reconstruction ability.

Therefore, for this experiment, we consider that a model is only usable after it has processed all the training examples, and regard the training time of each model as its time to convergence. Figure 5.8

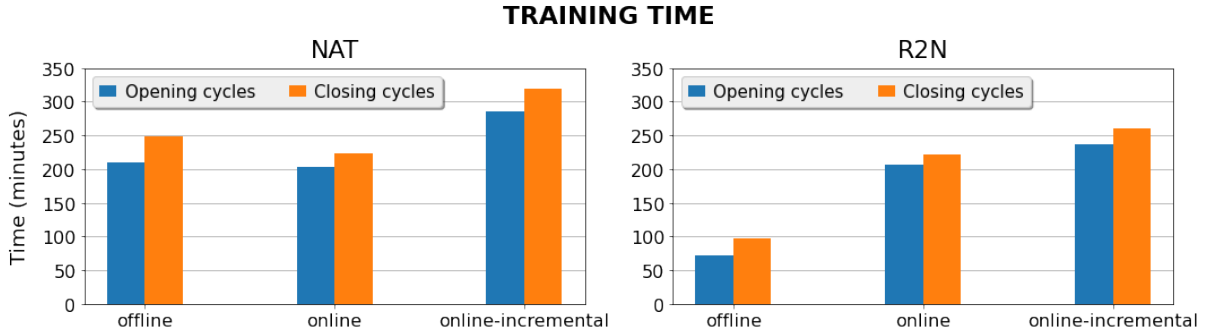


FIGURE 5.8: Training time of each model on both data sets

plots the training time of each model, for each cycle, of each data set. In both cases, OLI has the longest training time. ONL and OFF are close in training time on NAT data, but the gap is larger on R2N data, with OFF finishing its training much faster - we do not yet know the reason behind its fast execution.

Unfortunately, the hypothesis H_f^2 is not validated, as the offline version of the LSTM-AE achieves a better performance both in terms of reconstruction capacity and of convergence time.

4.3 Reduced information loss

In this experiment, we compare whether the features learned by the LSTM-AE reduce information loss better than the indicators identified by a human expert. For the feature vectors, we use the offline LSTM-AE to extract the features, since it outperforms its online counterparts. The expert indicators are extracted by an existing expert system that was made for both data sets. The size of the feature vectors by LSTM-AE and of the expert indicator vectors is displayed in Table 5.2.

TABLE 5.2: The size of the feature vectors and of the expert indicator vectors

		NAT	**R2N**
Door opening	LSTM-AE	40	40
	Expert indicators	25	83
Door closing	LSTM-AE	40	40
	Expert indicators	24	70

Because the original cycles cannot be reconstructed solely from the expert indicators, we cannot directly quantify the amount of information loss via the reconstruction loss used to train the LSTM-AE. Instead, we assess the information loss via a *ranking* evaluation. The intuition is that, if the features are truthfully learned from the original cycles, anomalous cycles will result in anomalous features, and normal cycles will result in normal features. Not only the abnormality should be preserved, but the *degree of abnormality* should be maintained as well. In other words, the top k most anomalous cycles should map to the top k most anomalous features.

Let $\mathcal{C}_k = [\, C^{(1)}, \ldots, C^{(k)} \,]$ and $\mathcal{X}_k = [\, X^{(1)}, \ldots, X^{(k)} \,]$ respectively be the ranking of the k most anomalous cycles and k most anomalous features, such that $\forall i, j \in [1, \ldots, k], i < j$, $C^{(i)}$ is more anomalous than $C^{(j)}$, and similarly for $X^{(i)}$ and $X^{(j)}$. Our goal is to verify whether the ordering of $\mathcal{C}_k$ matches with that of $\mathcal{X}_k$. We further distinguish the ranking of expert indicators $\mathcal{X}^E$ from the ranking of LSTM-AE features $\mathcal{X}^L$.

Let $\overline{C}$ be the reference profile of clean cycles that are used to train the LSTM-AE (Table 5.1), $\overline{X}^E$ the profile of clean expert indicators extracted from the clean cycles, and $\overline{X}^L$ that of the LSTM-AE features. The cycles used for testing are all excluded from this clean set. The ranking of the cycles, of the expert indicators, and of the LSTM-AE features are obtained as follows.

- The abnormality of a cycle is computed by the area under the curve of this cycle to the profile $\overline{C}$. The greater the area is, the more this cycle deviates from $\overline{C}$ and the more anomalous it is.
- The abnormality of an expert indicator vector is the Euclidean distance between this vector to the profile $\overline{X}^E$. The greater the distance is, the more anomalous this vector is with respect to $\overline{X}^E$.
- The abnormality of an LSTM-AE feature vector is the Euclidean distance between this vector to the profile $\overline{X}^L$. The greater the distance is, the more anomalous this vector is with respect to $\overline{X}^L$.

The experiment is set up as shown in Figure 5.9. The ground-truth ranking $\mathcal{C}$ is obtained by sorting the cycles in the test set (C_{test}) by their area under the curve to the profile $\overline{C}$ in the descending order. The ranking of the LSTM-AE feature vectors $\mathcal{X}^L$ is obtained by sorting the distance of the test feature vectors $\hat{X}^L$ to $\overline{X}^L$, similarly for $\mathcal{X}^E$ via $\hat{X}^E$ and $\overline{X}^E$. We want to see, between $\mathcal{X}^E$ and $\mathcal{X}^L$, which ranking is closer to the ground-truth $\overline{C}$.

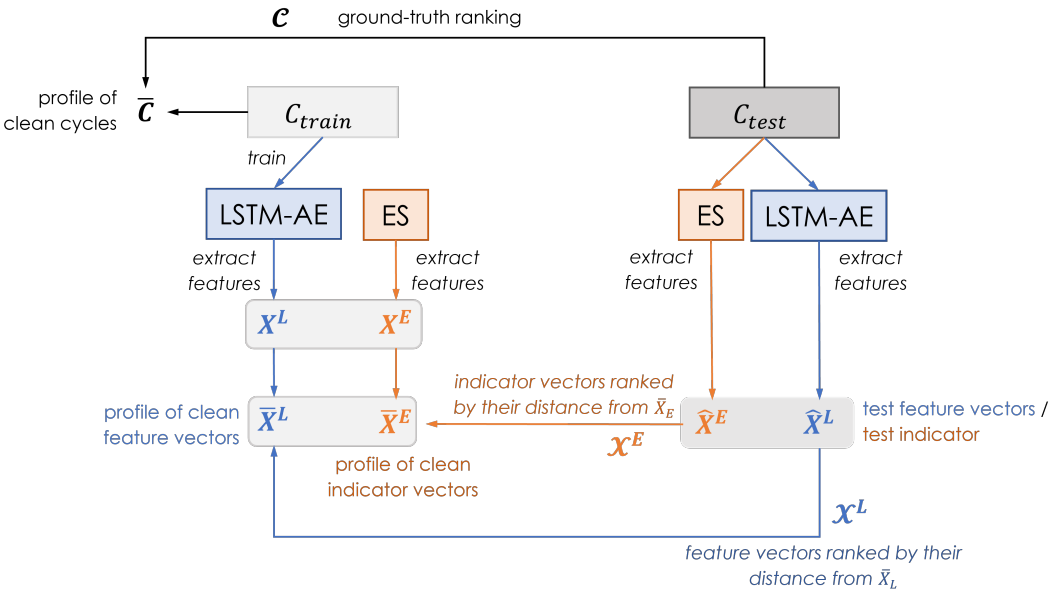


FIGURE 5.9: Configuration of the cycle-feature ranking experiment

To compare two rankings, we use the normalized discounted cumulative gain (nDCG) [114] that evaluates the quality of an algorithm-based ranking with respect to a ground-truth ranking. The score is bounded between 0 and 1, with 1 being the perfect match. The nDCG is commonly used in information retrieval and assesses the quality of a ranking of items based on the relevance score (also called *gain*) of each item and their position. An item with a higher relevance score is more relevant to the user query and should be placed near the top of the ranking.

First, the discounted cumulative gain of the top k items is computed by summing, for each position from 1 to k, the cumulative gain (relevance score) divided by a logarithmic discount factor (5.9). The idea is that the lower the position of a document, the less likely a user consults it, thus the need of a rank-based discount factor. The logarithm base can be any value; we choose the common base 2.

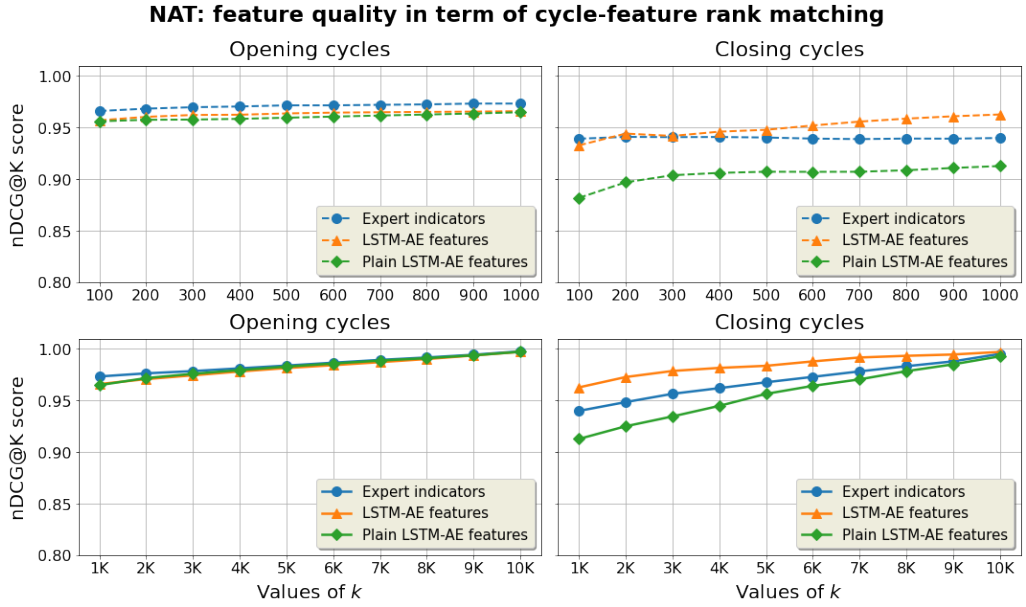
$$DCG@k = \sum_{i=1}^{k} \frac{G_i}{\log_2(i+1)} = G_1 + \sum_{i=2}^{k} \frac{G_i}{\log_2(i+1)} \tag{5.9}$$

The ideal discounted cumulative gain (iDCG) is computed similarly, but the relevance score G_i at each position i is replaced by the relevance score of the ground-truth ranking. The iDCG returns the maximum amount of gain on an ideal ranking. The nDCG score is equal to $\frac{DCG@k}{iDCG@k}$.

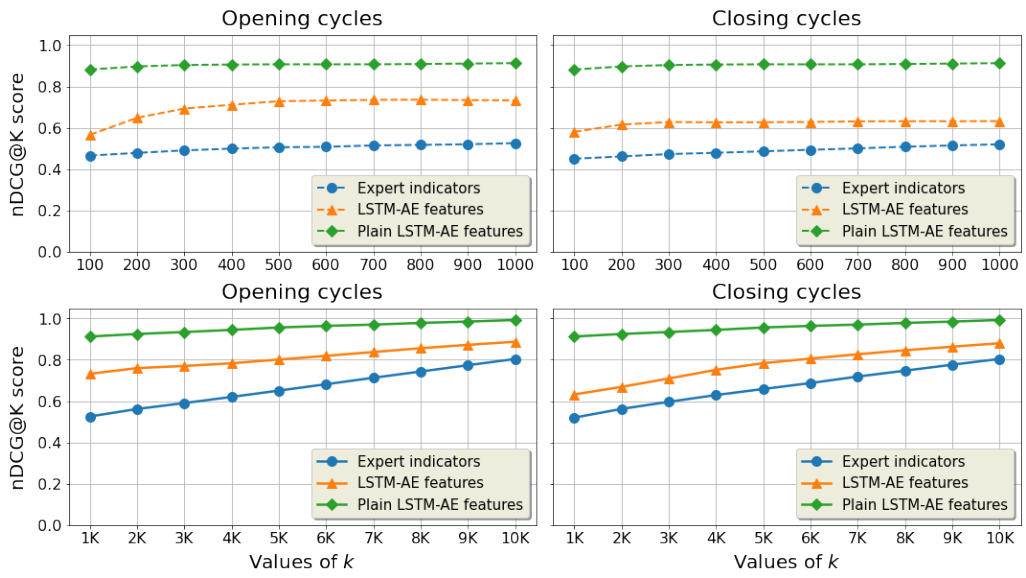
In this experiment, the relevance score of each item is the area under the curve of the cycles in C_k. The ranking of the cycles and of the vectors are obtained by sorting the cycles/vectors in a descending order of their area/distance. That is, we rank the test cycles by their area to the profile $\overline{C}$, the test indicator vectors and the test feature vectors by their Euclidean distance to the profiles $\overline{X}^E$ and $\overline{X}^L$, respectively. We refer to the ranking of cycles, of the indicator vectors, and of the feature vectors

respectively as $\mathcal{C}$, $\mathcal{X}^E$, and $\mathcal{X}^L$. We omit the notation k as in top k because the ranking is done on the entirety of the test data.

Once we have the ranking, we use the areas as the relevance scores for nDCG@k. The ideal relevance scores are the ranked areas of $\overline{\mathcal{C}}$. Then, we compare nDCG@k($\mathcal{X}^E, \mathcal{C}$) to nDCG@k($\mathcal{X}^L, \mathcal{C}$) with varying values of k, on a small range [100, ..., 1000] and on a large range [1000, ..., 10000]. Figure 5.10 shows the results on the small range of k on the first row, and on the large range of k on the second row. The features and indicators of the opening and closing cycles are evaluated independently. Please note that the score range is different for each data set: for NAT data, the nDCG@k score is shown in [0.8, 1.0], while for R2N data, the nDCG@k score is shown in [0.0, 1.0].



(A) Results on NAT cycles and features



(B) Results on R2N cycles and features

FIGURE 5.10: Evaluation of cycle-feature ranking via the nDCG@k metrics

The LSTM-AE features outperform the expert indicators in almost all cases, except in the opening cycles of NAT data, albeit by only a small gap (approximately 0.01 in nDCG score). This result highlights the strength of the LSTM-AE over the expert system for learning features. As the LSTM-AE produces features whose ranking is closer to that of the original cycles, its ability to preserve information is better than manual feature engineering.

Both the LSTM-AE and the expert system perform very well on NAT data, achieving nDCG@k scores above 0.85 for the opening and closing cycles. On R2N data, the expert system and the joint LSTM-AE are less successful, but interestingly, the plain LSTM-AE achieves significantly higher scores than the other two models. This is surprising, considering that R2N data contain more perturbations than NAT data and have fewer train stations. Fairly speaking, the plain LSTM-AE has a competitive performance to the joint version; even if the latter produces better features on NAT data, the difference in performance is not substantial.

The LSTM-AE can be potentially improved by having more than 40 features, or by adding more contextual variables in the classifier. For now, we only use the information of the train stations. Other contextual variables can be of use, such the outside temperatures, the mission code of each train, the creation time of the cycles (useful to recognize peak and off-peak hours), et cetera.

This experiment shows that the LSTM-AE is able to produce features whose characteristics are closer to the original cycles than the expert system, and thus is better at reducing information loss. Therefore, the hypothesis H_f^3 is validated.

5 Conclusion

In this chapter, we seek a method to learn relevant features from the cycles. To this end, we develop two versions of a long short-term memory autoencoder (LSTM-AE). The plain version has the traditional structure of encoder-decoder and solely focuses on learning to reconstruct cycles. The joint version connects a classifier that maps a cycle to its contextual train station to the encoder, in order to encourage the encoder to learn features that are truthful to the original cycles and robust against contextual noises. We evaluate both versions of the LSTM-AE on the NAT and R2N data sets to validate the following hypotheses:

- ($\mathbf{H_f^1}$) Online feature learning performs superior to, or at least on part with, offline feature learning in terms of accuracy.
- ($\mathbf{H_f^2}$) Online feature learning is more reactive than offline feature learning.
- ($\mathbf{H_f^3}$) Feature learning results in better information preservation than feature engineering.

We implement three paradigms of the LSTM-AE for each version (plain and joint): offline, online, and online incremental. The offline model is trained on the entirety of training cycles on multiple (k) epochs. The online model updates its weights once on each incoming training cycle. The online incremental model updates its weights once on a small batch of training cycles, including the new one and $k-1$ previous cycles.

The experimental results show that the offline model is the most performing, achieving the highest reconstruction capacity and having a lower training time to the online counterparts. Unfortunately, this means that H_f^1 and H_f^2 cannot be validated. On the other hand, the online incremental model is better than the online model in terms of reconstruction capacity. This implies that it is better to update a neural network on several training examples than on a single one.

To study the hypothesis H_f^3, we compare the features extracted by the (offline) plain and joint LSTM-AE to the indicators engineered by an expert system. Information preservation means the characteristics, or the anomalous ranking of cycles, should be preserved in their resulting features. We evaluate this criterion via a ranking assessment. The results show that the LSTM-AE features are better at preserving the ranking than the expert indicators in most cases, except for the opening cycles of NAT data, but the gap is small (0.01 in difference via the normalized discounted cumulative gain score). Therefore, H_f^3 is validated. Nonetheless, we note that the joint version does not largely outperforms the plain version.

In conclusion, the experiments highlight that a learning algorithm can outperform the baseline expert system in extracting good features in the offline setting. The online setting needs further improvement to close the gap between offline and online feature learning.

Also, we discover a crucial flaw that is inherent to autoencoders: it erases perturbations from the inputs in favor of a smoother reconstruction. Because autoencoders aim to learn the most representative abstraction of the inputs, they tend to be invariant to perturbations, for instance, a sudden peak in the signals. Yet, the noises in the input cycles may indicate an anomaly manifesting in a system. Future works include, but not limited to, adding explanability to the LSTM-AE to interpret the magnitude of change in the features with respect to the perturbations in the cycles, adjusting the loss function and/or the architecture of the network to retain important perturbations and to improve the feature quality, modifying the online training setting to make the online models competitive to the offline counterparts.

Finally, the features extracted by the LSTM-AE become the inputs to the next task, which is health detection. The goal is to discover and maintain a set of health profiles from the stream of feature vectors, and from these health profiles, to continuously monitor the health of each system.

Chapter 6

Health detection

Contents

Summary

To address adaptive and continuous learning for predictive maintenance on an unlabeled stream of feature vectors, we propose Continuous Health Monitoring using Online Clustering (CheMoc) as an unsupervised method that captures evolving health profiles of the monitored systems incrementally, assesses their working condition continuously via an adaptive health score, and works efficiently on streaming data. The works in this chapter address four hypotheses: H_d^1, H_d^2, H_d^3, and H_d^4.

The experimental results show that CheMoc succeed to discover a set of health profiles of the systems that correspond to the underlying physics of the anomalies linked to the PASs, as confirmed by a railway expert. CheMoc attains a satisfactory accuracy as it consumes only a moderate amount of computational resource to process the entire stream. Although we do not have a ground-truth to assess the accuracy of the health score computation, the evolution of the scores is coherent to the data trajectory of the systems.

1 Introduction

After cycle extraction and feature learning, we obtain a stream of unlabeled feature vectors from all systems in the fleet. On this stream, we determine the evolving health of the systems and identify the anomalies affecting them.

We define the *health* of a system as *its extent of being free from anomaly* (Section 3.4), quantified by a score bounded between 0 and 1 that locates the system health on the spectrum of being normal (close to 0) or being anomalous (close to 1). Because the railway operational constraint enforces that most of the fleet must be functional at any time, the majority of the data from the fleet defines the normal health. A normal system is one that produce data similar to the data of most systems in the fleet. We refer to this normalcy as a *reference* or *typical* profile. An anomalous system is one whose data deviate from the reference; an anomaly is any manifestation observable from the data that deviates a system from the typical behavior.

Determining the health of a system can be tackled in various ways. A system can be classified as "healthy" or "unhealthy" at a given moment. Alternatively, the health score can be projected via regression. However, there are no labels related to the system health, and it is difficult to label the health scores manually due to various factors (the amount of data, the speed of the stream, the uncertainty in identifying a system health, human inaccuracy). As a consequence, supervised or semi-supervised methods are not easily applicable in this scenario.

Therefore, we propose Continuous Health Monitoring using Online Clustering (CheMoc) as an unsupervised method that employs online clustering to continuously monitor the system health on an unlabeled stream of feature vectors. We formulate machinery health monitoring as a clustering task. Our intuition is that the data of different systems under the same health profile form a cluster, and the assignment of data in the clusters reflects not only the system health but also the type of manifesting anomalies and their severity.

We use DenStream [44] as the core online clustering algorithm for CheMoc. Due to the complexity of machinery health monitoring and railway operational constraints, we modify DenStream as follows: we omit the offline clustering process because the online phase is sufficient to maintain relevant clusters; we adjust the density parameters ϵ dynamically based on the radii of the clusters; we omit cluster pruning to stabilize the learning of health profiles; we store, in a cluster, the summary statistics of each system that produces data in it while ensuring fast and incremental cluster update.

The four hypotheses addressed in this chapter are:

- ($\mathbf{H_d^1}$) The clusters represent the health profiles of the fleet.
- ($\mathbf{H_d^2}$) Computing the health score by considering the data a system creates in the health profiles results in an accurate health detection, within an observable perimeter from the data.
- ($\mathbf{H_d^3}$) Online clustering reaches convergence earlier than offline clustering, if no drift occurs.
- ($\mathbf{H_d^4}$) Online clustering keeps track of the temporal evolution of the health profiles via cluster updates, while offline clustering cannot.

Due to the lack of ground-truth and manual examination of the clusters, we evaluate CheMoc only on the R2N data set. The results show that CheMoc is able to capture relevant health profiles of the fleet, as confirmed by a railway expert. Also, CheMoc finishes processing the data of one year with more than 1 million data points in approximately one hour, using a total of 600 MB memory to store internal data.

This chapter is organized as follows. Section 2 reviews previous works on machinery health monitoring. Section 3 describes CheMoc in detail. Section 4 shows the experimental evaluation. Section 5 concludes this chapter.

2 Related works

Machinery health monitoring is often studied under the lens of anomaly detection or fault diagnostics. For these tasks, the ground truth is usually unavailable or is very limited. Relying on supervised methods is not practical, as sensorized systems may generate an enormous volume of unlabeled data in a short period of time, making manual labeling infeasible. Because the data arrive continuously, repeatedly retraining an intricate model is suboptimal, in comparison to online methods that learn incrementally on infinite amount of data. Nonetheless, a substantial literature body in health monitoring adopts the traditional offline approach.

Supervised offline learning is a popular choice if labeled data are available. Yang, Habibullah, Zhang, Xu, Lim, and Nadarajan [241] trained a neural network on hand-picked features to learn a smoothed non-linear approximation of the system health. Wei, Wu, and Terpenny [236] implemented a Dynamic Variational AutoEncoder with a custom objective function to learn time-dependent health index of complex systems and to estimate the remaining useful life. Zhang, Jiang, Li, and Zhang [250] trained a multi-head self-attentive neural networks with gated recurrent units and variational mode decomposition on run-to-failure data to build a health index for pumped storage units. Zhang, Gupta, Farahat, Ristovski, and Ghosh [248] learned the health indicators as a credit assignment problem from unlabeled sensor data using reinforcement learning, but assuming that failure data were available. A regressor was trained on the learned indicators to estimate the remaining useful life. All models were trained offline. In spite of significant results of supervised models, labeled data remain difficult to obtain in practice.

On the other hand, works using unsupervised methods exist, but most are not online-compatible. Lima, Paredes Crovato, Goytia Mejia, Rosa Righi, Oliveira Ramos, André da Costa, and Pesenti [143] detected deterioration trends via signal decomposition and used k-means to group the extracted trends into clusters and to build the health index, but the entire process must be rerun on new data. Melani, Michalski, Silva, and Souza [157] used Moving Window Principle Component Analysis to detect atypical non-sudden changes in the monitored parameters and forwarded the output to a hand-crafted Bayesian network to infer the most likely failure modes. Bayesian networks are not easy to build and learning their parameters (conditional probability tables) is a non-trivial task. Feng, Weng, He, Han, Lu, Ren, and Ouyang [68] estimated the state of health of Li-ion batteries by comparing new charging data to the support vectors learned offline on clean data from fresh batteries cell. Diez, Khoa, Makki Alamdari, Wang, Chen, and Runcie [59] detected structural damage in bridges via k-means clustering on clean data preprocessed offline. The models were trained offline on past data before being deployed online for damage detection and stayed constant. Later, Tian, Khoa, Anaissi, Wang, and Chen [221] extended this work to perform real-time online damage detection using one-class support vector machines that classified new data as normal or anomalous. They coined the problem of online anomaly detection as concept drift adaptation and devised the rules to incrementally update the support vectors.

The dominance of offline methods notwithstanding, works on health monitoring in fully online scenario have started to emerge. Aydemir and Acar [21] detected machinery anomalies using a simple control chart called Cumulative Sum (CUSUM). CUSUM continuously monitored the sensor signals and issued an alert when the signals exceeded the predefined limits of healthy operation. Despite (or due to) its simplicity, CUSUM disables finer analysis on the temporal evolution of the system health. Ben Ali,

Saidi, Harrath, Bechhoefer, and Benbouzid [25] performed online monitoring of high-speed bearings of wind turbines using Adaptive Resonance Theory 2 (ART2). Relevant features were extracted and fed to an ART2 model to categorize the inputs as healthy, degraded, or failure. ART2 could run online and did not require training data nor offline learning. However, it did not capture health profiles and focused solely on classifying the system health into one of the three predefined states. Ribeiro, Pereira, and Gama [193] predicted failure of train doors via sequential anomaly detection. The inputs were binned and classified as normal or abnormal, then a low-pass filter smoothed the classification outputs to issue the final anomaly alerts. They provided no insights into the degradation pattern. Most recently, Putina and Rossi [190] employed DenStream to detect anomalies from streams of network telemetry data. The emphasized on anomaly detection in a routing network rather than on monitoring the system health. Still, their work provided us useful guidelines on hyperparameter selection for DenStream.

In general, previous works on this vein train complex offline models on past data or build online unsupervised models with a primary focus on anomaly detection. To tackle machinery health monitoring, we use online clustering to discover the health profiles of a fleet on an unlabeled stream and build a health index that quantifies the working condition of a system on the basis of these clusters.

We would like to note that our work goes beyond anomaly detection. Anomaly detection is the task to find occurrences of data points that are "significantly different from the remaining data" [4]. Such anomaly often appears as a "peak" in the data. Examples of anomaly detection are fraud detection in banking or network intrusion.

Machinery health monitoring has a larger scope than anomaly detection: beside detecting which system is straying from a normal health, we also characterize the manifesting anomalies and quantify their impact. Moreover, a degradation-induced anomaly occurs on the long run. It is not a peak that suddenly appears; instead, it is a phenomenon that happens gradually and aggravates over time (if no maintenance is carried out). We can also approach our task with *sequential anomaly detection* [36, 193], the goal of which is to detect a sequence of anomalies from a larger sequence of data. Yet, sequential anomaly detection assumes that only one anomaly is occurring, or makes no distinction on anomaly types at all, whereas in the railway, a system can be impacted by multiple anomalies simultaneously at different degrees.

3 Continuous health monitoring using online clustering

In this section, we introduce the fundamental concepts of CheMoc (Section 3.1) and explain how we modify DenStream to align it to the railway constraints (Section 3.2). The input of CheMoc is a stream of feature vectors. Hereafter, we refer to "feature vectors" and "data points" interchangeably.

The objective of CheMoc is twofold: to discover a set of evolving health profiles as clusters from the stream, and to compute an any-time health score of any system using these clusters. To cluster data streams, an algorithm must be able to update the clusters continuously, by creating a new cluster, removing an outdated cluster, or growing/shrinking an existing cluster. CheMoc uses a modified version of DenStream to evolve the clusters on the stream, then computes the adaptive health score of the monitored systems from the statistics collected from these clusters (Figure 6.1).

The constraints on which we build CheMoc are the following:

- **Fleet-level analysis:** The data produced by systems of the same kind[1] are analyzed in the same pipeline. Despite being identically manufactured, these systems degrade differently due to their

[1]Two PASs are two systems of the same kind, but a PAS and a battery are two distinct types of system.

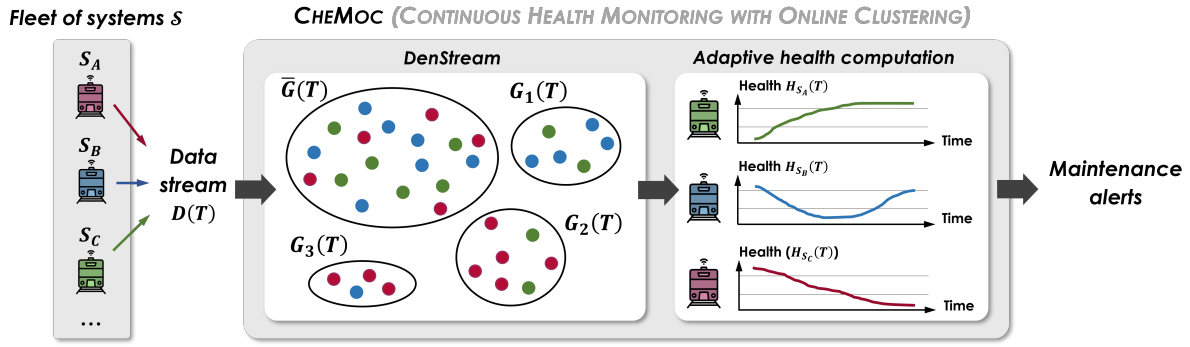


Figure 6.1: From a stream $D(T)$ containing the data from all the systems in the fleet, CheMoc uses a modified version of DenStream to update the clusters on new data, then computes the health score $H_{S_m}(T)$ of each system S_m adaptively.

diverse operational context. Using the data from all the systems allows to learn the common behavior of the entire fleet. Even so, the health of each system is assessed individually.

- **Cycle type distinction:** The data describing one function of the systems is analyzed in its own pipeline, because each cycle type is subject to different kinds of anomaly. For the PASs, we analyze the data of the door opening cycles and of the door closing cycles independently.
- **Multiple anomalies:** A system can be affected by multiple anomalies simultaneously.
- **Unique reference profile:** As a system has only one normal behavior, we identify one unique cluster of good health at any given time, denoted $\overline{G}$.

3.1 Fundamental concepts

As a reminder, $\mathcal{S} = \{S_1, \dots, S_M\}$ is the fleet of M systems. Each system produces cycles that are transformed to feature vectors of dimension P, and the sequence of feature vectors constitutes a data stream $D(T)$ containing the data from the fleet.

Definition 3.1 (Data stream). *A data stream $D(T)$ is a sequence of data points produced by the systems in $\mathcal{S}$ from $t = 1$ to $t = T$, arriving in chronological order.*

$$D(T) = \left(X^t_{S_m} \mid S_m \in \mathcal{S}, 1 \leqslant t \leqslant T, X^t_{S_m} \in \mathbb{R}^P\right)$$

Partitioning the stream $D(T)$ yields a set of clusters characterizing normal and anomalous behaviors commonly encountered in the systems. $D(T)$ can be partitioned into K evolving clusters $\mathcal{G}(T) = \{G_1(T), \dots, G_K(T)\}$, including $\overline{G}(T)$.

Definition 3.2 (Cluster). *A cluster $G_k(T)$ is a set of data points grouped together at the time T, based on a cluster membership condition ϕ depending on the algorithm being used.*

$$G_k(T) = \{X^t_{S_m} \mid 1 \leqslant t \leqslant T, S_m \in \mathcal{S}, \phi\}$$

To determine the degree of anomaly of a cluster, we assign a value $0 \leqslant \omega_k(T) \leqslant 1$ to each cluster $G_k(T)$. Intuitively, a cluster of few data hints an anomaly or a noise. Nonetheless, as new data arrive from the stream, a small cluster at T may precede the emergence of a new normal behavior. Judging the abnormality of a cluster solely by its size is misleading; instead, the judgment should be based on

the data contained in a cluster. Because an anomaly is a deviation from what is normal, the anomaly degree of $G_k(T)$ is the difference between $G_k(T)$ and $\overline{G}(T)$. To compute such deviation, each cluster holds a *statistic matrix* that collects the statistics of the data grouped in this cluster until T, denoted $stat_k(T)$. The matrix of $\overline{G}(T)$ is denoted $\overline{stat}(T)$.

Definition 3.3 (Statistic matrix). *A statistic matrix $stat_k(T) \in \mathbb{R}^{N_{stat} \times P}$ of a cluster $G_k(T)$ stores a number of N_{stat} statistics collected incrementally for each of P features from the data points in $G_k(T)$, such that one entry $A_{i,j}$ in the matrix is the i^{th} incremental statistic updated until T of the j^{th} feature.*

$$stat_k(T) = [\ A_{i,j}\ ]_{\substack{1 \leqslant i \leqslant N_{stat} \\ 1 \leqslant j \leqslant P}}$$

The deviation of a cluster $G_k(T)$ to the reference profile $\overline{G}(T)$ is the Frobenius norm of the difference between their statistic matrix, denoted $\Delta_k(T)$ (6.1).

$$\Delta_k(T) = \parallel \overline{stat}(T) - stat_k(T) \parallel_F \geqslant 0 \tag{6.1}$$

We scale $\Delta_k(T)$ to the range of $[0, 1]$ to make it the anomaly degree of $G_k(T)$. Because the clustering algorithm may not find the perfect partitioning of the data, we are not always certain that a cluster is truly an anomalous behavior. The anomaly degree of a cluster therefore implies *the possibility that a cluster is an anomaly*, estimated by the deviation of this cluster from the reference $\overline{G}(T)$.

Definition 3.4 (Anomaly degree). *The anomaly degree $\omega_k(T)$ of a cluster $G_k(T)$ is the extent to which $G_k(T)$ is an anomaly as observed so far on $D(T)$. A degree close to 1 means that $G_k(T)$ is very likely an anomalous behavior, and inversely if close to 0. The anomaly degree of $\overline{G}(T)$ is always 0.*

$$\omega_k(T) = \frac{\Delta_k(T)}{\max\limits_{1 \leqslant k' \leqslant K} \Delta_{k'}(T)} \in [0, 1] \tag{6.2}$$

Because we define the health of a system its extent of being free from anomalies, the health of a system S_m at T is aggregated from its *anomaly scores*. Let $A_k^{S_m}(T)$ be the anomaly score that quantifies the severity of an anomaly $G_k(T)$ that is occurring in S_m at T.

Definition 3.5 (Anomaly score). *The anomaly score $A_k^{S_m}(T)$ is a real-valued number in $[0, 1]$ that quantifies the severity of an anomaly $G_k(T)$ occurring in S_m at T, with 0 being the most normal and 1 being the most severe.*

Intuitively, the anomaly is severe in S_m if S_m mainly produces data in $G_k(T)$ and fewer in $\overline{G}(T)$. Let $G_k^{S_m}(T)$ and $\overline{G}^{S_m}(T)$ be the set of data points from S_m in $G_k(T)$ and in $\overline{G}(T)$ respectively, the anomaly score $A_k^{S_m}(T)$ is the ratio of $|G_k^{S_m}(T)|$ over the sum $|G_k^{S_m}(T)| + |\overline{G}_k^{S_m}(T)|$. If S_m produces very few data in $\overline{G}(T)$, S_m is functioning under the anomaly profile $G_k(T)$ at the time T and its anomaly score will be close to 1 (very anomalous); inversely, $A_k^{S_m}(T)$ is close to 0 if S_m primarily produces data in the reference profile, thus being close to healthy. We do not compute the anomaly score on the reference cluster, so $\bar{A}^{S_m}(T)$ does not exist.

$$A_k^{S_m}(T) = \frac{|G_k^{S_m}(T)|}{|G_k^{S_m}(T)| + |\overline{G}^{S_m}(T)|} \in [0, 1] \tag{6.3}$$

Figure 6.2 illustrates an example, in which three monitored systems (S_A, S_B, S_C) generate data that are partitioned into a set of clusters $\mathcal{G}(T) = \{ \overline{G}(T), G_1(T), G_2(T), G_3(T) \}$ (whose anomaly degree is $\overline{\omega} = 0$, $\omega_1 = 0.7$, $\omega_2 = 0.3$, $\omega_3 = 1.0$, respectively), where $\overline{G}(T)$ is the reference profile, and the others represent anomaly profiles. In this example, we compute the anomaly scores of the system S_C on three types of anomaly G_1, G_2, and G_3. Following (6.3), we have:

$$|\overline{G}^{S_C}(T)| = 6 \qquad |G_1^{S_C}(T)| = 3 \qquad |G_2^{S_C}(T)| = 2 \qquad |G_3^{S_C}(T)| = 1$$

and therefore, the anomaly scores $A_1^{S_C}(T)$, $A_2^{S_C}(T)$, and $A_3^{S_C}(T)$ are:

$$A_1^{S_C}(T) = \frac{3}{3+6} \approx 0.333 \qquad A_2^{S_C}(T) = \frac{2}{2+6} = 0.25 \qquad A_3^{S_C}(T) = \frac{1}{1+6} \approx 0.143$$

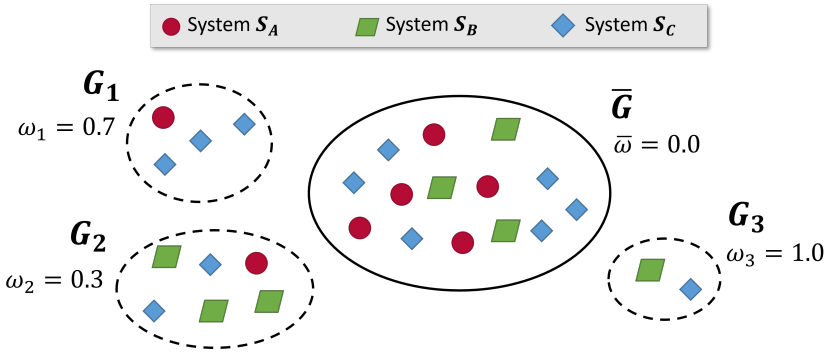


FIGURE 6.2: An example on how to compute the anomaly scores. We omit the notation of time T in the figure for simplicity.

However, (6.3) is solely based on the amount of data in the clusters, disregarding the temporality of the data. Equation (6.3) considers old and recent data equally relevant, whereas more recent data should should be more relevant regarding the current health of the system. To insist on the data temporality, we damp the anomaly score by a decay factor $f(t) = 2^{-\lambda t}$, where t is a time interval and $\lambda > 0$ dictates how fast the data are forgotten (higher λ means faster decay). We replace the count of data points in a cluster by the sum of the decay factor of each data point in $G_k^{S_m}(T)$ (6.4). The time interval to compute the decay is between the timestamp of the latest data point from S_m on $D(T)$, denoted T_m, to the creation time of each data point T_i by S_m in the cluster. We will explain why we take T_m as the anchor point in Section 3.2. In (6.4), i denotes the i^{th} data point in $G_k^{S_m}$ and j denotes the j^{th} data point in $\overline{G}^{S_m}$.

$$A_k^{S_m}(T) = \frac{\sum_i^{G_k^{S_m}} f(T_m - T_i)}{\sum_i^{G_k^{S_m}} f(T_m - T_i) + \sum_j^{\overline{G}^{S_m}} f(T_m - T_j)} = \frac{1}{1 + \frac{\sum_j^{\overline{G}^{S_m}} f(T_m - T_j)}{\sum_i^{G_k^{S_m}} f(T_m - T_i)}} \in [0, 1] \tag{6.4}$$

Finally, we define the health score. For a system S_m, we can estimate its health score $H_{S_m}(T)$ at a moment T by computing the average of the anomaly scores of S_m on all clusters $G_k(T)$, in which S_m

has produced at least one data point, weighted by $\omega_k(T)$. We exclude the reference profile $\overline{G}(T)$ from the health computation because the anomaly score is not computed on $\overline{G}(T)$.

Definition 3.6 (Health score)**.** *The health score $H_{S_m}(T)$ of a system S_m at a time T expresses the deviation of S_m from the good health and is the weighted average of the anomaly scores computed from the clusters in $\mathcal{G}(T) \setminus \overline{G}(T)$.*

$$H_{S_m}(T) = \frac{1}{K-1} \sum_{G_k}^{\mathcal{G} \setminus \overline{G}} \left(\omega_k(T) \times A_k^{S_m}(T) \right) \in [0, 1] \tag{6.5}$$

The anomaly degree $\omega_k(T)$ nuances the impact of an anomaly such that clusters with higher degree, i.e., those that are more likely anomalous, cast a larger impact to the health score. If $H_{S_m}(T) = 0$, S_m is free of all anomalies and is closest to the normal profile at the time T. If $H_{S_m}(T)$ is close to 1, S_m is severely degraded and needs maintenance.

3.2 Online clustering with DenStream

From the fundamental concepts, we implement the core clustering in CheMoc with an online clustering algorithm. We study a number of candidate online clustering algorithms against the following criteria:

(1) able to detect cluster evolution,
(2) being efficient (low execution time, low to moderate memory usage),
(3) having self-adaptive hyperparameters (adjusting the initial values dynamically).

We also include DBSCAN in the study for completeness, because DBSCAN is one of the most well-known offline clustering algorithms and is the foundation of several online clustering algorithms.

Table 6.1 shows the list[2] of online clustering algorithms we studied and checks whether they address the aforementioned criteria. The first three algorithms include DBSCAN and its two extensions with self-adaptive density (ADBCAN [122]) and incremental update (IDBSCAN [63]). The rest is online clustering algorithms.

Table 6.1: Online clustering algorithms. (1) = incremental, (2) = cluster evolution, (3) = efficient, (4) = self-adaptive hyperparameters

Algorithm	Summary	(1)	(2)	(3)	(4)
DBSCAN [65]	The cluster's border is propagated to include data points that are reachable within a predefined radius (density-based). All points that are not assigned to any cluster are noises. Single pass over all points.				
IDBSCAN (1998) [63]	Incremental version of DBSCAN to update the clusters on new data points. Able to track the evolution of clusters. Scalable only if the number of data points is bounded.	x	x		
ADBSCAN (2018) [122]	Changes the density ε of DBSCAN at each iteration to capture clusters of varying density. Start at a random value of ε and increment by 0.5 at each iteration until 95% of the data are processed. All other unassigned data points are noises. Requires a predefined number of clusters.				x

[2]This list is by no means exhaustive. We recommend checking these excellent surveys [13, 46, 47, 78, 209, 257] for an extensive review on online clustering.

TABLE 6.1: (continued)

Algorithm	**Summary**	**(1)**	**(2)**	**(3)**	**(4)**
DenStream (2006) [45]	A micro-cluster's density is estimated via its weight and radius. Periodic pruning discards outliers. Suboptimal values of parameters may lead to a large number of clusters.	x	x	x	
ESA-Stream (2020) [140]	A cluster is a region of dense grids sharing borders. It defines three self-adaptive parameters: upper and lower bound of grid density, and a time gap to update the clusters. Lightweight, fully online.	x	x	x	x
ClusTree (2009) [131]	Adaptive to the speed of the stream. Using index structure (R-Tree*) to store history of the stream in form of micro-clusters.	x	x	x	
PDSDBCAN (2012) [181]	Using disjoint sets to break the sequential access of DBSCAN to allow random access and parallelization, also enables to change the object membership quickly without specific ordering. Achieve significant speedup when running on 10+ cores.	x	x	x	
C-DenStream (2009) [196]	Similar to DenStream but requiring domain knowledge about instance-level constraints ("must-link" versus "cannot-link").	x	x	x	
OpticsStream (2007) [217]	Extending cluster visualization from OPTICS [17] by considering time a projecting dimension, allowing to track cluster evolution. Basing on notions such as core distance, reachability distance, potential micro-clusters.	x	x	x	
D-Stream [50]	Dividing the data space into small grids of fixed size and distinguishing dense, sparse, and transitional grids depending on their density. Dense grids that shared adjacent facets constituted a cluster. Performing periodic pruning was to remove infrequent or mostly empty grids.	x	x	x	
DBSTREAM [88]	Introducing the shared density graph to explicitly capture the density area between micro-clusters. Updating the shared density graph via competitive learning to adapt to concept drifting.	x	x	x	
CluStream [5]	Two-phase clustering. Capturing summary statistics of the stream via micro-clusters in the online phase. Applying k-means in the offline phase on the micro-clusters to return clusters at request time.	x	x	x	
Young, Arel, Karnowski, and Rose [245] (2010)	Incremental clustering that constraints the movement of centroids once the clusters reach stability (a centroid is changing with very small tolerance). Using competitive learning to adaptively set the learning rate on new data. Assuming a fixed and known number of centroids.	x		x	
rDenStream (2009) [146]	Outlier micro-clusters are saved and revised to verify their outlierness. Three steps: online clustering (like DenStream), offline clustering (like DBSCAN), retrospect (using a classifier to classify outliers). High memory usage and time-consuming because of the retrospect. Improved accuracy, applicable for noisy stream.	x	x		
SDStream (2009) [192]	Two-phase clustering, similar to DenStream. Using sliding windows to discard clusters outside of the current window. Limited number of clusters, low memory usage, exponential histograms.	x		x	
SOStream (2012) [105]	Based on self-organizing maps and competitive learning. Finding overlapping clusters via winner clusters and merging clusters if possible. Full online to create, merge, and remove clusters. Dynamically defined threshold, but it is time-consuming.	x	x		
HDDStream (2012) [174]	Dealing with high-dimensional streams. Projecting micro-clusters on a subspace of the feature space. Only the most varying variables are considered. Time-consuming pruning process.	x	x	x	

TABLE 6.1: (continued)

Algorithm	Summary	(1)	(2)	(3)	(4)
PreDeConStream (2012) [92]	Extending HDDStream by expanding the clusters during the offline phase. Expensive pruning process.	x	x	x	
FlockStream (2009) [71]	Bio-inspired from flocking models: agents move within a predefined range, and if an agent visits another, a cluster is formed. Less comparisons between data points.	x	x	x	
Re-DBSCAN (2020) [161]	Differential update to the cluster using k-dist graph. Low execution time but high memory usage to store the k-dist values.	x	x		
DUCStream (2005) [77]	Grid-based clustering. Density of one unit is the number of points it contains. A local dense unit may become a dense unit. A cluster is a connected component of the graph, where the nodes are units, the edges are the common border between two units.	x	x		
RepStream (2009) [150]	Relying on a sparse, directed graph to store information about the relationship between data points (a node is a data point, an edge is the distance to the nearest neighbor of a point). Using a knowledge repository to reconstruct historic clusters.	x			
HCDStream (2014) [12]	Two-phase clustering. Combing DenStream and D-Stream (grid-like density-based clustering).	x	x	x	

Among these algorithms, some that appear most suitable are DenStream [44], D-Stream [50], and ESA-Stream [140]. We favor density-based clustering algorithms because they do not require inputting a predefined number of clusters, as the number of anomalies is unknown beforehand.

After considering these three algorithms, we decide to implement DenStream as the core clustering of CheMoc, for the following reasons. First, DenStream is density-based and does not need a predefined number of clusters. Secondly, DenStream relies on lightweight and decaying micro-clusters to efficiently cope with dynamic changes; this simplicity makes DenStream the building block of many other online clustering algorithms (SDSStream, rDenStream, C-DenStream, and so on). Thirdly, we would like to obtain a baseline result with DenStream, from which we can try other clustering algorithms to improve CheMoc; for instance, once the performance of CheMoc is confirmed using DenStream, we can implement ESA-Stream as a next improvement.

In this section, we will first describe the methodology of DenStream, then discuss why it does not fit completely to the railway constraints, and explain how we amend the misalignment.

3.2.1 DenStream

DenStream is a two-phase clustering algorithm. During the online phase, DenStream finds clusters from a data stream by maintaining a special data structure called *micro-clusters* and evolves these micro-clusters using a damping window model. On a stream, the importance of the data decreases exponentially over time via a fading function $f(t) = 2^{-\lambda t}$, $\lambda > 0$. A high value of λ makes past data quickly forgotten. The offline phase of DenStream considers the center of the micro-clusters as virtual data points and applies the traditional DBSCAN on these points to obtain the final clusters.

DenStream maintains lightweight statistics from the stream in the form of micro-clusters. A micro-cluster mc_i at a time t for a group of n points $p_{i_1}, \dots, p_{i_n}$ created at timestamps $T_{i_1}, \dots, T_{i_n}$ comprises the cluster features $\{w, LS, SS\}$ including the weight (6.6), the weighted linear sum (6.7), and the weighted squared sum (6.8). If the data points are multidimensional (with P variables), the linear sum and squared

sum are computed for each dimension, that is, $LS \in \mathbb{R}^P$ and $SS \in \mathbb{R}^P$.

$$w = \sum_{j=1}^{n} f(t - T_{i_j}) \tag{6.6}$$

$$LS = \sum_{j=1}^{n} f(t - T_{i_j})\, p_{i_j} \tag{6.7}$$

$$SS = \sum_{j=1}^{n} f(t - T_{i_j})\, (p_{i_j})^2 \tag{6.8}$$

From these cluster features, we can compute the center (6.9) and radius (6.10) of a micro-cluster at any time. A micro-cluster is dense if its radius is lower than the density threshold ϵ, inputted by the users. Therefore, the density ϵ is the most important hyperparameter of DenStream because it defines the desired density of the micro-clusters.

$$\bar{c} = LS \,/\, W \tag{6.9}$$

$$r = \sqrt{\frac{SS}{W} - \left(\frac{|LS|}{W}\right)^2} \leqslant \epsilon \tag{6.10}$$

A micro-cluster is updated in two cases: when it receives a new data point and when it decays.

When a new data point p_{n+1} is added in mc_i at a time t, because t is also the creation time T_{n+1} of p_{n+1}, we have $f(t - T_{n+1}) = f(0) = 2^{-\lambda \times 0} = 1$, the cluster features become:

$$w' = \sum_{j=1}^{n+1} f(t - T_{i_j}) = \sum_{j=1}^{n} f(t - T_{i_j}) + f(0) = w + 1$$

$$LS' = \sum_{j=1}^{n+1} f(t - T_{i_j})\, p_{i_j} = \sum_{j=1}^{n} f(t - T_{i_j})\, p_{i_j} + f(0) p_{i_{n+1}} = LS + p_{i_{n+1}}$$

$$SS' = \sum_{j=1}^{n+1} f(t - T_{i_j})\, (p_{i_j})^2 = \sum_{j=1}^{n} f(t - T_{i_j})\, (p_{i_j})^2 + f(0)\, (p_{i_{n+1}})^2 = SS + (p_{i_{n+1}})^2$$

When the micro-cluster does not receive any new data after an interval δt, it is possible that this micro-cluster is becoming obsolete and no longer corresponds to the current characteristics of the stream. Consequently, its cluster features decay. The decay is implemented via $f(\delta t + t - T_{i_j})$. For example, decaying the weight w of mc_i after an interval δt is done as follows.

$$w' = \sum_{j=1}^{n} f(\delta t + t - T_{i_j}) = \sum_{j=1}^{n} 2^{-\lambda(\delta t + t - T_{i_j})} = \sum_{j=1}^{n} 2^{-\lambda \delta t} 2^{-\lambda(t - T_{i_j})} = 2^{-\lambda \delta t} \sum_{j=1}^{n} 2^{-\lambda(t - T_{i_j})} = 2^{-\lambda \delta t} w$$

Similarly, we obtain the decayed cluster features for the linear sum and squared sum.

$$LS' = 2^{-\lambda \delta t} LS \qquad\qquad SS' = 2^{-\lambda \delta t} SS$$

A micro-cluster receiving a new point (6.11) or decaying (6.12) is incrementally updatable, which makes micro-clusters an efficient data structure for data stream clustering.

$$mc_i = \{ w + 1,\ LS + p_{i_{n+1}},\ SS + (p^2_{i_{n+1}}) \} \tag{6.11}$$

$$mc_i = \{ w \times 2^{-\lambda \delta t},\ LS \times 2^{-\lambda \delta t},\ SS \times 2^{-\lambda \delta t} \} \tag{6.12}$$

DenStream distinguishes two types of micro-clusters, depending on their weight. Let $\mu > 0$ be the minimum weight defining a dense micro-cluster, $0 < \beta \leqslant 1$ the outlierness threshold:

- If $w \geqslant \beta\mu$, mc_i is a *potential micro-cluster* (PMC) of high density.
- If $w < \beta\mu$, mc_i is an *outlier micro-cluster* (OMC) that may evolve into a PMC or is a noise.

On the stream, a PMC may be demoted to an OMC if it is not updated after a while, or an OMC can evolve into a PMC if it has become sufficiently dense. An OMC that is not updated after a while will be discarded to free up space. These operations are performed during the periodic pruning phase of DenStream, except promoting an OMC to a PMC that is done when merging a new data point in an existing micro-cluster. The interval that dictates the pruning frequency is called *the minimal time span* T_p (6.13), and is determined by the equation $2^{-\lambda T_p}\beta\mu + 1 = \beta\mu$, that is, the interval after which a micro-cluster becomes dense enough (its weight equal to $\beta\mu$). A PMC is demoted to an OMC if its weight at the pruning time falls below $\beta\mu$.

$$T_p = \left\lceil \frac{1}{\lambda} \log\left(\frac{\beta}{\beta\mu - 1}\right) \right\rceil \tag{6.13}$$

To decide which OMCs to discard, DenStream uses a lower weight limit ξ (6.14) that is a function of the current time t_c and the creation time of the OMC t_p. An OMC will not likely evolve into a PMC if its weight is below $\xi(t_c, t_o)$ and will be safely deleted.

$$\xi(t_c, t_o) = \frac{2^{-\lambda(t_c - t_o + T_p)} - 1}{2^{-\lambda T_p} - 1} \tag{6.14}$$

The full algorithm of DenStream is described in Algorithm 6.1. For every new data point p arriving at a time t from the stream, DenStream tries to add it to an existing PMC, only if the new radius of this PMC with p added is below the density ϵ (line 1–3). Otherwise, DenStream tries to add it in an existing OMC using the same logic; if adding p increases the weight of this OMC above $\beta\mu$, DenStream promotes this OMC to a PMC (line 5–10). If p cannot be added to any existing OMC either, DenStream initializes a new OMC with p (line 12–13). Every T_p, DenStream performs a periodic pruning to demote PMCs and to discard obsolete OMCs. Any PMC with a weight below $\beta\mu$ is demoted to an OMC. Any OMC with a weight below $\xi(t_c, t_o)$ is discarded and forgotten.

During the offline phase, the PMCs are turned into virtual points by keeping their centroid. A traditional DBSCAN is performed on these virtual points to return the official clusters when requested. The OMCs are omitted during the offline phase.

3.2.2 Adjusting DenStream to the railway scenario

Using DenStream as it was originally proposed does not fit the constrains of railway maintenance. This obliges us to modify DenStream to amend the misalignment.

Algorithm 6.1: DenStream [44]

Data: A data point p at a time t

```
/* add p to an existing micro-cluster or create a new one */
c_p ← get nearest PMC to p
r'_p ← compute radius of c_p if adding p
if r'_p ⩽ ϵ then add p to c_p
else
    c_o ← get nearest OMC to p
    r'_o ← compute radius of c_o if adding p
    if r'_o ⩽ ϵ then
        add p to c_o
        w_o ← weight of c_o after adding p
        if w_o ⩾ βμ then promote c_o to PMC
    else
        c_o ← newOMC()
        add p to c_o
/* perform periodic pruning */
if t % T_p = 0 then
    foreach PMC c_p with weight w_p do
        decay c_p
        if w_p < βμ then demote c_p to OMC
    foreach OMC c_o with weight w_o created at t_o do
        decay c_o
        ξ ← (2^{-λ(t-t_o+T_p)} - 1) / (2^{-λT_p} - 1)
        if w_o < ξ then delete c_o
```

(1) Offline clustering omitted Precisely, there are two offline processes in DenStream: one to *warm-start* the clusters at the beginning and the other to do a traditional batch clustering on the micro-clusters when requested, both using DBSCAN [65].

Warm-start We retain the warm-start to initialize CheMoc on a set of initial clusters. Through experimentation, we notice that the warm-start stabilizes the growth of clusters by CheMoc. Without it, CheMoc tends to create an excessive amount of OMCs and cannot discover meaningful clusters. Because the warm-start is not time-consuming, we deem it not an important bottleneck of CheMoc.

Offline clustering We omit the second offline process on virtual points with DBSCAN because we find out that the micro-clusters discovered online are sufficiently stable to monitor the systems. DenStream uses this offline clustering phase to eliminate noises from the OMCs to produce a clean set of clusters from the PMCs.

In the railway context, a system can only be in a finite set of health profiles and a system changes from one profile to another gradually. Even if the change might be sudden, this abrupt change will be mitigated by the data coming from other systems from $\mathcal{S}$, which reduces the noisiness on the stream and produces stable micro-clusters during the online phase. Not only unnecessary, offline clustering can be harmful because it collapses one meaningful micro-cluster representing one health profile into a single point, causing information loss.

Furthermore, the clusters obtained between each offline clustering are disconnected. It is not easy to conclude which clusters have evolved from the previous offline clustering. This disrupts the inherent continuity of the evolving micro-clusters.

Considering these factors, we only maintain the warm-start and online phase of DenStream to update the micro-clusters continuously. Thereafter, we refer to a micro-cluster as a cluster, since there is no more distinction between an official cluster obtained offline and a micro-cluster captured online.

(2) Dynamic density threshold DenStream has a crucial hyperparameter ϵ that decides the boundary of a cluster. Still, it is difficult to guess the true density of a data stream. Setting a high ϵ to find the clusters on sparse data can lead to excessively large clusters that group data of different health profiles together when the data become denser. Ideally, ϵ should be adjusted dynamically based on the current density of the stream.

To this end, we adapt ϵ dynamically following Putina and Rossi [190], by revising ϵ continuously based on the radius of the clusters. Let r be a random variable that records the radii of the clusters maintained thus far (from $t = 1$ to $t = T$), $\bar{r}$ and σ_r the mean and the standard deviation of r computed incrementally, the new density is set to:

$$\epsilon = \bar{r} + k\sigma_r \qquad \text{with} \quad k > 0 \tag{6.15}$$

We set $k = 3$ as recommended by Putina and Rossi [190].

(3) Pruning omitted To discards stale clusters, DenStream periodically prunes PMCs and OMCs that have not been recently updated to limit the number of micro-clusters maintained online and to retain only those consistent with the current characteristics of the stream. However, because we want to learn the health profiles of the systems, discarding clusters is to be avoided.

A cluster not having received data for some time is not necessarily outdated; it is likely that no system falls in that profile at a given period. This is a common scenario. If we prune idle clusters, CheMoc loses the information about the health profiles and must relearn them if those profiles are activated again in the future, leading to suboptimal behavior of CheMoc. As a result, we completely omit the periodic pruning from DenStream. It implies that the clusters also do not decay over time (Algorithm 6.1).

Nonetheless, we need to monitor the temporal evolution of the system health. Even if the clusters do not lose their relevance over time, the data from a system do. A data point produced by a system long ago no longer captures its current health. For instance, a system was healthy some weeks ago but has degraded since. Hence, the decay must occur *on the data themselves*, such that old data contribute less to the anomaly scores than recent data. This is effectively captured by (6.4): instead of simply counting the number of data points in a cluster as in (6.3), (6.4) incorporates the decay factor in the computation and decreases the impact of old data over time, thus addressing the adaptivity of CheMoc.

(4) Varying anchor timestamp by system The weight of a cluster mc_i of n data points at t is $w = \sum_{j=1}^{n} f(t - T_{i_j})$, where T_{i_j} is the timestamp of the point p_{i_j} in mc_i. Normally, t applies to all points in mc_i such that a point p_{i_j} has lower relevance the further its timestamp T_{i_j} is to t. Nevertheless, a system that does not produce new data for some time does not mean its data are less relevant than a system that produces data more recently. It is possible that the former is put to rest and stops producing data, while the latter continues working (Figure 6.3). In this case, the recency of the data on these two

systems does not depend on a global timestamp, but it rather depends on the timestamp *at which each system produces the latest data point.*

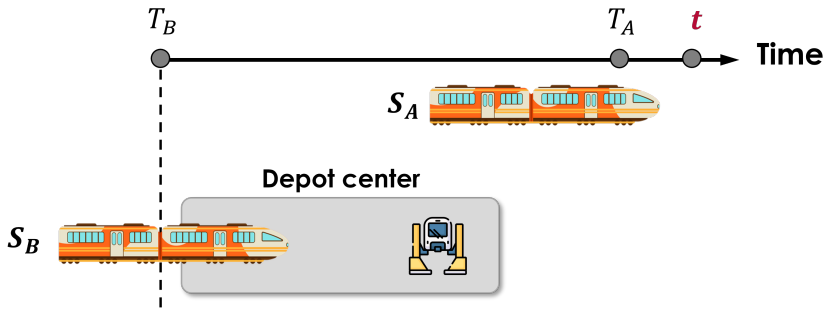


FIGURE 6.3: The system S_B enters a depot center at T_B and stops producing data, while the system S_A continues operating until T_A. The most recent data of S_B are not indexed at t but at T_B.

Therefore, a global timestamp t cannot be used to decay the data points from different systems in mc_i, but t must vary by system. As a result, it leads to a different definition of a cluster mc_i. As a reminder, $\mathcal{S} = \{S_1, \dots, S_M\}$ is the set of M systems being monitored. Let $orig(p)$ be a function that maps a point p to the system that produces it. Let us define, for a cluster mc_i:

$$\mathcal{S}_i = \{ S_m \in \mathcal{S} \mid \exists p \in mc_i,\ orig(p) = S_m \}$$

the set of M_i systems, such that each system $S_{i_m} \in \mathcal{S}_i$ produces at least one data point in mc_i ($S_{i_m} \in \mathcal{S}_i$, $M_i \leqslant M$, and $\mathcal{S}_i \subseteq \mathcal{S}$). We denote $\mathcal{P}_{i_m}$ the set of n_{i_m} data points produced by a system S_{i_m} in $\mathcal{S}_i$.

$$\mathcal{P}_{i_m} = \{ p \in mc_i \mid orig(p) = S_{i_m} \}$$

Let $mc_{i_m} = \{ w_{i_m}, LS_{i_m}, SS_{i_m} \}$ be a tuple of cluster features that summarize the data points *produced by a system* $S_{i_m} \in \mathcal{S}_i$ in mc_i, such that w_{i_m} is the weight by S_{i_m} (6.16), LS_{i_m} the linear sum by S_{i_m} (6.17), and SS_{i_m} the squared sum by S_{i_m} (6.18).

$$w_{i_m} = \sum_{j=1}^{n_{i_m}} f(t_m - T_{i_m}^j) \tag{6.16}$$

$$LS_{i_m} = \sum_{j=1}^{n_{i_m}} f(t_m - T_{i_m}^j)\, p_{i_m}^j \tag{6.17}$$

$$SS_{i_m} = \sum_{j=1}^{n_{i_m}} f(t_m - T_{i_m}^j)\, (p_{i_m}^j)^2 \tag{6.18}$$

For the decay, t_m is the timestamp of the latest point produced by S_{i_m} on the stream $D(T)$. Because the latest timestamp of a system t_m is not local to any cluster, the index i is omitted. Figure 6.4 illustrates an example of the data from different systems assigned to the clusters at T, where each point is marked with its timestamp. The latest timestamp t_m of a system S_m is not the same as the global latest timestamp t of the stream. In this example, we have $t = 26$, $t_A = 7$, $t_B = 7$, $t_C = 12$.

Having one tuple of cluster features for each S_{i_m} in mc_i, the cluster features of mc_i is the sum of all such tuples by S_i.

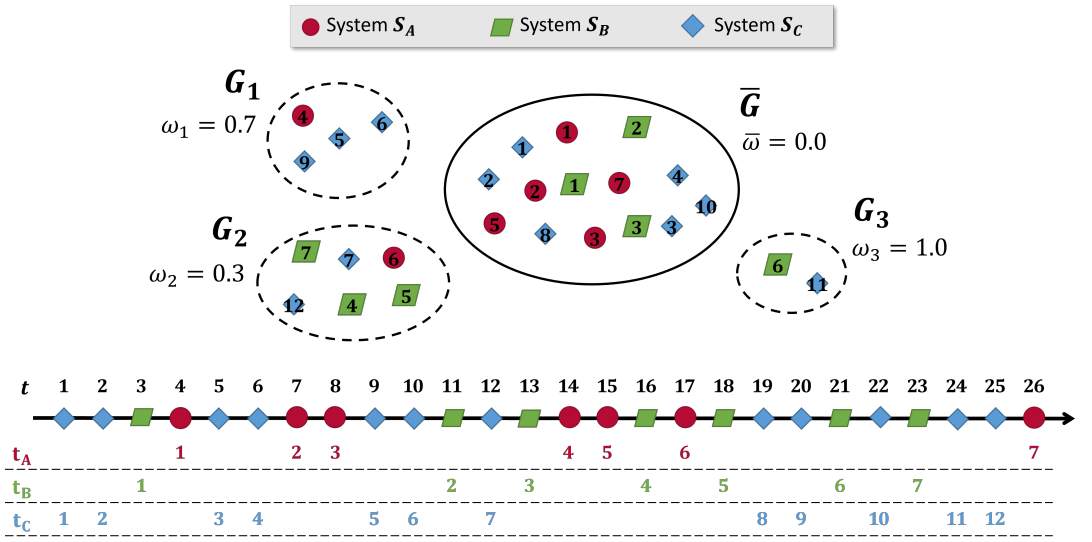


FIGURE 6.4: The data from three systems S_A, S_B, and S_C are assigned to four clusters. Each data point is marked with its creation timestamp. The timestamp of the latest data point by a system is local to that system only ($t_A = 7, t_B = 7, t_C = 12$).

$$mc_i = \{w, LS, SS\} = \left\{ \sum_{S_{i_m}}^{S_i} w_{i_m}, \sum_{S_{i_m}}^{S_i} LS_{i_m}, \sum_{S_{i_m}}^{S_i} SS_{i_m} \right\} \tag{6.19}$$

This definition still enables incremental update of a cluster. Let us consider two cases of update: a new point p is added to a cluster mc_i, and no new point is added so mc_i decays after a while.

If a point p is added to mc_i at the time T, p may or may not come from a system that already has data in mc_i. We distinguish two sub-cases, where $orig(p) \in \mathcal{S}_i$ and $orig(p) \notin \mathcal{S}_i$.

- Let $orig(p) = S_{i_m}$ be a system in $\mathcal{S}_i$. When S_{i_m} creates p, the reference timestamp t_m of S_{i_m} is updated and is the same as the timestamp T of p. Therefore, the decay factor when p is merged in mc_i becomes $f(t_m - T) = f(0) = 1$, and mc_{i_m} thus becomes:

  $$\{w_{i_m} + 1, LS_{i_m} + p, SS_{i_m} + p^2\}$$

  Following (6.19), mc_i after merging p is:

  $$\{w + 1, LS + p, SS + p^2\} \tag{6.20}$$

- If $orig(p) = S_l \notin \mathcal{S}$, the new cluster features of S_l is:

  $$mc_{i_l} = \{w_{i_l}, LS_{i_l}, SS_{i_l}\} = \{1, p, p^2\}$$

  Following (6.19), mc_i after merging p (and S_l) is:

  $$\left\{ \sum_{S_{i_m}}^{S_i} w_{i_m} + w_{i_l}, \sum_{S_{i_m}}^{S_i} LS_{i_m} + LS_{i_l}, \sum_{S_{i_m}}^{S_i} SS_{i_m} + SS_{i_l} \right\} = \{w + 1, LS + p, SS + p^2\} \tag{6.21}$$

If no new point is added in mc_i for a while, mc_i decays, leading to the subsequent decay of its cluster features. For each system $S_{i_m} \in \mathcal{S}_i$, let $\delta t_{i_m} = t_m - T_{i_m}$ be the time interval from the latest point by S_{i_m}

on the stream (t_m) to the latest point of S_{i_m} in a specific cluster mc_i (T_{i_m}). To illustrate how to compute δt_{i_m}, consider the scenario in Figure 6.4. Suppose that the cluster features in mc_1 decay at the universal time $t = 26$, we have $\delta t_{1_A} = t_A - T_{1_A} = 7 - 4 = 3$ for the system S_A, and $\delta t_{1_C} = t_C - T_{1_C} = 12 - 9 = 3$ for the system S_C. The global timestamp $t = 26$ does not play any role in the decay of the cluster features of a specific system.

To decay mc_i, we simply sum the decayed cluster features of each system in mc_i. For example, the decayed linear sum LS' of mc_i becomes:

$$\begin{aligned} LS' = \sum_{S_{i_m}}^{S_i} LS'_{i_m} &= \sum_{S_{i_m}}^{S_i} \sum_{j=1}^{n_{i_m}} f(\delta t_m + t_m - T_{i_m}^j) p_{i_m}^j \\ &= \sum_{S_{i_m}}^{S_i} \sum_{j=1}^{n_{i_m}} 2^{-\lambda(\delta t_m + t_m - T_{i_m}^j)} p_{i_m}^j \\ &= \sum_{S_{i_m}}^{S_i} \sum_{j=1}^{n_{i_m}} 2^{-\lambda \delta t_m} 2^{-\lambda(t_m - T_{i_m}^j)} p_{i_m}^j \\ &= \sum_{S_{i_m}}^{S_i} 2^{-\lambda \delta t_m} \sum_{j=1}^{n_{i_m}} 2^{-\lambda(t_m - T_{i_m}^j)} p_{i_m}^j \\ &= \sum_{S_{i_m}}^{S_i} 2^{-\lambda \delta t_m} LS_{i_m} \end{aligned}$$

The squared sum SS' and weight w' are decayed similarly. Therefore, mc_i after δt is:

$$\left\{ \sum_{S_{i_m}}^{S_i} 2^{-\lambda \delta t_m} w_{i_m}, \sum_{S_{i_m}}^{S_i} 2^{-\lambda \delta t_m} LS_{i_m}, \sum_{S_{i_m}}^{S_i} 2^{-\lambda \delta t_m} SS_{i_m} \right\} \tag{6.22}$$

It suffices to multiply the features $\{ w_{i_m}, LS_{i_m}, SS_{i_m} \}$ of each system S_{i_m} in mc_i to $2^{-\lambda \delta t_m}$ and sum the decayed features of all the systems in mc_i to decay mc_i incrementally.

In conclusion, even if the cluster features are defined by system in a cluster, it is possible to update a cluster on-the-fly efficiently via (6.20), (6.21), and (6.22).

3.2.3 Full algorithm

The algorithm of CheMoc is sketched in Algorithm 6.2. It works on a stream $D(T)$ and requires as input a buffer size W to warm start CheMoc, a set of hyperparameters θ for the online clustering algorithm (denoted M), an alert threshold γ to issue maintenance alerts on the systems. θ depends on the chosen algorithm clustering algorithm.

First, M is initialized on θ. An empty buffer WS is allocated to store the data for warm-starting CheMoc. The warm-start process is done exactly as described in [44]. After the warm-start (line 5–8), CheMoc updates the clusters on each new data point (line 10). Then, CheMoc reassesses the clusters and recomputes the health score of the systems for every interval T_u[3] During this reassessment phase (line 12), ϵ is re-estimated using (6.15), a new reference cluster $\overline{G}(T)$ (the one with the most data points)

[3]This can be done at every new data point or by micro-batch. Updating at every point follows the principle of real-time monitoring, but it can cause a data communication bottleneck if the data must be persisted on hard disk. T_u must be set depending on the application.

Algorithm 6.2: CheMoc

Data: A stream of data points $D(T)$
Input: Buffer size W, clustering hyperparameters θ, threshold $0 < \gamma < 1$
Output: An alert if $H_{S_m}(T) > \gamma$ for a system S_m

```
1  M.init(θ)                                              /* init DenStream */
2  WS ← ∅                                                 /* init warm start buffer */
3  while stream D(T) is active do
4      x ← new data point from D(T)
5      if not yet warm started then                       /* warm start */
6          WS ← WS ∪ {x}
7          if |WS| = W then M.warmStart(WS)
8          else continue
9      else
10         M.process(x)                                   /* update G(T) */
11         if is update time after an interval T_u then
12             M.reassessClusters()
13             M.updateHealth()
14             foreach system S_m ∈ S do
15                 if H_{S_m}(T) > γ then Issue an alert on S_m
```

is re-elected. Once $\overline{G}(T)$ is identified, the cluster weight $\omega_k(T)$ of each cluster $G_k(T)$ is recomputed using (6.1) and (6.2). Then, the health score of the systems are updated using (6.4) and (6.5) (line 13). If S_m has a new health score that exceeds γ, an alert is issued (line 14–15).

4 Experimental results

The hypotheses addressed by CheMoc are:

- ($\mathbf{H}_\mathbf{d}^1$) The clusters represent the health profiles of the fleet.
- ($\mathbf{H}_\mathbf{d}^2$) Computing the health score by considering the data a system creates in the health profiles results in an accurate health detection, within an observable perimeter from the data.
- ($\mathbf{H}_\mathbf{d}^3$) Online clustering reaches convergence earlier than offline clustering, if no drift occurs.
- ($\mathbf{H}_\mathbf{d}^4$) Online clustering keeps track of the temporal evolution of the health profiles via cluster updates, while offline clustering cannot.

First, we explain how the experiments are set up using R2N data (Section 4.1). Then, we discuss how we choose the hyperparameters (Section 4.2). Finally, we show the experimental results of CheMoc and verify whether CheMoc validates the aforementioned hypotheses.

4.1 Experiment setup

To evaluate CheMoc, we only use the data of the R2N fleet, and we use the expert indicators instead of the LSTM-AE features as a starting point, for the following reasons.

The foremost reason is that there is no readily available ground-truth to systematically compare the health assessment results and the clusters of CheMoc. As a result, every time we conduct an experiment, a domain expert must manually verify all the clusters produced by CheMoc. We adopt a micro-batching approach to feed CheMoc the data of one week at a time. After CheMoc processes one weekly batch, we log the cluster footprints in the database (MongoDB) and make one visualization

to asses the clusters in this batch. An example cropped from one cluster visualization is shown in Figure 6.5 and displays the profiles and envelops of the cycles each cluster contains[4]. Please note that this visualization is for one week only. Each experiment of CheMoc is run on many weeks, therefore after each experiment the domain expert must re-check the clusters across multiple batches, which is a tedious and time-consuming task.

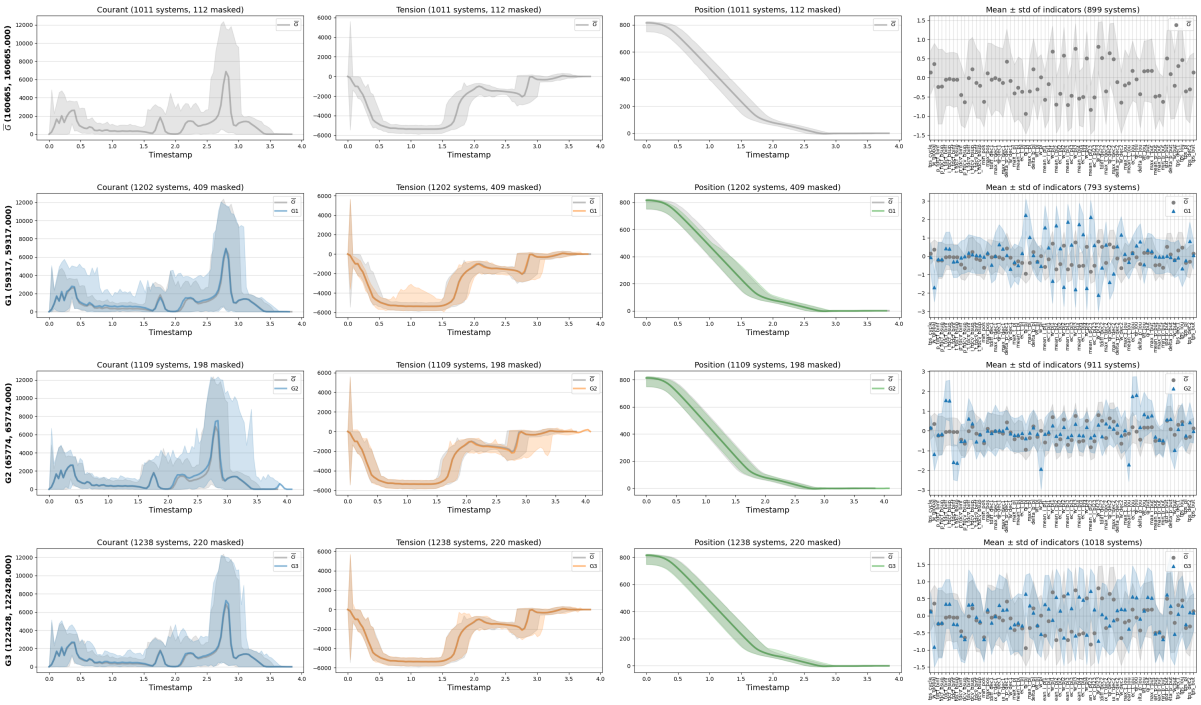


FIGURE 6.5: Visualization of some clusters obtained in one weekly batch. The columns from left to right: the motor current intensity, the motor voltage intensity, the motor position intensity, the mean and standard deviation of the expert indicators. The solid lines are the profiles and the colored regions the envelopes of a variable.

We have a database of alerts associated to the R2N fleet, but it is incomplete and include only three months of alerts, starting from September 2021. Meanwhile, we test CheMoc on the data of the full year 2021. If we test CheMoc on the same three months, CheMoc will capture clusters that are very different to those discovered from one year of data, which inevitably affects the health computation of the systems. Furthermore, an alert in this database does not map directly to the threshold-based alert of CheMoc. The former defines several levels of severity, which, unfortunately, is more sophisticated than the simplicity of CheMoc's alerting mechanism. Given these obstacles, we decide to limit the experiment to one data set, and to one type of cycle (door closing).

Although in Chapter 5 we have demonstrated that the LSTM-AE features outperform the expert indicators in terms of information preservation via a cycle-feature ranking, we use the expert indicators to test CheMoc to ensure the cluster interpretability when the domain expert studies the results. The expert indicators convey a clear, explicit meaning, whereas the feature vectors returned by the LSTM-AE are non-interpretable and not human-understandable. A workaround is to visualize a cluster by mapping the feature vectors it contains to the original cycles. We will consider this in future works. For now, it is more practical to visualize both the cycles and the expert indicators that form a cluster.

[4]Due to the size of each figure we cannot show the full visualization with all the clusters here. The full figures are found at this link.

Once we can find a more robust evaluation framework and/or build a reliable ground-truth, manually studying the clusters will no longer be necessary and we will expand the tests on both NAT and R2N data sets. Also, we will carry out the experiments on the LSTM-AE features as well. The clusters obtained with the expert indicators will then become the baseline.

Finally, we evaluate CheMoc on the data of R2N fleet, consisting of 1409 systems ($|\mathcal{S}| = 1409$), collected during the year 2021. There are 1074279 data points in total with 192 numerical and categorical features hand-crafted by a domain expert. We remove all categorical features because DenStream is not optimized to process categorical values. We also remove features with missing values. Finally, we obtain a set of 70 non-trivial numerical features.

We feed these data to CheMoc continuously in the chronological order to simulate a stream. Instead of performing the health reassessment at every data point (line 11–15 in Algorithm 6.2), we adopt a *micro-batching* approach: CheMoc receives a batch of data of one week at a time, updates the clusters on every data point in this one-week batch, reassesses the clusters, and updates the health scores at the end of the batch. Because we log the data of CheMoc in a database (MongoDB) for post-analysis, micro-batching reduces the bottleneck of the database writing procedure.

4.2 Hyperparameter tuning

First and foremost, we use the number of cycles as the time unit in CheMoc. Cao, Ester, Qian, and Zhou [45] leaves time unit as an application-dependent choice. In the railway scenario, the number of cycles are more relevant than the real-world time as in minutes or hours. A system only degrades if it is in operation and continuously produces new cycles. In contrast, if the system is idle and does not create any data, its degradation does not evolve even when real-world time elapses. Therefore, all notions of time in CheMoc are in the unit of number of cycles.

Because CheMoc uses DenStream as its core clustering, four crucial hyperparameters are the desired density ϵ, the decay factor λ dictating how fast a data point is forgotten, the outlierness threshold β to distinguish a PMC to an OMC, and the minimum weight μ for a cluster to be considered dense. The hyperparameters of CheMoc are set as in Table 6.2.

Table 6.2: Hyperparameters (Param) of CheMoc

Param	Value	Meaning
ϵ	15	The initial density for the warm-start
λ	0.06	Forgetting a cycle after the 100 next cycles
β	0.4	Threshold on the outlierness of a cluster
μ	5	Required number of points for a system in a cluster

To select the initial value of ϵ, we compute the mean and standard deviation of the pairwise distances on a random sample of the data and set $\epsilon_0 = \lceil \mu_{dist} + \sigma_{dist} \rceil$. Then, ϵ is continuously re-estimated using (6.15).

λ is computed such that a cycle is forgotten after a window of 100 next cycles, i.e., $\delta t = 100$, as suggested by a domain expert. We consider that a cycle is forgotten if its weight $\beta\mu = 0.01$; the value 0.01 is our own choice. Hence, we have $2^{-\lambda\delta t} = 2^{-100\lambda} = 0.01 \Rightarrow \lambda \approx 0.06$.

As β dictates whether a cluster is dense enough to become a PMC, we select the value 0.4 to be on the tolerant side when promoting an OMC to a PMC. Based on the weight threshold $\beta\mu$, the higher the value of β is, the denser a cluster must become to be considered a PMC.

For μ, instead of considering it as the minimum weight of a dense cluster, we define μ as the least number of points a system must have in a cluster to be allowed to update the cluster features. It is to eliminate noises from systems that have only one or two points in a cluster. Yet, even if a system does not reach μ, we still store its cluster features $\{ w_{i_m}, LS_{i_m}, SS_{i_m} \}$ in the cluster, but we do not add them to the final features of the cluster $\{ w, LS, SS \}$. It implies that a system with very few points is saved and masked in the cluster containing it, until the number of data it creates exceeds μ.

The data we stored in the local statistic matrices $stat_k(T)$ and $\overline{stat}(T)$ are the mean of the data from all the systems in the cluster, which is simply its centroid.

CheMoc is warm-started on the first batch of data that has 1589 data points. The warm-start finishes after three seconds.

4.3 Evaluation of CheMoc

To validate the hypotheses, we expect CheMoc to produce clusters relevant to the health profiles of the systems (H_d^1), to produce the health scores consistent with the data trajectory of the systems across the clusters (H_d^2), to reach a stable set of clusters earlier than an offline clustering algorithm (H_d^3), and to evolve the clusters according to the stream (H_d^4). Moreover, because CheMoc works on a fast data stream, it must be efficient, both in terms of execution time and memory usage.

4.3.1 Cluster analysis

CheMoc captures 12 clusters, among which 11 are PMCs ($\overline{G}$, G_1–G_{10}) and one is an OMC (G_{11}). We use PCA to embed the entirety of the data on a 3D space. The data appear to exist in a large cloud (Figure 6.6a), with significant outliers scattering on the border, but they do not exhibit intricate topology. On these data, CheMoc discovers a set of tight-knit clusters (Figure 6.6c). We visualize the clusters obtained after running CheMoc on the data of 2021. The cluster distribution implies that when a system degrades, its health deviates gradually from the normal state but is not completely separated. The temporal trajectory of each cluster shows that the clusters reach a stable state after the first few batches and move slowly afterwards (Figure 6.6b).

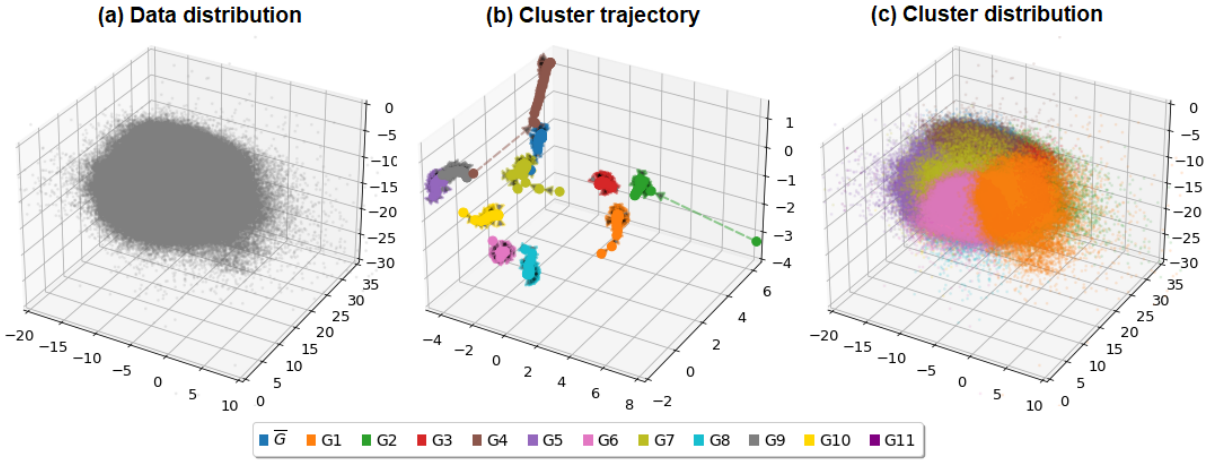


FIGURE 6.6: (a) The distribution of one-year data; (b) The movement of the clusters over batches; (c) The distribution of the clusters on one-year data.

A small ϵ is therefore suitable to detect the gradual trajectory of a system's data via small clusters that change their position over time. Dynamically adjusting ϵ makes it increasingly smaller, hinting

that the clusters become denser over time (Figure 6.7b). Their radii concentrate around 0.5 to 1.5 and plateau after the first few batches. From this observation, the clusters seem to have a stable structure.

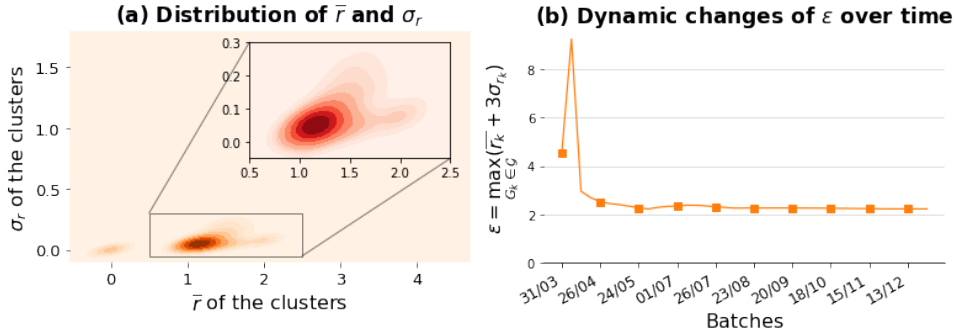


FIGURE 6.7: ϵ dynamically adjusted based on the mean $\bar{r}$ and standard deviation σ_r of the clusters' radii (the initial value $\epsilon_0 = 15$ was adjusted is not included)

Then, we look into the anomaly degrees of the clusters and how they evolve temporally. The degrees remain stable without significant fluctuations and tend to increase slightly over time (Figure 6.8a). Looking at the Euclidean distances between a cluster's centroid to the reference's centroid (Figure 6.8b), we see that G_1 is the most anomalous cluster in $\mathcal{G}$, because it is always the farthest from $\overline{G}$, and the second-most anomalous cluster is G_7. This checks out with the observation that the anomaly degree of G_1 and G_7 is the most and second-most highest among all the clusters.

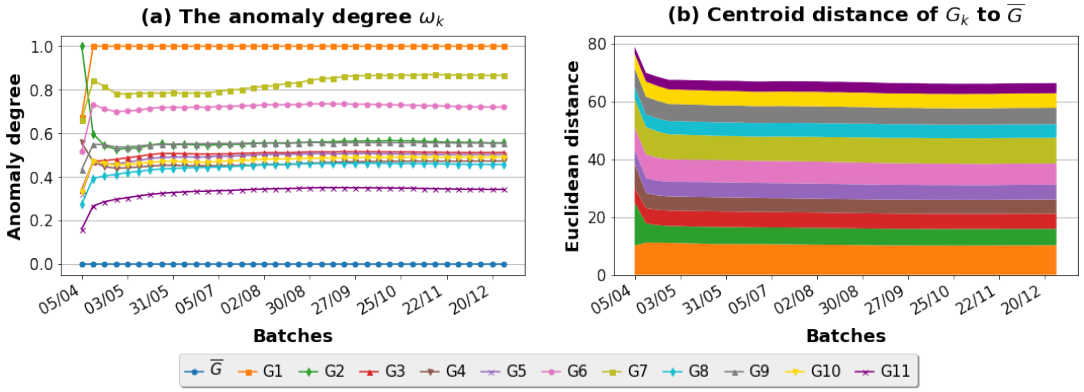


FIGURE 6.8: The anomaly degrees of the clusters (a) and the distance between a cluster's centroid to the reference's centroid (b).

We also ask for the domain expert's opinion on the quality of the clusters by visualizing the cycles and their indicator vectors each cluster contains. With CheMoc set up on the hyperparameters shown in Table 6.2, the domain expert positively confirms that CheMoc has converged to a set of relevant health profiles, and that the clusters reflect the underlying physics of known anomalies on the PASs. Therefore, the hypothesis H_d^1 is validated.

Yet, we admit that these health profiles are far from perfect, as CheMoc does not allow merging or splitting clusters, which could potentially refine the health profiles. These resulting clusters are as best as one could get purely from the data without an expert's guidance. In the future, we plan to incorporate the expert's feedback in CheMoc to enable a more sophisticated cluster fine-tuning.

4.3.2 Health evolution

The interest of finding clusters is to track the evolving health of the monitored systems. The health score of a given system at any moment is calculated using (6.5). Among 1409 systems, we pick one whose health shows changes for analysis. We denote the system S_m. We test both the static version (6.3) and decaying version (6.4) of the anomaly scores.

Figure 6.9 plots the health score H_{S_m} of S_m updated at each batch against the amount and percentage of data from S_m in each cluster. Since S_m mostly produces data in anomalous profiles, most notably in G_6, H_{S_m} is relatively high and is increasing as S_m continues to produce data in anomalous profiles.

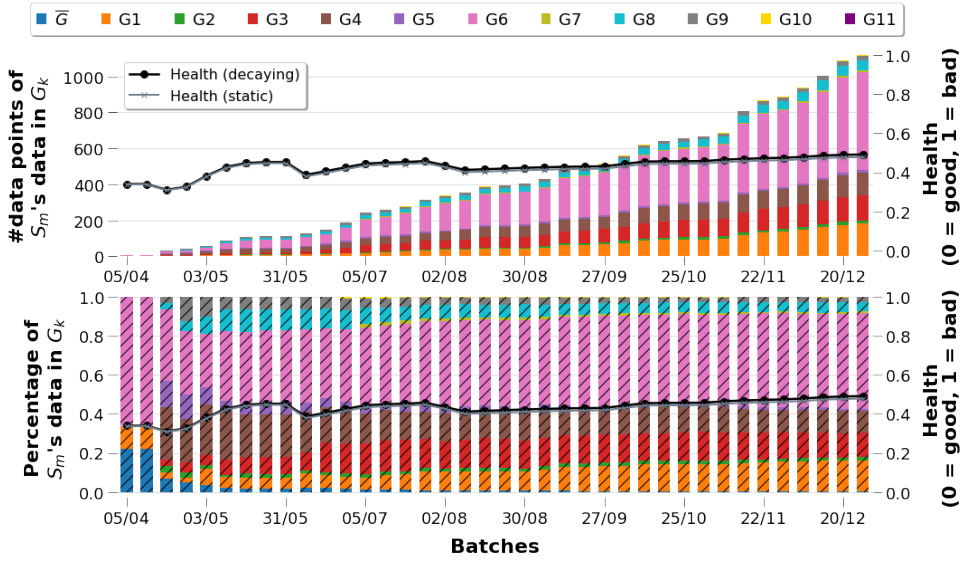


FIGURE 6.9: Health evolution of the system S_m: plotted against the amount (top) and percentage (bottom) of data points S_m has in each cluster.

Surprisingly, the weighted (decaying) and unweighted (static) scores are overlapping, meaning there is little to no difference between the two versions. For this particular system, because it primarily produces data in anomalous clusters, the term $|G_k^{S_m}|$ in (6.3) and $\sum_i^{G_k^{S_m}} f(T_m - T_i)$ in (6.4) dominate over $|\overline{G}^{S_m}|$ and $\sum_j^{\overline{G}^{S_m}} f(T_m - T_j)$, respectively. Hence, the anomaly scores of both versions are very close to 1.0, and consequently it results in almost the same health scores (6.5). Another plausible reason is that we did not choose an adequate decay factor λ, so the data are not forgotten quickly enough.

To examine the type of anomalies under which S_m might have been, we select one batch and visualize the data in the clusters that contribute to H_{S_m}. We choose the batch just before 03/05/2021, when H_{S_m} starts rising. In this batch, S_m appears in multiple clusters, so we visualize the most notable ones, namely G_1, G_3, G_4, and G_6 (Figure 6.10); these four clusters do not appear dominant for S_m in this particular batch, but will become so over time. We compute the profile and envelope of these clusters from the cycles it contains, on three most relevant measurements: the current, the voltage, and the position of the electric motor in the PAS. We plot both the profile of a cluster G_k and that of the reference cluster $\overline{G}$ for comparison.

Figure 6.10 shows that G_1, G_3, G_4, and G_4 display behaviors different from that of the normal profile $\overline{G}$ (light gray), especially G_4. The presence of S_m's data in these clusters means that S_m has started to deviate from the good health and, consequently, has its score H_{S_m} increased.

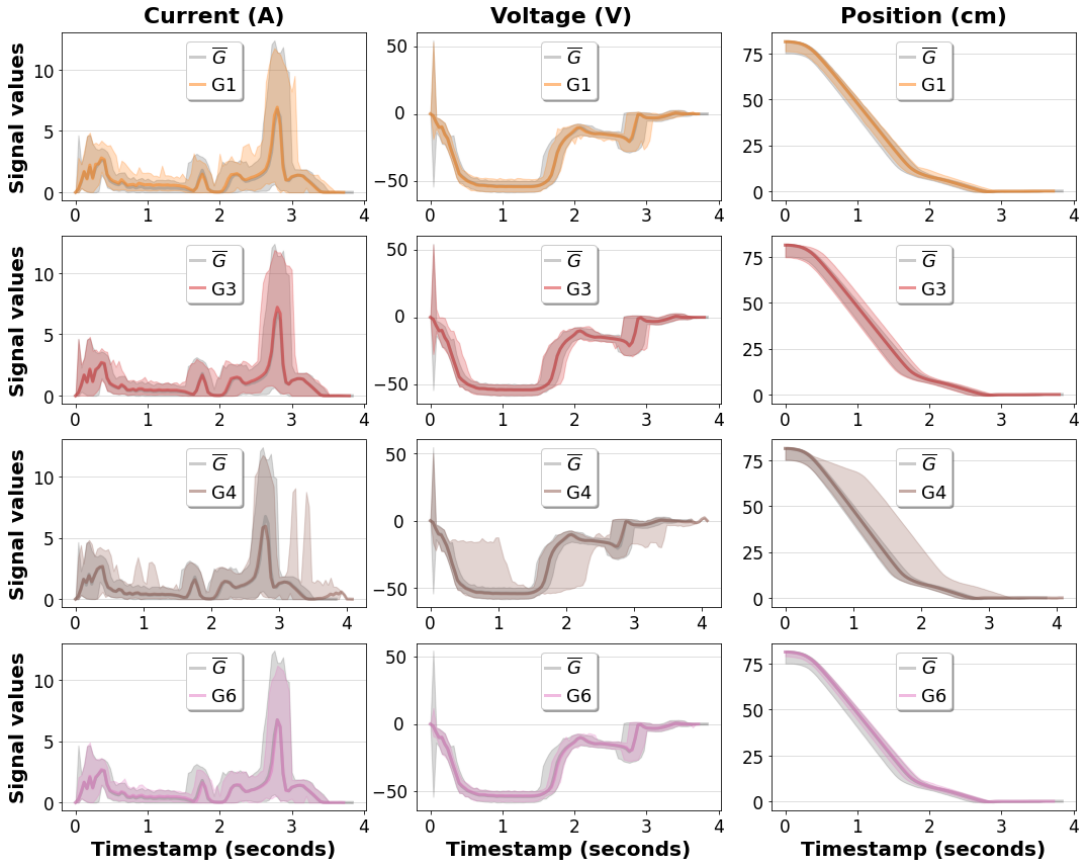


FIGURE 6.10: Visualization of the profiles of G_1, G_3, G_4, and G_4 via the current, voltage, and position measurements. Thick lines are the profiles, and colored regions are the envelopes of the variables.

We can therefore validate the hypothesis H_d^2, given that the health evolution is in accordance with the trajectory of a system's data in the health profiles. Since the clusters are confirmed by the domain expert to represent the underlying physics of known anomalies, our formula used to estimate the health score are able to assess a system's condition at any given moment.

4.3.3 Reactivity and continuity of online clustering

The reactivity of CheMoc is assessed by the time elapsed since CheMoc receives the first data point until it reaches a stable set of clusters. A stable cluster is one that does not undergo significant changes on the stream in the absence of data drifting. To quantify the stability of the clusters, we use a cluster validity index (CVI) and observe its evolution over time to determine CheMoc's convergence. A CVI evaluates a clustering result on a data set, either by matching it to a ground-truth (ideal partitioning), or by assessing the cohesion and separation between clusters. The former is known as external CVI, and the latter internal CVI. Due to the lack of ground-truth clusters, we work with internal CVIs only.

Among the existing internal CVIs, we pick the Davies-Bouldin (DB) index [55] and Xie-Beni (XB) index [239]. The DB index (6.23) estimates the cluster cohesion based on the distance between centroids and between each point to its centroid. The XB index calculates the ratio of compactness to separation (6.24). Smaller values of DB and XB indicate better clustering. We denote K the number of clusters, $\bar{c}_k$ the centroid of a cluster G_k, $d(x, y)$ the Euclidean distance between x and y.

$$DB = \frac{1}{K} \sum_{G_k \in \mathcal{G}} \max_{G_l \in \mathcal{G} \setminus G_k} \frac{S(G_k) + S(G_l)}{d(\bar{c}_k, \bar{c}_l)} \qquad \text{where} \qquad S(G_k) = \frac{1}{|G_k|} \sum_{i \in G_k} d(x_i, \bar{c}_k) \tag{6.23}$$

$$XB = \frac{WGSS/N}{\min\limits_{k \neq l} \| \bar{c}_k - \bar{c}_l \|_2^2} \qquad \text{where} \qquad WGSS = \sum_{k=1}^{K} \sum_{i \in G_k} \| \bar{c}_k - x_i \|_2^2 \tag{6.24}$$

Because CheMoc uses DenStream as its core clustering and DenStream is based on DBSCAN [65], we compare the reactivity of CheMoc to that of DBSCAN. We set ϵ of DBSCAN to 3.0 to make it consistent to the dynamic density ϵ of CheMoc (Figure 6.7). CheMoc is more reactive than DBSCAN if the DB and XB scores of CheMoc's clusters become stable earlier than those of DBSCAN.

We collect the DB and XB scores of each method on two separate approaches: on the data as a stream (all data accumulated from $t = 0$), and on the data of weekly batches. We name the four approaches `CheMoc-inc`, `CheMoc-batch`, `DBSCAN-inc`, and `DBSCAN-batch`. The DB and XB scores at a time T of each approach are calculated as follows.

- `CheMoc-inc` and `DBSCAN-inc`: the scores are calculated on all the clusters and data from the beginning until T.
- `CheMoc-batch`: the scores are calculated on all the clusters but considering the data in the current batch only.
- `DBSCAN-batch`: the scores are calculated on the clusters obtained in one batch.

Figure 6.11 shows the scores alongside the number of clusters of each approach. The number of clusters of `CheMoc-batch` is the number of *new* clusters that appears in a batch with respect to the previous one.

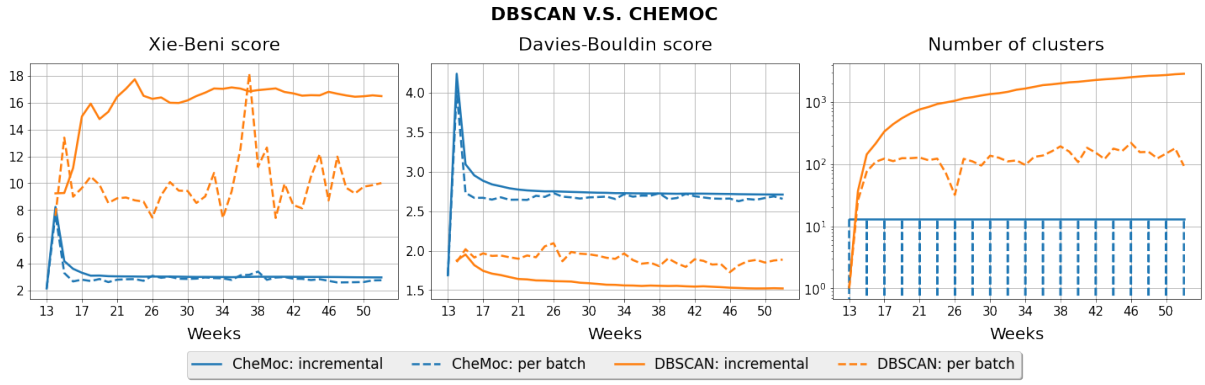


FIGURE 6.11: Left to right: XB scores, DB scores, and number of clusters of CheMoc versus DBSCAN

On the XB scores, `DBSCAN-inc` fluctuates strongly, implying that clusters obtained separately at each batch do not have a strong connection from one batch to another. Meanwhile, the XB scores of both version of CheMoc plateau quickly and are lower than those of DBSCAN.

In the contrary, the DB scores of DBSCAN are lower than those of CheMoc, but the number of clusters of `DBSCAN-inc` is particularly high and rises up to 2000 clusters on the full data. This may explain the low values of the DB scores: because the data are over-partitioned, each cluster has very few data points, which makes the clusters denser and the centroids close to each other, consequently

lowering the DB scores. The DB scores of DBSCAN-inc converge after the first five weeks, but the DB scores of DBSCAN-batch and the XB scores of DBSCAN-batch and DBSCAN-inc do not converge at all.

Both scores have a consistent tendency for the clusters of CheMoc. They increase sharply after the first warm-start batch, but become stable quickly after the first few weeks. The scores of CheMoc-inc are smoother than those of CheMoc-batch. Despite producing substantially fewer clusters than DBSCAN, CheMoc results in CVI scores that are quite close to those of DBSCAN, which hint at a higher quality of the clusters found by CheMoc.

Therefore, CheMoc is at least equally reactive to a traditional batch clustering. Even if DBSCAN-inc seems to be competitive to or better than CheMoc, DBSCAN-inc must accumulate all data from the start of the stream to perform a clustering, which does not scale in the long run because a stream is infinite, whereas CheMoc is designed to learn incrementally. The hypothesis H_d^3 is partly[5] validated.

This experiment also shows that CheMoc ensures the continuity of the cluster evolution by maintaining the clusters to novelties on the stream, instead of re-creating the clusters at every batch like DBSCAN-batch. The number of clusters by CheMoc is stable whereas that of DBSCAN-batch is erratic and changes at every batch, without any indication that a cluster grows, shrinks, or disappears. Hence, the hypothesis H_d^4 on the continuity of CheMoc is validated.

4.3.4 Time and memory efficiency

To monitor a large number of systems, CheMoc must be computationally efficient: updating the clusters must be done quickly and consumes a reasonable amount of memory. As CheMoc is designed to run on a server, we tolerate a moderate use of computational resource.

Because CheMoc uses DenStream as its inner clustering algorithm, it is able to quickly update the clusters. Further analysis on the performance of DenStream can be found in [44].

The time complexity of CheMoc is estimated on a single data point x arriving at T and is the complexity of its three main operations (Algorithm 6.2): updating the clusters $\mathcal{G}(T)$ on x, reassessing the clusters, recomputing the health score of all systems. Considering a worst-case scenario, we assume that the last two operations are carried out for every new data point, e.g., $T_u = 1$. Please note that we use the decaying version of the anomaly score in this estimation.

Let us denote K, N_p, N_o the number of clusters, of PMCs, and of OMCs maintained thus far such that $N_p + N_o = K$, $N = |D(T)|$ the total data points from the stream until T, $M = |\mathcal{S}|$ the number of systems in the fleet $\mathcal{S}$, N_{stat} and P respectively the number of rows and columns of a statistic matrix. The worst-case time complexity of each operation in CheMoc is:

- Updating $\mathcal{G}(T)$ on x: $O(\max(N_p, N_o)) = O(K)$[6].
- Cluster reassessment: $O(KN_{stat}P)$.
- Health score update: $O(MKN)$.

The overall worst-case time complexity of updating CheMoc on a single data point x at a time T is:

$$O(K + KN_{stat}P + MKN) = O(KN_{stat}P + KNM)$$

[5] It is validated only partly because we cannot prove that CheMoc outperforms DBSCAN on both CVIs.

[6] In the worst-case scenario, we may have all PMCs or all OMCs in $\mathcal{G}(T)$ and must iterate over all the clusters to merge a data point to an existing one.

which is non-scalable because it involves N, due to (6.4) that requires computing $\sum_i^{G_k^{S_m}} f(T_m - T_i)$ and $\sum_i^{\overline{G}^{S_m}} f(T_m - T_i)$, both of which in the worst case become N because a cluster may contain all data from the stream. If we use the unweighted version of the anomaly score (6.3), the time complexity of health recomputation becomes $O(MK)$ because it only needs to store and access the number of points by system in a cluster. There is an option to reduce $O(MKN)$ to $O(MK)$ as well for the weighted version (6.4), but it requires communicating with the maintenance workers which is not yet implemented at this stage. It is part of our future works and we will discuss it at the end of this section.

Figure 6.12 shows the processing time and memory of CheMoc on each batch. The processing time is consistent with the batch size: an update takes a constant time processing each data point. The total execution time of CheMoc is almost entirely dominated by that of DenStream. A small gap exists due to the time CheMoc takes to update the health scores of the systems. The only offline process is the warm-start on the first batch and takes negligible time to finish (three seconds).

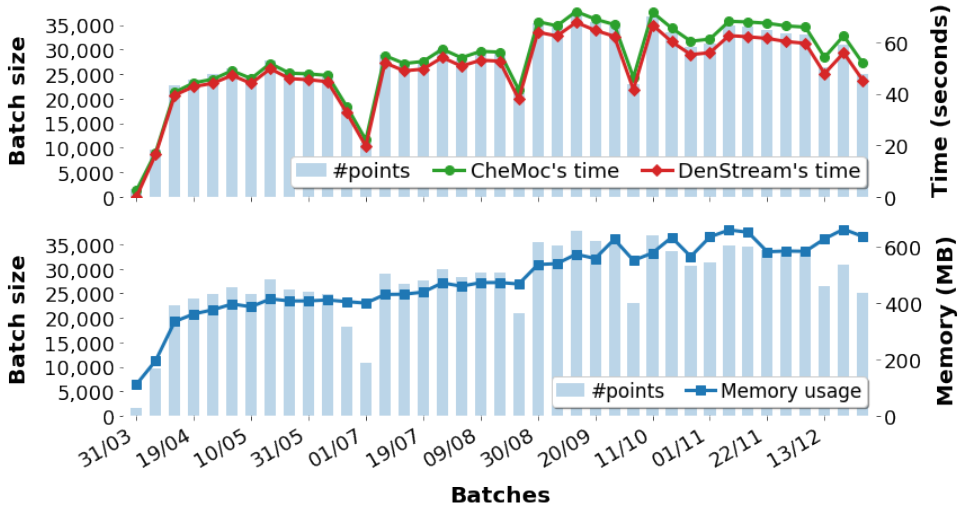


FIGURE 6.12: Processing time and memory usage of CheMoc

The memory usage of CheMoc tends to increase slightly. When data from unseen systems arrive, CheMoc creates and stores the cluster features of new systems in the clusters, thus expanding the memory usage over time, but not infinitely because the number of systems is finite. Processing the data of an entire year from a fleet of 1409 systems requires only 600 MB in total.

Therefore, if we adapt a micro-batching approach ($T_u \neq 1$), CheMoc is efficient in terms of execution time and memory usage and can be used to monitor a fleet of systems generating a fast data stream.

4.4 Limits of CheMoc and potential improvements

Having studied CheMoc on the R2N data set, we have identified its three most important drawbacks.

4.4.1 Storing individual relevance weights

The first and foremost drawback of CheMoc is that it stores the data assignment in the clusters to quickly compute the decaying anomaly scores (6.4). In the simple version without decaying (6.3), storing the number of points per system is sufficient to compute the score via $|G_k^{S_m}|$ and $|\overline{G}^{S_m}|$. However, the decaying version (6.4) needs to access the relevance weight of individual data points. Although we

store only the weight value of each point (instead of storing its entire data), this solution will not work in the long run as data arrive infinitely.

It is possible to transform the terms in (6.4) to avoid storing individual relevance weights. Let $dG_k^{S_m} = \sum_i^{|G_k^{S_m}|} f(t_m - T_i)$ be the sum of the relevance weights of all the points from S_m in G_k, where t_m is the latest timestamp of S_m on $D(T)$ and T_i is the timestamp of the point p_i of S_m in G_k.

$$\begin{aligned} dG_k^{S_m} &= \sum_i^{G_k^{S_m}} f(t_m - T_i) = \sum_i^{G_k^{S_m}} 2^{-\lambda(t_m - T_i)} \\ &= \sum_i^{G_k^{S_m}} 2^{-\lambda t_m + \lambda T_i} = \sum_i^{G_k^{S_m}} 2^{-\lambda t_m} 2^{\lambda T_i} \\ &= 2^{-\lambda t_m} \sum_i^{G_k^{S_m}} 2^{\lambda T_i} \end{aligned} \tag{6.25}$$

The term $2^{\lambda T_i}$ in (6.25) can be computed incrementally by simply keeping the sum of $2^{\lambda T_i}$ for each point p_i from S_m at T_i in G_k, overcoming the issue of storing individual relevance weights. This will reduce the time complexity of health computation from $O(SN_pN)$ to $O(SN_p)$ because we no longer need to iterate over all the data points of a system to compute the anomaly scores. However, computing $2^{\lambda T_i}$ becomes numerically infeasible as $T_i \to \infty$. To circumvent this, we introduce a "landmark" timestamp T_o such that:

$$dG_k^{S_m} = \sum_i^{G_k^{S_m}} f((t_m - T_o) - (T_i - T_o)) = 2^{-\lambda(t_m - T_o)} \sum_i^{G_k^{S_m}} 2^{\lambda(T_i - T_o)}$$

where $T_i - T_o$ should be much smaller than T_i. Beside reducing the numerical value of $2^{\lambda T_i}$, T_o brings a subtlety to the anomaly scores: all the data recorded before T_o have become obsolete and are ignored when computing (6.4). Concretely, we can set T_o to the timestamp at which a maintenance took place and restored the health state of a system. $T_i - T_o$ thus becomes the interval from the *last-known time of good health* of a system to its current time. Without an explicit T_o, T_i implies that the last-known time of good health of a system is at 0, which is the start of its lifecycle. Nevertheless, T_o is difficult to obtain, because maintenance feedback is not always accessible in practice.

4.4.2 Refining the clusters on expert's feedback

For now, CheMoc discovers and updates the clusters on its own. The resulting clusters are as good as it could be solely from the data without help from the domain experts. To evaluate the clusters of CheMoc, we collect the cluster footprints in each batch and show them to a domain expert. They verify the clusters based on their understanding of the systems. We use their feedback to tweak the hyperparameters of CheMoc in hope of finding better clusters.

It would be better to incorporate the feedback directly in CheMoc, forming a continuous feedback loop that allows CheMoc to adapt itself. A feedback may trigger CheMoc to re-estimate the hyperparameters, to split and/or merge clusters. Reinforcement learning is a potential direction because it is by nature compatible with the online learning paradigm having a feedback loop. We believe that the expert's feedback would improve CheMoc significantly. In the end, the purpose of CheMoc is not to replace the experts completely but to be a tool that helps them in the decision-making process.

Moreover, we plan to integrate a continuous monitoring of the cluster quality via multiple incremental cluster validity indices. A sudden change in the indices likely indicates a drift in the data distribution, which in turn may alter the clusters significantly. Once such drift is detected, CheMoc can adapt its hyperparameters to novelties from the stream and/or query the expert on how the clusters should be refined.

4.4.3 Fuzzy and/or hierarchical clustering

Given the multifault assumption, a system likely generates a cycle that is affected by multiple anomalies simultaneously, thus one data point may belong to more than one cluster. Using fuzzy clustering to soften cluster membership may improve the accuracy of the health profiles. Another way to increase the clustering accuracy is by hierarchical clustering, because an anomaly can be part of another high-level fault. This may require integrating prior information about system specifications, such as expected types of fault, hierarchy of faults, or affected components in the systems.

5 Conclusion

Aiming at real-time machinery monitoring for the railway, we propose CheMoc as an unsupervised method that employs online clustering to capture the health profiles of the monitored systems and to assess their working condition continuously from a stream of unlabeled sensor data.

We use DenStream to detect online clusters via density propagation on the sensor stream. Because the stream contains the data from multiple systems, we modify DenStream such that a cluster considers the contribution of individual systems. We show that, even if the cluster features are separated by system, the clusters remain incrementally updatable without explicit data storage. We omit the pruning operation from DenStream to stabilize the discovery of online clusters, but we incorporate the decay factor in the computation of the health scores to ensure the adaptivity of CheMoc.

We evaluate CheMoc on the data of the closing cycles of the R2N data set, using the expert indicators for better explanability. Via experimental results, we see that CheMoc succeeds to discover clusters that correspond to the underlying physics of known anomalies, which are positively confirmed by a domain expert. These clusters form the health profiles from which CheMoc assesses the health of individual systems accordingly. Being incremental, DenStream as the core clustering of CheMoc continuously adapts the clusters on novelties from the stream and finds a stable set of clusters earlier than the traditional offline DBSCAN. Therefore, the hypotheses H_d^1, H_d^2, and H_d^4 are validated on the closing cycles of R2N data. The results obtained to validate H_d^3 are less conclusive, so we decide that H_D^3 is only partially validated.

Yet, there remains important shortcomings to be addressed. CheMoc stores the relevance weight of the data points in memory to compute the anomaly scores quickly at the expense of non-scalable time and space complexity. CheMoc cannot integrate human feedback that would have enabled it to further refine the clusters. Adding incremental cluster validity indices to monitor the clusters would allow timely drift detection, based on which we can prompt for expert feedback more efficiently. Besides, under multifault assumption, hard partitioning is not the most suitable approach; instead, fuzzy clustering will allow a greater flexibility to cluster the data stream. In addition, hierarchical clustering will enhance the hierarchy of health profiles and add more diagnostics capacity to CheMoc. Overall, future improvements seek to implement an interaction between CheMoc and railway experts.

Chapter 7

Prognostics

In this chapter, we discuss shortly about prognostics and what it entails for predictive maintenance. At the time of writing, our prognostics method has not been implemented, due to the lack of a well-built ground-truth on the two data sets NAT and R2N for systematic evaluation. Once a ground-truth is established for each data set, we will start implementing and evaluating the online prognostics method to complete the pipeline for online predictive maintenance on complex railway systems.

Prognostics is the task of predicting failures [183], most commonly tackled by estimating the remaining useful life (RUL) of a piece of equipment [207]. RUL estimation is a regression task, which belongs to the supervised learning paradigm and thus requires labels. In order to implement an online regressor for estimating the RUL of any given system, it is crucial to have labeled data as inputs. Nonetheless, the RUL labels are not abundant; in our case, it is unlikely to have one RUL label for each cycle (in the form of feature vector) on the stream.

Firstly, it is infeasible to manually annotate the RUL for all data points on a fast stream. Secondly, the RUL labels must be supplied by the maintenance workers who know the moment where a system fails or has severely degraded, but such feedback may not always be provided (incomplete records or maintenance logs not digitized). Thirdly, the feedback may not be accurate due to human errors, for instance, a system has a problem that is difficult to detect and is missed by the maintenance team. As a workaround, we devise a self-supervised annotation scheme to create the RUL labels automatically without completely depending on the maintenance team's feedback. Please note that the self-annotation is done on each system independently.

Regarding the time unit of the RUL, the time in seconds or days is usually chosen. However, in the railway, such time units are not the most intuitive. Let us consider a scenario as shown in Figure 7.1. at an instant t, a system is put at rest for three hours, then it is put back to operation at t' and generates six cycles during 10 minutes before failing at t_F. If we choose hours to be the RUL time unit, the RUL of the system at t is 3 hours 10 minutes, which appears less alarming than it should and does not leave

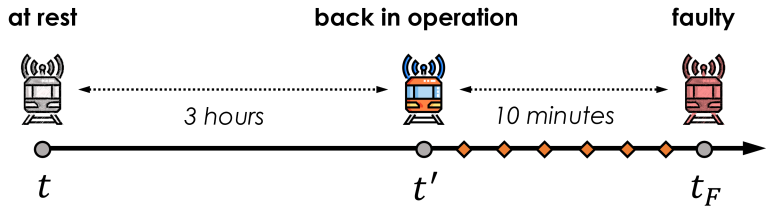


Figure 7.1: Starting from an instant t, a system is put at rest until it is back in operation three hours later at t', generates six cycles, then fails at t_F.

enough time to plan a maintenance inspection. On the contrary, if we choose the number of cycles as the RUL time unit, the RUL at t becomes six cycles, which is more relevant to the working condition of the system and creates a sense of alertness to the maintenance team. Therefore, the RUL of a system will be estimated in terms of *the remaining number of cycles the system can still generate until its failure.*

To self-annotate the RUL, we count the RUL backward from a cycle captured at a known moment of failure ($C_{t_{F_k}}$) and stops when reaching the cycle at the previous known moment of failure ($C_{t_{F_{k-1}}}$). Precisely, when a cycle of a system is recognized as "faulty" (when the health score of this system reaches the maximum value), the RUL self-annotation starts immediately. First, it assigns the label RUL = 0 to the cycle at $C_{t_{F_k}}$. Then, it increments the RUL going backward on previous cycles and stops once it reaches the cycle *just before* of the previous moment of known failure ($C_{t_{F_{k-1}}}$).

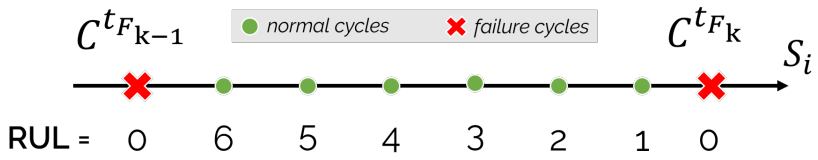


FIGURE 7.2: From a cycle at a known moment of failure ($C_{t_{F_k}}$), we increment the RUL backward, starting from the value 0 at $C_{t_{F_k}}$, until we reach the cycle of the previous known moment of failure ($C_{t_{F_{k-1}}}$). The annotation stops one cycle before $C_{t_{F_{k-1}}}$.

These newly generated RUL labels, coupled with the associated feature vectors and health scores, become the new training data for the online regressor. That is, a labeled data example (X, y) to feed to the regressor has the form $\left([X_{S_m}^T, H_{S_m}(T)], RUL_{S_m}(T) \right)$. Even if the self-annotation is done separately for each system, the resulting labeled data are used to update a single online regressor[1], so that the regressor learns to recognize RUL patterns across different systems.

Nevertheless, two issues arise.

(i) Two failures may be months apart, with tens of thousands of cycles in between[2]. It is not intuitive nor useful to create the RUL labels at such a magnitude.
(ii) The moments of failure are not always known. It is possible that a maintenance inspection happens before a failure occurs, which shadows the real failure moment.

To tackle the issue (i), we limit the RUL backward count and stops it upon reaching the *first anomalous cycle* since the previous failure. In other words, only anomalous cycles are assigned an RUL label, while normal cycles are not annotated, or annotated with a neutral label indicating that the system is in a good condition. It also implies that the self-annotation does not proceed until the previous known failure $C_{t_{F_{k-1}}}$, but only until the first cycle that is marked anomalous since $C_{t_{F_{k-1}}}$. A cycle of a system is anomalous if after processing it, the system's health score exceeds a tolerance threshold, for instance, when the health score crosses 0.5. From this point, we face two options. We may automatically consider that all the cycles that follow this one are anomalous, but this risks false positives if the health score then decreases below, or fluctuate around the threshold. The second option is that we only recognize a cycle as the start of an anomaly if all the cycles that follow also maintain the health score above the threshold, i.e., a sequential anomaly detection. We will study these options through experimentation.

[1]Following the logic of CheMoc, we have one regressor for each cycle type.
[2]Suppose that two failures are three months apart and that a cycle is created every three minutes, we will have 43200 cycles between two failures.

To overcome the issue (ii), we replace the exact moments of failures by *the moments a maintenance inspection was carried out* to be the anchor point. Nonetheless, a system is not always repaired after a maintenance, if it is deemed sufficiently good to be put back in operation. In this case, the backward count must not be started if the system does not undergo any change of state after an inspection. In some cases, we may not receive any feedback about maintenance inspections, but a drift in the data when the system switches from one state to another is a hint that an inspection has taken place. Hence, the important factor is whether the health score of a system eventually reverses its trend, with or without maintenance feedback. The self-annotation of a system S_m only starts when such change is detected from $H_{S_m}(T)$. Whether a maintenance feedback is made explicit or not decides the *degree of confidence* in the RUL self-annotation.

- If $H_{S_m}(T)$ changes and a maintenance feedback is affirmed, the RUL annotation has the highest degree of confidence.
- If $H_{S_m}(T)$ changes but there is no maintenance feedback, a degree of confidence is added to each RUL annotation, depending on the magnitude of the change in $H_{S_m}(T)$.
- If a maintenance feedback is given but $H_{S_m}(T)$ remains on the same trajectory as before, the maintenance might not have been done properly and the RUL self-annotation is not triggered.

Then, not only the RUL labels are given to the online regressor, but a degree of confidence $\gamma \in [0, 1]$ is also included. Therefore, a training data example becomes $\big([X^T_{S_m}, H_{S_m}(T)], [RUL_{S_m}(T), \gamma(\Delta H)] \big)$, where ΔH denotes the magnitude of change in the health scores.

Figure 7.3 describes the modified self-annotation scheme. Let $C^{t_{F_{k-1}}}$ and $C^{t_{F_k}}$ be two consecutive failures associated to two cycles (we omit the notation of the system S_m for simplicity), such that $C^{t_{F_{k-1}}}$ and $C^{t_{F_k}}$ are the last cycles generated by a system before it enters a failure state, or before a maintenance inspection is carried out. Once $C^{t_{F_k}}$ is known, either via maintenance feedback or drift detection on the health scores, the self-annotation is triggered and assigns the RUL labels backward to every cycle preceding $C^{t_{F_k}}$ until we reach $C^{t_{F_{k-1}}}$. However, explicit RUL labels are only assigned to anomalous cycles. Other cycles belonging to the healthy region are labeled with a generic label, for instance, `healthy` or ∞, to distinguish them from anomalous cycles and/or to be excluded from updating the regressor.

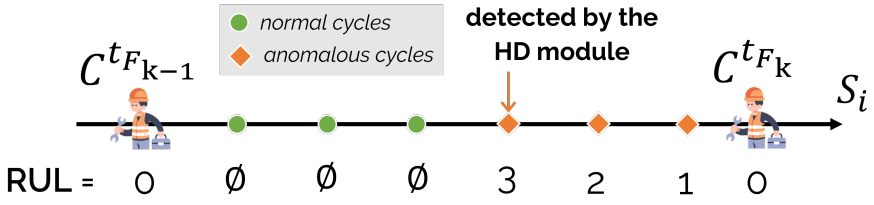


FIGURE 7.3: The modified self-annotation scheme of the RUL, taking into account the aforementioned issues

Having the self-annotated RUL labels, we use them to update the online regressor. Starting from the beginning, the regressor is continuous updated on the training examples (including the feature vectors, the health scores, the RUL labels, and the degree of confidence), until it attains a predefined accuracy threshold at a time T, starting from which we can use the regressor to estimate the RUL of any given system. The regressor is trained on the data of all systems from the fleet, but one RUL prediction is linked to one individual system.

Chapter 8

Conclusion and Perspectives

This chapter summarizes the works implemented thus far to address the central research question of the thesis. We also discuss the perspectives for post-thesis works.

1 Conclusion

The overall topic addressed in this thesis is the applicability of online machine learning to predictive maintenance in the railway industry. To find the research question, we perform a literature study on the current progress of data-driven predictive maintenance, the state-of-the-art in online machine learning, and the related works linking these two domains. We also study the concepts surrounding maintenance research, such as complex systems, reliability theory, existing maintenance strategies, and industrial maintenance standards (Figure 8.1).

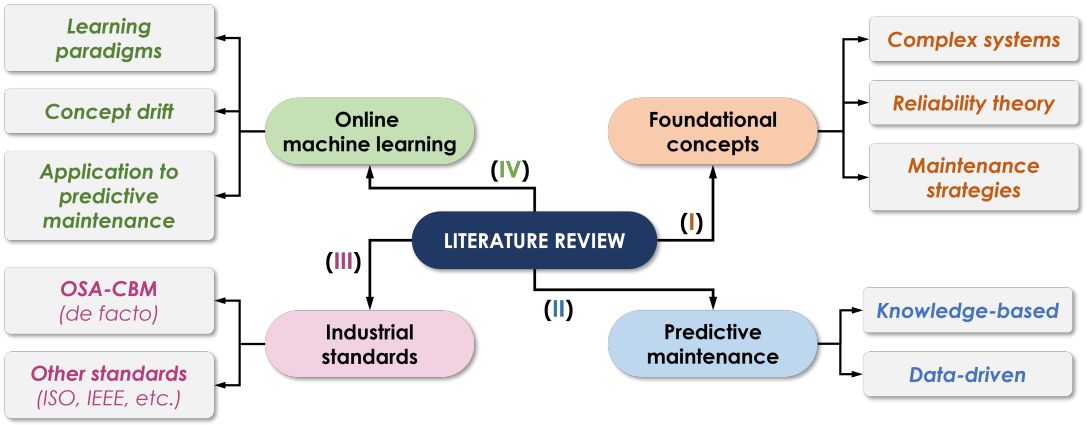


FIGURE 8.1: Our literature study covers four domains: (I) foundational concepts of maintenance research, (II) predictive maintenance, (III) industrial standards, (IV) and online machine learning.

Our observations distilled after the literature study are published in:

➔ Minh Huong Le Nguyen, Fabien Turgis, Pierre-Emmanuel Fayemi, and Albert Bifet. "Challenges of Stream Learning for Predictive Maintenance in the Railway Sector". In: *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning*. Communications in Computer and Information Science. Springer International Publishing, 2020, pp. 14–29. ISBN: 978-3-030-66770-2. DOI: 10.1007/978-3-030-66770-2_2,

and allow us to state the research question:

> Given that online machine learning can overcome certain limits of traditional machine learning, could we use online machine learning to achieve satisfactory results for railway predictive maintenance?

To tackle the research question, we formulate the hypotheses by blocks of railway characteristics, namely *granularity*, *cyclicity*, *features*, and *health* (Chapter 3). To test our hypotheses, we use two real-world railway data sets collected from the passenger access systems on two fleets of passenger trains NAT and R2N, supplied by SNCF - the national railway company of France. A hypothesis is validated only if it reaches satisfactory results on both data sets.

At the time of writing, some hypotheses are validated, and some need further investigation. Figure 8.2 schematizes our hypotheses: the hypotheses in boxes with thick lines are fully validated, those in boxes with dashed lines are not validated or only partially validated, those in boxes with no border line are not yet addressed.

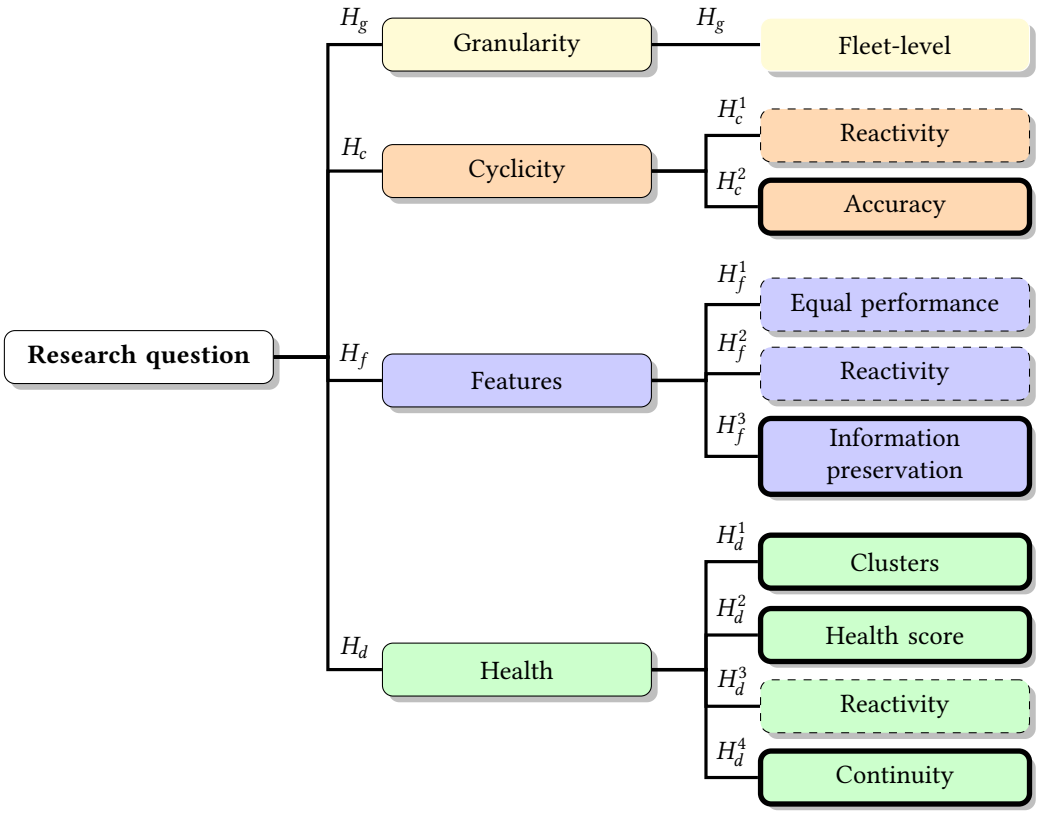


FIGURE 8.2: Thick lines are validated hypotheses, dashed lines are those that cannot be validated or are only partly validated, and H_g is not yet addressed

Studying and validating the hypotheses result in a byproduct processing pipeline for the purpose of predictive maintenance. In Figure 8.3, one module maps to one block of railway characteristics and is linked to several hypotheses. The input to this pipeline is a stream of sensor data generated by a fleet of railway complex systems; the output is an alert issued for a system in which a problem is detected. The results obtained thus far on implementing the pipeline are published in:

➔ Minh-Huong Le-Nguyen, Fabien Turgis, Pierre-Emmanuel Fayemi, and Albert Bifet. "Real-time learning for real-time data: online machine learning for predictive maintenance of railway systems". In: *Transport Research Arena (TRA)*. Lisbon, Portugal, Nov. 2022,

and the extended results are under review for:

➔ *Minh-Huong Le-Nguyen, Fabien Turgis, Pierre-Emmanuel Fayemi, and Albert Bifet. "Exploring the potentials of online machine learning for predictive maintenance: A case study in the railway industry". In: Applied Intelligence (Springer).*

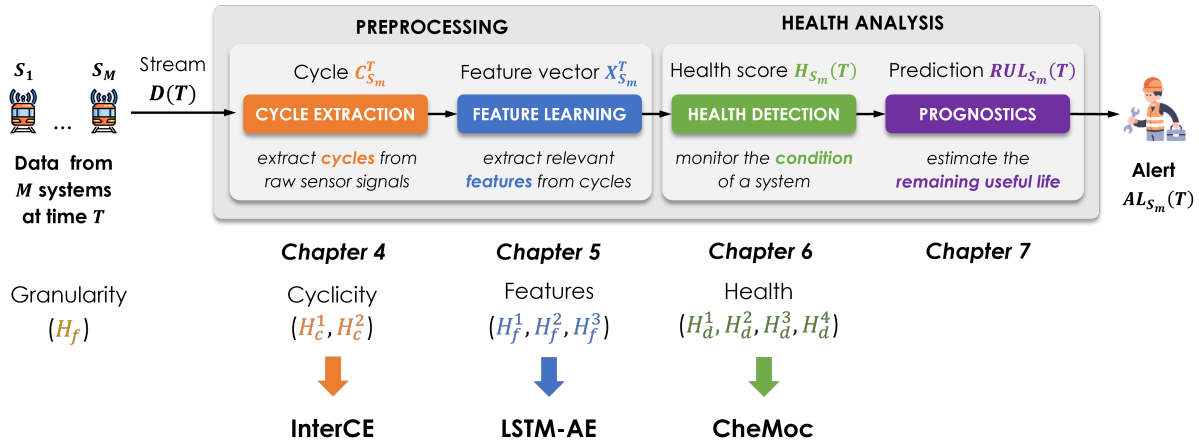


FIGURE 8.3: The pipeline of four modules are the byproduct result of validating and implementing the hypotheses, where each module is linked to a method that addresses a set of hypotheses of one railway characteristics.

CYCLICITY The *cyclicity* characteristic maps to the module Cycle extraction in the pipeline, with the aim to detect and label cycles from a stream of sensor data. The hypotheses H_c^1 and H_c^2 address, respectively, the reactivity and accuracy of online machine learning for cycle extraction. We implement the framework **InterCE** (*Interactive Cycle Extraction*) that learns to extract cycles from such stream using an active learning-based approach (Chapter 4).

- Given a new input, InterCE sends a query to a human expert if the input contains a novel motif that InterCE does not know how to process; otherwise, InterCE attempts to extract cycles from this input automatically using past feedback.
- InterCE combines multiple extractors via ensemble learning to reduce the variance of individual extractor and to increase the overall accuracy.
- InterCE relies on a memory that stores familiar data motifs so that queries are only created for strictly unseen motifs, in order to limit the number of queries sent to the human expert.
- The experimental results show that:
  - ✖ (H_c^1) When compared to its offline counterpart, online InterCE does not show a clear superiority in reactivity, but it remains competitive nonetheless. H_c^1 is only partially validated.
  - ☑ (H_c^2) InterCE substantially improves the baseline accuracy of the expert system on both data sets, which validates H_c^2.

➔ Linked publication: Minh Huong Le Nguyen, Fabien Turgis, Pierre-Emmanuel Fayemi, and Albert Bifet. "A Complete Streaming Pipeline for Real-time Monitoring and Predictive Maintenance". In: *Proceedings of the 31st European Safety and Reliability Conference*. 2021, p. 2119. DOI: 10.3850/978-981-18-2016-8_400-cd.

FEATURES The *features* block maps to the Feature learning module, the purpose of which is to extract a compact yet representative feature vector from each cycle extracted by InterCE. The hypotheses H_f^1 and H_f^2 compare the performance of offline and online feature learning on two criteria - representation

performance and reactivity, while H_f^3 answers whether learned features preserve information better than manually engineered features (also called expert indicators). For this, we implement an **LSTM-AE** (*long short-term memory autoencoder*) that leverages the LSTM cells in an autoencoder architecture to learn representation from sequential data (cycles) (Chapter 5).

- In the LSTM-AE, each layer is an LSTM cell to capture the temporality of the cycles, which are per se time series.
- We propose two versions of the LSTM-AE, including:
  - a *plain* model that has a traditional encoder-decoder architecture and 14 layers in total, and
  - a *joint* model that has an additional classifier connected to the encoder to map a cycle to its context (train stations).
- We train both versions of the LSTM-AE in three settings, that are:
  - the *offline* setting trains the LSTM-AE on all the training cycles over K epochs ($K = 300$),
  - the *online* setting trains the LSTM-AE on each incoming cycle over one epoch,
  - the *online incremental* setting trains the LSTM-AE on a micro-batch, containing the incoming cycle and $K-1$ previous cycles; the purpose is to let the LSTM-AE process one cycle as many time as the number of epochs and to enact a fair comparison with the offline setting.
- The experimental results show that:
  - ✖ (H_f^1 and H_f^2) After comparing the results of the LSTM-AE on three learning settings, we see that the offline version clearly outperforms the online versions in reconstruction capacity and reactivity, therefore H_f^1 and H_f^2 cannot be validated.
  - ☑ (H_f^3) We assess the quality of the (offline) LSTM-AE features and the expert indicators via a cycle-feature ranking evaluation and observe that the LSTM-AE features are better at preserving the information of the cycles than the expert indicators, which validates H_f^3.

Health We define the *health* of a system its extent of being free from anomaly. Identifying a system's health is linked to the Health detection module. The aim is to discover a set of evolving health profiles in the form of adaptive clusters from the stream of feature vectors, then to compute the health score of each system from these profiles. To monitor the health of a fleet of systems continuously, we formulate four hypotheses: H_d^1 (health profiles as online, evolving clusters), H_d^2 (cluster-based health score computation), H_d^3 (superior reactivity of online clustering to offline clustering), and H_d^4 (continuity in cluster evolution as an advantage of online clustering). We implement **CheMoc** (*Continuous Health Monitoring using Online Clustering*) to tackle these hypotheses (Chapter 6).

- CheMoc uses DenStream as the core online clustering algorithm to capture the health profiles of the fleet as evolving clusters.
- Using DenStream as it is does not align with the railway operational constraints, thus we modify DenStream to amend the misalignment; the resulting algorithm still enables incremental update of the clusters.
- Based on the clusters, CheMoc updates the health score of every system such that the scores reflect the deviation of a system from a reference health profile.
- The experimental results show that:
  - ☑ (H_d^1) CheMoc succeeds to discover clusters that correspond to known anomalies of the systems, as confirmed by a domain expert, which validates H_d^1.
  - ☑ (H_d^2) Since the clusters are correctly identified, CheMoc uses them to compute the health score of a system accordingly at any time, which validate H_d^2.

✖ (H_d^3) When compared to the offline DBSCAN via two cluster validity indices, the reactivity of CheMoc remains inconclusive, as it does not outperform DBSCAN in all cases, so H_d^3 is only partially validated.

☑ (H_d^4) Being incremental, CheMoc continuously adapts the clusters on novelties from the stream, which its offline counterpart DBSCAN cannot, thus validating H_d^4.

➔ Linked publication: Minh-Huong Le-Nguyen, Fabien Turgis, Pierre-Emmanuel Fayemi, and Albert Bifet. "Continuous Health Monitoring of Machinery using Online Clustering on Unlabeled Data Streams". In: *2022 IEEE International Conference on Big Data (Big Data).* Dec. 2022, pp. 1866–1873. DOI: 10.1109/BigData55660.2022.10021002.

Granularity The hypothesis H_g that tackles the granularity of the analysis (fleet-level versus individual-level) is not yet addressed, and will be included in future works.

General conclusion The works conducted in this thesis demonstrate the potentials of online machine learning to enable adaptive and lifelong learning for predictive maintenance, with applications in the railway industry.

Among ten hypotheses that we formulated, five of them are fully validated ($H_c^2, H_f^3, H_d^1, H_d^2, H_d^3$) and highlight the superior accuracy of online, adaptive learning in comparison to offline, static learning.

- By incorporating human feedback on-the-fly, InterCE is able to update its knowledge on the task without requiring prior training.
- CheMoc continuously updates the clusters on incoming data - this allows CheMoc to follow the evolution of the health profiles and of a system's health more accurately than an offline process.
- We would like to note that although H_f^3 is validated, the features are obtained from the offline LSTM-AE, therefore online machine learning lags behind in all aspects for the *features* block.
  - This could be due to choosing a neural network as a feature learner: since the current state-of-the-art of neural network training favors training a complex model on a vast volume of data over hundreds or even thousands epochs, a neural network is not likely optimized for a continuous data stream.
  - Future works may replace the LSTM-AE by another feature learning method other than neural networks to enact a fairer comparison.

Nonetheless, we find it surprising that the reactivity of online machine learning is not higher than that of offline machine learning, as the hypotheses that fail to be fully validated mostly concern the reactivity of online machine learning ($H_c^1, H_f^1, H_f^2, H_d^3$), with an exception of H_f^1 that points out the low accuracy a neural network learned online.

- For InterCE, LSTM-AE, and CheMoc, the online methods do not outperform, or at best are competitive to, the offline ones.
  - In terms of temporal accuracy, the offline version of InterCE outruns online InterCE on NAT data, while online InterCE is slightly better on R2N data; because online InterCE is not the clear winner on both data sets, we do not fully validate H_c^1.
  - The offline LSTM-AE has an approximately equal training time as the online ones, even though the latter should be faster as they are trained on fewer data; however, we are not certain if this is due to the nature of online learning, or to the technical implementation in `tensorflow` that does not privilege online training; we made the conservative choice not to validate H_f^2.

  - We compare the cluster quality by CheMoc and by DBSCAN via two cluster validity indices, Xie-Beni (XB) and Davies-Bouldin (DB), and find out that the clusters of CheMoc are better then those of DBSCAN on the XB index only, but lag behind on the DB index, although the gap is not large; that is why H_d^3 is only partially validated.
- We believe this is related to the *convergence* of a learning algorithm:
  - An offline learning algorithm is trained on a static batch of data until it reaches convergence (e.g., the loss cannot be further reduced), after which it is deployed for online use and maintains a stable accuracy, unless a drift occurs.
  - An online learning algorithm is deployed online from the beginning and updates itself on new data examples continuously, but its accuracy inevitably fluctuates until it has seen enough data to perform the intended task properly.

Henceforth, we are facing a compromise: offline machine learning requires prior training but is stable after validation yet demands retraining in the case of data drifting, whereas online machine learning is unstable until reaching convergence but can adapt to drifts by itself.

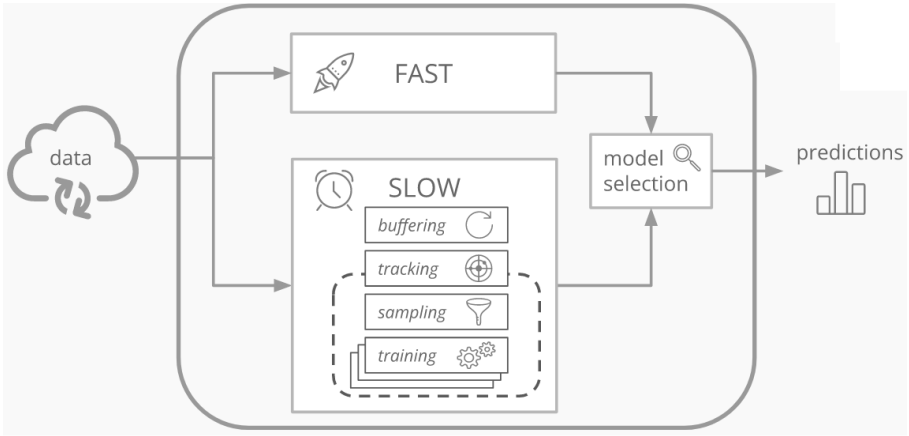


FIGURE 8.4: A framework that combines a fast and slow learner to improve the overall accuracy of data stream mining [162].

Given that both paradigms do not possess an absolute superiority, a potential direction is to have a hybrid approach by creating a framework with two parallel systems: an online, highly reactive method that adapts very fast to drifting, and an offline, retrospective method that carefully distills information passed from the online method to yield more reliable results. This idea has been explored by Montiel, Bifet, Losing, Read, and Abdessalem [162] (Figure 8.4), and we deem it very relevant to the future research direction of reactive and accurate data-driven predictive maintenance.

2 Perspectives

There are multiple research directions opening at the end of this thesis. We will first discuss immediate future works that aim to improve of InterCE, LSTM-AE, and CheMoc to overcome existing issues, and to implement the missing methods (prognostics, granularity investigation). Then, we will look at the perspectives of this thesis on a larger scope, such as implementing the human-in-the-loop mechanism to each module to enable the methods to self-improve with the help of expert feedback, or realizing a hybrid framework with the parallel online-offline learners.

2.1 Improvements of each module

The methods we implemented are functional proof-of-concepts of the applicability of online machine learning to railway predictive maintenance. Yet, improvements are necessary to make them more robust and efficient to deal with real-world scenarios, where the input stream is no longer simulated but comes from a fleet of railway equipment in operation in real-time.

InterCE InterCE relies heavily on memorizing instead of learning. Storing distinct data motifs and labeled cycles from human feedback in memory are the backbone of InterCE. Instead of storing data explicitly, we could implement proper learning mechanics in InterCE. For instance, the motifs are not saved in memory as is but a classifier learns to recognize unique or seen motifs; the knowledge of InterCE not only saves the feedback cycles based on their label but apply clustering on these cycles to become more robust against human errors.

Then, the update on human feedback affects only the labeling functionality. The extractors currently used in InterCE are static: they do not adapt nor evolve and will likely repeat the same mistakes. To improve the overall accuracy of InterCE, making the learners updatable is of utmost necessity. The human feedback would serve as ground-truth to adjust the hyperparameters of the extractors in InterCE accordingly. Additional extractors could be developed to diversify the extraction algorithms in the ensemble of InterCE.

The scalability of InterCE is also a concern. InterCE takes several hours to process 100000 data files of NAT data and more than a day for R2N data. Adding parallelism could reduce the execution time, but as long as the motif search remains a bottleneck, InterCE may not scale on fast, infinite data streams. This stresses on the need to convert the memorizing-and-searching approach of InterCE to a learning-driven approach.

LSTM-AE Although the LSTM-AE outperforms the expert system in information preservation, it is by nature a black-box model and its features are not interpretable, unlike the expert indicators that convey exact meaning. Explaining the LSTM-AE features is important to facilitate model debugging and raise interpretability when presenting the results to the end-users.

In addition, the online training paradigm unfortunately does not lead to results that are more satisfactory than its offline version. Despite being online-compatible, neural networks appear to reach their best performance only when trained in the traditional offline setting, where the networks iteratively revise the same set of training examples until an optimum is found. We may think of a hybrid approach, in which the LSTM-AE is trained offline until convergence, but it stays updatable on new examples after deployment. This falls in the category of *continual learning*, the goal of which is to ensure the neural network can continue to learn new data while not forgetting past data [87].

Most importantly, a crucial drawback of the LSTM-AE is that it erases perturbations from the data to produce a smooth representation. Although this is desirable in some cases, for example, to denoise the inputs, such perturbations may hint at an anomaly manifesting in the systems, and smoothing out these perturbations leads to information loss. To amend this issue, we can adjust the LSTM-AE architecture to make it more sophisticated, or craft a loss function that enables the network to retain relevant perturbations.

CheMoc To quickly compute the decaying version of the anomaly score, CheMoc stores the relevance weight of each data point in the clusters, but this will not scale as data arrive infinitely. Several

solutions exist to circumvent this issue. We can compute the terms in the anomaly score formula cumulatively and introduce a landmark timestamp to make it numerically feasible. The relevance scores can be saved in a database and retrieved only when the health scores are recomputed, but this will undoubtedly incur a delay. Or, we can simply use the static version of the anomaly scores, given that the static and decaying versions make little difference in the health scores, as observed from the experimental results.

Moreover, CheMoc does not have a feedback loop. Unlike InterCE that updates its knowledge on human feedback, CheMoc discovers clusters by itself on the data stream, and the resulting clusters are as good as it could get without human guidance. Even if the density threshold is dynamically adjusted, there is no guarantee that the adjusted density reflects the true density of the data, as it is computed based on the clusters and biases may be inevitable. Making CheMoc updatable on human feedback, such that the density threshold can be reset on command, or a cluster is requested to split or to merge, may bring significant improvement in building accurate health profiles.

Prognostics At the end of the thesis, an online prognostics method has not been implemented, but we have sketched the idea of a self-annotating framework to learn an online regressor that estimates the remaining useful life of a system on a stream of feature vectors and health scores. This is the final module that will enable failure prediction and thus complete our predictive maintenance solution based on online machine learning. This module partly relies on human feedback to identify the anchor moment on which the self-annotation starts, therefore human-in-the-loop will play a crucial role in our prognostics solution.

Granularity Once the prognostics method is implemented and thus completes the full pipeline, we will study the hypothesis of granularity H_g, by comparing the results of a fleet-level pipeline to those of individual pipelines (one pipeline per system).

2.2 On a larger scope

Besides separate improvement by module, it is essential to have all modules integrated in a seamless pipeline on a processing server - this is on the course of realization at IKOS Lab. The pipeline is being integrated to the framework **IKIM** (IKOS Intelligent Maintenance) module by module. We would like to note that our entire pipeline fits in the `Analyzing` node of IKIM (Figure 8.5) Currently, InterCE has been deployed on IKIM and is undergoing extensive tests to prepare it for production.

Figure 8.5: The IKIM platform dedicated to big data processing for railway predictive maintenance, under development at IKOS Consulting

Once the pipeline is fully deployed on IKIM - given that the prognostics module and the granularity hypothesis have been validated, we will test the pipeline on the data from other railway complex

systems, such as the batteries, compressors, or the heating, ventilation, and air conditioning systems, to assess how well our methods generalizes to other types of systems.

As mentioned, we also plan to integrate a feedback loop with human experts to each method, so that the expert feedback will fine-tune the methods and, in overall, bring improvement to the accuracy of the outputted maintenance alerts. We have shown that the human-in-the-loop approach helps InterCE to attain an accuracy higher than that of the expert system. It is our expectation that such approach will also strengthen LSTM-AE, CheMoc, and our prognostics method.

Furthermore, as we have observed that using only online machine learning or offline machine learning is unlikely to yield optimal results in all cases, it is worth considering a hybrid approach that has a fast, reactive learner to handle novelties from the data stream, and a robust, sophisticated learner working in batch mode that aggregates results of the fast learner and past data to refine the results. For example, the fast learner may detect changes from the incoming data and signals a potential drift; the drift detection alert is sent to the batch learner (such as an expert system), and by revising past patterns, the batch learner decides that it is a false alert caused by contextual noise and does not emit maintenance orders. After the experimentations done in this thesis, we have come to the belief that being fast is not equivalent to being accurate, and that a harmony between reactivity and retrospection is necessary to guarantee reliable results.

To the best of our knowledge, there has been no previous work that tackles various stages in a predictive maintenance solution using online machine learning, for both the training *and* prediction phases. With this thesis, we have attempted to bridge this gap in the related literature. We hope that our works will attract the attention of practitioners to online machine learning - a paradigm that enables lifelong, adaptive learning and deserves much consideration, and its potentials yet to be explored for predictive maintenance in the railway industry.

Appendix A

Résumé en français

Contents

1 Motivation

Le transport ferroviaire permet de transporter de nombreux passagers tout en laissant une faible empreinte carbone dans l'environnement, ce qui en fait un mode de transport en commun efficace et écologique. Donc, le chemin de fer continuera à se développer dans les décennies suivantes, d'où le besoin d'une maintenance efficace. La maintenance est une fonction support essentielle car elle aide à assurer la fiabilité des équipements, la disponibilité du service, et la sécurité des humains. Dans le ferroviaire, la maintenance corrective et la maintenance préventive périodique sont dominantes. Le premier vise à mettre en place les interventions urgentes ayant lieu après l'occurrence d'une panne dans le système, alors que le dernier a pour l'objectif de planifier des inspections par intervalles régulières pour prévenir toutes défaillances potentielles.

Récemment, émerge une nouvelle stratégie de maintenance qui est la *maintenance prévisionnelle*. Cette stratégie émet des ordre de maintenance en s'appuyant sur l'état actuel d'un système, révélé par la surveillance en continu de l'état du système, et sur la prévision de ses conditions futures. La maintenance prévisionnelle a attiré une attention croissante des praticiens du domaine, en particulier dans l'ère de l'Industrie 4.0 où les systèmes sont équippés de capteurs qui génèrent *un flux de données* en temps réel, facilitant la surveillance en continue de l'état des systèmes. Les techniques réalisant la maintenance prévisionnelle peuvent se diviser en deux catégories: l'approche basée sur les connaissances et l'approche basée sur les données. Pour cette thèse, nous nous concentrons sur le dernier, pour lequel *l'apprentissage automatique* a pris de l'importance.

predictive maintenance

data stream

machine learning

La pratique commune consiste à collecter un certain volume de données, puis à entraîner un modèle sur ce lot de données. Après l'entraînement et la validation du modèle, celui-ci est déployé en production pour émettre des prévisions de défaillances tout en restant constant, en dépit des nouveautés venant de nouvelles données d'entrée. Cela est *le paradigme hors-ligne de l'apprentissage automatique*. Dans le contexte où les données sont générées en continu par les capteurs installés dans les systèmes surveillés, les changements dynamiques peuvent affecter négativement la précision du modèle. Par exemple, si le modèle ne voit que les données sur quelques modes de défaillance des systèmes, le modèle échouera à reconnaître les autres modes de défaillances plus rares qui n'ont pas apparu dans les

offline machine learning

données d'entraînement. Un autre exemple est quand les modifications et améliorations fonctionnelles sont implémentées dans les systèmes, ce qui change en conséquence leur comportement et les motifs de données qu'il génèrent, le modèle doit être re-entraîné sur les nouvelles données, d'où un délai inévitable à re-collecter les données et l'effort à re-valider le modèle. Ce phénomène s'appelle *la dérive des données* et arrive souvent sur un flux de données dynamique. *data drifting*

De l'autre côté, la nature infinie et rapide d'un flux de données entrave *l'étiquetage des données*. Pour l'apprentissage automatique, une étiquette est une valeur associée à un exemple de données, par exemple, l'état d'un système au moment où il produit cet exemple de données, ou la durée de vie résiduelle à partir de cet exemple. Ces étiquettes permettent l'usage des algorithmes de l'apprentissage automatique supervisé, qui est la classes des techniques les plus développées de l'apprentissage automatique. Toutefois, la vitesse et le volume d'un flux de données rendent l'étiquetage manuel irréalisable. En outre, dans le ferroviaire, chaque occurrence d'une défaillance est suivie par la procédure FRACAS[1] pour empêcher sa réapparition; par conséquent, cette défaillance (et son étiquette) n'apparaîtra plus dans les données, ce qui rend plus difficile l'application des algorithmes supervisés sur le flux de données ferroviaires. *data labelling*

Considérant ces désavantages de l'apprentissage automatique hors-ligne, nous portons l'attention sur *l'apprentissage automatique en ligne*, qui consiste à apprendre de façon continue, en mettant à jour le modèle incrémentalement sur chaque nouvel exemple de données, et à s'adapter automatiquement aux nouveautés sur le flux de données. Par résultat, l'apprentissage automatique en ligne permet aussi à un modèle d'interagir avec les humains en collectant leur retour pour ajuster ses paramètres si nécessaire. Cette thèse étudie l'applicabilité de l'apprentissage automatique en ligne pour la maintenance prévisionnelle dans le ferroviaire, utilisant les données des systèmes d'accès passager[2] sur deux flottes de trains NAT et R2N, fournies par la SNCF de la France, comme cas d'études. *online machine learning*

Donc, la question de recherche de cette thèse est comme suit.

> Étant donné que l'apprentissage automatique en ligne peut surmonter des limites de l'apprentissage automatique hors ligne traditionnel, pourrions-nous utiliser l'apprentissage automatique en ligne pour atteindre les résultats satisfaisant pour la maintenance prévisionnelle dans le ferroviaire ?

2 Défis

Afin d'implémenter l'apprentissage automatique en ligne pour la maintenance prévisionnelle dans le ferroviaire, il faut considérer les contraintes opérationnelles et caractéristiques spécifiques aux ferroviaire. Nous les divisons en quatre blocs: granularité, cyclicité, indicateurs, et santé.

Granularité

La granularité de l'analyse concerne le niveau auquel nous analysons les données venant de toute la flotte. Si les données de tous les systèmes sont analysées ensemble, cela est une analyse au niveau de la flotte. Si les données de chaque système sont analysées indépendamment, cette analyse est au niveau individuel. Chaque option a ses points forts et faibles.

[1] Failure reporting, analysis and corrective action system

[2] Un système d'accès passager est une porte électrique automatique sur une voiture.

L'analyse au niveau de la flotte supprime l'individualité des systèmes et implique qu'il existe un ensemble de comportements partagés par tous systèmes dans la flotte. Il faut noter que l'évolution particulière d'un système peut être cachée par l'information de la masse. L'analyse au niveau individuel crée un modèle pour chaque système et entraîne ce modèle sur les données de ce système exclusivement. En conséquence, un modèle peut capter les motifs particuliers à chaque système. Cependant, un modèle peut rater l'information qui a été apprise par un autre modèle et doit tout re-apprendre.

En comparant les deux options, nous nous rendons compte que l'analyse au niveau de la flotte est plus réalisable, car une flotte avec un grand nombre de systèmes rend l'autre option au niveau individuel impossible à monter en échelle (milliers de modèles à maintenir séparément). En outre, un modèle appris sur les données d'un seul système risque à être sur-ajusté ou à apprendre des motifs incohérents. Donc, toutes les analyses effectuées pour cette thèse se font au niveau de la flotte.

Cyclicité

Tous les systèmes typiques dans le ferroviaire sont cycliques: ils sont conçus pour effectuer un nombre de fonctions défini périodiquement. Par exemple, un système d'accès passager s'ouvre et se ferme, un batterie se charge et se décharge, et cetera. Chaque réalisation d'une fonction par un système constitue un *cycle*. Techniquement, un cycle est représenté par un segment de données dont le motif se répète sur le flux de données d'entrée. Un cycle est l'unité d'analyse la plus petite permettant de révéler l'état d'un système, car un cycle enregistre le comportement d'un système pendant que celui-ci effectue une fonction. Toutes anomalies peuvent se voir dans les cycles qu'un système génère.

cycle

Donc, la première tâche à accomplir est *l'extraction des cycles* à partir d'un flux de données. Une extraction comprend deux tâches: il faut tout d'abord détecter la présence des cycles dans les données de capteurs brutes, et puis il faut associer un cycle détecté à une fonction du système (le type du cycle). Les défis liés à l'extraction de cycles sont:

cycle extraction

Défi 1: Utiliser l'apprentissage automatique hors-ligne introduit un délai important car il faut collecter assez de données sur tous les cycles puis entraîner et valider un modèle approprié.

Défi 2: Le flux de données ne contient pas d'étiquettes sur les types de cycles, ce qui complique la mise en oeuvre des algorithmes d'apprentissage supervisés.

Défi 3: Nouveaux types de cycle peuvent émerger, ou les types existant peuvent changer en raison d'une mise à jour fonctionnelle dans les systèmes. Un modèle statique ne peut pas s'adapter à ces changements et exige un re-entraînement.

L'entrée de la tâche de l'extraction de cycles est un flux de données brutes venant des capteurs, contenant les données de tous les systèmes dans une flotte. La sortie attendue est tous les cycles détectables sur ce flux et étiquetés avec leur type correspondant.

Indicateurs

Par nature, un cycle est une série temporelle multivariée. Bien qu'un cycle soit l'unité d'analyse la plus petite dans les données, comparer les millions de cycles paire par paire n'est pas efficace, car la comparaison doit se faire à travers des pas de temps et des variables, alors que comparer les cycles via des *indicateurs de résumé* peut atteindre le même niveau de performance en réduisant les ressources de calcul. Donc, la deuxième tâche qui suit l'extraction des cycles est de les transformer en vecteurs d'indicateurs, de manière que ces vecteurs soient plus compacts que les cycles d'origine tout en préservant leur information. Nous appellons cette tâche *l'apprentissage des indicateurs.*

features

feature learning

Les défis liées à l'apprentissage des indicateurs sont comme suit.

Défi 1: Nous ne savons pas quels sont les indicateurs pertinents à extraire. L'ensemble des indicateurs identifiés peut être incomplet et manque des indicateurs importants, ou en contient trop et produit des indicateurs redondants. En plus, un indicateur peut perdre et gagner la pertinence au cours de temps - ce phénomène s'appelle *l'évolution des indicateurs* et fait partie de la dérive des données. *feature evolution*

Défi 2: Les bruits venant du contexte opérationnel d'un système affectent aussi la qualité des indicateurs. Un contexte est tous les facteurs qui entourent un système en opération, tels que la température, la charge de travail, la courbure des rails. Un contexte peut faire un cycle normal apparaître anormal, et vice-versa.

Défi 3: Comme un cycle est projeté d'une dimension plus haute (espace des séries temporelles) à une dimension plus basse (espace des vecteurs), il faut réduire la perte d'information.

Défi 4: Utiliser l'apprentissage automatique hors-ligne introduit un délai important car il faut collecter assez de cycles pour entraîner et valider un modèle d'apprentissage d'indicateurs approprié.

Santé

Inspiré par la définition littérale de la santé [189], nous définissons la santé d'un système comme *sa mesure d'être exempt d'anomalie*, quantifiée par un score de santé borné entre 0 et 1, dont 0 indique la bonne santé (exempt de toute anomalie) et 1 indique la plus mauvaise santé (largement affecté par les anomalies). Une anomalie est tout ce qui dévie du comportement normal, ou typique, déterminé par la majorité des systèmes dans la flotte.

À part d'un score de santé calculé pour chaque système à un moment donné, nous cherchons aussi à identifier *les profils de santé* de la flotte. Un profil de santé est une enveloppe de caractéristiques des systèmes dans le même état de santé, tel que les données dans le même profil est plus similaire entre elles que celles situés dans les différents profils de santé. Intuitivement, nous considérons un profil de santé en tant qu'un *cluster* groupant les cycles (sous forme de vecteurs d'indicateurs) fortement similaires. Puisqu'un système est conçu pour fonctionner dans un état nominal unique, la santé typique est définie via un cluster de référence unique. Tous les autres clusters sont considérés des clusters d'anomalie.

Il existe deux défis liés au calcul de santé d'un système sur un flux de donnés: un défi associé à la découverte des profils de santé, et l'autre le calcul de la santé selon les profils détectés.

Défi 1: Les profils de santé doivent être découverts automatiquement sur le flux de données sans connaissance du domaine. L'algorithme doit aussi pouvoir s'adapter à toute évolution des profils de santé au cours du temps.

Défi 2: Étant donnés les profils de santé détectés, comment le score de santé d'un système peut-il être calculé, de manière que le score reflète l'impact de toutes les anomalies qui affectent le système à un moment donné? De plus, le score de santé doit être calculé de façon efficace pour permettre la surveillance en temps réel de toute la flotte.

3 Contributions

Ayant considéré les défis susmentionnés, nous proposons dix hypothèses, divisées par bloc opérationnel du ferroviaire comme défini dans la section précédente. Figure A.1) montre ces dix hypothèses et indique celles qui sont validées, celle qui ne sont pas validées et ne sont validées que partiellement.

La validation de ces hypothèses est basée sur les résultats expérimentaux collectés sur les données des systèmes d'accès passager de deux flottes de trains NAT et R2N.

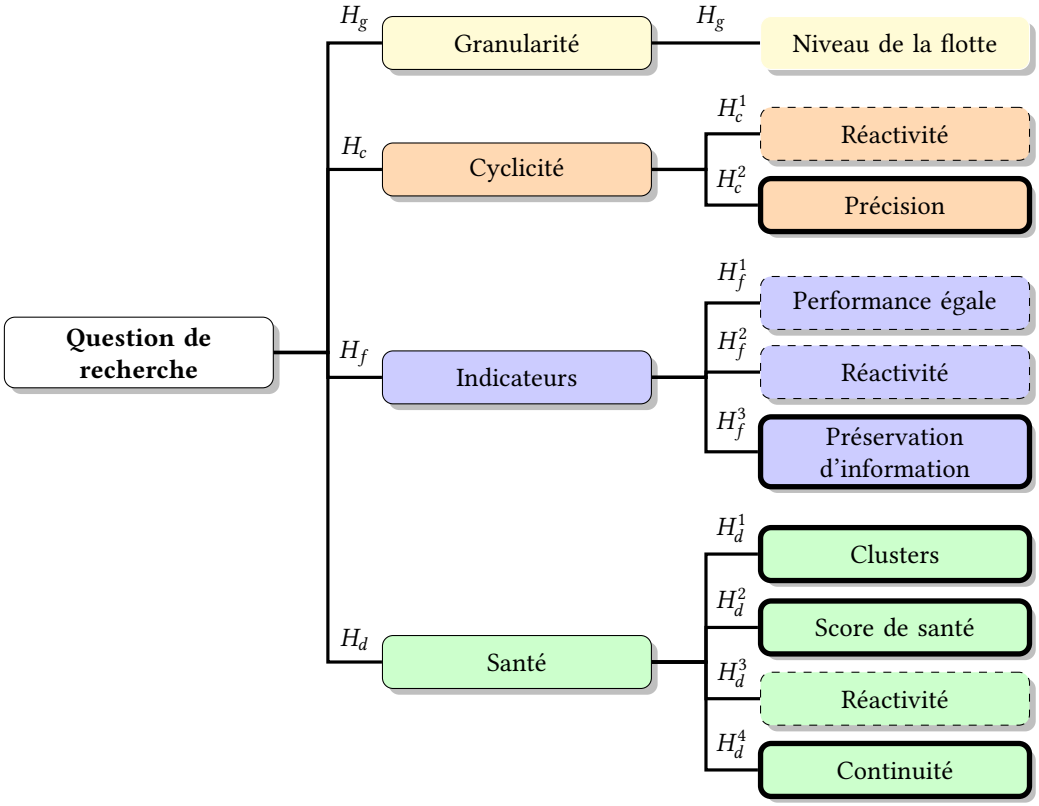


FIGURE A.1: Les lignes épaisses sont des hypothèses validées. Les lignes en pointillé sont celles qui ne sont pas validées et sont partiellement validées. H_g n'est pas encore adressée à la fin de cette thèse.

Les sections suivantes décrivent les méthodes que nous avons proposées pour valider les hypothèses. Nous avons conçu une méthode pour chaque bloc thématique.

InterCE

Pour entamer les hypothèses de cyclicité, nous avons proposé **InterCE** (*Interactive Cycle Extraction*) pour automatiser l'extraction des cycles à partir d'un flux de donnés, en appliquant le principe de l'apprentissage actif [201] pour apprendre à extraire des cycles au fur et à mesure.

Étant donné une nouvelle entrée X, InterCE décide s'il peut extraire des cycles à partir de X automatiquement, selon la connaissance qu'il possède sur la tâche. Si X est similaire à une entrée que InterCE a vu dans le passé, InterCE cherche à extraire des cycles lui-même. Sinon, InterCE crée un requête $Qr(X)$ et l'envoie à un humain qui s'engage activement à répondre aux requêtes. Une fois un retour $Fb(Qr(X))$ pour la requête $Qr(X)$ est renvoyée à InterCE, InterCE met à jour sa connaissance en ajoutant X et des cycles potentiels qui s'y trouvent, et désormais InterCE sait comment traiter les entrées similaires à X sans envoyer des requêtes redondantes à l'humain.

Pour valider l'hypothèse H_c^1, nous avons comparé InterCE à sa version hors-ligne sur 1000 fichiers de données bruts. La version hors-ligne de InterCE est entraînée sur 100, 200, et 300 premiers fichiers, aboutissant à trois modèles hors-ligne. Le modèle hors-ligne commence à retourner des cycles seulement après avoir fini la phase d'entraînement. La réactivité se mesure par la durée depuis le début du

flux de données (le premier fichier) jusqu'à la convergence du modèle (quand le modèle commence à atteindre une précision acceptable de façon cohérente). Les résultats montrent que Inter en ligne n'est pas supérieur à ses versions hors-ligne, mais leur réactivité est presque égale. Donc, l'hypothèse H_c^1 n'est validée que partiellement.

Pour valider l'hypothèse H_c^2, nous avons comparé, sur 1000 fichiers, les cycles extraits par InterCE et le système expert, déjà développé pour les deux flottes NAT et R2N. Les résultats montrent que InterCE améliorer la base de performance du système expert significativement sur toutes les deux flottes (98.2% de InterCE versus 42.53% du système expert pour la flotte NAT, et 78.3% de InterCE versus 40.6% du système expert pour la flotte R2N). Donc, l'hypothèse H_c^2 est validée.

Nous avons aussi mesuré l'efficacité de InterCE via le temps d'exécution et le nombre de requêtes créées, en testant InterCE sur 100000 fichiers de données de chaque flotte. Les résultats montrent que InterCE met environ 1 - 1.5 seconds pour traiter un fichier. Cela peut être amélioré en parallélisant les processus dans InterCE. Le ratio de requêtes par fichiers d'entrée est 0.03% et 0.04% pour les données NAT et R2N, respectivement, par conséquent, InterCE émet un nombre très limité de requêtes et rend la tâche d'annotation moins pénible pour les humains.

Cependant, InterCE fonctionne en mémorisant au lieu d'apprendre et généraliser. Par exemple, pour l'instant, InterCE décide si une entrée X doit être requêtée en cherchant dans son mémoire s'il a déjà vu un motif similaire à X; si oui, InterCE confirme que X est un motif connu, sinon, InterCE sauvegarde X dans son mémoire comme un nouveau motif. Il est préférable que InterCE soit capable d'extraire des caractéristiques des motifs traités, à partir desquelles InterCE pourrait plus rapidement décider si une entrée X a été traitée en comparant les caractéristiques de X à celles que InterCE a appris, et donc nous pourrions re-cadrer la recherche de motifs en tant qu'une tâche de classification et non qu'une tâche de recherche à force brute.

Une autre amélioration importante consiste à rendre les extracteurs individuels dans InterCE adaptatifs. À présent, l'adaptation de InterCE se trouve plutôt dans sa façon d'apprendre les nouveaux types de cycle et les nouveaux motifs, mais InterCE utilise un ensemble d'extracteurs statiques qui restent constants face aux nouveautés du flux de données. Cela pourrait se faire en deux manières: soit nous autoriserions InterCE à ajuster les hyperparamètres de ses extracteurs dynamiquement grâce à un détecteur de dérive, soit les extracteurs individuels eux-mêmes sont ajustables sans avoir besoin d'intervention de InterCE.

LSTM-AE

Pour entamer les hypothèses du bloc indicateurs, nous avons implémenté le modèle **LSTM-AE** (*Long short-term memory autoencoder*) pour apprendre des indicateurs de façon non-supervisée à partir des cycles détectés.

Un LSTM-AE est un réseau neuronal qui apprend à reproduire ses propres entrées, ce qui y permet à apprendre à déterminer les indicateurs abstraits capable à produire des reconstructions fidèles. Puisque les cycles sont des données séquentielles (séries temporelles), nous avons renforcé l'architecture traditionnelle d'un autoencodeur en ajoutant les cellules *long short-term memory* dans les couches cachées du réseau. Non seulement à reconstruire un cycle, nous cherchons aussi à apprendre au modèle à classer le contexte d'un cycle, pour but de rendre le modèle plus robuste contre les bruits contextuels. Cela a aboutit à deux versions du LSTM-AE: l'une avec seulement le décodeur, et l'autre avec un classeur partageant le même encodeur avec le décodeur.

Pour valider l'hypothèse H_f^1, nous voulons montrer que le LSTM-AE appris de façon incrémentale produit les mêmes indicateurs que celui appris hors-ligne traditionnellement. La version apprise en ligne met à jour ses poids sur un seul cycle ou un mini-lot de cycles chaque fois. Cependant, les résultats montrent que le LSTM-AE hors-ligne produit la meilleure reconstruction, tandis que la version en ligne aboutit à une reconstruction encore fautive. Donc, l'hypothèse H_f^1 n'est pas validée, et nous avons effectués les autres tests en utilisant la version hors-ligne.

L'hypothèse H_f^2 a pour but de vérifier si le LSTM-AE en ligne peut atteindre la convergence, et donc être capable de retourner des indicateurs utilisables, avant la version hors-ligne. Nous avons mesure la convergence via le temps d'entraînement de chaque modèle. Encore une fois, le modèle hors-ligne s'achève son entraînement plus vite que sa contrepartie en ligne. Donc, l'hypothèse H_f^2 n'est pas validée.

L'hypothèse H_f^3 cherche à montrer que le LSTM-AE arrive à réduire la perte d'information mieux que le système expert. Nous avons quantifié la capacité de préservation d'information de chaque modèle par une évaluation de rang: si un modèle préserve bien l'information, il doit produire des indicateurs anormaux pour des cycles anormaux, et vice-versa. Les résultats montrent que le LSTM-AE surpasse le système expert dans ce cadre d'expérimentation, ce qui confirme que le LSTM-AE apprend des indicateurs qui sont plus fidèles aux cycles d'origine que les indicateurs identifiés manuellement par le système expert. Donc, l'hypothèse H_f^3 est validée.

Un défaut crucial des autoencodeurs est qu'ils ont tendance à écraser toutes petites perturbations dans les données, pour but de produire des reconstructions plus lisses des données d'entrée. C'est pour cette raison que les autoencodeurs sont souvent utilisés pour réduire la dimensionnalité des données et pour débruiter les entrées. Toutefois, dans notre cas, les perturbations sont un facteur important qui indique la présence probable d'une ou plusieurs anomalies dans un système. Un autoencodeur qui supprime ces bruits risque de supprimer aussi les précurseurs de défaillances d'un système et affecte la précision des prévisions de défaillances. Nos travaux futurs consisteront à modifier l'architecture du réseau pour que ces bruits soient préservés et/ou à développer une fonction de perte qui atteint le même objectif.

Le LSTM-AE entraîné hors-ligne surpasse sa contre-partie en ligne car la majorité de recherche portée sur les réseaux neuronaux se concentrent sur la création d'un réseau sophistiqué, avec des millions et billions de paramètres, hors-ligne en utilisant un volume de données énorme, ce qui contredit les contraintes de l'apprentissage automatique en ligne. Pour pouvoir mieux adapter le LSTM-AE au paradigme en ligne, il faudrait favoriser l'entraînement du LSTM-AE de manière que la version en ligne est mise à jour sur le même nombre (ou presque) d'exemples de données que sa version hors-ligne pendant une itération. Nous pourrions aussi penser à ajouter l'explicabilité au LSTM-AE pour interpréter la magnitude d'impact qu'une perturbation aurait dans les indicateurs appris, ce qui enrichirait les prévisions de défaillances.

CheMoc

Pour entamer les hypothèses de santé, nous avons développé **CheMoc** (*Continuous Health Monitoring using Online Clustering*) qui découvre les profils de santé sur le flux de données et qui calcule un score de santé adaptatif pour chaque système.

Nous avons utilisé DenStream [44] comme l'algorithme de clustering central de CheMoc pour détecter et maintenir les profils de santé sous forme de clusters évoluant. Nous avons apporté quelques ajustements à DenStream pour l'aligner avec les contraintes opérationnelles dans le ferroviaire. Nous

avons aussi proposé des formules pour calculer, à n'importe quel moment donné, le degré d'anomalie d'un cluster, le score d'anomalie liée à un profil de santé spécifique d'un système, et le score de santé d'un système.

Nous avons évalué CheMoc sur les cycles de fermetures des systèmes d'accès passager de la flotte R2N. Par manque de vérité terrain, nous avons décidé de laisser les tests sur la flotte NAT pour les travaux futurs.

Les résultats expérimentaux montrent que CheMoc est capable de capter les profils de santé pertinents de la flotte, qui sont vérifiés et confirmés par un expert du domaine. Les clusters ont été aussi contre-vérifiés par l'évolution de leur degré d'anomalie et leur distance au cluster de référence. Donc, l'hypothèse H_d^1 est validée.

Pour savoir si la santé d'un système est correctement estimée, nous avons pris un système aléatoire et visualisé son évolution de santé. Nous avons observé que sa santé s'aggrave car ce système a tendance à générer des données dans les profils d'anomalie. En visualisant les clusters où ce système génère des données, nous avons vu que ceux-ci dévient effectivement du comportement typique déterminé par le cluster de référence. Cependant, faute de vérité terrain, il est difficile à conclure définitivement que la santé de ce système a été estimée bien correctement. Donc, l'hypothèse H_f^2 n'est validée que partiellement.

Pour mesure la réactivité du clustering en ligne contre le clustering hors-ligne, nous avons comparé la vitesse de convergence de CheMoc contre celle de DBSCAN [65]. Un algorithme converge s'il produit des clusters de bonne qualité, mesurée par les indices de validité de clusters (Davies-Bouldin [55] et Xie-Beni [239]). Les résultats montrent que CheMoc et DBSCAN sont compétitifs en réactivité et ni l'un ni l'autre se montre gagnant décidément. Donc, l'hypothèse H_d^3 n'est validée que partiellement.

Le test décrit précédemment permet aussi d'examiner la continuité de CheMoc et DBSCAN. Le nombre de clusters découverts par CheMoc reste stable et correspond à la nature des anomalies dans les systèmes, tandis que DBSCAN a tendance à créer des clusters excessivement pour chaque lot de données et ne donne aucun indice pour connecter les clusters entre deux lots consécutifs. Donc, CheMoc surpasse DBSCAN en continuité et l'hypothèse H_d^4 est validée.

Comme CheMoc a été développé en vue d'une surveillance en temps réel de la condition des systèmes, il est attendu que CheMoc soit efficace en termes de temps d'exécution et de mémoire consommé. Les résultats montrent que CheMoc est capable de traiter les données d'une année entière (avec plus de 1 millions de points de données) dans approximativement une heure, en mettant environ 1 minute pour chaque lot de données d'une semaine qui contient au moins 10000 exemples de données. Ensuite, le mémoire vif consommée par CheMoc se mesure jusqu'à 600 Mo en total pour traiter les données d'une année. Ainsi, CheMoc est efficace en sa demande computationnelle.

Une des améliorations importantes pour CheMoc est de permettre à un expert du domaine de donner ses retours sur la construction de clusters, et d'ajuster les clusters par conséquent. Pour l'instant, CheMoc découvre les clusters à partir des données sans être orienté par un expert du domaine. Donc, les clusters sont aussi bons qu'ils pourraient l'être uniquement sur la base des données. Par exemple, un expert peut demander de fusionner deux petits clusters qui doivent constituer le même profil de santé, de diviser un cluster trop grand qui groupe beaucoup de sous-profils de santé ensemble, ou d'ajuster la densité de détection pour pouvoir découvrir des clusters d'une plus fine granularité. L'idée est d'implémenter une boucle de rétroaction comme ce qui a été fait pour InterCE.

Étant donnée la supposition de multi-faute, un système peut générer un cycle affecté par plusieurs anomalies simultanément. En conséquence, un cycle pourrait appartenir à plusieurs clusters, ce qui n'est pas possible avec un algorithme de clustering du type partitionnement dur comme DenStream. Au lieu de cela, utiliser les algorithmes de clustering flous donnerait une plus grande flexibilité à la construction des clusters (et des profils de santé) et améliorerait la précision du calcul de santé des systèmes. À part le clustering flou, le clustering hiérarchique est une autre direction de recherche potentielle, car une anomalie peut faire partie d'une autre anomalie de plus haut niveau, ce qui permettrait d'implémenter une meilleure identification de fautes.

Bibliography

[1] Riccardo Accorsi, Riccardo Manzini, Pietro Pascarella, Marco Patella, and Simone Sassi. "Data Mining and Machine Learning for Condition-based Maintenance". In: *Procedia Manufacturing* 11 (2017). 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy, pp. 1153–1161. ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2017.07.239. URL: http://www.sciencedirect.com/science/article/pii/S235197891730447X.

[2] Ryan Prescott Adams and David J. C. MacKay. "Bayesian Online Changepoint Detection". In: *arXiv:0710.3742 [stat]* (Oct. 2007). arXiv: 0710.3742. URL: http://arxiv.org/abs/0710.3742 (visited on 10/02/2020).

[3] Sumeet Agarwal, V. Vijaya Saradhi, and Harish Karnick. "Kernel-based online machine learning and support vector reduction". In: *Neurocomputing*. Progress in Modeling, Theory, and Application of Computational Intelligenc 71.7 (Mar. 1, 2008), pp. 1230–1237. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2007.11.023. URL: https://www.sciencedirect.com/science/article/pii/S0925231208000581 (visited on 03/19/2022).

[4] Charu C. Aggarwal. "An Introduction to Outlier Analysis". en. In: *Outlier Analysis*. Ed. by Charu C. Aggarwal. Cham: Springer International Publishing, 2017, pp. 1–34. ISBN: 978-3-319-47578-3. DOI: 10.1007/978-3-319-47578-3_1. URL: https://doi.org/10.1007/978-3-319-47578-3_1 (visited on 02/13/2023).

[5] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. "A framework for clustering evolving data streams". In: *Proceedings of the 29th international conference on Very large data bases - Volume 29*. VLDB '03. Berlin, Germany: VLDB Endowment, Sept. 2003, pp. 81–92. ISBN: 978-0-12-722442-8. (Visited on 03/25/2020).

[6] Rosmaini Ahmad and Shahrul Kamaruddin. "An overview of time-based and condition-based maintenance in industrial application". In: *Computers & Industrial Engineering* 63.1 (2012), pp. 135–149. ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2012.02.002. URL: http://www.sciencedirect.com/science/article/pii/S0360835212000484.

[7] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. "Unsupervised real-time anomaly detection for streaming data". In: *Neurocomputing* 262 (2017). Online Real-Time Learning Strategies for Data Streams, pp. 134–147. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2017.04.070. URL: http://www.sciencedirect.com/science/article/pii/S0925231217309864.

[8] Suzan Alaswad and Yisha Xiang. "A review on condition-based maintenance optimization models for stochastically deteriorating system". In: *Reliability Engineering & System Safety* 157 (2017), pp. 54–63. ISSN: 0951-8320. DOI: https://doi.org/10.1016/j.ress.2016.08.009. URL: http://www.sciencedirect.com/science/article/pii/S0951832016303714.

[9] Ezilda Almeida, Carlos Ferreira, and João Gama. "Adaptive Model Rules from Data Streams". en. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 480–492. ISBN: 978-3-642-40988-2. DOI: 10.1007/978-3-642-40988-2_31.

[10] Emanuel F. Alsina, Manuel Chica, Krzysztof Trawiński, and Alberto Regattieri. "On the use of machine learning methods to predict component reliability from data-driven industrial case studies". In: *The International Journal of Advanced Manufacturing Technology* 94.5 (Feb. 2018), pp. 2419–2433. ISSN: 1433-3015. DOI: 10.1007/s00170-017-1039-x. URL: https://doi.org/10.1007/s00170-017-1039-x.

[11] Edgar J. Amaya and Alberto J. Alvares. "SIMPREBAL: An expert system for real-time fault diagnosis of hydrogenerators machinery". In: *2010 IEEE 15th Conference on Emerging Technologies Factory Automation (ETFA 2010)*. ISSN: 1946-0759. Sept. 2010, pp. 1–8. DOI: 10.1109/ETFA.2010.5641302.

[12] Amineh Amini, Hadi Saboohi, Teh Wah, and Tutut Herawan. "A Fast Density-Based Clustering Algorithm for Real-Time Internet of Things Stream". In: *TheScientificWorldJournal* 2014 (June 2014), p. 926020. DOI: 10.1155/2014/926020.

[13] Amineh Amini, Teh Ying Wah, and Hadi Saboohi. "On Density-Based Data Streams Clustering Algorithms: A Survey". en. In: *Journal of Computer Science and Technology* 29.1 (Jan. 2014), pp. 116–141. ISSN: 1860-4749. DOI: 10.1007/s11390-014-1416-y. URL: https://doi.org/10.1007/s11390-014-1416-y (visited on 03/25/2022).

[14] Samaneh Aminikhanghahi and Diane J. Cook. "A survey of methods for time series change point detection". In: *Knowledge and Information Systems* 51.2 (May 2017), pp. 339–367. ISSN: 0219-1377. DOI: 10.1007/s10115-016-0987-z. URL: https://doi.org/10.1007/s10115-016-0987-z (visited on 09/30/2020).

[15] N. Amruthnath and T. Gupta. "Fault class prediction in unsupervised learning using model-based clustering approach". In: *2018 International Conference on Information and Computer Technologies (ICICT)*. Mar. 2018, pp. 5–12. DOI: 10.1109/INFOCT.2018.8356831.

[16] Dawn An, Nam H. Kim, and Joo-Ho Choi. "Practical options for selecting data-driven or physics-based prognostics algorithms with reviews". In: *Reliability Engineering & System Safety* 133 (2015), pp. 223–236. ISSN: 0951-8320. DOI: https://doi.org/10.1016/j.ress.2014.09.014. URL: http://www.sciencedirect.com/science/article/pii/S0951832014002245.

[17] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. "OPTICS: ordering points to identify the clustering structure". In: *ACM SIGMOD Record* 28.2 (June 1999), pp. 49–60. ISSN: 0163-5808. DOI: 10.1145/304181.304187. URL: https://doi.org/10.1145/304181.304187 (visited on 02/15/2023).

[18] J. F. Archard. "Contact and Rubbing of Flat Surfaces". In: *Journal of Applied Physics* 24.8 (1953), pp. 981–988. DOI: 10.1063/1.1721448. eprint: https://doi.org/10.1063/1.1721448. URL: https://doi.org/10.1063/1.1721448.

[19] *ASTM G40-10b: Standard Terminology Relating to Wear and Erosion*. www.astm.org. ASTM International. West Conshohocken, PA, 2010.

[20] V. Atamuradov, F. Camci, S. Baskan, and M. Sevkli. "Failure diagnostics for railway point machines using expert systems". In: *2009 IEEE International Symposium on Diagnostics for Electric Machines, Power Electronics and Drives*. Aug. 2009, pp. 1–5. DOI: 10.1109/DEMPED.2009.5292755.

[21] Gurkan Aydemir and Burak Acar. "Anomaly monitoring improves remaining useful life estimation of industrial machinery". In: *Journal of Manufacturing Systems* 56 (July 1, 2020), pp. 463–469. ISSN: 0278-6125. DOI: 10.1016/j.jmsy.2020.06.014. URL: https://www.sciencedirect.com/science/article/pii/S0278612520301060 (visited on 09/28/2021).

[22] Maroua Bahri, Albert Bifet, Silviu Maniu, and Heitor Murilo Gomes. "Survey on feature transformation techniques for data streams". In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. IJCAI'20. Yokohama, Yokohama, Japan, Jan. 7, 2021, pp. 4796–4802. ISBN: 978-0-9992411-6-5. (Visited on 03/15/2022).

[23] Marcia Baptista, Shankar Sankararaman, Ivo. P. de Medeiros, Cairo Nascimento, Helmut Prendinger, and Elsa M.P. Henriques. "Forecasting fault events for predictive maintenance using data-driven techniques and ARMA modeling". In: *Computers & Industrial Engineering* 115 (2018), pp. 41–53. ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2017.10.033. URL: http://www.sciencedirect.com/science/article/pii/S036083521730520X.

[24] P. Baraldi, F. Mangili, and E. Zio. "A Kalman Filter-Based Ensemble Approach With Application to Turbine Creep Prognostics". In: *IEEE Transactions on Reliability* 61.4 (Dec. 2012), pp. 966–977. ISSN: 1558-1721. DOI: 10.1109/TR.2012.2221037.

[25] Jaouher Ben Ali, Lotfi Saidi, Salma Harrath, Eric Bechhoefer, and Mohamed Benbouzid. "Online automatic diagnosis of wind turbine bearings progressive degradations under real experimental conditions based on unsupervised machine learning". In: *Applied Acoustics* 132 (Mar. 1, 2018), pp. 167–181. ISSN: 0003-682X. DOI: 10.1016/j.apacoust.2017.11.021. (Visited on 02/09/2022).

[26] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation Learning: A Review and New Perspectives". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (Aug. 2013). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1798–1828. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2013.50.

[27] Marcus Bengtsson, Ella Olsson, Peter Funk, and Mats Jackson. "Technical design of condition based maintenance system—A case study using sound analysis and case-based reasoning". In: (Jan. 2004).

[28] Elmer L. Peterson Benjamin S. Blanchard Dinesh C. Verma. *Maintainability: A Key to Effective Serviceability and Maintenance Management*. John Wiley & Sons, 1995. ISBN: 0-471-59132-7.

[29] Andrew Van Benschoten, Austin Ouyang, Francisco Bischoff, and Tyler Marrs. "MPA: a novel cross-language API for time series analysis". In: *Journal of Open Source Software* 5.49 (2020), p. 2179. DOI: 10.21105/joss.02179. URL: https://doi.org/10.21105/joss.02179.

[30] Jürgen Beringer and Eyke Hüllermeier. "An Efficient Algorithm for Instance-Based Learning on Data Streams". en. In: *Advances in Data Mining. Theoretical Aspects and Applications*. Ed. by Petra Perner. Berlin, Heidelberg: Springer, 2007, pp. 34–48. ISBN: 978-3-540-73435-2. DOI: 10.1007/978-3-540-73435-2_4.

[31] Andrey Besedin, Pierre Blanchart, Michel Crucianu, and Marin Ferecatu. "Evolutive deep models for online learning on data streams with no storage". In: ECML/PKDD 2017 Workshop on Large-scale Learning from Data Streams in Evolving Environments. Sept. 18, 2017. URL: https://hal-cea.archives-ouvertes.fr/cea-01832986 (visited on 03/19/2022).

[32] Albert Bifet and Ricard Gavaldà. "Adaptive Learning from Evolving Data Streams". en. In: *Advances in Intelligent Data Analysis VIII*. Ed. by Niall M. Adams, Céline Robardet, Arno Siebes, and Jean-François Boulicaut. Berlin, Heidelberg: Springer, 2009, pp. 249–260. ISBN: 978-3-642-03915-7. DOI: 10.1007/978-3-642-03915-7_22.

[33] Albert Bifet and Ricard Gavaldà. "Learning from Time-Changing Data with Adaptive Windowing". In: vol. 7. Apr. 2007. DOI: 10.1137/1.9781611972771.42.

[34] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen, and Thomas Seidl. "MOA: Massive Online Analysis, a Framework for Stream Classification and Clustering". In: *Proceedings of the First Workshop on Applications of Pattern Analysis*. Proceedings of the First Workshop on Applications of Pattern Analysis. ISSN: 1938-7228. PMLR, Sept. 30, 2010, pp. 44–50. URL: https://proceedings.mlr.press/v11/bifet10a.html (visited on 03/15/2022).

[35] Avrim Blum and Tom Mitchell. "Combining labeled and unlabeled data with co-training". In: *Proceedings of the eleventh annual conference on Computational learning theory*. COLT' 98. New York, NY, USA: Association for Computing Machinery, July 24, 1998, pp. 92–100. ISBN: 978-1-58113-057-7. DOI: 10.1145/279943.279962. URL: https://doi.org/10.1145/279943.279962 (visited on 03/20/2022).

[36] Paul Boniol, Michele Linardi, Federico Roncallo, and Themis Palpanas. "Automated Anomaly Detection in Large Sequences". In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. ISSN: 2375-026X. Apr. 2020, pp. 1834–1837. DOI: 10.1109/ICDE48307.2020.00182.

[37] Léon Bottou. "Stochastic Gradient Descent Tricks". In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 421–436. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_25. URL: https://doi.org/10.1007/978-3-642-35289-8_25 (visited on 03/19/2022).

[38] Ikram Bouchikhi, André Ferrari, Cédric Richard, Anthony Bourrier, and Marc Bernot. "Kernel Based Online Change Point Detection". In: *2019 27th European Signal Processing Conference (EUSIPCO)*. ISSN: 2076-1465. Sept. 2019, pp. 1–5. DOI: 10.23919/EUSIPCO.2019.8902582.

[39] R. H. W. Brook and J. S. C. Parry. "Cumulative Damage in Fatigue: A Step towards Its Understanding". In: *Journal of Mechanical Engineering Science* 11.3 (1969), pp. 243–255. DOI: 10.1243/JMES_JOUR_1969_011_032_02. URL: https://doi.org/10.1243/JMES_JOUR_1969_011_032_02.

[40] The British Standards Institution. *Maintenance: Maintenance terminology*. Standard. The British Standards Institution, Dec. 2017.

[41] Carl S. Byington, Matthew Watson, Michael J. Roemer, Thomas R. Galie, Jack J. McGroarty, and Christopher Savage. *Prognostic Enhancements to Gas Turbine Diagnostic Systems*. en. Tech. rep. IMPACT TECHNOLOGIES LLC STATE COLLEGE PA, Jan. 2003. URL: https://apps.dtic.mil/docs/citations/ADA457841 (visited on 03/31/2020).

[42] A. Cachada, J. Barbosa, P. Leitño, C. A. S. Gcraldcs, L. Deusdado, J. Costa, C. Teixeira, J. Teixeira, A. H. J. Moreira, P. M. Moreira, and L. Romero. "Maintenance 4.0: Intelligent and Predictive Maintenance System Architecture". In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. Sept. 2018, pp. 139–146. DOI: 10.1109/ETFA.2018.8502489.

[43] M. Canizo, E. Onieva, A. Conde, S. Charramendieta, and S. Trujillo. "Real-time predictive maintenance for wind turbines using Big Data frameworks". In: *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*. June 2017, pp. 70–77. DOI: 10.1109/ICPHM.2017.7998308.

[44] Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. "Density-Based Clustering over an Evolving Data Stream with Noise". In: *Proceedings of the Sixth SIAM International Conference on Data Mining, April 20-22, 2006, Bethesda, MD, USA*. Vol. 2006. 2006, pp. 328–339. DOI: 10.1137/1.9781611972764.29.

[45] Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. "Density-Based Clustering over an Evolving Data Stream with Noise". In: *Proceedings of the Sixth SIAM International Conference on Data Mining, April 20-22, 2006, Bethesda, MD, USA*. Vol. 2006. Apr. 2006. DOI: 10.1137/1.9781611972764.29.

[46] Matthias Carnein, Dennis Assenmacher, and Heike Trautmann. "An Empirical Comparison of Stream Clustering Algorithms". In: *Proceedings of the Computing Frontiers Conference*. CF'17. New York, NY, USA: Association for Computing Machinery, May 2017, pp. 361–366. ISBN: 978-1-4503-4487-6. DOI: 10.1145/3075564.3078887. URL: https://doi.org/10.1145/3075564.3078887 (visited on 03/25/2022).

[47] Matthias Carnein and Heike Trautmann. "Optimizing Data Stream Representation: An Extensive Survey on Stream Clustering Algorithms". In: *Business & Information Systems Engineering* 61.3 (June 1, 2019), pp. 277–297. ISSN: 1867-0202. DOI: 10.1007/s12599-019-00576-5. URL: https://doi.org/10.1007/s12599-019-00576-5 (visited on 03/15/2022).

[48] Thyago P. Carvalho, Fabrízzio A. A. M. N. Soares, Roberto Vita, Roberto da P. Francisco, João P. Basto, and Symone G. S. Alcalá. "A systematic literature review of machine learning methods applied to predictive maintenance". In: *Computers & Industrial Engineering* 137 (2019), p. 106024. ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2019.106024. URL: http://www.sciencedirect.com/science/article/pii/S0360835219304838.

[49] P.K. Chande and S.V. Tokekar. "Expert-based maintenance: a study of its effectiveness". In: *IEEE Transactions on Reliability* 47.1 (Mar. 1998). Conference Name: IEEE Transactions on Reliability, pp. 53–58. ISSN: 1558-1721. DOI: 10.1109/24.690904.

[50] Yixin Chen and Li Tu. "Density-based clustering for real-time stream data". In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '07. San Jose, California, USA: Association for Computing Machinery, Aug. 2007, pp. 133–142. ISBN: 978-1-59593-609-7. DOI: 10.1145/1281192.1281210. URL: https://doi.org/10.1145/1281192.1281210 (visited on 03/25/2020).

[51] Heng-Tze Cheng, Feng-Tso Sun, Martin Griss, Paul Davis, Jianguo Li, and Di You. "NuActiv: recognizing unseen new activities using semantic attribute-based learning". In: *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. MobiSys '13. New York, NY, USA: Association for Computing Machinery, June 2013, pp. 361–374. ISBN: 978-1-4503-1672-9. DOI: 10.1145/2462456.2464438. URL: https://doi.org/10.1145/2462456.2464438 (visited on 09/10/2021).

[52] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. "Online Passive-Aggressive Algorithms". In: *The Journal of Machine Learning Research* 7 (Dec. 2006), pp. 551–585. ISSN: 1532-4435.

[53] Adrian Cubillo, Suresh Perinpanayagam, and Manuel Esperon-Miguez. "A review of physics-based models in prognostics: Application to gears and bearings of rotating machinery". In: *Advances in Mechanical Engineering* 8.8 (2016), p. 1687814016664660. DOI: 10.1177/1687814016664660. eprint: https://doi.org/10.1177/1687814016664660. URL: https://doi.org/10.1177/1687814016664660.

[54] Narjes Davari, Bruno Veloso, Gustavo De Assis Costa, Pedro Pereira, Rita Ribeiro, and João Gama. "A Survey on Data-Driven Predictive Maintenance for the Railway Industry". In: *Sensors* 21 (Sept. 8, 2021), p. 5739. DOI: 10.3390/s21175739.

[55] David L. Davies and Donald W. Bouldin. "A Cluster Separation Measure". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1.2 (Apr. 1979). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 224–227. ISSN: 1939-3539. DOI: 10.1109/TPAMI.1979.4766909.

[56] E. Deloux, B. Castanier, and C. Bérenguer. "Predictive maintenance policy for a gradually deteriorating system subject to stress". In: *Reliability Engineering & System Safety* 94.2 (2009), pp. 418–431. ISSN: 0951-8320. DOI: https://doi.org/10.1016/j.ress.2008.04.002. URL: http://www.sciencedirect.com/science/article/pii/S0951832008001336.

[57] F. Desobry, M. Davy, and C. Doncarli. "An online kernel change detection algorithm". In: *IEEE Transactions on Signal Processing* 53.8 (Aug. 2005). Conference Name: IEEE Transactions on Signal Processing, pp. 2961–2974. ISSN: 1941-0476. DOI: 10.1109/TSP.2005.851098.

[58] F. Di Marco, L. Fortuna, A. Gallo, and G. Nunnari. "An Expert System for On-Line Fault Diagnosis and Control of a Railway Locomotive". en. In: *IFAC Proceedings Volumes*. IFAC Workshop on Motion Control for Intelligent Automation, Perugia, Italy, 27-29 October 1992 25.29, Part 1 (Oct. 1992), pp. 217–221. ISSN: 1474-6670. DOI: 10.1016/S1474-6670(17)50569-8. URL: https://www.sciencedirect.com/science/article/pii/S1474667017505698 (visited on 03/05/2022).

[59] Alberto Diez, Nguyen Lu Dang Khoa, Mehrisadat Makki Alamdari, Yang Wang, Fang Chen, and Peter Runcie. "A clustering approach for structural health monitoring on bridges". In: *Journal of Civil Structural Health Monitoring* 6.3 (July 1, 2016), pp. 429–445. ISSN: 2190-5479. DOI: 10.1007/s13349-016-0160-0. (Visited on 02/09/2022).

[60] Pedro Domingos and Geoff Hulten. "Mining high-speed data streams". In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '00. Boston, Massachusetts, USA: Association for Computing Machinery, Aug. 2000, pp. 71–80. ISBN: 978-1-58113-233-5. DOI: 10.1145/347090.347107. URL: https://doi.org/10.1145/347090.347107 (visited on 03/17/2020).

[61] Agnieszka Duraj and Piotr S. Szczepaniak. "Outlier Detection in Data Streams — A Comparative Study of Selected Methods". en. In: *Procedia Computer Science*. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 25th International Conference KES2021 192 (Jan. 2021), pp. 2769–2778. ISSN: 1877-0509. DOI: 10.1016/j.procs.2021.09.047. URL: https://www.sciencedirect.com/science/article/pii/S1877050921017841 (visited on 10/02/2021).

[62] Charles E. Ebeling. *An Introduction To Reliability and Maintainability Engineering*. McGraw-Hill, 1997. ISBN: 0070188521.

[63] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Michael Wimmer, and Xiaowei Xu. "Incremental Clustering for Mining in a Data Warehousing Environment". In: *Proceedings of the 24rd International Conference on Very Large Data Bases*. VLDB '98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Aug. 1998, pp. 323–333. ISBN: 978-1-55860-566-4. (Visited on 07/18/2021).

[64] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, Aug. 1996, pp. 226–231. (Visited on 03/25/2020).

[65] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, Aug. 1996, pp. 226–231. (Visited on 03/25/2020).

[66] Johann Faouzi and Hicham Janati. "pyts: A Python Package for Time Series Classification". In: *Journal of Machine Learning Research* 21.46 (2020), pp. 1–6. ISSN: 1533-7928. URL: http://jmlr.org/papers/v21/19-763.html (visited on 02/12/2023).

[67] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. "Testing the manifold hypothesis". en. In: *Journal of the American Mathematical Society* 29.4 (Oct. 2016), pp. 983–1049. ISSN: 0894-0347, 1088-6834. DOI: 10.1090/jams/852. URL: https://www.ams.org/jams/2016-29-04/S0894-0347-2016-00852-4/ (visited on 12/15/2022).

[68] Xuning Feng, Caihao Weng, Xiangming He, Xuebing Han, Languang Lu, Dongsheng Ren, and Minggao Ouyang. "Online State-of-Health Estimation for Li-Ion Battery Using Partial Charging Segment Based on Support Vector Machine". In: *IEEE Transactions on Vehicular Technology* 68.9 (Sept. 2019). Conference Name: IEEE Transactions on Vehicular Technology, pp. 8583–8592. ISSN: 1939-9359. DOI: 10.1109/TVT.2019.2927120.

[69] Marta Fernandes, Alda Canito, Verónica Bolón-Canedo, Luís Conceição, Isabel Praça, and Goreti Marreiros. "Data analysis and feature selection for predictive maintenance: A case-study in the metallurgic industry". In: *International Journal of Information Management* 46 (2019), pp. 252–262. ISSN: 0268-4012. DOI: https://doi.org/10.1016/j.ijinfomgt.2018.10.006. URL: http://www.sciencedirect.com/science/article/pii/S0268401218304699.

[70] Kyle D. Feuz, Diane J. Cook, Cody Rosasco, Kayela Robertson, and Maureen Schmitter-Edgecombe. "Automated Detection of Activity Transitions for Prompting". In: *IEEE Transactions on Human-Machine Systems* 45.5 (Oct. 2015). Conference Name: IEEE Transactions on Human-Machine Systems, pp. 575–585. ISSN: 2168-2305. DOI: 10.1109/THMS.2014.2362529.

[71] Agostino Forestiero, Clara Pizzuti, and Giandomenico Spezzano. "FlockStream: A Bio-Inspired Algorithm for Clustering Evolving Data Streams". In: *2009 21st IEEE International Conference on Tools with Artificial Intelligence*. ISSN: 2375-0197. Nov. 2009, pp. 1–8. DOI: 10.1109/ICTAI.2009.60.

[72] Diego Galar, Adithya Thaduri, Marcantonio Catelani, and Lorenzo Ciani. "Context awareness for maintenance decision making: A diagnosis and prognosis approach". en. In: *Measurement* 67 (May 2015), pp. 137–150. ISSN: 0263-2241. DOI: 10.1016/j.measurement.2015.01.015. URL: https://www.sciencedirect.com/science/article/pii/S0263224115000408 (visited on 03/05/2022).

[73] Joao Gama. *Knowledge Discovery from Data Streams*. 1st. Chapman & Hall/CRC, 2010. ISBN: 978-1-4398-2611-9.

[74] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. "Issues in evaluation of stream learning algorithms". In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '09. New York, NY, USA: Association for Computing Machinery, June 2009, pp. 329–338. ISBN: 978-1-60558-495-9. DOI: 10.1145/1557019.1557060. URL: https://doi.org/10.1145/1557019.1557060 (visited on 03/15/2022).

[75] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. "A survey on concept drift adaptation". In: *ACM Computing Surveys* 46.4 (Mar. 2014), 44:1–44:37. ISSN: 0360-0300. DOI: 10.1145/2523813. URL: https://doi.org/10.1145/2523813 (visited on 03/19/2020).

[76] Jie Gao, Myeongsu Kang, Jing Tian, Lifeng Wu, and Michael Pecht. "Unsupervised Locality-Preserving Robust Latent Low-Rank Recovery-Based Subspace Clustering for Fault Diagnosis". In: *IEEE Access* 6 (2018). Conference Name: IEEE Access, pp. 52345–52354. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2869923.

[77] Jing Gao, Jianzhong Li, Zhaogong Zhang, and Pang-Ning Tan. "An Incremental Data Stream Clustering Algorithm Based on Dense Units Detection". en. In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Tu Bao Ho, David Cheung, and Huan Liu. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 420–425. ISBN: 978-3-540-31935-1. DOI: 10.1007/11430919_49.

[78] Mohammed Ghesmoune, Mustapha Lebbah, and Hanene Azzag. "State-of-the-art on clustering data streams". In: *Big Data Analytics* 1.1 (Dec. 2016), p. 13. ISSN: 2058-6345. DOI: 10.1186/s41044-016-0011-3. URL: https://doi.org/10.1186/s41044-016-0011-3 (visited on 03/25/2022).

[79] Heitor Murilo Gomes, Jean Paul Barddal, Luis Boiko Ferreira, and Albert Bifet. "Adaptive random forests for data stream regression". In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*. Apr. 2018.

[80] Heitor Murilo Gomes, Maciej Grzenda, Rodrigo Mello, Jesse Read, Minh Huong Le Nguyen, and Albert Bifet. "A Survey on Semi-supervised Learning for Delayed Partially Labelled Data Streams". In: *ACM Computing Surveys* 55.4 (Nov. 2022), 75:1–75:42. ISSN: 0360-0300. DOI: 10.1145/3523055. URL: https://dl.acm.org/doi/10.1145/3523055.

[81] Heitor Murilo Gomes, Jesse Read, Albert Bifet, Jean Paul Barddal, and João Gama. "Machine learning for streaming data: state of the art, challenges, and opportunities". In: *ACM SIGKDD Explorations Newsletter* 21.2 (Nov. 2019), pp. 6–22. ISSN: 1931-0145. DOI: 10.1145/3373464.3373470. URL: https://doi.org/10.1145/3373464.3373470 (visited on 03/17/2020).

[82] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[83] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[84] Liang Guo, Yaguo Lei, Naipeng Li, Tao Yan, and Ningbo Li. "Machinery health indicator construction based on convolutional neural networks considering trend burr". In: *Neurocomputing* 292 (May 31, 2018), pp. 142–150. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2018.02.083. URL: https://www.sciencedirect.com/science/article/pii/S0925231218302583 (visited on 09/30/2021).

[85] Liang Guo, Naipeng Li, Feng Jia, Yaguo Lei, and Jing Lin. "A recurrent neural network based health indicator for remaining useful life prediction of bearings". In: *Neurocomputing* 240 (May 31, 2017), pp. 98–109. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2017.02.045. URL: https://www.sciencedirect.com/science/article/pii/S0925231217303363 (visited on 09/30/2021).

[86] Manish Gupta, Jing Gao, Charu C. Aggarwal, and Jiawei Han. "Outlier Detection for Temporal Data: A Survey". In: *IEEE Transactions on Knowledge and Data Engineering* 26.9 (Sept. 2014). Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 2250–2267. ISSN: 1558-2191. DOI: 10.1109/TKDE.2013.184.

[87] Raia Hadsell, Dushyant Rao, Andrei A. Rusu, and Razvan Pascanu. "Embracing Change: Continual Learning in Deep Neural Networks". en. In: *Trends in Cognitive Sciences* 24.12 (Dec. 2020), pp. 1028–1040. ISSN: 1364-6613. DOI: 10.1016/j.tics.2020.09.004. URL: https://www.sciencedirect.com/science/article/pii/S1364661320302199 (visited on 03/05/2023).

[88] M. Hahsler and M. Bolaños. "Clustering Data Streams Based on Shared Density between Micro-Clusters". In: *IEEE Transactions on Knowledge and Data Engineering* 28.6 (2016), pp. 1449–1461.

[89] Ahsanul Haque, Latifur Khan, and Michael Baron. "SAND: semi-supervised adaptive novel class detection and classification over data stream". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI'16. Phoenix, Arizona: AAAI Press, Feb. 2016, pp. 1652–1658. (Visited on 03/31/2022).

[90] Ahsanul Haque, Latifur Khan, Michael Baron, Bhavani Thuraisingham, and Charu Aggarwal. "Efficient handling of concept drift and concept evolution over Stream Data". In: *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. May 2016, pp. 481–492. DOI: 10.1109/ICDE.2016.7498264.

[91] H. M. Hashemian and W. C. Bean. "State-of-the-Art Predictive Maintenance Techniques". In: *IEEE Transactions on Instrumentation and Measurement* 60.10 (Oct. 2011), pp. 3480–3492.

[92] Marwan Hassani, Pascal Spaus, Mohamed Medhat Gaber, and Thomas Seidl. "Density-Based Projected Clustering of Data Streams". en. In: *Scalable Uncertainty Management*. Ed. by Eyke Hüllermeier, Sebastian Link, Thomas Fober, and Bernhard Seeger. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 311–324. ISBN: 978-3-642-33362-0. DOI: 10.1007/978-3-642-33362-0_24.

[93] F. O. Heimes. "Recurrent neural networks for remaining useful life estimation". In: *2008 International Conference on Prognostics and Health Management*. ISSN: null. 2008, pp. 1–6. DOI: 10.1109/PHM.2008.4711422.

[94] F. O. Heimes. "Recurrent neural networks for remaining useful life estimation". In: *2008 International Conference on Prognostics and Health Management*. Oct. 2008, pp. 1–6. DOI: 10.1109/PHM.2008.4711422.

[95] Aiwina Heng, Sheng Zhang, Andy C.C. Tan, and Joseph Mathew. "Rotating machinery prognostics: State of the art, challenges and opportunities". In: *Mechanical Systems and Signal Processing* 23.3 (2009), pp. 724–739. ISSN: 0888-3270. DOI: https://doi.org/10.1016/j.ymssp.2008.06.009. URL: http://www.sciencedirect.com/science/article/pii/S0888327008001489.

[96] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735 (visited on 09/02/2021).

[97] M. R. Hoeprich. "Rolling Element Bearing Fatigue Damage Propagation". In: *Journal of Tribology* 114.2 (Apr. 1992), pp. 328–333. ISSN: 0742-4787. DOI: 10.1115/1.2920891. eprint: https://asmedigitalcollection.asme.org/tribology/article-pdf/114/2/328/5583421/328_1.pdf. URL: https://doi.org/10.1115/1.2920891.

[98] Steven C. H. Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. "Online learning: A comprehensive survey". en. In: *Neurocomputing* 459 (Oct. 2021), pp. 249–289. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2021.04.112. URL: https://www.sciencedirect.com/science/article/pii/S0925231221006706 (visited on 10/15/2021).

[99] Kevin Shen Hoong Ong, Dusit Niyato, and Chau Yuen. "Predictive Maintenance for Edge-Based Sensor Networks: A Deep Reinforcement Learning Approach". In: *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*. 2020 IEEE 6th World Forum on Internet of Things (WF-IoT). June 2020, pp. 1–6. DOI: 10.1109/WF-IoT48130.2020.9221098.

[100] Geoff Hulten, Laurie Spencer, and Pedro Domingos. "Mining time-changing data streams". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '01. New York, NY, USA: Association for Computing Machinery, Aug. 2001, pp. 97–106. ISBN: 978-1-58113-391-2. DOI: 10.1145/502512.502529. URL: https://doi.org/10.1145/502512.502529 (visited on 03/16/2022).

[101] *1232-2010 - IEEE Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE)*. Standard. Institute of Electrical and Electronics Engineers, Dec. 2010.

[102] "IEEE Standard Framework for Prognostics and Health Management of Electronic Systems". In: *IEEE Std 1856-2017* (2017), pp. 1–31.

[103] Elena Ikonomovska, João Gama, and Sašo Džeroski. "Learning model trees from evolving data streams". en. In: *Data Mining and Knowledge Discovery* 23.1 (July 2011), pp. 128–168. ISSN: 1573-756X. DOI: 10.1007/s10618-010-0201-y. URL: https://doi.org/10.1007/s10618-010-0201-y (visited on 05/03/2020).

[104] Vamsi Inturi, N. Shreyas, Karthick Chetti, and G. R. Sabareesh. "Comprehensive fault diagnostics of wind turbine gearbox through adaptive condition monitoring scheme". In: *Applied Acoustics* (Nov. 1, 2020), p. 107738. ISSN: 0003-682X. DOI: 10.1016/j.apacoust.2020.107738. URL: http://www.sciencedirect.com/science/article/pii/S0003682X20308422 (visited on 11/03/2020).

[105] Charlie Isaksson, Margaret H. Dunham, and Michael Hahsler. "SOStream: Self Organizing Density-Based Clustering over Data Stream". en. In: *Machine Learning and Data Mining in Pattern Recognition*. Ed. by Petra Perner. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 264–278. ISBN: 978-3-642-31537-4. DOI: 10.1007/978-3-642-31537-4_21.

[106] *Condition monitoring and diagnostics of machines - Vocabulary*. Standard. International Organization for Standardization, Sept. 2012.

[107] *Condition monitoring and diagnostics of machines — Data interpretation and diagnostics techniques — Part 2: Data-driven applications*. Standard. International Organization for Standardization, Apr. 2015.

[108] *Condition monitoring and diagnostics of machines — Data processing, communication and presentation — Part 1: General guidelines*. Standard. International Organization for Standardization, Mar. 2003.

[109] *Condition monitoring and diagnostics of machines — Data processing, communication and presentation — Part 2: Data processing*. Standard. International Organization for Standardization, July 2007.

[110] *Condition monitoring and diagnostics of machines — Data processing, communication and presentation — Part 3: Communication*. Standard. International Organization for Standardization, Feb. 2012.

[111] *Condition monitoring and diagnostics of machine systems — Data processing, communication and presentation — Part 4: Presentation*. Standard. International Organization for Standardization, Dec. 2015.

[112] *Condition monitoring and diagnostics of machines — Prognostics — Part 1: General guidelines*. Standard. International Organization for Standardization, Sept. 2015.

[113] Andrew K.S. Jardine, Daming Lin, and Dragan Banjevic. "A review on machinery diagnostics and prognostics implementing condition-based maintenance". In: *Mechanical Systems and Signal Processing* 20.7 (2006), pp. 1483–1510. ISSN: 0888-3270. DOI: https://doi.org/10.1016/j.ymssp.2005.09.012. URL: http://www.sciencedirect.com/science/article/pii/S0888327005001512.

[114] Kalervo Järvelin and Jaana Kekäläinen. "Cumulated gain-based evaluation of IR techniques". In: *ACM Transactions on Information Systems* 20.4 (Oct. 2002), pp. 422–446. ISSN: 1046-8188. DOI: 10.1145/582415.582418. URL: https://doi.org/10.1145/582415.582418 (visited on 04/16/2022).

[115] George H. John and Pat Langley. "Estimating continuous distributions in Bayesian classifiers". In: *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*. UAI'95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Aug. 1995, pp. 338–345. ISBN: 978-1-55860-385-1. (Visited on 03/16/2022).

[116] Ireneusz J. Jóźwiak. "An introduction to the studies of reliability of systems using the Weibull proportional hazards model". In: *Microelectronics Reliability* 37.6 (1997), pp. 915–918. ISSN: 0026-2714. DOI: https://doi.org/10.1016/S0026-2714(96)00285-5. URL: http://www.sciencedirect.com/science/article/pii/S0026271496002855.

[117] E. Kandare, S. Feih, B.Y. Lattimer, and A.P. Mouritz. "Larson–Miller Failure Modeling of Aluminum in Fire". In: *Metallurgical and Materials Transactions A* 41.12 (Dec. 1, 2010), pp. 3091–3099. ISSN: 1543-1940. DOI: 10.1007/s11661-010-0369-1. URL: https://doi.org/10.1007/s11661-010-0369-1 (visited on 03/05/2022).

[118] Richard M. Karp. "On-Line Algorithms Versus Off-Line Algorithms: How Much is it Worth to Know the Future?" In: *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture - Information Processing '92, Volume 1 - Volume I*. NLD: North-Holland Publishing Co., Sept. 1992, pp. 416–429. ISBN: 978-0-444-89747-3. (Visited on 03/17/2020).

[119] Karamjit Kaur, Matt Selway, Georg Grossmann, Markus Stumptner, and Alan Johnston. "Towards an Open-standards Based Framework for Achieving Condition-based Predictive Maintenance". In: *Proceedings of the 8th International Conference on the Internet of Things*. IOT '18. ACM, 2018, 16:1–16:8. ISBN: 978-1-4503-6564-2. DOI: 10.1145/3277593.3277608. URL: http://doi.acm.org/10.1145/3277593.3277608.

[120] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. "Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases". en. In: *Knowledge and Information Systems* 3.3 (Aug. 2001), pp. 263–286. ISSN: 0219-1377. DOI: 10.1007/PL00011669. URL: https://doi.org/10.1007/PL00011669 (visited on 09/02/2022).

[121] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. "Segmenting time series: a survey and novel approach". In: *Data Mining in Time Series Databases*. Vol. Volume 57. Series in Machine Perception and Artificial Intelligence Volume 57. WORLD SCIENTIFIC, June 2004, pp. 1–21. ISBN: 978-981-238-290-0. DOI: 10.1142/9789812565402_0001. URL: https://www.worldscientific.com/doi/abs/10.1142/9789812565402_0001 (visited on 03/26/2021).

[122] Mohammad Mahmudur Rahman Khan, Md. Abu Bakr Siddique, Rezoana Bente Arif, and Mahjabin Rahman Oishe. "ADBSCAN: Adaptive Density-Based Spatial Clustering of Applications with Noise for Identifying Clusters with Varying Densities". In: *2018 4th International Conference on Electrical Engineering and Information Communication Technology (iCEEiCT)*. Sept. 2018, pp. 107–111. DOI: 10.1109/CEEICT.2018.8628138.

[123] Samir Khan and Takehisa Yairi. "A review on the application of deep learning in system health management". In: *Mechanical Systems and Signal Processing* 107 (July 1, 2018), pp. 241–265. ISSN: 0888-3270. DOI: 10.1016/j.ymssp.2017.11.024. (Visited on 02/06/2021).

[124] R. Killick, P. Fearnhead, and I. A. Eckley. "Optimal Detection of Changepoints With a Linear Computational Cost". In: *Journal of the American Statistical Association* 107.500 (Dec. 2012). Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/01621459.2012.737745, pp. 1590–1598. ISSN: 0162-1459. DOI: 10.1080/01621459.2012.737745. URL: https://doi.org/10.1080/01621459.2012.737745 (visited on 08/25/2022).

[125] J. Kivinen, A.J. Smola, and R.C. Williamson. "Online learning with kernels". In: *IEEE Transactions on Signal Processing* 52.8 (Aug. 2004). Conference Name: IEEE Transactions on Signal Processing, pp. 2165–2176. ISSN: 1941-0476. DOI: 10.1109/TSP.2004.830991.

[126] Martin Kleppmann. *Designing Data-Intensive Applications*. 1st ed. Vol. 1. O'Reilly Media Inc., Aug. 2018. ISBN: 978-1-449-37332-0.

[127] G. A. Klutke, P. C. Kiessler, and M. A. Wortman. "A critical look at the bathtub curve". In: *IEEE Transactions on Reliability* 52.1 (Mar. 2003), pp. 125–129. DOI: 10.1109/TR.2002.804492.

[128] Panagiotis Korvesis. "Machine Learning for Predictive Maintenance in Aviation". These de doctorat. Université Paris-Saclay (ComUE), Nov. 2017. URL: https://www.theses.fr/2017SACLX093 (visited on 03/11/2022).

[129] Panagiotis Korvesis, Stephane Besseau, and Michalis Vazirgiannis. "Predictive Maintenance in Aviation: Failure Prediction from Post-Flight Reports". In: *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. 2018 IEEE 34th International Conference on Data Engineering (ICDE). ISSN: 2375-026X. Apr. 2018, pp. 1414–1422. DOI: 10.1109/ICDE.2018.00160.

[130] Dominik Kozjek, Andreja Malus, and Rok Vrabič. "Multi-objective adjustment of remaining useful life predictions based on reinforcement learning". In: *Procedia CIRP*. 53rd CIRP Conference on Manufacturing Systems 2020 93 (Jan. 1, 2020), pp. 425–430. ISSN: 2212-8271. DOI: 10.1016/j.procir.2020.03.051. URL: http://www.sciencedirect.com/science/article/pii/S2212827120306582 (visited on 09/28/2020).

[131] Philipp Kranen, Ira Assent, Corinna Baldauf, and Thomas Seidl. "Self-Adaptive Anytime Stream Clustering". In: *2009 Ninth IEEE International Conference on Data Mining*. 2009 Ninth IEEE International Conference on Data Mining. ISSN: 2374-8486. Dec. 2009, pp. 249–258. DOI: 10.1109/ICDM.2009.47.

[132] Vikram Krishnamurthy, Kusha Nezafati, and Vikrant Singh. "Application of Machine Learning and Spatial Bootstrapping to Image Processing for Predictive Maintenance". In: *2019 IEEE International Conference on Big Data (Big Data)*. Dec. 2019, pp. 4395–4401. DOI: 10.1109/BigData47090.2019.9006439.

[133] Minh Huong Le Nguyen, Fabien Turgis, Pierre-Emmanuel Fayemi, and Albert Bifet. "A Complete Streaming Pipeline for Real-time Monitoring and Predictive Maintenance". In: *Proceedings of the 31st European Safety and Reliability Conference*. 2021, p. 2119. DOI: 10.3850/978-981-18-2016-8_400-cd.

[134] Minh Huong Le Nguyen, Fabien Turgis, Pierre-Emmanuel Fayemi, and Albert Bifet. "Challenges of Stream Learning for Predictive Maintenance in the Railway Sector". In: *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning*. Communications in Computer and Information Science. Springer International Publishing, 2020, pp. 14–29. ISBN: 978-3-030-66770-2. DOI: 10.1007/978-3-030-66770-2_2.

[135] C. Lee, Yi Cao, and Kam K.H. Ng. "Big Data Analytics for Predictive Maintenance Strategies". In: Jan. 2017. DOI: 10.4018/978-1-5225-0956-1.ch004.

[136] Wei-Han Lee, Jorge Ortiz, Bongjun Ko, and Ruby Lee. "Time Series Segmentation through Automatic Feature Learning". In: *arXiv:1801.05394 [cs, stat]* (Jan. 2018). arXiv: 1801.05394. URL: http://arxiv.org/abs/1801.05394 (visited on 11/23/2020).

[137] Yaguo Lei, Bin Yang, Xinwei Jiang, Feng Jia, Naipeng Li, and Asoke K. Nandi. "Applications of machine learning to machine fault diagnosis: A review and roadmap". In: *Mechanical Systems and Signal Processing* 138 (Apr. 1, 2020), p. 106587. ISSN: 0888-3270. DOI: 10.1016/j.ymssp.2019.106587. URL: https://www.sciencedirect.com/science/article/pii/S0888327019308088 (visited on 09/30/2021).

[138] Hongfei Li, Dhaivat Parikh, Qing He, Buyue Qian, Zhiguo Li, Dongping Fang, and Arun Hampapur. "Improving rail network velocity: A machine learning approach to predictive maintenance". In: *Transportation Research Part C: Emerging Technologies* 45 (2014). Advances in Computing and Communications and their Impact on Transportation Science and Technologies, pp. 17–26. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2014.04.013. URL: http://www.sciencedirect.com/science/article/pii/S0968090X14001107.

[139] Y. Li, S. Billington, C. Zhang, T. Kurfess, S. Danyluk, and S. Liang. "Adaptive prognostics for rolling element bearing condition". In: *Mechanical Systems and Signal Processing* 13.1 (1999), pp. 103–113. ISSN: 0888-3270. DOI: https://doi.org/10.1006/mssp.1998.0183. URL: http://www.sciencedirect.com/science/article/pii/S0888327098901832.

[140] Yanni Li, Hui Li, Zhi Wang, Bing Liu, Jiangtao Cui, and Hang Fei. "ESA-Stream: Efficient Self-Adaptive Online Data Stream Clustering". In: *IEEE Transactions on Knowledge and Data Engineering* (2020). Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 1–1. ISSN: 1558-2191. DOI: 10.1109/TKDE.2020.2990196.

[141] Zhenglin Liang and Ajith Parlikad. "A Markovian model for power transformer maintenance". en. In: *International Journal of Electrical Power & Energy Systems* 99 (July 2018), pp. 175–182. ISSN: 0142-0615. DOI: 10.1016/j.ijepes.2017.12.024. URL: http://www.sciencedirect.com/science/article/pii/S0142061517321312 (visited on 05/05/2020).

[142] Haitao Liao, Elsayed A. Elsayed, and Ling-Yau Chan. "Maintenance of continuously monitored degrading systems". In: *European Journal of Operational Research* 175.2 (2006), pp. 821–835. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2005.05.017. URL: http://www.sciencedirect.com/science/article/pii/S0377221705005059.

[143] Miromar Jose de Lima, Cesar David Paredes Crovato, Rodrigo Ivan Goytia Mejia, Rodrigo da Rosa Righi, Gabriel de Oliveira Ramos, Cristiano André da Costa, and Giovani Pesenti. "HealthMon: An approach for monitoring machines degradation using time-series decomposition, clustering, and metaheuristics". en. In: *Computers & Industrial Engineering* 162 (Dec. 2021), p. 107709. ISSN: 0360-8352. DOI: 10.1016/j.cie.2021.107709. URL: https://www.sciencedirect.com/science/article/pii/S0360835221006136 (visited on 10/04/2021).

[144] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. "A symbolic representation of time series, with implications for streaming algorithms". In: *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*. DMKD '03. New York, NY, USA: Association for Computing Machinery, June 2003, pp. 2–11. ISBN: 978-1-4503-7422-4. DOI: 10.1145/882082.882086. URL: https://doi.org/10.1145/882082.882086 (visited on 10/01/2020).

[145] Song Liu, Makoto Yamada, Nigel Collier, and Masashi Sugiyama. "Change-point detection in time-series data by relative density-ratio estimation". en. In: *Neural Networks* 43 (July 2013), pp. 72–83. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2013.01.012. URL: https://www.sciencedirect.com/science/article/pii/S0893608013000270 (visited on 08/25/2022).

[146] Li-xiong Liu, Hai Huang, Yun-fei Guo, and Fu-cai Chen. "rDenStream, A Clustering Algorithm over an Evolving Data Stream". In: *2009 International Conference on Information Engineering and Computer Science*. ISSN: 2156-7387. Dec. 2009, pp. 1–4. DOI: 10.1109/ICIECS.2009.5363379.

[147] Abbas Loghman and Mehdi Moradi. "Creep damage and life assessment of thick-walled spherical reactor using Larson–Miller parameter". en. In: *International Journal of Pressure Vessels and Piping* 151 (Mar. 2017), pp. 11–19. ISSN: 0308-0161. DOI: 10.1016/j.ijpvp.2017.02.003. URL: https://www.sciencedirect.com/science/article/pii/S0308016117300571 (visited on 03/05/2022).

[148] Viktor Losing, Barbara Hammer, and Heiko Wersing. "Incremental on-line learning: A review and comparison of state of the art algorithms". en. In: *Neurocomputing* 275 (Jan. 2018), pp. 1261–1274. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2017.06.084. URL: http://www.sciencedirect.com/science/article/pii/S0925231217315928 (visited on 03/17/2020).

[149] Viktor Losing, Barbara Hammer, and Heiko Wersing. "Incremental on-line learning: A review and comparison of state of the art algorithms". In: *Neurocomputing* 275 (Jan. 31, 2018), pp. 1261–1274. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2017.06.084. URL: https://www.sciencedirect.com/science/article/pii/S0925231217315928 (visited on 03/15/2022).

[150] Sebastian Lühr and Mihai Lazarescu. "Incremental clustering of dynamic data streams using connectivity based representative points". In: *Data Knowl. Eng.* 68 (Jan. 2009), pp. 1–27. DOI: 10.1016/j.datak.2008.08.006.

[151] X. J. Luo, K. F. Fong, Y. J. Sun, and M. K. H. Leung. "Development of clustering-based sensor fault detection and diagnosis strategy for chilled water system". en. In: *Energy and Buildings* 186 (Mar. 2019), pp. 17–36. ISSN: 0378-7788. DOI: 10.1016/j.enbuild.2019.01.006. URL: https://www.sciencedirect.com/science/article/pii/S0378778818329207 (visited on 01/14/2022).

[152] Junshui Ma, James Theiler, and Simon Perkins. "Accurate On-line Support Vector Regression". In: *Neural Computation* 15.11 (Nov. 2003). Conference Name: Neural Computation, pp. 2683–2703. ISSN: 0899-7667. DOI: 10.1162/089976603322385117.

[153] Emaad Manzoor, Hemank Lamba, and Leman Akoglu. "xStream: Outlier Detection in Feature-Evolving Data Streams". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '18. London, United Kingdom: Association for Computing Machinery, July 2018, pp. 1963–1972. ISBN: 978-1-4503-5552-0. DOI: 10.1145/3219819.3220107. URL: https://doi.org/10.1145/3219819.3220107 (visited on 05/03/2020).

[154] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[155] Avin Mathew, Liqun Zhang, Sheng Zhang, and Lin Ma. "A Review of the MIMOSA OSA-EAI Database for Condition Monitoring Systems". en. In: *Engineering Asset Management*. Ed. by Joseph Mathew, Jim Kennedy, Lin Ma, Andy Tan, and Deryk Anderson. London: Springer, 2006, pp. 837–846. ISBN: 978-1-84628-814-2. DOI: 10.1007/978-1-84628-814-2_88.

[156] Pawel Matuszyk, Georg Krempl, and Myra Spiliopoulou. "Correcting the Usage of the Hoeffding Inequality in Stream Mining". en. In: *Advances in Intelligent Data Analysis XII*. Ed. by Allan Tucker, Frank Höppner, Arno Siebes, and Stephen Swift. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 298–309. ISBN: 978-3-642-41398-8. DOI: 10.1007/978-3-642-41398-8_26.

[157] Arthur Henrique de Andrade Melani, Miguel Angelo de Carvalho Michalski, Renan Favarão da Silva, and Gilberto Francisco Martha de Souza. "A framework to automate fault detection and diagnosis based on moving window principal component analysis and Bayesian network". en. In: *Reliability Engineering & System Safety* 215 (Nov. 2021), p. 107837. ISSN: 0951-8320. DOI: 10.1016/j.ress.2021.107837. (Visited on 01/13/2022).

[158] MIMOSA. *MIMOSA OSA-CBM*. http://www.mimosa.org/mimosa-osa-cbm/. Accessed: 2019-11-22. 2001.

[159] MIMOSA. *MIMOSA OSA-EAI*. http://www.mimosa.org/mimosa-osa-eai/. Accessed: 2019-11-22. 2001.

[160] Melanie Mitchell and Zoltan Toroczkai. "Complexity: A Guided Tour". In: *Physics Today* 63 (Feb. 2010), pp. 47–. DOI: 10.1063/1.3326990.

[161] Yasushi Miyata and Hiroshi Ishikawa. "Concept Drift Detection on Data Stream for Revising DBSCAN Cluster". In: *Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics*. WIMS 2020. New York, NY, USA: Association for Computing Machinery, Aug. 2020, pp. 104–110. ISBN: 978-1-4503-7542-9. DOI: 10.1145/3405962.3405990. URL: https://doi.org/10.1145/3405962.3405990 (visited on 02/16/2023).

[162] Jacob Montiel, Albert Bifet, Viktor Losing, Jesse Read, and Talel Abdessalem. "Learning Fast and Slow: A Unified Batch/Stream Framework". In: *2018 IEEE International Conference on Big Data (Big Data)*. Dec. 2018, pp. 1065–1072. DOI: 10.1109/BigData.2018.8622222.

[163] Jacob Montiel, Hoang-Anh Ngo, Minh-Huong Le-Nguyen, and Albert Bifet. "Online Clustering: Algorithms, Evaluation, Metrics, Applications and Benchmarking". In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. KDD '22. New York, NY, USA: Association for Computing Machinery, Aug. 2022, pp. 4808–4809. ISBN: 978-1-4503-9385-0. DOI: 10.1145/3534678.3542600. URL: https://doi.org/10.1145/3534678.3542600.

[164] J. Moubray. *Reliability-centered Maintenance*. Industrial Press, 2001. ISBN: 9780831131463. URL: https://books.google.fr/books?id=bNCVF0B7vpIC.

[165] "Dynamic Time Warping". en. In: *Information Retrieval for Music and Motion*. Ed. by Meinard Müller. Berlin, Heidelberg: Springer, 2007, pp. 69–84. ISBN: 978-3-540-74048-3. DOI: 10.1007/978-3-540-74048-3_4. URL: https://doi.org/10.1007/978-3-540-74048-3_4 (visited on 03/26/2021).

[166] M. E. J. Newman. "Complex Systems: A Survey". In: *American Journal of Physics* 79.8 (Aug. 2011), pp. 800–810. ISSN: 0002-9505, 1943-2909. DOI: 10.1119/1.3590372. arXiv: 1112.1440. URL: http://arxiv.org/abs/1112.1440 (visited on 02/28/2022).

[167] Minh-Huong Le-Nguyen, Fabien Turgis, Pierre-Emmanuel Fayemi, and Albert Bifet. "Continuous Health Monitoring of Machinery using Online Clustering on Unlabeled Data Streams". In: *2022 IEEE International Conference on Big Data (Big Data)*. Dec. 2022, pp. 1866–1873. DOI: 10.1109/BigData55660.2022.10021002.

[168] Minh-Huong Le-Nguyen, Fabien Turgis, Pierre-Emmanuel Fayemi, and Albert Bifet. "Real-time learning for real-time data: online machine learning for predictive maintenance of railway systems". In: *Transport Research Arena (TRA)*. Lisbon, Portugal, Nov. 2022.

[169] Gang Niu and Bo-Suk Yang. "Intelligent condition monitoring and prognostics system based on data-fusion strategy". en. In: *Expert Systems with Applications* 37.12 (Dec. 2010), pp. 8831–8840. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2010.06.014. URL: http://www.sciencedirect.com/science/article/pii/S095741741000518X (visited on 03/30/2020).

[170] Gang Niu, Bo-Suk Yang, and Michael Pecht. "Development of an optimized condition-based maintenance system by data fusion and reliability-centered maintenance". In: *Reliability Engineering & System Safety - RELIAB ENG SYST SAFETY* 95 (July 2010), pp. 786–796. DOI: 10.1016/j.ress.2010.02.016.

[171] Gang Niu, Bo-Suk Yang, and Michael Pecht. "Development of an optimized condition-based maintenance system by data fusion and reliability-centered maintenance". In: *Reliability Engineering & System Safety* 95.7 (2010), pp. 786–796. ISSN: 0951-8320. DOI: https://doi.org/10.1016/j.ress.2010.02.016. URL: http://www.sciencedirect.com/science/article/pii/S0951832010000591.

[172] Mohammed A. Noman, Emad S. Abouel Nasr, Adel Al-Shayea, and Husam Kaid. "Overview of predictive condition based maintenance research using bibliometric indicators". In: *Journal of King Saud University - Engineering Sciences* 31.4 (2019), pp. 355–367. ISSN: 1018-3639. DOI: https://doi.org/10.1016/j.jksues.2018.02.003. URL: http://www.sciencedirect.com/science/article/pii/S1018363917303720.

[173] Kento Nozawa and Issei Sato. *Empirical Evaluation and Theoretical Analysis for Representation Learning: A Survey*. arXiv:2204.08226 [cs]. Apr. 2022. DOI: 10.48550/arXiv.2204.08226. URL: http://arxiv.org/abs/2204.08226 (visited on 07/29/2022).

[174] Irene Ntoutsi, Arthur Zimek, Themis Palpanas, Peer Kröger, and Hans-Peter Kriegel. "Density-based Projected Clustering over High Dimensional Data Streams". In: *Proceedings of the 2012 SIAM International Conference on Data Mining (SDM)*. Proceedings. Society for Industrial and Applied Mathematics, Apr. 2012, pp. 987–998. ISBN: 978-1-61197-232-0. DOI: 10.1137/1.9781611972825.85. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611972825.85 (visited on 02/16/2023).

[175] Basquin OH. "The exponential law of endurance tests". In: *American Society for Testing Materials* 10 (1910), pp. 625–630.

[176] Christopher Olah. *Understanding LSTM Networks – colah's blog*. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 09/16/2022).

[177] Olufemi A. Omitaomu, Myong K. Jeong, and Adedeji B. Badiru. "Online Support Vector Regression With Varying Parameters for Time-Dependent Data". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 41.1 (Jan. 2011). Conference Name: IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, pp. 191–197. ISSN: 1558-2426. DOI: 10.1109/TSMCA.2010.2055156.

[178] Charles H. Oppenheimer and Kenneth A. Loparo. "Physically based diagnosis and prognosis of cracked rotor shafts". In: *Component and Systems Diagnostics, Prognostics, and Health Management II*. Ed. by Peter K. Willett and Thiagalingam Kirubarajan. Vol. 4733. International Society for Optics and Photonics. SPIE, 2002, pp. 122–132. DOI: 10.1117/12.475502. URL: https://doi.org/10.1117/12.475502.

[179] Gopalakrishna Palem. "Condition-Based Maintenance using Sensor Arrays and Telematics". In: *International Journal of Mobile Network Communications & Telematics* 3 (June 2013), pp. 19–28. DOI: 10.5121/ijmnct.2013.3303.

[180] Paul Paris, Mario Gomez, and William Anderson. "A rational analytic theory of fatigue". In: *The trend inf Engineering* 13 (1961), pp. 9–14.

[181] Md. Mostofa Ali Patwary, Diana Palsetia, Ankit Agrawal, Wei-keng Liao, Fredrik Manne, and Alok Choudhary. "A new scalable parallel DBSCAN algorithm using the disjoint-set data structure". In: *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ISSN: 2167-4337. Nov. 2012, pp. 1–11. DOI: 10.1109/SC.2012.9.

[182] Louis-Francois Pau. "Survey of expert systems for fault detection, test generation and maintenance". In: *Expert Systems* 3 (Apr. 2007), pp. 100–110. DOI: 10.1111/j.1468-0394.1986.tb00199.x.

[183] Ying Peng, Ming Dong, and Ming Jian Zuo. "Current status of machine prognostics in condition-based maintenance: a review". In: *The International Journal of Advanced Manufacturing Technology* 50.1 (Sept. 2010), pp. 297–313. ISSN: 1433-3015. DOI: 10.1007/s00170-009-2482-0. URL: https://doi.org/10.1007/s00170-009-2482-0.

[184] Per Anders Akersten. "Maintenance Related IEC Dependability Standards". en. In: *Engineering Asset Management*. Ed. by Joseph Mathew, Jim Kennedy, Lin Ma, Andy Tan, and Deryk Anderson. London: Springer, 2006, pp. 115–119. ISBN: 978-1-84628-814-2. DOI: 10.1007/978-1-84628-814-2_12.

[185] J. Phillips, E. Cripps, John W. Lau, and M. R. Hodkiewicz. "Classifying machinery condition using oil samples and binary logistic regression". In: *Mechanical Systems and Signal Processing* 60-61 (Aug. 1, 2015), pp. 316–325. ISSN: 0888-3270. DOI: 10.1016/j.ymssp.2014.12.020. URL: https://www.sciencedirect.com/science/article/pii/S0888327014005093 (visited on 09/30/2021).

[186] Robi Polikar. "Ensemble Learning". en. In: *Ensemble Machine Learning: Methods and Applications*. Ed. by Cha Zhang and Yunqian Ma. Boston, MA: Springer US, 2012, pp. 1–34. ISBN: 978-1-4419-9326-7. DOI: 10.1007/978-1-4419-9326-7_1. URL: https://doi.org/10.1007/978-1-4419-9326-7_1 (visited on 03/25/2021).

[187] Ashok Prajapati, James Bechtel, and Subramaniam Ganesan. "Condition based maintenance: a survey". In: *Journal of Quality in Maintenance Engineering* 18.4 (Jan. 2012). Publisher: Emerald Group Publishing Limited, pp. 384–400. ISSN: 1355-2511. DOI: 10.1108/13552511211281552. URL: https://doi.org/10.1108/13552511211281552 (visited on 03/05/2022).

[188] Mahardhika Pratama, Andri Ashfahani, Yew Soon Ong, Savitha Ramasamy, and Edwin Lughofer. "Autonomous Deep Learning: Incremental Learning of Denoising Autoencoder for Evolving Data Streams". In: *arXiv:1809.09081 [cs, stat]* (Sept. 24, 2018). arXiv: 1809.09081. URL: http://arxiv.org/abs/1809.09081 (visited on 03/19/2022).

[189] Oxford University Press. *Oxford English Dictionary*. Oxford: Oxford University Press, 2012.

[190] Andrian Putina and Dario Rossi. "Online Anomaly Detection Leveraging Stream-Based Clustering and Real-Time Telemetry". In: *IEEE Transactions on Network and Service Management* 18.1 (Mar. 2021). Conference Name: IEEE Transactions on Network and Service Management, pp. 839–854. ISSN: 1932-4537. DOI: 10.1109/TNSM.2020.3037019.

[191] Jing Qiu, Brij B. Seth, Steven Y. Liang, and Cheng Zhang. "Damage mechanics approach for bearing lifetime prognostics". In: *Mechanical Systems and Signal Processing* 16.5 (2002), pp. 817–829. ISSN: 0888-3270. DOI: https://doi.org/10.1006/mssp.2002.1483. URL: http://www.sciencedirect.com/science/article/pii/S0888327002914834.

[192] Jiadong Ren and Ruiqing Ma. "Density-Based Data Streams Clustering over Sliding Windows". In: *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery*. Vol. 5. Aug. 2009, pp. 248–252. DOI: 10.1109/FSKD.2009.553.

[193] Rita P. Ribeiro, Pedro Pereira, and João Gama. "Sequential anomalies: a study in the Railway Industry". In: *Machine Learning* 105.1 (Oct. 1, 2016), pp. 127–153. ISSN: 1573-0565. DOI: 10.1007/s10994-016-5584-6. URL: https://doi.org/10.1007/s10994-016-5584-6 (visited on 02/23/2022).

[194] Frank Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain". en. In: *Psychological review* (1958). URL: https://doi.apa.org/doiLanding?doi=10.1037%2Fh0042519 (visited on 03/16/2022).

[195] Sam T. Roweis and Lawrence K. Saul. "Nonlinear Dimensionality Reduction by Locally Linear Embedding". In: *Science* 290.5500 (Dec. 2000). Publisher: American Association for the Advancement of Science, pp. 2323–2326. DOI: 10.1126/science.290.5500.2323. URL: https://www.science.org/doi/10.1126/science.290.5500.2323 (visited on 12/15/2022).

[196] Carlos Ruiz, Ernestina Menasalvas, and Myra Spiliopoulou. "C-DenStream: Using Domain Knowledge on a Data Stream". en. In: *Discovery Science*. Ed. by João Gama, Vítor Santos Costa, Alípio Mário Jorge, and Pavel B. Brazdil. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 287–301. ISBN: 978-3-642-04747-3. DOI: 10.1007/978-3-642-04747-3_23.

[197] Leszek Rutkowski, Lena Pietruczuk, Piotr Duda, and Maciej Jaworski. "Decision Trees for Mining Data Streams Based on the McDiarmid's Bound". In: *IEEE Transactions on Knowledge and Data Engineering* 25.6 (June 2013). Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 1272–1279. ISSN: 1558-2191. DOI: 10.1109/TKDE.2012.66.

[198] Radhya Sahal, John G. Breslin, and Muhammad Intizar Ali. "Big data and stream processing platforms for Industry 4.0 requirements mapping for a predictive maintenance use case". en. In: *Journal of Manufacturing Systems* 54 (Jan. 2020), pp. 138–151. ISSN: 0278-6125. DOI: 10.1016/j.jmsy.2019.11.004. URL: http://www.sciencedirect.com/science/article/pii/S0278612519300937 (visited on 03/24/2020).

[199] Doyen Sahoo, Quang Pham, Jing Lu, and Steven C. H. Hoi. "Online Deep Learning: Learning Deep Neural Networks on the Fly". In: (2018), pp. 2660–2666. URL: https://www.ijcai.org/proceedings/2018/369 (visited on 03/19/2022).

[200] Mahsa Salehi and Lida Rashidi. "A Survey on Anomaly detection in Evolving Data: [with Application to Forest Fire Risk Prediction]". In: *ACM SIGKDD Explorations Newsletter* 20.1 (May 29, 2018), pp. 13–23. ISSN: 1931-0145. DOI: 10.1145/3229329.3229332. URL: https://doi.org/10.1145/3229329.3229332 (visited on 11/12/2020).

[201] Burr Settles. *Active Learning Literature Survey*. Technical Report. Accepted: 2012-03-15T17:23:56Z. University of Wisconsin-Madison Department of Computer Sciences, 2009. URL: https://minds.wisconsin.edu/handle/1793/60660 (visited on 03/25/2021).

[202] Jianjun Shen, Shihao Li, Feng Jia, Hao Zuo, and Junxing Ma. "A Deep Multi-Label Learning Framework for the Intelligent Fault Diagnosis of Machines". In: *IEEE Access* 8 (2020). Conference Name: IEEE Access, pp. 113557–113566. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3002826.

[203] Xuejin Shen, Lei Cao, and Ruyan Li. "Numerical simulation of sliding wear based on archard model". In: *2010 International Conference on Mechanic Automation and Control Engineering*. June 2010, pp. 325–329. DOI: 10.1109/MACE.2010.5535855.

[204] Jong-Ho Shin and Hong-Bae Jun. "On condition based maintenance policy". In: *Journal of Computational Design and Engineering* 2.2 (Apr. 2015), pp. 119–127. ISSN: 2288-4300. DOI: 10.1016/j.jcde.2014.12.006.

[205] Xiao-Sheng Si, Wenbin Wang, Chang-Hua Hu, Mao-Yin Chen, and Dong-Hua Zhou. "A Wiener-process-based degradation model with a recursive filter algorithm for remaining useful life estimation". In: *Mechanical Systems and Signal Processing* 35.1 (2013), pp. 219–237. ISSN: 0888-3270. DOI: https://doi.org/10.1016/j.ymssp.2012.08.016. URL: http://www.sciencedirect.com/science/article/pii/S0888327012003226.

[206] Xiao-Sheng Si, Wenbin Wang, Chang-Hua Hu, and Dong-Hua Zhou. "Remaining useful life estimation – A review on the statistical data driven approaches". In: *European Journal of Operational Research* 213.1 (2011), pp. 1–14. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2010.11.018. URL: http://www.sciencedirect.com/science/article/pii/S0377221710007903.

[207] Xiao-Sheng Si, Wenbin Wang, Chang-Hua Hu, and Dong-Hua Zhou. "Remaining useful life estimation – A review on the statistical data driven approaches". en. In: *European Journal of Operational Research* 213.1 (Aug. 2011), pp. 1–14. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2010.11.018. URL: https://www.sciencedirect.com/science/article/pii/S0377221710007903 (visited on 02/28/2023).

[208] Cláudio R. Ávila da Silva and Giuseppe Pintaude. "Uncertainty analysis on the wear coefficient of Archard model". en. In: *Tribology International* 41.6 (June 2008), pp. 473–481. ISSN: 0301-679X. DOI: 10.1016/j.triboint.2007.10.007. URL: https://www.sciencedirect.com/science/article/pii/S0301679X07001740 (visited on 03/05/2022).

[209] Jonathan A. Silva, Elaine R. Faria, Rodrigo C. Barros, Eduardo R. Hruschka, André C. P. L. F. de Carvalho, and João Gama. "Data stream clustering: A survey". In: *ACM Computing Surveys* 46.1 (July 2013), 13:1–13:31. ISSN: 0360-0300. DOI: 10.1145/2522968.2522981. URL: https://doi.org/10.1145/2522968.2522981 (visited on 03/25/2022).

[210] Rimjhim Singh and Manoj B. Chandak. "Classification and Novel Class Detection in Data Streams Using Strings". en. In: *Open Access Library Journal* 2.5 (May 2015). Number: 5 Publisher: Scientific Research Publishing, pp. 1–8. DOI: 10.4236/oalib.1101507. URL: http://www.scirp.org/Journal/Paperabs.aspx?paperid=68406 (visited on 03/31/2022).

[211] Manson SS. *Behavior of materials under conditions of thermal stress.* Tech. rep. National Advisory Committee for Aeronautics. Lewis Flight Propulsion Lab.; Cleveland, OH, United States, 1954.

[212] A. G. Starr. "A structured approach to the selection of condition based maintenance". In: *Fifth International Conference on Factory 2000 - The Technology Exploitation Process.* Apr. 1997, pp. 131–138. DOI: 10.1049/cp:19970134.

[213] Chuan-Jun Su and Shi-Feng Huang. "Real-time big data analytics for hard disk drive predictive maintenance". en. In: *Computers & Electrical Engineering* 71 (Oct. 2018), pp. 93–101. ISSN: 0045-7906. DOI: 10.1016/j.compeleceng.2018.07.025. URL: http://www.sciencedirect.com/science/article/pii/S0045790617328409 (visited on 05/04/2020).

[214] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone, and A. Beghi. "Machine Learning for Predictive Maintenance: A Multiple Classifier Approach". In: *IEEE Transactions on Industrial Informatics* 11.3 (June 2015), pp. 812–820. DOI: 10.1109/TII.2014.2349359.

[215] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, second edition: An Introduction.* Google-Books-ID: uWV0DwAAQBAJ. MIT Press, Nov. 13, 2018. 549 pp. ISBN: 978-0-262-35270-3.

[216] Jian-Zhong Tang and Qing-Feng Wang. "Online fault diagnosis and prevention expert system for dredgers". en. In: *Expert Systems with Applications* 34.1 (Jan. 2008), pp. 511–521. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2006.09.032. URL: http://www.sciencedirect.com/science/article/pii/S0957417406003022 (visited on 05/06/2020).

[217] Dimitris K. Tasoulis, Gordon Ross, and Niall M. Adams. "Visualising the Cluster Structure of Data Streams". en. In: *Advances in Intelligent Data Analysis VII.* Ed. by Michael R. Berthold, John Shawe-Taylor, and Nada Lavrač. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pp. 81–92. ISBN: 978-3-540-74825-0. DOI: 10.1007/978-3-540-74825-0_8.

[218] Agnes Tegen, Paul Davidsson, and Jan A. Persson. "The Effects of Reluctant and Fallible Users in Interactive Online Machine Learning". eng. In: CEUR Workshops, 2020, pp. 55–71. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:mau:diva-17673 (visited on 09/05/2021).

[219] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. "A Global Geometric Framework for Nonlinear Dimensionality Reduction". In: *Science* 290.5500 (Dec. 2000). Publisher: American Association for the Advancement of Science, pp. 2319–2323. DOI: 10.1126/science.290.5500.2319. URL: https://www.science.org/doi/10.1126/science.290.5500.2319 (visited on 12/15/2022).

[220] Michael Thurston, Mitchell Lebold, and P O Box. "Standards Developments for Condition-Based Maintenance Systems". en. In: (2001), p. 12.

[221] Hongda Tian, Nguyen Lu Dang Khoa, Ali Anaissi, Yang Wang, and Fang Chen. "Concept Drift Adaption for Online Anomaly Detection in Structural Health Monitoring". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM '19. New York, NY, USA: Association for Computing Machinery, Nov. 3, 2019, pp. 2813–2821. ISBN: 978-1-4503-6976-3. DOI: 10.1145/3357384.3357816. (Visited on 03/26/2021).

[222] Tiedo Tinga. *Principles of loads and failure mechanisms. Applications in maintenance, reliability and design*. English. Springer Series in Reliability Engineering. Springer, 2013. ISBN: 978-1-4471-4916-3. DOI: 10.1007/978-1-4471-4917-0.

[223] Sahar Torkamani and Volker Lohweg. "Survey on time series motif discovery". en. In: *WIREs Data Mining and Knowledge Discovery* 7.2 (2017), e1199. ISSN: 1942-4795. DOI: https://doi.org/10.1002/widm.1199. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1199 (visited on 03/25/2021).

[224] Charles Truong, Laurent Oudre, and Nicolas Vayatis. "ruptures: change point detection in Python". In: *arXiv:1801.00826 [cs, stat]* (Jan. 2018). arXiv: 1801.00826. URL: http://arxiv.org/abs/1801.00826 (visited on 10/02/2020).

[225] Albert H.C. Tsang. "Condition-based maintenance: tools and decision making". In: *Journal of Quality in Maintenance Engineering* 1.3 (Jan. 1995), pp. 3–17.

[226] Fabien Turgis, Pierre Auder, Quentin Coutadeur, and Cyril Verdun. "Industrialization of Condition based Maintenance for Complex Systems in a Complex Maintenance Environment, Example of NAT". In: *12th World Congress on Railway Research*. 2019.

[227] Fabien Turgis, Pierre Audier, Quentin Coutadeur, and Cyril Verdun. "Industrialization of Condition Based Maintenance for Complex Systems in a Complex Maintenance Environment, Example of NAT". In: Dec. 2019.

[228] Fabien Turgis, Pierre Audier, and Rémy Marion. "Prognostic Expert System for Railway Fleet Maintenance". In: *Proceedings of the 31st European Safety and Reliability Conference*. Pages: 2111. Anger, France, Sept. 2021. DOI: http://dx.doi.org/10.3850/978-981-18-2016-8_391-cd.

[229] Fabien Turgis, Pierre Audier, Valentin Nemoz, and Rémy Marion. "Health state characterization using clustering algorithms for railway maintenance". In: Birmingham, United Kingdom, 2022.

[230] M. Vasudevan, S. Venkadesan, P. V. Sivaprasad, and S. L. Mannan. "Use of the Larson-Miller parameter to study the influence of ageing on the hardness of cold-worked austenitic stainless steel". en. In: *Journal of Nuclear Materials* 211.3 (Aug. 1994), pp. 251–255. ISSN: 0022-3115. DOI: 10.1016/0022-3115(94)90355-7. URL: https://www.sciencedirect.com/science/article/pii/0022311594903557 (visited on 03/05/2022).

[231] *Fieldbus neutral reference architecture for Condition Monitoring in production automation*. Standard. VDMA, Apr. 2014.

[232] Gido M. van de Ven and Andreas S. Tolias. "Three scenarios for continual learning". In: *arXiv:1904.07734 [cs, stat]* (Apr. 15, 2019). arXiv: 1904.07734. URL: http://arxiv.org/abs/1904.07734 (visited on 03/19/2022).

[233] Cyril Verdun, Fabien Turgis, and Pierre Audier. "Remote Diagnosis and Condition-based Maintenance for Rolling Stock at SNCF". In: Tokyo, Japan, 2019.

[234] Hongzhou Wang. "A survey of maintenance policies of deteriorating systems". In: *European Journal of Operational Research* 139.3 (2002), pp. 469–489. ISSN: 0377-2217. DOI: https://doi.org/10.1016/S0377-2217(01)00197-7. URL: http://www.sciencedirect.com/science/article/pii/S0377221701001977.

[235] Jinjiang Wang, Laibin Zhang, Lixiang Duan, and Robert X. Gao. "A new paradigm of cloud-based predictive maintenance for intelligent manufacturing". In: *Journal of Intelligent Manufacturing* 28.5 (2017), pp. 1125–1137. ISSN: 1572-8145. DOI: 10.1007/s10845-015-1066-0. URL: https://doi.org/10.1007/s10845-015-1066-0.

[236] Yupeng Wei, Dazhong Wu, and Janis Terpenny. "Learning the health index of complex systems using dynamic conditional variational autoencoders". In: *Reliability Engineering & System Safety* 216 (Dec. 1, 2021), p. 108004. ISSN: 0951-8320. DOI: 10.1016/j.ress.2021.108004. (Visited on 01/13/2022).

[237] Martin Wollschlaeger, Stefan Theurich, Albrecht Winter, Frank Lubnau, and Christoph Paulitsch. "A reference architecture for condition monitoring". In: *2015 IEEE World Conference on Factory Communication Systems (WFCS)*. May 2015, pp. 1–8. DOI: 10.1109/WFCS.2015.7160555.

[238] Guangrong Wu. "Design on Fault Diagnosis Expert System for Railway Signal Equipment". en. In: ISSN: 2352-5401. Atlantis Press, June 2018, pp. 36–41. ISBN: 978-94-6252-531-3. DOI: 10.2991/icmmct-18.2018.7. URL: https://www.atlantis-press.com/proceedings/icmmct-18/25898536 (visited on 03/05/2022).

[239] X.L. Xie and G. Beni. "A validity measure for fuzzy clustering". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.8 (Aug. 1991). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 841–847. ISSN: 1939-3539. DOI: 10.1109/34.85677.

[240] R. C. M. Yam, P.W. Tse, L. Li, and P. Tu. "Intelligent Predictive Decision Support System for Condition-Based Maintenance". In: *The International Journal of Advanced Manufacturing Technology* 17.5 (Feb. 2001), pp. 383–391. ISSN: 1433-3015. DOI: 10.1007/s001700170173. URL: https://doi.org/10.1007/s001700170173.

[241] Feng Yang, Mohamed Salahuddin Habibullah, Tianyou Zhang, Zhao Xu, Pin Lim, and Sivakumar Nadarajan. "Health Index-Based Prognostics for Remaining Useful Life Predictions in Electrical Machines". In: *IEEE Transactions on Industrial Electronics* 63.4 (Apr. 2016). Conference Name: IEEE Transactions on Industrial Electronics, pp. 2633–2644. ISSN: 1557-9948. DOI: 10.1109/TIE.2016.2515054.

[242] Jiachen Yao, Baochun Lu, and Junli Zhang. "Tool remaining useful life prediction using deep transfer reinforcement learning based on long short-term memory networks". In: *The International Journal of Advanced Manufacturing Technology* 118.3 (Jan. 1, 2022), pp. 1077–1086. ISSN: 1433-3015. DOI: 10.1007/s00170-021-07950-2. URL: https://doi.org/10.1007/s00170-021-07950-2 (visited on 03/03/2022).

[243] David Yarowsky. "Unsupervised word sense disambiguation rivaling supervised methods". In: *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*. ACL '95. USA: Association for Computational Linguistics, June 26, 1995, pp. 189–196. DOI: 10.3115/981658.981684. URL: https://doi.org/10.3115/981658.981684 (visited on 03/20/2022).

[244] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. "Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. ISSN: 2374-8486. Dec. 2016, pp. 1317–1322. DOI: 10.1109/ICDM.2016.0179.

[245] Steven Young, Itamar Arel, Thomas P. Karnowski, and Derek Rose. "A Fast and Stable Incremental Clustering Algorithm". In: *2010 Seventh International Conference on Information Technology: New Generations*. Apr. 2010, pp. 204–209. DOI: 10.1109/ITNG.2010.148.

[246] Hang Yu, Jie Lu, and Guangquan Zhang. "Continuous Support Vector Regression for Nonstationary Streaming Data". In: *IEEE Transactions on Cybernetics* (2020). Conference Name: IEEE Transactions on Cybernetics, pp. 1–14. ISSN: 2168-2275. DOI: 10.1109/TCYB.2020.3015266.

[247] Yu Zang, Wei Shangguan, Baigen Cai, Huashen Wang, and Michael G Pecht. "Methods for fault diagnosis of high-speed railways: A review". en. In: *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability* 233.5 (Oct. 2019). Publisher: SAGE Publications, pp. 908–922. ISSN: 1748-006X. DOI: 10.1177/1748006X18823932. URL: https://doi.org/10.1177/1748006X18823932 (visited on 03/05/2022).

[248] Chi Zhang, Chetan Gupta, Ahmed Farahat, Kosta Ristovski, and Dipanjan Ghosh. "Equipment Health Indicator Learning Using Deep Reinforcement Learning". In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Ulf Brefeld, Edward Curry, Elizabeth Daly, Brian MacNamee, Alice Marascu, Fabio Pinelli, Michele Berlingerio, and Neil Hurley. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 488–504. ISBN: 978-3-030-10997-4. DOI: 10.1007/978-3-030-10997-4_30.

[249] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. "BIRCH: an efficient data clustering method for very large databases". In: *ACM SIGMOD Record* 25.2 (June 1, 1996), pp. 103–114. ISSN: 0163-5808. DOI: 10.1145/235968.233324. URL: https://doi.org/10.1145/235968.233324 (visited on 03/27/2022).

[250] Xiaoyuan Zhang, Yajun Jiang, Chaoshun Li, and Jinhao Zhang. "Health status assessment and prediction for pumped storage units using a novel health degradation index". In: *Mechanical Systems and Signal Processing* 171 (May 15, 2022), p. 108910. ISSN: 0888-3270. DOI: 10.1016/j.ymssp.2022.108910. URL: https://www.sciencedirect.com/science/article/pii/S0888327022000978 (visited on 02/15/2022).

[251] Pushe Zhao, Masaru Kurihara, Junichi Tanaka, Tojiro Noda, Shigeyoshi Chikuma, and Tadashi Suzuki. "Advanced correlation-based anomaly detection method for predictive maintenance". In: *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*. June 2017, pp. 78–83. DOI: 10.1109/ICPHM.2017.7998309.

[252] Rui Zhao, Ruqiang Yan, Zhenghua Chen, Kezhi Mao, Peng Wang, and Robert X. Gao. "Deep learning and its applications to machine health monitoring". In: *Mechanical Systems and Signal Processing* 115 (Jan. 15, 2019), pp. 213–237. ISSN: 0888-3270. DOI: 10.1016/j.ymssp.2018.05.050. URL: https://www.sciencedirect.com/science/article/pii/S0888327018303108 (visited on 02/09/2022).

[253] Aoying Zhou, Feng Cao, Weining Qian, and Cheqing Jin. "Tracking clusters in evolving data streams over sliding windows". In: *Knowledge and Information Systems* 15.2 (May 1, 2008), pp. 181–214. ISSN: 0219-3116. DOI: 10.1007/s10115-007-0070-x. URL: https://doi.org/10.1007/s10115-007-0070-x (visited on 03/27/2022).

[254] Xingquan Zhu, Peng Zhang, Xiaodong Lin, and Yong Shi. "Active Learning from Data Streams". In: *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. ISSN: 2374-8486. Oct. 2007, pp. 757–762. DOI: 10.1109/ICDM.2007.101.

[255] Indrė Žliobaitė, Albert Bifet, Bernhard Pfahringer, and Geoff Holmes. "Active Learning with Evolving Streaming Data". en. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis. Berlin, Heidelberg: Springer, 2011, pp. 597–612. ISBN: 978-3-642-23808-6. DOI: 10.1007/978-3-642-23808-6_39.

[256] Tiago Zonta, Cristiano André da Costa, Rodrigo da Rosa Righi, Miromar José de Lima, Eduardo Silveira da Trindade, and Guann Pyng Li. "Predictive maintenance in the Industry 4.0: A systematic literature review". en. In: *Computers & Industrial Engineering* 150 (Dec. 2020), p. 106889. ISSN: 0360-8352. DOI: 10.1016/j.cie.2020.106889. (Visited on 10/12/2020).

[257] Alaettin Zubaroğlu and Volkan Atalay. "Data stream clustering: a review". en. In: *Artificial Intelligence Review* 54.2 (Feb. 2021), pp. 1201–1236. ISSN: 1573-7462. DOI: 10.1007/s10462-020-09874-x. URL: https://doi.org/10.1007/s10462-020-09874-x (visited on 03/25/2022).

DOCTORAL SCHOOL

Titre: Maintenance prévisionnelle basée sur l'apprentissage automatique en ligne dans le secteur ferroviaire

Mots clés: maintenance prévisionnelle, ferroviaire, apprentissage automatique en ligne, flux de données

Résumé: En tant que moyen de transport en commun efficace sur de longues distances, le chemin de fer continuera de prospérer pour son empreinte carbone limitée dans l'environnement. Assurer la fiabilité des équipements et la sécurité des passagers fait ressortir la nécessité d'une maintenance efficace. Outre la maintenance corrective et périodique courante, la maintenance prédictive a pris de l'importance ces derniers temps. Les progrès récents de l'apprentissage automatique et l'abondance de données poussent les praticiens à la maintenance prédictive basée sur les données. La pratique courante consiste à collecter des données pour former un modèle d'apprentissage automatique, puis à déployer le modèle pour la production et à le conserver inchangé par la suite. Nous soutenons qu'une telle pratique est sous-optimale sur un flux de données. Le caractère illimité du flux rend le modèle sujet à un apprentissage incomplet. Les changements dynamiques sur le flux introduisent de nouveaux concepts invisibles pour le modèle et diminuent sa précision. La vitesse du flux rend l'étiquetage manuel impossible et désactive les algorithmes d'apprentissage supervisé. Par conséquent, il est nécessaire de passer d'un paradigme d'apprentissage statique et hors ligne à un paradigme adaptatif en ligne, en particulier lorsque de nouvelles générations de trains connectés générant en continu des données de capteurs sont déjà une réalité. Nous étudions l'applicabilité de l'apprentissage automatique en ligne pour la maintenance prédictive sur des systèmes complexes typiques du secteur ferroviaire. Tout d'abord, nous développons InterCE en tant que framework basé sur l'apprentissage actif pour extraire des cycles d'un flux non étiqueté en interagissant avec un expert humain. Ensuite, nous implémentons un auto-encodeur à mémoire longue et courte durée pour transformer les cycles extraits en vecteurs de caractéristiques plus compacts tout en restant représentatifs. Enfin, nous concevons CheMoc comme un framework pour surveiller en permanence l'état des systèmes en utilisant le clustering adaptatif en ligne. Nos méthodes sont évaluées sur les systèmes d'accès voyageurs sur deux flottes de trains gérés par la société nationale des chemins de fer SNCF de la France.

Title: Online machine learning-based predictive maintenance for the railway industry

Keywords: predictive maintenance, railway, online machine learning, data stream

Abstract: Being an effective long-distance mass transit, the railway will continue to flourish for its limited carbon footprint in the environment. Ensuring the equipment's reliability and passenger safety brings forth the need for efficient maintenance. Apart from the prevalence of corrective and periodic maintenance, predictive maintenance has come into prominence lately. Recent advances in machine learning and the abundance of data drive practitioners to data-driven predictive maintenance. The common practice is to collect data to train a machine learning model, then deploy the model for production and keep it unchanged afterward. We argue that such practice is suboptimal on a data stream. The unboundedness of the stream makes the model prone to incomplete learning. Dynamic changes on the stream introduce novel concepts unseen by the model and decrease its accuracy. The velocity of the stream makes manual labeling infeasible and disables supervised learning algorithms. Therefore, switching from a static, offline learning paradigm to an adaptive, online one is necessary, especially when new generations of connected trains continuously generating sensor data have already been a reality. We investigate the applicability of online machine learning for predictive maintenance on typical complex systems in the railway. First, we develop InterCE as an active learning-based framework that extracts cycles from an unlabeled stream by interacting with a human expert. Then, we implement a long short-term memory autoencoder to transform the extracted cycles into feature vectors that are more compact yet remain representative. Finally, we design CheMoc as a framework that continuously monitors the condition of the systems using online adaptive clustering. Our methods are evaluated on the passenger access systems on two fleets of passenger trains managed by the national railway company SNCF of France.

Institut Polytechnique de Paris
91120 Palaiseau, France