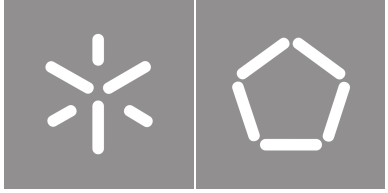
Universidade do Minho
Escola de Engenharia

Luís Fernando de Faria Ferreira

An Automated and Efficient Machine Learning Framework for One-Class Classification Tasks

September, 2023

An Automated and Efficient Machine Learning Framework for One-Class Classification Tasks
Luís Fernando de Faria Ferreira
UMinho | 2023



Universidade do Minho
Escola de Engenharia

Luís Fernando de Faria Ferreira

An Automated and Efficient Machine Learning Framework for One-Class Classification Tasks

Doctorate Thesis

Doctoral Program in Information Systems and Technology

Work developed under the supervision of:
Professor Doctor Paulo Alexandre Ribeiro Cortez

September, 2023

Acknowledgements

I am deeply grateful for this opportunity of pursuing a doctoral degree and for the many individuals who have contributed to my success along the way. Through their guidance, support, and belief in my abilities, we have achieved significant milestones and made meaningful contributions to our field of study.

I would like to express my sincere gratitude to Professor Paulo Cortez for his invaluable guidance, support, and encouragement throughout my PhD journey. His expertise, patience, and dedication to my academic success have been instrumental in shaping my research and professional development.

This heartfelt appreciation is also extended to my dear colleagues at Centro de Computação Gráfica for their generous support, insightful discussions, and willingness to collaborate. Their contributions and expertise have been invaluable to my research, and I am grateful for the opportunities to learn and grow alongside such talented individuals. I feel fortunate to have worked with colleagues who not only possess exceptional skills and knowledge but are also outstanding individuals.

I am also deeply grateful to my friends, who have been a constant source of support, laughter, and inspiration. While the pursuit of a PhD can be all-consuming, having good friends to share life's joys and challenges with has been essential to my well-being and overall success.

I would also like to express my heartfelt gratitude to my family, particularly my brother and mother, for their consistent love, support, and encouragement throughout my PhD journey. Their belief in my abilities, even during challenging times, has been an incredible source of motivation and inspiration for me, and I feel incredibly fortunate to have them in my life.

Finally, I don't have enough words (and probably never will) to thank my wife Águeda, who has been with me every step of the way throughout this journey. From boyfriend and girlfriend to now being husband and wife, she has been a constant source of love, encouragement, and support. Her understanding and her sacrifices have been instrumental and I am forever grateful for her presence in my life. I know that I could not have achieved this milestone without her by my side. I love you more than words can say and I will always cherish our journey together.

"To go wrong in one's own way is better than to go right in someone else's."
Fyodor Dostoevsky, Crime and Punishment

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the Universidade do Minho.

Abstract

An Automated and Efficient Machine Learning Framework for One-Class Classification Tasks

The present era of Machine Learning (ML) is defined by the availability of copious amounts of data, powerful algorithms, and high-speed processing machines that enable accurate predictions. Two crucial features of modern ML applications are automation and efficiency. Automation is essential for streamlining the ML workflow, particularly for non-specialists. Because of this, the emergence of Automated Machine Learning (AutoML), which automates various components of the workflow, has gained significant attention in recent years. Efficiency plays a vital role in dealing with Big Data or hardware constraints and it is often achieved through distributed or parallel learning across multiple machines or processors.

This thesis aims to contribute to the field of ML by designing and implementing an automated and efficient ML framework for One-Class Classification (OCC) tasks. A first set of initial experiments was performed to gain insights into the application of AutoML for supervised learning tasks, as well as to identify a robust and reliable evaluation method for the proposed AutoML framework. In these experiments, we defined an architecture to deal with the typical steps of the ML workflow and we performed a robust benchmark of existing AutoML frameworks.

Afterwards, in a second set of experiments, we propose a novel AutoML framework (AutoOneClass) that applies a Grammatical Evolution (GE) to design and evolve different OCC ML algorithms by using both single and multi-objective optimization applied to a real-world Predictive Maintenance (PdM) dataset. Then, we proposed an improved version of the framework (AutoOC), that is exclusively focused on a multi-objective optimization. Several computational experiments were held to evaluate the effectiveness of the AutoOC framework, using eight public datasets from several domains and two distinct validation modes (unsupervised and supervised).

Keywords: Automated Machine Learning, Efficient Machine Learning, Multi-objective Optimization, One-Class Classification, Grammatical Evolution.

Resumo

Um *Framework* de *Machine Learning* Automatizado e Eficiente Para Tarefas de *One-Class Classification*

A era atual da área de *Machine Learning (ML)* é definida pela disponibilidade de uma vasta quantidade de dados, algoritmos poderosos, e máquinas com processamento de alta velocidade que permitem previsões precisas. Duas características cruciais nas aplicações de *ML* modernas são a automatização e a eficiência. A automatização é essencial para a simplificação do fluxo de trabalho de *ML*, particularmente para os não-especialistas. Devido a isto, o surgimento da área de *Automated Machine Learning (AutoML)*, que automatiza vários componentes do fluxo de trabalho, tem ganho uma atenção significativa nos últimos anos. A eficiência desempenha um papel vital para lidar com *Big Data* ou restrições de *hardware* e é frequentemente alcançada através de aprendizagem distribuída ou em paralelo, usando várias máquinas ou processadores.

Esta tese visa contribuir para o campo de *ML* através da conceção e implementação de um *framework* de *ML* automatizado e eficiente para tarefas de *One-Class Classification (OCC)*. Um primeiro conjunto de experiências iniciais foi realizado para obter conhecimentos acerca da aplicação de *AutoML* para tarefas de aprendizagem supervisionada, bem como para identificar um método de avaliação robusto e fiável para o *framework* de *AutoML* a ser proposto. Nestas experiências, definiu-se uma arquitetura para lidar com as etapas tradicionais do fluxo de trabalho de *ML* e realizou-se um *benchmark* robusto de *frameworks* de *AutoML* existentes.

Posteriormente, num segundo conjunto de experiências, propôs-se um novo *framework* de *AutoML* (*AutoOneClass*) que aplica *Grammatical Evolution (GE)* para criar e evoluir differentes algoritmos *OCC* de *ML* usando otimização uni e multiobjetivo aplicados a um *dataset* real de Manutenção Preventiva. Em seguida, propôs-se uma versão melhorada do *framework* (*AutoOC*), que se centra exclusivamente em otimização multiobjetivo. Várias experiências computacionais foram realizadas para avaliar a eficácia do *framework AutoOC*, usando oito *datasets* públicos de vários domínios de aplicação e dois modos de validação distintos (não supervisionado e supervisionado).

Palavras-chave: *Automated Machine Learning*, *Efficient Machine Learning*, Otimização Multiobjetivo, *One-Class Classification*, *Grammatical Evolution*.

Contents

List of Figures

List of Tables

Acronyms

AE	Autoencoder
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
AUC	Area Under the Curve
AutoDL	Automated Deep Learning
AutoML	Automated Machine Learning
AutoOC	Automated One-Class tool (version 2)
AutoOneClass	Automated One-Class tool (version 1)
BIC	Bayesian Information Criterion
BNF	Backus–Naur Form
CASH	Combined Algorithm Selection and Hyperparameter Optimization
CCG	Centro de Computação Gráfica
CMMS	Computerized Maintenance Management System
CRISP-DM	Cross-Industry Standard Process for Data Mining
DL	Deep Learning
DSRM-IS	Design Science Research Methodology for Information Systems
DT	Decision Tree
EC	Evolutionary Computation
EPMQ	IT Engineering - Process, Data, Maturity, and Quality
GA	Genetic Algorithm
GBM	Gradient Boosting Machine
GBT	Gradient-Boosted Tree
GE	Grammatical Evolution

GLM	Generalized Linear Model
GML	General Machine Learning
IDF	Inverse Document Frequency
IF	Isolation Forest
IRMDA	Intelligent Risk Management for the Digital Age
IS	Information Systems
LOF	Local Outlier Factor
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
ML	Machine Learning
NAS	Neural Architecture Search
NE	Neuroevolution
NN	Neural Network
NSGA-II	Non-dominated Sorting Genetic Algorithm II
OC-SVM	One-Class SVM
OCC	One-Class Classification
PdM	Predictive Maintenance
pp	Percentage Points
RF	Random Forest
RMSE	Root Mean Squared Error
ROC	Receiver Operating Characteristic
SMOTE	Synthetic Minority Oversampling Technique
SVM	Support Vector Machine
VAE	Variational Autoencoder
XGB	XGBoost

Chapter 1

Introduction

This chapter starts by presenting the main motivation for this PhD work (Section 1.1) and the main objectives (Section 1.2). Then, it presents the adopted research methodology (Section 1.3), followed by a detailed description of the scientific contributions (Section 1.4) and the organization of this PhD thesis (Section 1.5).

1.1 Motivation

Currently, there is growing success in the usage of **Machine Learning (ML)** algorithms in a wide range of real-world application domains. This success resulted from three important phenomena: the availability of Big Data, increase of computational power and more sophisticated learning techniques (Darwiche, 2018). When targeting a real-world application context, there are two important aspects that impact the effectiveness of the ML approach: automation and efficiency. Both these aspects are addressed in this PhD.

Automation is an essential ML modeling aspect, allowing people with limited knowledge in the field to easily deal with tasks of the ML workflow. The development of a ML application includes typically several data processing steps, such as data preparation, feature engineering, algorithm selection, and hyperparameter tuning. Most of these steps require trial and error approaches, especially for non-ML experts. More experienced practitioners often use heuristics to exploit the vast dimensional space of parameters (Lin et al., 2018). With the increasing number of non-specialists working with ML (Thornton et al., 2013), in the last years there has been a focus on automating the components of the ML workflow, giving rise to the area of **Automated Machine Learning (AutoML)** (Guyon et al., 2019).

Efficiency is another relevant aspect that is particularly useful for ML applications in the context of Big Data or when there are hardware constraints. Efficiency is often addressed by using distributed or parallel learning, making use of multiple machines or processors to process parts of the ML algorithm or the data. The fact that it is possible to adjust the number of processing units enables ML applications to surpass time and memory restrictions (Peteiro-Barral & Guijarro-Berdiñas, 2013).

It is also important to mention that this PhD work was partially funded and developed within the contexts of two R&D projects, each involving a distinct real-world ML application domain:

- **Intelligent Risk Management for the Digital Age (IRMDA)**, initially approached, with a duration of 2 years and related with the telecommunications risk management domain; and
- **Computerized Maintenance Management System (CMMS)**, executed at a later stage, with a duration of around one year and related with the predictive maintenance domain.

This research context naturally motivated and influenced the execution of this PhD, with several of the analyzed R&D ML requisites, datasets and tasks giving rise to research opportunities that were approached in this work.

1.2 Objectives

This PhD thesis assumed an initial and more general goal that consisted in the design, development and evaluation of a novel ML framework that uses automated methods to search for computationally efficient ML models. As shown in Chapter 2, there are several recent research works that address AutoML or efficiency in ML by means of a distributed computing. However, there is scarce research that combines both automation and efficiency aspects within the scope of real-world ML applications.

It should be noted that the ML field includes a diverse range of learning paradigms and tasks. Originally, this PhD was focused on a supervised learning, which is a popular learning paradigm that is highly targeted by AutoML proposals. During the execution of the PhD work, and motivated by the involvement in second R&D project (CMMS), the PhD goal was redefined towards a less studied and more specific task: **One-Class Classification (OCC)**. Thus, the refined and more specific Research Question (RQ) of this PhD work is: **How to design a framework that allows an automated and efficient development of OCC ML models?** In order to answer this RQ, this PhD proposes a computationally efficient AutoML tool to automate the design of lightweight OCC ML models using a **Grammatical Evolution (GE)**.

To achieve the proposed GE framework, several intermediate objectives were addressed during the execution of this PhD, namely:

- Search for state-of-the-art AutoML frameworks or open-source tools that allow an automated and efficient execution of ML algorithms (e.g., via distributed computing).
- Perform a robust benchmark of existing AutoML frameworks.
- Develop an initial version of a novel AutoML framework focused on OCC tasks. The main goal of the framework is to search for the best OCC ML algorithms and its associated hyperparameters, assuming a single or multi-objective search.

- Develop an improved version of the novel AutoML framework, focusing exclusively on multi-objective optimization to generate lightweight ML models.

- Evaluate the proposed OCC framework, comparing the performance against existing AutoML tools, using metrics to evaluate the predictive performance and efficiency of the framework.

1.3 Research Methodology

Given that this PhD results in the development of an artifact that addresses a specific problem, namely a ML framework, the adopted approach was a Design Research, namely **Design Science Research Methodology for Information Systems (DSRM-IS)**. The DSRM-IS methodology consists of a set of principles, practices, and procedures for performing design science research in the area of **Information Systems (IS)**. DSRM-IS (Fig. 1) includes six main activities: Problem Identification and Motivation, Define the Objectives for a Solution, Design and Development, Demonstration, Evaluation, and Communication (Peffers et al., 2008). This process is iterative, meaning that is possible to restart from a previous step until the artifact is finished. In this section, we summarize how each of these steps was applied in this PhD.

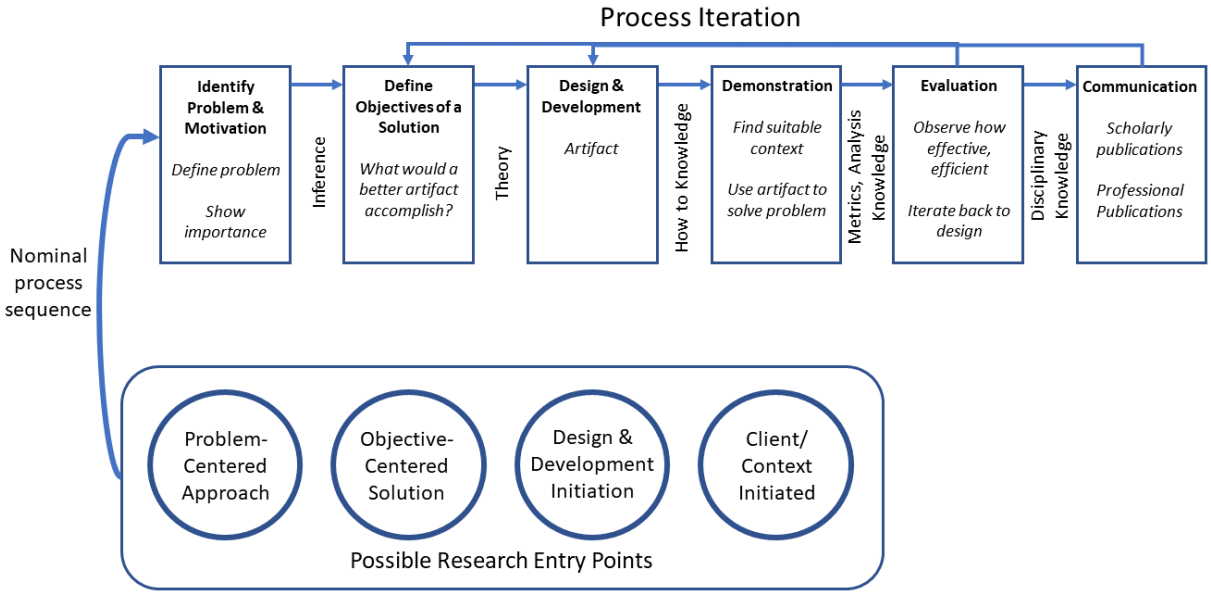


Figure 1: DSRM-IS Process Model. Adapted from Peffers et al. (2008).

In recent years, automation and efficiency have been increasingly considered important capabilities of real-world ML applications. With the increasing number of non-specialists working with ML (Thornton et al., 2013), automation is essential to allow people with limited knowledge in the field to easily deal with tasks of the ML workflow. AutoML aims to help solve this issue and it is particularly relevant when constant model updates are required. Efficiency is also necessary to scale ML problems when there are memory

or time limitations. For example, by using multiple computers or multi-core processors in parallel, each processor can process a different portion of the data or ML algorithm. This solution allows overcoming memory and time constraints to growing data just by adding new machines or processors (Peteiro-Barral & Guijarro-Berdiñas, 2013).

The first step of this PhD was the awareness of the critical importance of automation and efficiency in the field of ML. After analyzing the state-of-the-art, it became apparent that utilizing efficient AutoML algorithms could lead to significant improvements in the ML body of knowledge. This could lead to more effective decision-making and increased efficiency in various applications. Furthermore, exploring and implementing these algorithms could also contribute to the advancement of the field of ML. A solution was formulated, leading to the next step of the methodology.

The main goal of the second activity of DSRM-IS is to infer the objectives of the solution from the defined problem, aiming to increase the existing body of knowledge. In this PhD, the suggested solution consisted of proposing a computationally efficient AutoML tool to automate the design of OCC ML models using a multi-objective optimization approach. Specifically, a GE algorithm was used to generate and train OCC predictive models, such as **Autoencoder (AE)**, **Isolation Forest (IF)**, **Local Outlier Factor (LOF)**, **One-Class SVM (OC-SVM)**, or **Variational Autoencoder (VAE)**. This decision was taken based on a research gap identified in the state-of-the-art studies, showing that OCC is rarely approached by AutoML tools.

The next activity consists of the development of the artifact, namely the automated and efficient OCC ML framework. In this PhD, the development was performed iteratively, by experimenting and applying different ML approaches, optimization techniques, or combinations of both.

The fourth activity requires the demonstration of the use of the artifact to solve at least one of the instances of the problem. Since this PhD project had more than one task, we ensured that at least one of the tasks used real-world data involved in an R&D project, which was also used to evaluate the proposed framework. In other tasks, we focused on using open-source data from several industries and domains, to demonstrate the versatility of the proposed artifact.

For the evaluation activity of DSRM-IS, different approaches were compared using several measures, such as computational effort and predictive performance. To ensure a robust and fair comparison, the best approaches were then selected and compared with relevant state-of-the-art baseline models, other AutoML tools, and also the best ML modeling as performed by humans (via the OpenML platform). A novel multi-objective AutoML method was developed, as further detailed in Chapters 4 and 5.

The final activity of the DSRM-IS methodology is achieved when the evaluated artifact produces satisfactory results. In the context of this PhD project, satisfactory refers to the development of a framework that demonstrates better performance when compared to existing methods in a robust benchmark, while also meeting all established objectives. As identified by the methodology, several iterations were required until the desired level of success was achieved. Concluding remarks were then drawn, which resulted in scientific publications. Indeed, the findings of this PhD work were published in several international scientific conferences and journals, as summarized in Section 1.4. The scientific work associated with

these articles is detailed in Sections 3.2 and 3.3 and Chapters 4 and 5.

1.4 Contributions

The main goal of this thesis is to give a strong contribution to the body of knowledge in the area of ML with the design and implementation of an automated and efficient ML framework. Under this context, a collection of research papers were written during the execution of this PhD project with the goal of reaching the research objectives outlined in Section 1.2.

This PhD work was partially developed under the IRMDA R&D project, with a duration of two years. The project was developed by a leading Portuguese company in the area of software and analytics. The purpose of the project was to develop a ML system to assist the company's telecommunications clients. Both scalability and automation were central requirements of the ML system since the company had many clients with diverse amounts of data (both large or small) and that are typically non-ML experts. A ML technological architecture was proposed to identify and automate all typical tasks of a common supervised ML application, with minimum human input. Also, the architecture was developed to work within a computational cluster with several processing nodes. Nevertheless, during the project execution time, which approximately corresponded to this doctoral thesis first two years, it was not possible to execute more mature PhD tasks, such as proposing and evaluating a novel AutoML framework. Hence, we decided to start the PhD work with the exploration of existing AutoML frameworks, which is the main topic of this thesis, giving particular focus to automated and distributed ML and using three distinct datasets from a real-world domain, related to Telecom risk management. The purpose was to gain insights into the application of AutoML for supervised learning tasks, as well as to identify a robust and reliable evaluation method for the proposed AutoML framework. The PhD work developed under the IRMDA project is detailed in Chapter 3 and resulted in the following three publications, consisting of two conference papers and an invitation to an extended publication (book chapter):

- Ferreira, L., Pilastri, A. L., Martins, C., Santos, P., & Cortez, P. (2020b). An Automated and Distributed Machine Learning Framework for Telecommunications Risk Management. In A. P. Rocha, L. Steels, & H. J. van den Herik (Eds.), *Proceedings of the 12th International Conference on Agents and Artificial Intelligence, ICAART 2020, Volume 2, Valletta, Malta, February 22-24, 2020* (pp. 99–107). SCITEPRESS. https://doi.org/10.5220/0008952800990107 *(indexed at Scopus, Clarivate and DBLP scientific databases)*

- Ferreira, L., Pilastri, A. L., Martins, C., Santos, P., & Cortez, P. (2020a). A Scalable and Automated Machine Learning Framework to Support Risk Management. In A. P. Rocha, L. Steels, & H. J. van den Herik (Eds.), *Agents and Artificial Intelligence, 12th International Conference, ICAART 2020, Valletta, Malta, February 22-24, 2020, Revised Selected Papers* (pp. 291–307). Springer. https://doi.org/10.1007/978-3-030-71158-0_14 *(indexed at Scopus)*

- Ferreira, L., Pilastri, A. L., Martins, C. M., Pires, P. M., & Cortez, P. (2021). A Comparison of AutoML Tools for Machine Learning, Deep Learning and XGBoost. *International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021*, 1–8. https://doi.org/10.1109/IJCNN52387.2021.9534091 *(indexed at Scopus, Clarivate, DBLP and IEEE Xplore; ranked CORE B 2021)*

The next contributions were developed under the CMMS R&D project, related to the application of ML for **Predictive Maintenance (PdM)**. This project also had a short duration, approximately one year, which corresponded to the third year of this PhD project. Under the CMMS project, the first preliminary experiments related to a novel AutoML framework were conducted by applying a GE to design and evolve different OCC ML algorithms using both single and multi-objective searches. We first applied several state-of-the-art AutoML tools using a real-world PdM dataset. Then, we proposed **Automated One-Class tool (version 1) (AutoOneClass)**, an AutoML framework that focuses on OCC using three algorithms: deep AEs, IF, and OC-SVM. The method used GE to optimize the search for the best OCC ML algorithm and its associated hyperparameters, assuming a single or multi-objective search. The single-objective approach only uses the predictive performance to select the best ML model, while the multi-objective variant considers two objectives simultaneously, predictive performance and training time. The proposed method also uses two validation setups: unsupervised - using unlabeled data during validation and anomaly scores to evaluate the ML models; and supervised validation - using a labeled validation set to assess model performance. This R&D project provided practical knowledge about the usage of OCC algorithms within AutoML and contributed to advancing the state-of-the-art regarding the implementation of AutoML approaches in the PdM industry. The work developed under the CMMS project is further described in Chapter 4 and resulted in two papers, consisting of one conference paper (winner of Best Paper Award) and one Q1 journal article:

- Ferreira, L., Pilastri, A. L., Sousa, V., Romano, F., & Cortez, P. (2021). Prediction of Maintenance Equipment Failures Using Automated Machine Learning. *Intelligent Data Engineering and Automated Learning - IDEAL 2021 - 22nd International Conference, IDEAL 2021, Manchester, UK, November 25-27, 2021, Proceedings*, *13113*, 259–267. https://doi.org/10.1007/978-3-030-91608-4_26 *(indexed at Scopus and DBLP)*

- Ferreira, L., Pilastri, A., Romano, F., & Cortez, P. (2022). Using Supervised and One-Class Automated Machine Learning for Predictive Maintenance. *Applied Soft Computing, Elsevier*, *131*, 109820. https://doi.org/10.1016/j.asoc.2022.109820 *(Clarivate Q1 in "Computer Science, Artificial Intelligence"; Scimago Q1 in "Software")*

In the fourth and final year of this doctoral program, the focus was to further develop the initial version of the AutoML framework proposed in Chapter 5. This work resulted in **Automated One-Class tool (version 2) (AutoOC)**, an improved version of the AutoOneClass method. This new contribution focused

exclusively on OCC ML algorithms, identified as a research gap in the previous research work since it was identified in the state-of-the-art that the vast majority of AutoML tools target a supervised learning (e.g., classification, regression) and do not handle an OCC. Moreover, this new contribution focused exclusively on a multi-objective optimization, using the **Non-dominated Sorting Genetic Algorithm II (NSGA-II)** algorithm to maximize the predictive performance of the OCC learners while minimizing their training time. The goal was to address the efficiency part of the PhD objectives, by generating lightweight ML models, an important aspect when working with real-world Big Data. Furthermore, the proposed AutoOC adopts two computationally efficient mechanisms to speed up the overall execution time: a continuous sampling of training data and a parallel fitness evaluation by adopting multi-core processors. Several computational experiments were held to evaluate the effectiveness of AutoOC, using eight public datasets from several domains and two distinct validation modes (unsupervised and supervised). The results were compared with a baseline state-of-the-art OCC algorithm and also with public predictive results. This work is described in Chapter 5 and resulted in a paper publication in an international Q1 journal:

- Ferreira, L., & Cortez, P. (2023). AutoOC: Automated Multi-objective Design of Deep Autoencoders and One-class Classifiers Using Grammatical Evolution. *Applied Soft Computing*, *144*, 110496. https://doi.org/10.1016/j.asoc.2023.110496 *(Clarivate Q1 in "Computer Science, Artificial Intelligence"; Scimago Q1 in "Software")*

Additionally, the work developed in Chapters 4 and 5 was adapted and released as a Python library named AutoOC, available both on GitHub[1] and on PyPi[2]. The AutoOC Python library is further described in Appendix B and resulted in a submission to an international journal:

- Ferreira & Cortez, P. (2023). AutoOC: A Python Module for Automated Multi-objective One-Class Classification.

Finally, it is also worth mentioning that during the PhD execution time there were some collaborative efforts with two Master of Science students and other R&D projects regarding the ML area of the **IT Engineering - Process, Data, Maturity, and Quality (EPMQ)**[3] domain of the **Centro de Computação Gráfica (CCG)** unit. These efforts resulted in the publication of one journal article and three conference papers that are not directly related with this PhD (e.g., involving a Federated ML) and thus are not included in this document.

1.5 Thesis Organization

This thesis is divided into six chapters. Chapter 1 presents the motivation and objectives of this PhD. It also provides an overview of the adopted research strategy and details the research contributions and document

[1] https://pepy.tech/project/autooc
[2] https://pypi.org/project/autooc
[3] https://ccg.pt/domains/epmq-presentation/?lang=en

structure. Chapter 2 presents the relevant state-of-the-art. It starts by presenting the bibliographic search strategy. Then, it provides a theoretical introduction of relevant concepts addressed during this project, such as Machine Learning (Section 2.2.1), Automated Machine Learning (Section 2.2.2), One-Class Classification (Section 2.2.3), **Neuroevolution (NE)** (Section 2.2.4), and multi-objective optimization (Section 2.2.5). The chapter ends with a state-of-the-art analysis of recent and relevant studies related to the application of AutoML techniques to different ML tasks and business domains. Specifically, distributed AutoML applied to risk management (Section 2.3.1), comparison of AutoML tools (Section 2.3.2), supervised and One-Class Classification AutoML applied to Predictive Maintenance (Section 2.3.3), and automated and multi-objective One-Class Classification (Section 2.3.4).

Next, Chapter 3 is related to our initial supervised AutoML experiments, presenting the methods, experiments, and results produced within this PhD work, which was partially developed under the IRMDA R&D project. This chapter is divided into two main parts. First, Section 3.2 describes a proposed technological architecture for the telecommunications risk management that addresses the ML challenges of automation and scalability. The proposed AutoML architecture delineates a set of steps to automate the typical workflow of a ML application (e.g., data preprocessing, model training) that uses supervised learning. The focus of this architecture was the model training module of the architecture, which was designed to use a distributed AutoML tool, resulting in a benchmark to compare two tools that allowed a distributed execution using three real-world datasets from the domain of telecommunications. The second part of this chapter (Section 3.3) presents a comparison study that considers eight recent open-source AutoML technologies (Auto-Keras, Auto-PyTorch, Auto-Sklearn, AutoGluon, H2O AutoML, rminer, TPOT, and TransmogrifAI). To evaluate these tools, twelve popular datasets were retrieved from the OpenML platform, divided into regression, binary, and multi-class classification tasks. Three main scenarios were designed for the benchmark study: **General Machine Learning (GML)** algorithm selection; **Deep Learning (DL)** selection, also known as **Automated Deep Learning (AutoDL)**; and **XGBoost (XGB)** hyperparameter tuning. Each tool was measured in terms of its predictive performance (using an external 10-fold cross-validation) and computational cost (measured in terms of time elapsed).

Then, Chapter 4 describes the first preliminary experiments related to a novel AutoML framework, which were conducted by applying GE to design and evolve different OCC ML algorithms using both single and multi-objective optimization. The main contribution of this chapter is the proposal of AutoOneClass, an AutoML framework that focuses on OCC using three algorithms: deep AEs, IF, and OC-SVM. The method used GE to optimize the search for the best OCC ML algorithm and its associated hyperparameters, assuming a single or multi-objective search. Furthermore, this chapter provides a robust benchmark, predicting equipment failures in different time windows (e.g., 3 days, 5 days) and comparing the results from the proposed AutoOneClass method with ten AutoML tools, focused on classical ML and DL.

Chapter 5 outlines the experiments performed regarding the AutoOC framework, an improved version of the AutoOneClass method, proposed in Chapter 4. AutoOC focuses exclusively on OCC ML algorithms, identified as a research gap in the previous research work. Moreover, this new version focuses exclusively on a multi-objective optimization, using the NSGA-II algorithm to maximize the predictive performance of

the OCC learners while minimizing their training time. The goal of this new contribution was to address the efficiency part of the PhD objectives, by generating lightweight ML models, an important aspect when working with real-world Big Data. Furthermore, the proposed AutoOC adopts two computationally efficient mechanisms to speed up the overall execution time: a continuous sampling of training data and a parallel fitness evaluation by adopting multi-core processors. The framework includes other enhancements when compared with AutoOC, such as the optmization of a larger number of ML base learners. Several computational experiments were held to evaluate the effectiveness of AutoOC, using eight public datasets from distinct domains and two validation modes (unsupervised and supervised). The results were compared with a baseline state-of-the-art OCC algorithm and also with public ML predictive human modeling results.

Finally, Chapter 6 presents the main conclusions of this doctoral project, identifies its limitations, and draws possible future work directions.

Chapter 2

Background

This chapter presents the relevant state-of-the-art. It starts by presenting the bibliographic search strategy (Section 2.1). Then, it provides a theoretical introduction of relevant concepts addressed during this project, such as Machine Learning (Section 2.2.1), Automated Machine Learning (Section 2.2.2), One-Class Classification (Section 2.2.3), Neuroevolution (Section 2.2.4), multi-objective optimization (Section 2.2.5), and evaluation metrics (Section 2.2.6). The chapter ends with a state-of-the-art analysis of recent and relevant studies related to the application of AutoML techniques to different ML tasks and business domains. Specifically, distributed AutoML applied to risk management (Section 2.3.1), comparison of AutoML tools (Section 2.3.2), supervised and One-Class Classification AutoML applied to Predictive Maintenance (Section 2.3.3), and automated and multi-objective One-Class Classification (Section 2.3.4).

2.1 Bibliographic Search Strategy

In order to identify the relevant state-of-the-art for this PhD and identify research gaps, a literature review was conducted. The approach used was a traditional literature review, as described by Jesson et al. (2011). The used references included material provided by the supervisor and search in scientific databases. The used bibliographic databases were Google Scholar[1], Scopus[2], Web of Science[3], ACM Digital Library[4], IEEE Xplore[5], and Science Direct[6]. The main keywords used in each search engine were: "automated machine learning", "automl", "one-class classification", "one-class learning", "evolutionary computation", "grammatical evolution", "efficient machine learning", "distributed machine learning", "machine learning scalability", among others. In each search filters were applied to retrieve recent publications, setting the year range between 2010 and 2022. For the cases where the time filter returned insufficient or unrelated results, the filter was removed.

[1] https://scholar.google.pt
[2] https://www.scopus.com
[3] https://apps.webofknowledge.com
[4] https://dl.acm.org
[5] https://ieeexplore.ieee.org
[6] https://sciencedirect.com

To be considered for the literature review, the works needed to be published in a journal, conference proceedings, book, or technological report. Also, the work should be published in English. In cases where the work has been published in more than one publication, only the most extensive work was considered (e.g., journal article). When two or more publications presented similarities between them, the works that had the most Google Scholar citations and that were published in journals and conferences with higher quality ranks were considered. For journals, the Scimago Journal & Country Rank[7] was used to check the journal ranks. For conferences, the used rank was the Computing Research and Education Association of Australasia (CORE)[8].

The final selection of works was made via manual inspection of the works retrieved from the bibliographic databases. The manual inspection was carried out in three steps. First, the title, abstract, and keywords of each work were read. If the document was considered relevant, it was saved for further reading, described in the second step. Otherwise, the document was discarded. Then, the second step included the full reading of the saved publications. The works that were considered irrelevant upon full reading were also discarded. The third and final step consisted of the search for relevant works based on the publication references. The references of the articles that were selected in step two were analyzed. If the title of the reference was considered potentially relevant, the three steps of the manual inspection were applied to that work. The result of the application of the document manual inspection resulted in the bibliography presented in this literature review.

2.2 Relevant Concepts

This section provides a theoretical introduction of relevant concepts addressed during this project, such as Machine Learning and the different types of learning (Section 2.2.1), Automated Machine Learning (Section 2.2.2), One-Class Classification algorithms (Section 2.2.3), Neuroevolution (Section 2.2.4), multi-objective optimization (Section 2.2.5), and evaluation metrics (Section 2.2.6).

2.2.1 Machine Learning

"The learning machine" was the term proposed by Turing (1950) when referring to a machine that could learn for itself, simulating the human mind. This seminal article paved the way for the development of new techniques related to learning machines. Years later, Samuel (1959, 1967) coined the term "Machine Learning" when applying learning techniques to allow machines to play checkers. ML is now considered a subfield of **Artificial Intelligence (AI)**. Whereas AI is related to the creation and development of intelligent agents (Poole et al., 1998), ML is the subfield that focuses on the techniques that allow machines to improve their performance when exposed to new experiences (e.g., more data) over time (Han et al., 2011).

[7] https://scimagojr.com
[8] https://www.core.edu.au

Nowadays, ML algorithms are present everywhere, when people query Google and Google answers them, when spam is filtered into the email platform, when Amazon or Netflix recommend a book or movie to the user, or even when Facebook and Twitter choose which messages to show (Domingos, 2018). There are different forms of learning according to the representation of the data, the existence of labels on the data, or the existence of learning feedback (Russell & Norvig, 2003). The following two subsections describe some of the different types of learning within the field of ML. Then, Section 2.2.1.3 describes some of the most common types of ML algorithms.

2.2.1.1 Supervised Learning

In supervised learning, the learning algorithm (also called learner) receives a set of labeled records consisting of pairs of input and output and tries to map the inputs to the outputs, generalizing their relation. The goal of supervised learning algorithms is to be able to generalize the relation between inputs and outputs and to be able to map new inputs to their correct outputs (Mohri et al., 2012). It is sometimes compared to a teacher-student type of learning. One example of supervised learning is email spam detection. In this example, the supervised learning algorithm uses a set of emails that are classified (either spam or not spam) and tries to learn how to classify the emails. If the learning is done properly, when the algorithm is given a new email, it can correctly predict if it is spam or not.

Within supervised learning, there are two main subgroups. When the output is a discrete variable (e.g., rainy, cloudy, snowy), it is considered a classification task. More specifically, when the output only has two possible values (e.g., spam or not spam), they are sometimes referred to as binary classification tasks. When the output has more than two possible values, it is called multi-class classification. On the other hand, supervised learning is considered a regression when the outputs are continuous. In regression, the answer to the question is represented by a quantity and not by a quality (i.e., a label). This type of problem can be used to predict house prices in a given country, the likelihood of a customer abandoning the service, or predict when the company will profit in the next quarter. Time series forecasting is a particular case of a regression task, in which the data consists of sets of observations ordered over time (Brockwell & Davis, 2016).

2.2.1.2 Unsupervised Learning

Unlike supervised learning, unsupervised learning only deals with unlabeled data. In other words, it does not deal with input-output pairs, only inputs (Mohri et al., 2012). It can be considered a type of learning without a teacher. Unsupervised learning algorithms try to infer patterns in the data without the help of a teacher (i.e., output) who gives the right answers or helps identify the right direction (Hastie et al., 2009).

One of the most common types of unsupervised learning algorithms is clustering algorithms. This type of algorithm organizes the data into groups according to their characteristics or similarities (Jain, 2010). One application example of clustering algorithms is the grouping of clients by purchasing behavior. Within clustering, K-means is one of the most used algorithms. K-means tries to find groups in the data,

using a predefined number of groups (k). Iteratively, K-means tries to associate each record of the data to one of the k groups according to its characteristics. Another group of algorithms of this type of learning is outlier detection. These algorithms attempt to identify observations that differ from most of the data (Hodge & Austin, 2004). In this PhD, we focus on the OCC unsupervised learning task, which can be used to perform anomaly or outlier detection and that is detailed in Section 2.2.3.

2.2.1.3 Machine Learning Algorithms

For a given ML problem, there are many possible algorithms to use. Some supervised learning algorithms can be used for both classification and regression. Other algorithms can be used for more than one type of learning, such as supervised and unsupervised learning. This section briefly describes some ML algorithms that have been used in AutoML state-of-the-art works and that are used in the context of this PhD.

- **Decision Tree (DT)**: is a supervised learning algorithm used for both classification and regression. In a classification task, the Decision Tree (DT) is usually named a classification tree. In a regression task, DT is usually referred to as a regression tree. DTs can be represented by nodes and edges. The nodes represent conditions and the edges represent possible answers to those questions. The leaf nodes are nodes that do not have edges, which correspond to the possible outputs of the data (Rokach & Maimon, 2007). In the case of a classification task, the leaf nodes correspond to the classes of the data. The fact that the algorithm's main decisions are divided into smaller decisions makes it easy to interpret by a human, unlike other algorithms. Besides that, it is a non-parametric algorithm that can handle outliers or insignificant attributes (Gama et al., 2003). For classification tasks, the time of training is very low (Ho, 1995).

- **Random Forest (RF)**: is a supervised learning algorithm that is used for classification and regression. It consists of a combination of DTs that are randomly applied to subspaces of the data (Biau, 2012; Breiman, 2001). The idea of Random Forest (RF) is to combine (bagging) the results of many DTs that are potentially noisy but can help reduce the variance (Hastie et al., 2009; Ho, 1995). The method for combining different trees can be different. For example, for regression, the method for combining trees can be the mean value. For classification, it is possible to apply the mode for joining trees. Since RF includes more than one DT, it is considered an ensemble algorithm.

- **Linear Regression**: is an algorithm that models the relationship between a set of inputs (explanatory variables) and the outputs (response variables). It tries to identify a linear combination of the input variables to predict the output variable (Bishop, 2007). It is used for supervised learning, mostly regression. When there is only one explanatory variable, the linear regression is called simple linear regression. In cases where there is more than one explanatory variable, the model is called multiple linear regression (Tranmer & Elliot, 2008).

- **Support Vector Machine (SVM)**: is a ML algorithm used for supervised learning that has a foundation in statistical learning theory (Cortes & Vapnik, 1995; Vapnik, 1998). The algorithm first transforms the data into a high-dimensional space using kernel functions (L. Wang, 2005). Then, it constructs a hyperplane that separates the support vectors of each class, in case of a classification task (Cristianini & Shawe-Taylor, 2000). Although Support Vector Machine (SVM) are mostly applied to supervised learning, there are modified versions of the algorithm to deal with unsupervised learning (e.g., Support Vector Clustering).

- **Neural Network (NN)**: also called **Artificial Neural Network (ANN)**, is a family of ML algorithms inspired by biological neural networks that exist in the human brain (Y. Chen et al., 2019). Neural Network (NN)s can be applied to most types of learning, such as supervised learning (either classification or regression), unsupervised learning, and reinforcement learning. They are used for a variety of applications, such as image classification or speech recognition (Szegedy et al., 2014). When there is the availability of a vast amount of data and processing power, NNs can be a very effective type of algorithm (Aggarwal, 2018). In ML, a NN can be represented by groups of processing units (also called neurons) organized by layers. Each neuron is connected to other neurons of the adjacent layers and each connection carries a weight that is changed during training (B. Cheng & Titterington, 1994). The topology of a NN is related to the way the neurons are connected. In supervised learning one of the most common topologies is a fully connected NN. In these types of networks, each neuron is connected to all neurons of the next layer (Miikkulainen, 2010). Besides topology, NNs can be classified according to other characteristics, such as the direction of information flow, type of learning, and degree of supervision (Basheer & Hajmeer, 2000). After the 2010s, there was a growing rediscovery of NNs, given that several multi-layered NNs have obtained substantially better results in several world competitions (e.g., computer vision, Natural Language Processing tasks). The term DL become used to term these types of NNs, which are considered the current state-of-the-art ML methods in diverse supervised and unsupervised learning tasks (Goodfellow et al., 2016). Some common types of NNs are Multilayer Perceptrons, Radial Basis Neural Networks, Convolutional Neural Networks, AEs, and **Long Short-Term Memory (LSTM)** Networks.

2.2.2 Automated Machine Learning

A typical ML application includes the phases of data preparation, feature engineering, feature selection, algorithm selection, and hyperparameter tuning. Most of these tasks when performed manually often become time-consuming processes. For beginners in ML, most of the phases of the typical workflow require the use of trial-and-error computer experiments to find the best configuration for a given problem. In the case of algorithm selection and hyperparameter tuning, the high number of available algorithms and hyperparameters makes the choice often based on intuition and often using pre-defined parameters. Also, most ML beginners use a trial-and-error approach to find the best configuration for a given problem.

Sometimes the more experienced professionals use handmade heuristics to exploit the large space of options for algorithms and hyperparameters (Lin et al., 2018).

In the last few years, there has been an attempt to automate some of the phases of the ML workflow. With the increasing number of non-specialists working with ML (Thornton et al., 2013), allowing people with more limited knowledge in the field to deal with these steps with ease is a current research interest in the ML field. To address the problem of automation in ML, the concept of AutoML arises.

AutoML is a fairly recent term and therefore lacks a commonly accepted definition. Even though the concept is always related to the automation within the field of ML, different authors approach the concept in different ways. AutoML is almost always considered the automation of the modeling phase of **Cross-Industry Standard Process for Data Mining (CRISP-DM)**, which includes algorithm selection and hyperparameter tuning. According to the definition of Feurer et al. (2015), AutoML is focused on the fundamental problems of ML of choosing the algorithm to be used in a given dataset, whether or not to preprocess its attributes and how to establish all hyperparameters. The authors define AutoML as the automatic search (without the need for human input) of a ML algorithm and respective hyperparameters, within a fixed computational effort, which includes memory usage and time limit. Other authors define AutoML as a **Combined Algorithm Selection and Hyperparameter Optimization (CASH)** problem (Thornton et al., 2013).

Within the ChaLearn AutoML Challenge, AutoML is associated with the progressive automation of all phases of ML (Guyon et al., 2015; Guyon et al., 2016; Guyon et al., 2019). Besides algorithm selection and hyperparameter tuning, the authors include other tasks within the field of AutoML. These tasks include data loading and formatting, handling of missing data, feature extraction, matching between algorithm and problem, active learning, data splitting, selection of algorithms according to time restrictions, generating reusable workflows, meta-learning, and explanatory report generation.

Other authors approach AutoML only within the domain of DL. For Jin et al. (2019), the goal of AutoML is to allow ML practitioners to automatically search for architectures and hyperparameters of DL models. The application of AutoML in the DL context is usually named **Neural Architecture Search (NAS)** (Elsken et al., 2019). There are few authors that use other terms for ML automation. For instance, Li and Moore (2007) use the term semi-automated ML in the context of an Internet traffic classification application. The application includes mechanisms to make the selection of the dataset features, as well as the algorithm to be used, which depends on factors such as latency and memory usage.

Although there is no general definition of the term AutoML, all definitions or applications agree that AutoML is a process of automating the application of ML to real problems. All definitions set the goal of getting good and fast solutions to a problem without the need to spend time and human resources. Despite the lack of a general definition, in this PhD AutoML is addressed as the automation of one or more phases of the ML workflow.

In recent years, many AutoML tools have been proposed. Here we provide a summary of the AutoML tools that have been studied and used in this PhD:

- **Auto-Keras** is an AutoML Python library based on Keras (Jin et al., 2019). It is designed to automate the construction on DL algorithms, commonly named AutoDL or Neural Architecture Search (NAS) (Elsken et al., 2019). Auto-Keras automatically tunes hyperparameters of NNs, such as the number of layers and neurons, activation functions, or dropout values.

- **Auto-PyTorch** is another AutoDL tool, based on the PyTorch framework. Auto-PyTorch uses multi-fidelity optimization with portfolio construction to automate the construction of DL networks (Zimmer et al., 2021).

- **Auto-Sklearn** is an AutoML library based on the popular Scikit-Learn framework (Pedregosa et al., 2011). It uses Bayesian optimization, meta-learning, and Ensemble Learning modules to automate algorithm selection and hyperparameter tuning (Feurer et al., 2019).

- **Auto-Weka** is a module of WEKA, a ML tool that provides data preprocessing functions and ML algorithms that allow users to quickly compare ML models and create predictions using new data (Witten et al., 2016). Auto-Weka aims to solve the CASH problem, first established by Thornton et al. (2013).

- **H2O AutoML** is one of the open-source modules of H2O, a ML analytics platform that uses in-memory data and implements a distributed and scalable architecture (Cook, 2016). H2O AutoML uses the H2O infrastructure to provide functions to automate algorithm selection and hyperparameter optimization. H2O AutoML runs several algorithms from H2O and several Stacked Ensembles, with subsets of the trained ML models (H2O.ai, 2021).

- **MLJar** provides an AutoML framework that includes algorithm selection, hyperparameter tuning, feature engineering, feature selection, and Explainable AI (XAI) capabilities. From the three available modes of MLJar, in this PhD experiments we used the "Perform" mode[9], since it is considered the most appropriate mode for a real-world usage (Płonska & Płonski, 2022).

- **PyCaret** is an open-source ML Python library that automates ML workflows using low code functions. PyCaret provides an AutoML function (`compare_models`) to automate the choice algorithms by comparing the performance of all available algorithms (Ali, 2022).

- **rminer** is a library for the R programming language, focused on facilitating the usage of ML algorithms (Cortez, 2010). Since version 1.4.4, rminer implements AutoML functions. In this PhD, we used the `automl3` template[10], which runs several ML algorithms and one Stacked Ensemble.

[9] https://supervised.mljar.com/features/modes/
[10] https://CRAN.R-project.org/package=rminer

- **TPOT** is a Python AutoML tool that uses Genetic Programming to automate several phases of the ML workflow, such as feature selection, feature engineering, algorithm selection, and hyper-parameter tuning. It uses the Python Scikit-Learn framework to produce ML pipelines (Le et al., 2020).

- **TransmogrifAI** is an end-to-end AutoML library that runs on top of Apache Spark. It was created to increase ML efficiency through automation and an **Application Programming Interface (API)** that ensures compile-time type safety, modularity, and reuse. It is written in Scala and focused on the automation of several phases of the ML workflow, such as algorithm selection, feature selection, and feature engineering (Salesforce, 2022).

2.2.3 One-Class Classification

OCC can be viewed as a subclass of unsupervised learning, where the ML model only learns using training examples from a single class (Moya & Hush, 1996; Zola et al., 2021). The aim of OCC algorithms is to learn a representation of the examples from that class that allows the identification of other classes during the inference phase (Perera et al., 2021). This type of learning is valuable in diverse real-world scenarios where labeled data is non-existent, infeasible, or difficult (e.g., requiring a costly and slow manual class assignment), such as fraud detection (Seliya et al., 2021), cybersecurity (Arregoces et al., 2022), Predictive Maintenance (Ferreira et al., 2022), or industrial quality assessment (Ribeiro et al., 2022). The following five subsections describe popular OCC algorithms that have been used in this PhD.

2.2.3.1 Autoencoders

Autoencoders are a type of ANN that is trained to reconstruct its input data. This is achieved by learning a compressed representation of the input data, which is then used to reconstruct the original data as closely as possible (Maleki et al., 2021). AEs are used for several applications, such as dimensionality reduction or removing noise from data. AEs can be applied to OCC scenarios, where the AE is trained with normal data and attempts to produce outputs similar to the inputs. For each input instance, there is an associated reconstruction error, where higher reconstruction errors represent a higher probability of being an anomaly (Ribeiro et al., 2022). Fig. 2 shows an example of an AE.

2.2.3.2 Isolation Forest

Isolation Forest was proposed in 2008 (Liu et al., 2008) and it works by isolating "anomalies" instead of identifying "normal" instances. In order to isolate the instances, IF recursively generates partitions on the training data by randomly selecting an attribute and then selecting a split value for that attribute. This strategy is based on two main assumptions regarding anomalies: they are a minority of the data and very different from the normal instances. This way, since anomalies are few and different, they are easier to

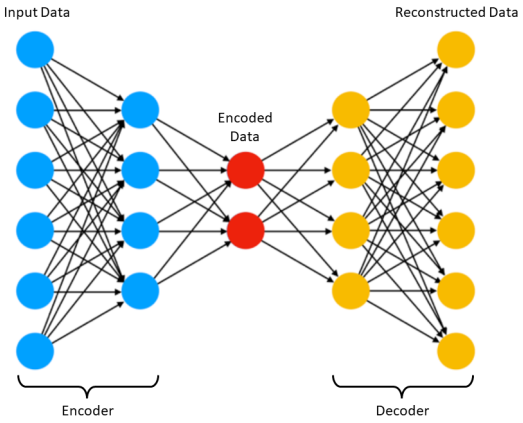


Figure 2: Example of an AE. The input data is encoded into a compressed representation and then it is decoded.

"isolate" compared to normal points. Fig. 3 exemplifies the IF partitioning on a dataset with two attributes. In the figure, x_0 is an anomaly (since it is easily isolated) and x_i is a normal point.

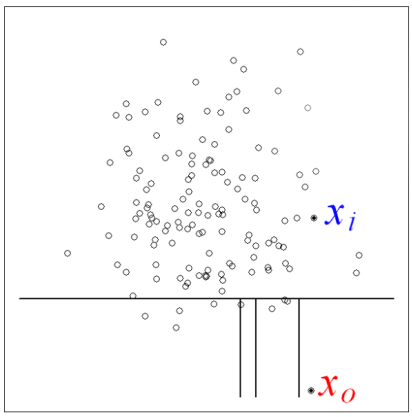


Figure 3: IF partitioning: x_0 is an anomaly (easily isolated) and x_i is a normal point. Adapted from Liu et al. (2008).

2.2.3.3 Local Outlier Factor

Local Outlier Factor is a density-based anomaly detection algorithm that is used to identify instances in a dataset that are significantly different from the majority of the instances. It works by calculating an anomaly score S_i for each instance i, which reflects the degree to which it is isolated from the rest of the examples in the dataset. LOF is particularly useful for detecting anomalies in high-dimensional datasets, as it is able to capture complex patterns in the data (Breunig et al., 2000). High LOF scores are considered to be outliers, as they are located in areas of the feature space that are less densely populated. Thus, in this work, we use the LOF S_i score as the anomaly degree measure.

2.2.3.4 One-Class Support Vector Machines

One-Class SVM is an extension of the SVM algorithm for unlabeled data (Schölkopf et al., 2001; S. Wang et al., 2018). OC-SVM learns a decision function from the training data (composed only of normal instances) and can classify new data as similar or different than the training data. Instead of using a hyperplane to separate two classes (such as the traditional SVM), OC-SVM uses the hyperspace to include all training instances.

2.2.3.5 Variational Autoencoders

Variational Autoencoders differ from traditional AEs in that they are trained to learn a distribution over the input data, rather than simply reconstructing the input data (Kingma & Welling, 2019). VAEs are composed of two parts: an encoder that maps the input data to a latent representation, and a decoder that maps the latent representation back to the original data space. The encoder and decoder are trained to optimize an objective function that encourages the generated data to be similar to the original data, while also encouraging the latent representation to be smooth and continuous. This allows VAEs to generate new data points that are similar to the original data, whereas traditional AEs are only able to reconstruct the input data. VAEs can be used for anomaly detection, since anomalies are expected to have different distributions when compared with normal training examples (Edun et al., 2022).

2.2.4 Neuroevolution

Neuroevolution (NE) is a field of AI related to the generation of Artificial Neural Networks using evolutionary algorithms (Risi & Togelius, 2017). NE addresses the automation of the design of ANNs (e.g., hyperparameters, structure, weights), often finding good solutions in complex and high-dimensional neural modeling spaces while using a reasonable amount of computational resources. NE has been successfully applied to a variety of tasks, including (Baymurzina et al., 2022; Cortez et al., 2020; Floreano et al., 2008): reinforcement learning, unsupervised learning, optimization, time series forecasting, supervised learning, and DL NAS.

A popular NE technique that is used in this PhD is Grammatical Evolution. GE is a biologically inspired evolutionary algorithm for generating computer programs. The algorithm was proposed by O'Neill and Ryan (2001) and has been widely used in both optimization and ML tasks. GE can handle complex optimization problems with a large number of objectives and constraints. It can also handle continuous and discrete optimization problems, as well as problems with mixed variables. Indeed, GE has been shown to be effective in finding high-quality solutions in a relatively short time, compared to other optimization methods (Nyathi & Pillay, 2018). In GE, a set of programs is represented as strings of characters, known as chromosomes. The chromosomes are encoded using a formal grammar, which defines the syntax and structure of the programs. The grammar is used to parse the chromosomes and generate the corresponding programs, which are then evaluated using a fitness function. The fitness function measures the

quality of the programs and is used to guide the evolution process toward better solutions.

A GE execution starts by creating an initial population of solutions (usually randomly), where each solution (usually named individual) corresponds to an array of integers (or genome) that is used to generate the program (or phenotype). In the evolutionary process of GE, each generation consists of two main phases: evolution and evaluation. During the evolution phase, new solutions are generated using operations such as crossovers and mutations. Crossover involves selecting pairs of individuals as parents and swapping their genetic material to produce new individuals, known as children. Mutation, which is applied to the children individuals after crossover, consists of randomly altering their genome to maintain genetic diversity. In the evaluation phase, the population of individuals is evaluated using the fitness function. Fig. 4 represents the steps for the GE optimization process.

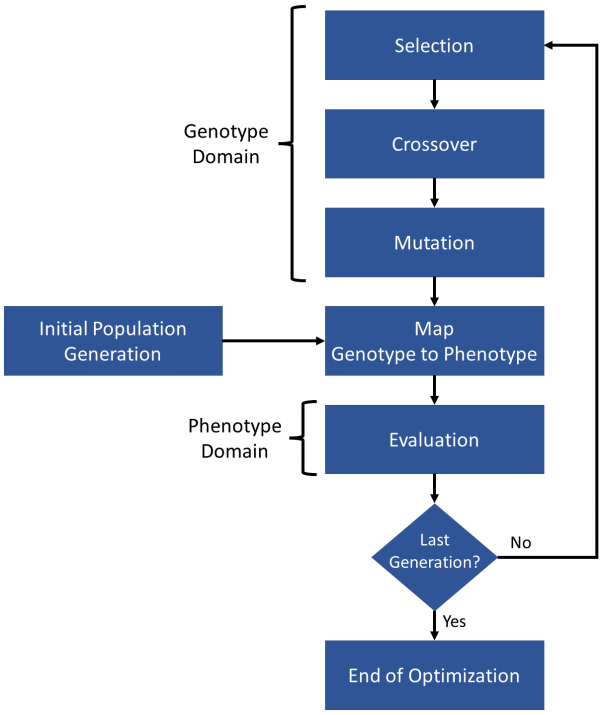


Figure 4: Steps for the GE optimization. Adapted from Anjum and Ryan (2021).

GE uses a mapping process to generate programs from a genome encoded using a formal grammar, typically in **Backus–Naur Form (BNF)** notation. This notation consists of terminals (items that can appear in language, such as the symbols + or −) and non-terminals (variables that include one or more terminals). An example of a BNF grammar is shown in Fig. 5.

$$< string >::=< letter > | < letter >< string >$$

$$< letter >::=< consonant > | < vowel >$$

$$< vowel >::= a|e|o|i|u$$

$$< consonant >::= b|c|d|f|g|h|j|k|l|m|n|p|q|r|s|t|v|w|x|y|z$$

Figure 5: Example of a BNF grammar to generate strings.

2.2.5 Multi-objective Optimization

Multi-objective optimization problems occur when two or more objectives are being optimized simultaneously. This is a common pattern in real-world domains, for example when a company intends to increase sales while reducing costs or maximizing the target audience while minimizing the budget. In these scenarios, typically the different objectives may conflict, where a gain in one objective might represent a loss in another one (Cortez, 2021).

One of the most popular multi-objective optimization algorithms is Non-dominated Sorting Genetic Algorithm II, which is used in this PhD. The NSGA-II algorithm was proposed in 2002 (Deb et al., 2002) and is based on the concept of non-dominance, which means that a solution is considered superior to another solution if it is not worse than the other solution in any objective and strictly better in at least one objective. The goal of NSGA-II is to find a set of non-dominated solutions, known as the Pareto front, which represents the trade-off between the different objectives. One of the main features of NSGA-II is its ability to handle constraints. The algorithm handles constraints by assigning a penalty value to solutions that violate the constraints. The penalty value is then used as an additional objective, which is minimized during the optimization process. NSGA-II also includes a crowding distance measure, which is used to preserve diversity among the solutions and avoid premature convergence. The algorithm has been widely used in various fields, including engineering, economics, and biology, and has shown promising results in a variety of multi-objective optimization problems (Coello et al., 2007). Fig. 6 shows an example of a multi-objective problem with two minimizing objectives. The dark line shows the Pareto front that includes the non-dominated solutions.

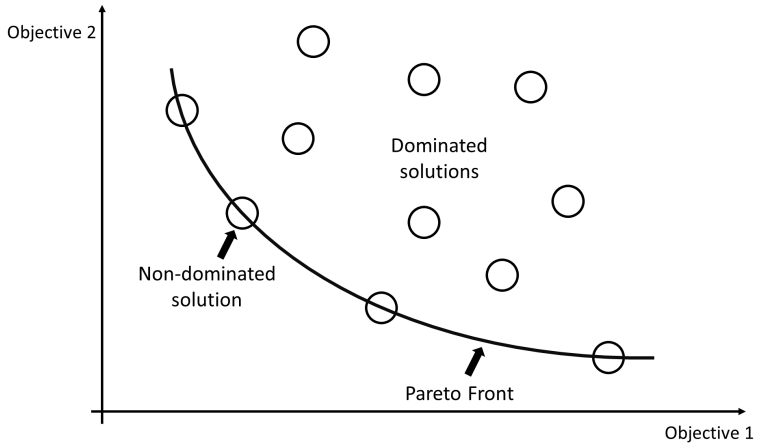


Figure 6: Example of a Pareto front in a multi-objective optimization problem with two minimizing objectives.

2.2.6 Evaluation Metrics

When developing ML applications, one of the most important steps is the assessment of the ML models quality. Within the task of supervised learning this step is typically achieved by comparing the predictions

of the ML model with the ground truth, using some kind of validation setup (e.g., holdout split, k-fold cross validation) (Dalianis, 2018). To compare the predictions of the ML models and the ground truth, quantitative methods based on mathematical functions are commonly applied. These functions (also named evaluation metrics in the context of ML applications) can vary depending on the ML task (e.g., supervised or unsupervised learning) and also on the type of model (e.g., classification, regression, time series).

In this PhD, most of the used evaluation metrics are based on popular supervised learning measures, which are further described in this section. For regression tasks, one of the most popular measures is **Mean Absolute Error (MAE)**. It measures the average absolute difference between the predicted and actual values. A lower Mean Absolute Error (MAE) value indicates better model performance, as it represents the average magnitude of the errors (Hastie et al., 2009). The MAE is a common choice for regression models as it is easy to interpret (e.g., a MAE of 5 means that the average error of the model is 5 units). MAE is calculated using the following formula:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

where y_i denotes the target values, x_i the predicted ones and n the number of predictions performed.

Root Mean Squared Error (RMSE) is another widely used metric in ML for evaluating the performance of regression models, as it measures the square root of the average of squared differences between predicted and actual values. RMSE penalizes larger errors more heavily than smaller errors and it is commonly used when larger errors are more significant than smaller errors (Karunasingha, 2022). A lower value of RMSE indicates better model performance. It is more difficult to interpret than MAE since the errors are squared errors instead of absolute errors. However, it represents the average magnitude of the errors in the same units as the predicted and actual values. It is calculated as follows:

$$RMSE = \frac{\sum_{i=1}^{n} (y_i - x_i)^2}{n}$$

The **Normalized Mean Absolute Error (NMAE)** is a scale independent version of MAE that shows the average error as a percentage of the response range. It is calculated using the following formula:

$$NMAE = MAE/(\max(y) - \min(y))$$

For binary classification tasks, one of the most used metrics is the **Area Under the Curve (AUC)** analysis of the **Receiver Operating Characteristic (ROC)** curve. The ROC curve plots the False Positive Rate versus the True Positive Rate for all possible threshold values ($Th \in [0, 1]$ for predicted binary class probability (e.g., normalized anomaly score when assuming that a positive class is the anomaly). The predicted probability $(x_i, i \in \{1, ..., n\})$ is considered a positive class if $x_i > Th$. The overall discrimination capability of the classifier is computed as (Fawcett, 2006):

$$AUC = \int_0^1 ROC\, dTh$$

AUC is a popular binary classification measure of performance, providing two main advantages (Coelho et al., 2022). Firstly, quality values are not influenced by the presence of unbalanced data, which occurs in OCC tasks. Secondly, the AUC values can be easily interpreted as follows: 50% – performance of a random classifier; 60% - reasonable; 70% - good; 80% - very good; 90% - excellent; and 100% - perfect.

Other metrics widely used for binary classification are Accuracy, Precision, and Recall. The three metrics are based on the confusion matrix, a table that compares the actual values from each actual and predicted class (Witten et al., 2016). Accuracy measures the proportion of correctly classified instances among all instances, indicating the overall performance of the model. Precision measures the proportion of true positive predictions among all positive predictions, indicating the model ability to avoid false positives. Recall, on the other hand, measures the proportion of true positive predictions among all actual positive instances, indicating the model ability to identify all positive instances. All of these metrics are represented by a number between 0 and 1, where 1 represents a perfect classifier. These metrics can be calculate using the following formulas:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

where TP, FP, TN, and FN denote the number of True Positives, False Positives, True Positives, and False Negatives.

The F1-score, also known as F-measure, is a harmonic mean of Precision and Recall, balancing both metrics and providing a single value that summarizes the overall performance of the model. F1-score can be used for both binary and multi-class classification and is particularly useful when the dataset is imbalanced, as it balances the contributions of Precision and Recall and penalizes extreme differences between both. A higher F1-score indicates better overall model performance, with the maximum score being 1, representing perfect precision and recall. F1-score is represented by the following formula:

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

When multi-class classification tasks are addressed, there is possible to compute an individual F1-score for each class. To aggregate all these scores, a popular adopted measure is the Macro F1-score, which is computed as the average of all individual F1-scores.

2.3 State-of-the-art Works

This section provides a state-of-the-art analysis of recent and relevant studies related to the application of AutoML techniques to different ML tasks and business domains. Specifically, distributed AutoML applied to risk management (Section 2.3.1), comparison of AutoML tools (Section 2.3.2), supervised and OCC AutoML applied to Predictive Maintenance (Section 2.3.3), and automated and multi-objective OCC (Section 2.3.4).

2.3.1 Distributed AutoML Applied to Risk Management

In a Big Data context, it is critical to create and use scalable ML algorithms to face the common constraints of memory and time (Peteiro-Barral & Guijarro-Berdiñas, 2013). To face that concern, classical distributed ML distributes the work among different processors, each performing part of the algorithm. Another current ML problem concerns the choice of ML algorithms and hyperparameters for a given task. For ML experts, this selection of algorithms and hyperparameters may use domain knowledge or heuristics, but it is not an easy task for non-ML experts. AutoML was developed to combat this relevant issue (He et al., 2021). The definition of AutoML can be described as the search for the best algorithm and hyperparameters for a given dataset with minimum human input.

In recent years, a large number of AutoML tools was developed, such as Auto-Keras (Jin et al., 2019), Auto-Sklearn (Feurer et al., 2015), Auto-Weka (Kotthoff et al., 2017), AutoGluon (AutoGluon, 2021), H2O AutoML (H2O.ai, 2021), Rminer (Cortez, 2020), TPOT (Olson et al., 2016), and TransmogrifAI (Salesforce, 2022). Within our knowledge, few studies directly compare AutoML tools. Most studies compare one specific AutoML framework with state-of-the-art ML algorithms (Feurer et al., 2015), do not present experimental tests (Elshawi et al., 2019; Yao et al., 2018), or are related to ML automation challenges (Guyon et al., 2015; Guyon et al., 2016; Guyon et al., 2019).

Recently, some studies focused on experimental comparisons of AutoML tools. In 2019, Gijsbers et al. (2019) and Truong et al. (2019) compare a set of AutoML tools using different datasets and ML tasks. In 2020, a benchmark was conducted using publicly available datasets from OpenML (Vanschoren et al., 2013), comparing different types of AutoML tools, which were grouped by their capabilities (Zöller & Huber, 2021). None of the mentioned comparison studies considered the distributed ML capability for the AutoML tools. Furthermore, none of the studies used datasets from the domain of telecommunications risk management, such as churn prediction or fraud detection.

2.3.2 Comparison of AutoML Tools

The state-of-the-art works that compare AutoML tools can be grouped into three major categories. The first category includes publications that introduce a novel AutoML tool and then compared it with existing ones. The second category is related to the comparison of distinct tools, without proposing a new AutoML

framework. Finally, the third category (less approached) focuses on the characteristics of the technologies rather than their predictive performances.

Table 1 summarizes the related works using the following columns: **Ref.** – the study reference; **Cat.** – the AutoML study category; **Dat.** – the number of analyzed datasets; **Tools** – the number of compared AutoML tools; **GML** – if General ML algorithms (not DL) were tested, such as Naïve Bayes, SVM, or XGB; **DL** – if DL was included in the comparison; **Ext.** – the external validation method used (if any); **C.** – if computational effort was measured; and **Description** – brief explanation of the comparison approach. The majority of the related works (14 studies) are from the year 2020, which confirms that AutoML tool comparison is a hot research topic. Some studies explore a large number of datasets (Das et al., 2020; Truong et al., 2019). Our comparison described in Section 3.3 adopts 12 datasets, which is below the two mentioned works but is still higher than used in eleven other studies, such as Dhir et al. (2020) and Liang et al. (2019). More importantly, we consider eight AutoML technologies, which is a number only outperformed by Waring et al. (2020), which tested only one dataset; and Y. Chen et al. (2020), which did not use any datasets. In particular, we benchmark the following recent tools: Auto-PyTorch - only studied by Zimmer et al. (2021) and compared by Y. Chen et al. (2020); rminer – not considered by the related works; and Transmogrifai – only compared by Ferreira et al. (2020b). Most works target GML and there are only four studies that address DL. Similar to our approach, there are seven studies that consider both GML and DL. Of the 21 surveyed works, only 12 employ an external validation. Most of these studies (8 of 12) use a single holdout train test split, which is less robust than a 10-fold cross-validation (adopted in four works). In addition, only 9 studies measure the computational effort. Furthermore, few studies contrast the AutoML results with the best human configured results. Kaggle competition results were included by Erickson et al. (2020), Liang et al. (2019), and Zöller and Huber (2021). Our work (Ferreira, Pilastri, Martins, et al., 2021) adopts open science (OpenML) best results, which was only performed by Hanussek et al. (2021).

2.3.3 Supervised and One-Class Classification AutoML Applied to Predictive Maintenance

Table 2 summarizes the related works that mention the usage of ML within the PdM domain in terms of the following columns: **Year** – the year in which the study was first published; **Ref.** – the study reference; **ML Algorithms** – which ML algorithms were used in the study (since some studies and tools do not disclose details to distinguish between shallow and deep structures, we adopt in this thesis the DL acronym to refer to both types of neural architectures); **FP** – if the study is applied to failure prediction (i.e., trying to identify when an equipment is going to fail); **Real Data** – if the study experiments analyze real-world data; and **ML Design** – the adopted ML modeling approach.

The related works are quite recent. In effect, Table 2 includes 16 studies published since 2017, including five works published in 2020 and three in 2021. Most works use real-world data and apply existing ML techniques to solve specific PdM tasks. Typically, classical supervised ML algorithms (e.g.,

Table 1: Summary of the related work: AutoML tool comparison.

Year	Ref.	Cat.	Dat.	Tools	GML	DL	Ext.	C.	Description
2019	(Drori et al., 2019)	1	8	5	✓				new AutoML tool
2019	(Liang et al., 2019)	1	2	4		✓	HO	✓	new AutoML tool
2019	(C. Wang & Wu, 2019)	1	53	4	✓	✓	10CV	✓	new AutoML tool
2019	(Gijsbers et al., 2019)	2	39	4	✓	✓			AutoML benchmark
2019	(Gimeno Saborit, 2019)	2	5	3	✓	✓			AutoML benchmark
2019	(Guyon et al., 2019)	2	n.d.	n.d.	✓	✓	HO	✓	AutoML competition
2019	(Truong et al., 2019)	2	300	6	✓	✓	HO		AutoML benchmark
2020	(Das et al., 2020)	1	175	2	✓		HO	✓	new AutoML tool
2020	(Dhir et al., 2020)	1	3	2	✓				new AutoML tool
2020	(Erickson et al., 2020)	1	50	6	✓		10CV	✓	new AutoML tool
2020	(Feurer et al., 2020)	1	39	2	✓		10CV	✓	new AutoML tool
2020	(Jing et al., 2020)	1	5	2		✓	HO	✓	new AutoML tool
2020	(Neto et al., 2020)	1	3	2		✓	HO		new AutoML tool
2020	(Yakovlev et al., 2020)	1	130	3	✓			✓	new AutoML tool
2020	(Zimmer et al., 2021)	1	8	4	✓	✓			new AutoML tool
2020	(Hanussek et al., 2021)	2	12	4	✓				AutoML benchmark
2020	(Ferreira et al., 2020b)	2	3	2	✓		HO	✓	AutoML benchmark (risk management)
2020	(Zöller & Huber, 2021)	2	137	5	✓		10CV		survey and benchmark
2020	(Waring et al., 2020)	3	1	12		✓	HO		literature review
2020	(Xanthopoulos et al., 2020)	3	0	7	✓				qualitative comparison
2020	(Y. Chen et al., 2020)	3	0	18	✓	✓			qualitative comparison
2021	(Ferreira, Pilastri, Martins, et al., 2021)	2	12	8	✓	✓	10CV	✓	AutoML benchmark

n.d. - not disclosed.
10CV - 10-fold Cross-Validation (CV).
HO - Hold-Out (HO) validation.

Linear Regression, DTs) are employed. Only five of the studies use DL, but none of these use this type of ML algorithm exclusively. Moreover, only four studies adopted unsupervised ML algorithms (Amruthnath & Gupta, 2018a; Cho et al., 2018; Makridis et al., 2020; Straus et al., 2018). This is a relevant issue for the PdM domain, since data labeling is often costly, requiring a manual effort.

Most studies aim to predict equipment failures, which is expected since it is one of the main challenges found in the PdM domain. Only two works did not try to predict when an equipment will fail: J. C. Cheng et al. (2020) tries to classify the condition of the equipment (e.g., excellent, good) and Arena et al. (2022) uses ML to suggest the type of maintenance needed for an equipment.

In terms of the ML modeling, the majority of the studies rely on a manual algorithm selection and hyper-parameter tuning (Expert-based). There are only two works apart from our work (described in Chapter 4) that use AutoML: Larocque-Villiers et al. (2021) is based on an existing AutoML framework (Auto-Sklearn), while Tornede et al. (2020) proposed an adaptation of the ML-Plan framework for PdM. In contrast with

Table 2: Summary of the related work: ML applied to PdM.

Year	Ref.	ML Algorithms	FP	Real Data	ML Design
2017	(Kanawaday & Sane, 2017)	ARIMA	✓	✓	Expert-based
2017	(Cline et al., 2017)	LR, LoR, DL, DT, RF, GBM	✓	✓	Expert-based
2018	(Butte et al., 2018)	GLM, RF, GBM, DL	✓	n.d.	Expert-based
2018	(Amruthnath & Gupta, 2018a)	K-means	✓	✓	Expert-based
2018	(Paolanti et al., 2018)	RF	✓	✓	Expert-based
2018	(Cho et al., 2018)	EM	✓	✓	Expert-based
2018	(Straus et al., 2018)	IF, LOF, OC-SVM	✓	✓	Expert-based
2020	(Tornede et al., 2020)	AdaBoost, RF	✓		AutoML
2020	(Gohel et al., 2020)	SVM, LogR	✓		Expert-based
2020	(J. C. Cheng et al., 2020)	DL, SVM		✓	Expert-based
2020	(Calabrese et al., 2020)	GBM, RF	✓	✓	Expert-based
2020	(Makridis et al., 2020)	DL, OC-SVM, XGB	✓	✓	Expert-based
2021	(Larocque-Villiers et al., 2021)	Classical ML, DL, Ensembles	✓	✓*	AutoML
2021	(Çakir et al., 2021)	SVM, LDA, RF, DT, KNN	✓	✓	Expert-based
2021	(Ayvaz & Alpay, 2021)	RF, XGB, GBM, MLP, SVM, Adaboost	✓	✓	Expert-based
2022	(Arena et al., 2022)	DT		✓	Expert-based
2022	(Ferreira et al., 2022)	Supervised and OCC	✓	✓	AutoML

ARIMA - Autoregressive Integrated Moving Average; DL - Deep Learning; DT - Decision Tree; EM - Expectation-Maximization; IF - Isolation Forest; Gradient Boosting Machine (GBM) - Gradient Boosting Machine; Generalized Linear Model (GLM) - General Linear Model; KNN - K-nearest Neighbors; LDA - Linear Discriminant Analysis; LOF - Local Outlier Factor; LR - Linear Regression; LogR - Logistic Regression; MLP - Multilayer Perceptron; n.d. - not disclosed; OC-SVM - One-Class SVM; RF - Random Forest; SVM - Support Vector Machines; XGB - XGBoost; ✓* - mixed data (both real and simulated).

our work (Ferreira et al., 2022), none of these studies compared more than one supervised AutoML tool. Moreover, the two works did not approach an unsupervised OCC, which is valuable for the PdM domain and that is here handled by using the proposed AutoOneClass method.

2.3.4 Automated and Multi-objective One-Class Classification

The related work can be grouped into three major categories. The first one (1) involves the application of **Evolutionary Computation (EC)** methods to perform an AutoML optimization, such as GE or **Genetic Algorithm (GA)**. The second category (2) includes research works that assume a multi-objective AutoML. The third and last category (3) focuses on studies that specifically target multi-objective OCC.

Table 3 summarizes the state-of-the-art works using the following columns: **Year** – the year in which the study was published; **Ref.** – the publication reference; **Cat.** – the study category (1, 2, or 3); **BL** – the number of distinct Base Learners (BL) or ML algorithms; **Dat.** – the number of analyzed datasets; **AutoML** – if the study performs an AutoML; **NAS** – if the study targets a NAS; **OCC** – if the study performs OCC (e.g., IF); **EC** – the type of EC algorithm (e.g., GE) used to search for the best ML design; and **MO** – if the study considers a Multi-objective optimization (more than one objective).

The related works are quite recent, with 19 studies published since 2016, including 4 works published

Table 3: Summary of the related work: automated and multi-objective OCC.

Year	Ref.	Cat.	BL	Dat.	AutoML	NAS	OCC	EC	MO
2016	(Balaprakash et al., 2016)	2	5	1	✓				✓
2017	(de Sá et al., 2017)	1	20	10	✓			GE	
2018	(de Lima Thomaz et al., 2018)	3	1	1			✓	GA	✓
2018	(Z. Chen et al., 2018)	3	5	4			✓		✓
2019	(Estevez-Velarde et al., 2019)	1	5	1	✓			GE	
2019	(Jr. & Barbosa, 2019)	1	11	10	✓			GA	
2019	(Cetto et al., 2019)	1	1	2	✓	✓		GE	✓
2019	(Gardner et al., 2019)	2	4	2	✓			GA	✓
2020	(Assunção et al., 2020)	1	22	10	✓			GE	
2020	(Moctezuma & Molinas, 2020)	3	1	1			✓	GA	✓
2021	(Estevez-Velarde et al., 2021)	1	3	1	✓			GE	
2021	(Marinescu et al., 2021)	1	8	50	✓			GE	
2021	(Mahjoubi et al., 2021)	2	1	3	✓				✓
2021	(Gardner et al., 2021)	2	1	2	✓				✓
2022	(Moyano & Ventura, 2022)	1	11	20	✓			GE	
2022	(Miranda et al., 2022)	1	1	1	✓	✓		GE	
2022	(Pfisterer, 2022)	2	1	1	✓				✓
2022	(Hirzel et al., 2022)	2	20	-	✓				✓
2022	(Ferreira et al., 2022)	1,2,3	3	1	✓	✓	✓	GE	*
2023	Work described in Chapter 5	1,2,3	5	8	✓	✓	✓	GE	✓

* – only partially studied.

in 2021 and 5 in 2022. In terms of study categories, from the analyzed first 18 works of Table 3, nine are related to the first category (EC to guide the optimization of the AutoML), six belong to the second category (multi-objective AutoML), and three works are from the third category (multi-objective OCC). From the first category, most works use EC to perform an hyperparameter tuning of the base learners or to build pipelines with data transformations (e.g., one-hot encoding) and algorithms (e.g., XGB). Apart from our work (described in Chapter 5), only two other studies apply a NAS optimization, thus approaching a pure NE. All the works from category 1 only target supervised learning algorithms (e.g., Linear Regression) and do not consider an OCC. From category 2, most works consider two optimization objectives, with the exception of Balaprakash et al. (2016) and Mahjoubi et al. (2021), which consider four or more objectives. Apart from the predictive performance, the works from this category consider other objectives within the AutoML optimization, such as fairness (Gardner et al., 2021; Hirzel et al., 2022; Pfisterer, 2022), domain-specific metrics (Balaprakash et al., 2016; Mahjoubi et al., 2021), or more than one predictive metric (Gardner et al., 2019). Regarding the third category, there are three works that use OCC in a multi-objective manner. From these works, two of them use GA to perform a multi-objective optimization. The considered objectives (apart from predictive performance) are related to the complexity of the ML model (Z. Chen et al., 2018), distances between solutions (de Lima Thomaz et al., 2018), or are domain-specific

(Moctezuma & Molinas, 2020). Our work assumes two objectives, predictive performance and the training computational effort.

In contrast with our research, the majority of the analyzed 18 related works approach supervised learning ML tasks. There are only two studies that employ an EC, namely a GA, to optimize OCC models (de Lima Thomaz et al., 2018; Moctezuma & Molinas, 2020). Thus, the work described in Chapter 5 is the only work that assumes a NE to evolve ANNs (performing a NAS). It also optimizes up to five base OCC classifiers, while the two most similar research works only optimize one ML algorithm: a Mahalanobis distance-based method (de Lima Thomaz et al., 2018); and OC-SVM (Moctezuma & Molinas, 2020). Moreover, we adopt a GE as the search engine, which is adopted by most of the related works but not by the two studies that evolve OCC models (de Lima Thomaz et al., 2018; Moctezuma & Molinas, 2020). As detailed in Section 5.4, GE provides two main advantages when evolving OCC classifiers: it assumes a variable-length solution representation (allowing to cope with different learners) and easy customization of the search space by means of an explicit grammar (allowing to adjust more or fewer base learners if needed). Finally, the two EC OCC optimization studies only adopt one dataset (e.g., from the medical domain), while our work explores eight datasets from different application domains.

Chapter 3

Supervised Automated Machine Learning

This chapter presents the main methods, experiments, and results obtained related to our initial supervised AutoML experiments and is divided into four sections. First, Section 3.1 describes the research context of the experiments. Then, Section 3.2 and Section 3.3 present the experiments and results. Finally, Section 3.4 presents the main conclusions from these experiments.

3.1 Research Context

This PhD thesis was partially conducted within the scope of the IRMDA R&D project. The project was undertaken by a leading Portuguese software and analytics company, spanning over a two-year period, also corresponding to the first two years of this PhD. The main objective of the project was to design a ML system to aid the company's telecommunications customers. The system was required to be automated and scalable as the company dealt with a wide range of customers, who possessed varying data sizes (both large and small), and lacked expertise in ML. We proposed a ML technological architecture, aiming to automate all typical tasks of a standard supervised ML application, with minimal human input. Additionally, the architecture was developed to function within a computational cluster, comprising multiple processing nodes.

However, given the project execution time, a novel AutoML framework that met the research objectives set for this PhD could not be proposed and evaluated within the scope of the project. As a result, we started the PhD work by exploring the existing AutoML frameworks, with a particular emphasis on automated and distributed ML, using three different datasets from the domain of Telecom risk management. The goal was to gain a deeper understanding of the application of AutoML for supervised learning tasks and to identify a robust evaluation method for the AutoML framework to be developed. The PhD work developed under the IRMDA project is detailed in this chapter and it is divided into two main research works. First, Section 3.2 describes a proposed technological architecture for telecommunications risk management that addresses the ML challenges of automation and scalability. Then, Section 3.3 presents a comparison study that considers eight recent open-source AutoML technologies (Auto-Keras, Auto-PyTorch, Auto-Sklearn,

AutoGluon, H2O AutoML, rminer, TPOT, and TransmogrifAI). The work associated with this chapter resulted in three publications: two conference papers (Ferreira et al., 2020a; Ferreira, Pilastri, Martins, et al., 2021) and an invitation to an extended publication in a book chapter (Ferreira et al., 2020b).

3.2 A Scalable and Automated Machine Learning Framework to Support Risk Management

3.2.1 Introduction

Nowadays, ML applications can make use of a great amount of data, complex algorithms, and machines with great processing power to produce effective predictions and forecasts (Darwiche, 2018). Currently, two of the most important features of real-world ML applications are distributed learning and AutoML. Distributed learning is particularly useful for ML applications in the context of Big Data or when there are hardware constraints. Distributed learning consists of using multiple machines or processors to process parts of the ML algorithm or parts of the data. The fact that it is possible to add new processing units enables ML applications to surpass time and memory restrictions (Peteiro-Barral & Guijarro-Berdiñas, 2013). AutoML intends to allow people that are not experts in ML to efficiently choose and apply ML algorithms. AutoML is particularly relevant since there is a growing number of non-specialists working with ML (Thornton et al., 2013). It is also important for real-world applications that require constant updates to ML models.

In this work, we propose a technological architecture that addresses these two ML challenges. The architecture was adapted to the area of telecommunications risk management, which is a domain that mostly uses supervised learning algorithms (e.g., for churn prediction). Moreover, the ML models are constantly updated by people that are not experts in ML and may involve Big Data. Thus, the proposed architecture delineates a set of steps to automate the typical workflow of a ML application that uses supervised learning. The architecture includes modules for task detection, data preprocessing, feature selection, model training, and deployment.

The focus of this work is the model training module of the architecture, which was designed to use a distributed AutoML tool. In order to select the ML tool for this module, we initially evaluated the characteristics of eight open-source AutoML tools (Auto-Keras, Auto-Sklearn, Auto-Weka, AutoGluon, H2O AutoML, Rminer, TPOT, and TransmogrifAI). We then performed a benchmark to compare the two tools that allowed a distributed execution (H2O AutoML and TransmogrifAI). The experiments used three real-world datasets from the domain of telecommunications. These datasets were related to churn (regression), event forecasting (time series), and fraud detection (binary classification).

The section is organized as follows. In Section 3.2.2, we detail the proposed ML architecture. Next, Section 3.2.3 describes the analyzed AutoML technologies and the datasets used during the experimental

tests. Then, Section 3.2.4 discusses the experimental results. Finally, Section 3.2.5 details the technological architecture.

3.2.2 Proposed Architecture

This work is part of IRMDA, a R&D project developed by a leading Portuguese company in the area of software and analytics. The purpose of the project is to develop a ML system to assist the company's telecommunications clients. Both scalability and automation are central requirements to the ML system since the company has many clients with diverse amounts of data (large or small) and that are typically non-ML experts.

The ML technological architecture that is proposed by this work identifies and automates all typical tasks of a common supervised ML application, with minimum human input (only the dataset and the target column). Also, since the architecture was developed to work within a cluster with several processing nodes, the users can handle any size of datasets just by managing the number of cluster nodes. The architecture is illustrated in Fig. 7.

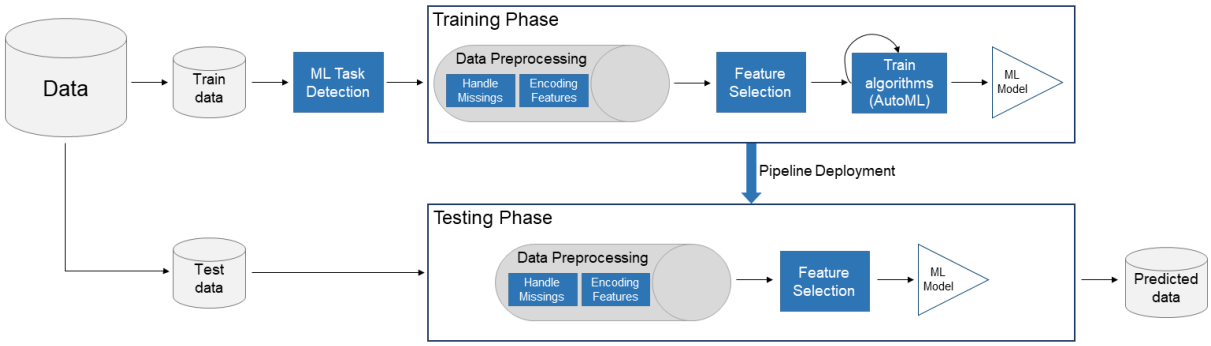


Figure 7: The proposed automated and scalable ML architecture.

3.2.2.1 Phases

The proposed architecture assumes two main phases (Fig. 7): a training phase and a testing phase.

The training phase includes the creation of a pipeline instance and the definition of its stages. The only human input needed by the user is the selection of the training dataset and the identification of the target column. Depending on the dataset columns, each module defines a set of stages for the pipeline. Each stage either transforms data or also creates a model based on the training data that will be used on the test phase to transform the data. When all stages are defined, the pipeline is fitted to the training data, creating a pipeline model. Finally, the pipeline model is exported to a file.

The execution of the testing pipeline assumes the same transformations that were applied to the training data. To execute the testing pipeline the user only needs to specify the test data and a pipeline model (and a forecasting horizon in the case of time series forecasting task). The last stage of the

testing pipeline is the application of the best model obtained during training, generating the predictions. Performance metrics are also computed and presented to the user.

3.2.2.2 Components

The proposed architecture includes five main components: task detection, data preprocessing, feature selection, model training (with the usage of AutoML), and pipeline deployment.

- **Machine Learning Task Detection:** set to detect the ML task of the pipeline (e.g., classification, regression, time series). This detection is made by analyzing the number of levels of the target column and the existence (or not) of a time column.
- **Data Preprocessing:** handles missing data, the encoding of categorical features, and the standardization of numerical features. The applied transformations depend on the data type of the columns, number of levels, and number of missing values.
- **Feature Selection:** deletes features from the dataset that may decrease the predictive performance of the ML models, using filtering methods. Filtering methods are based on individual correlations between each feature and the target, removing several features that present the lowest correlations (Blum & Langley, 1997).
- **Model Training:** automatically trains and tunes a set of ML models using a set of constraints (e.g., time limit, memory usage). The component also identifies the best model to be used on the test phase.
- **Pipeline Deployment:** manages the saving and loading of the pipelines to and from files. This module saves the pipeline that will be used on a test set, ensuring that the new data will pass through the same transformations as the training data. Also, the component stores the best model obtained during the training to make predictions, discarding all other ML models.

3.2.3 Materials and Methods

3.2.3.1 Experimental Evaluation

For the experimental evaluation, we first examined the characteristics of the open-source AutoML tools. Then, we used the tools that could be implemented in our architecture to perform a benchmark study. In order to be considered for the experimental evaluation, the tools have to implement distributed ML.

3.2.3.2 AutoML Tools

We first analyzed eight recent open-source AutoML tools, to verify their compliance with the project requirements: Auto-Keras, Auto-Sklearn, Auto-Weka, AutoGluon, H2O AutoML, rminer, TPOT, and TransmogrifAI.

Table 4 presents the characteristics of the analyzed AutoML related to interface language, associated platforms, current version and if it contains a Graphical User Interface and distributed ML mode. Additional information about the AutoML tools to is available in Section 2.2.2.

Table 4: Main characteristics of the analyzed AutoML tools.

Tool	Interface Language	Associated Platforms	Current Version	Graphical User Interface	Distributed Mode
Auto-Keras	Python	-	1.0.9	-	-
Auto-Sklearn	Python	-	0.10.0	-	-
Auto-Weka	Python R	WEKA	2.6.1	✓	-
AutoGluon	Python	-	0.0.14	-	-
H2O AutoML	Python R Scala	AWS Azure Google Cloud Apache Spark	3.30.1.3	✓	✓
rminer	R	-	1.4.6	-	-
TPOT	Python	-	0.11.5	-	-
TransmogrifAI	Scala	Apache Spark	0.7.0	-	✓

For the experimental study, we selected H2O AutoML and TransmogrifAI, as these were the only tools from Table 4 that meet the distributed ML requirement. Table Table 5 presents the ML algorithms implemented by both tools. The last two rows are related to the stacking ensembles implemented by H2O AutoML: all, which combines all trained algorithms; and best, which only combines the best algorithm per family.

3.2.3.3 Data

For the benchmark study, we used three real-world datasets from the domain of telecommunications, provided by the IRMDA project analytics company. The datasets are related to customer churn prediction (regression), event forecasting (univariate time series), and telecommunications fraud detection (binary classification).

The churn dataset contains 189 rows and 21 attributes. The attributes of each row characterize a client and the probability for canceling the company's analytics service (churn), as defined by the company. Table 6 describes each attribute of the churn dataset. The event forecasting dataset contains 1,418 rows that correspond to records about telecommunication events of a certain type (e.g., phone calls). The events occurred from February to April of 2019, aggregated on an hourly basis, ranged from 3,747 to 56,320. The only attributes are the timestamp and the number of events in that interval, as described in Table 7. The fraud detection dataset contains the identification of A (sender) and B (receiver), and the classification of the phone call ("fraud" or "normal"). The dataset contains more than 1 million examples,

Table 5: Algorithms implemented by H2O AutoML and TransmogrifAI.

Algorithm	**H2O AutoML**	**TransmogrifAI**
Decision Tree	-	✓
Deep Learning	✓	-
Extremely Randomized Forest	✓	-
Gradient-Boosted Tree	-	✓
Gradient Boosting Machine	✓	-
Generalized Linear Model	✓	-
Linear Regression	-	✓
Linear SVM	-	✓
Logistic Regression	-	✓
Naive Bayes	-	✓
Random Forest	✓	✓
XGBoost	✓ (only fully supported in Linux)	-
Stacking All (SA)	✓	-
Stacking Best (SB)	✓	-

Table 6: Description of the attributes of the churn dataset.

Attribute	**Description**
tenure	Time passed since the beginning of the contract
streaming_quality	Contractualized display resolution
ott_video	If OTT video is contractualized or not
contract	Duration of the contract
payment_method	Contractualized method of payment
product_name	Identification of the product
platform	Type of connectivity present in the contract
financial_status	If the payment is late or regularized
service_latency	Latency of the service
dropped_frames	Number of dropped frames
volume	Information about volume
duration	Information about duration
account_number	Account identification number
service_latency category	Category of the service latency attribute
dropped_frames category	Category of the dropped frames attribute
volume category	Category of the volume attribute
duration category	Category of the duration attribute
tenure category	Category of the tenure attribute
account_segment	Age segment of the client
equipment	Equipment used by the client
churn_probability	Probability of canceling the service ($\in [0, 1]$)

Table 7: Description of the attributes of the event forecasting.

Attribute	Description
Time	Timestamp (format: yyyy-mm-dd hh:mm)
datapoints	Number of events

which correspond to one day of phone calls from one of the company clients. The dataset attributes are described in Table 8.

Table 8: Description of the attributes of the fraud dataset.

Attribute	Description
A	Identification of the call sender
B	Identification of the call receiver
Result	Classification of the call (“fraud” or “normal”)

3.2.4 Results

3.2.4.1 Experimental Setup

To benchmark the AutoML tools, we executed several computational experiments using the three real-world datasets. The tools were compared under the same experimental setup, which was run on a machine with an i7-8700 Intel processor with 6 cores. A holdout split was used to divide the datasets into training (with 3/4 of the data) and test (1/4) sets. For churn and fraud dataset, the split was randomly selected, while for the event forecasting data a time order division was used (since the data is ordered in time). Using the training data, each AutoML tool optimizes a single performance measure, which was set as the MAE for the regression tasks (churn and event forecasting) and the AUC for the classification data (fraud detection). An internal 10-fold cross-validation was used by both AutoML tools in order to get a validation set where the selected performance measure is computed. For comparison purposes, for the test data we also computed the Normalized MAE (NMAE, in %, which is equal to the MAE divided by the target range) and RMSE values for regression tasks, and the Precision and Recall measures for the binary classification.

We tested all ML algorithms from Table 5, except for the DL, which was disabled from the H2O due to two main reasons. First, it required a huge computational effort, particularly for the large fraud detection dataset. Second, to achieve a more fair comparison, since TransmogrifAI does not include a DL algorithm. In order to allow the execution of all ML algorithms, no computational time execution limitation was used.

3.2.4.2 AutoML Results

For the Churn dataset, two scenarios were designed to test the performance of the AutoML tools. The first scenario (1) assumes all the attributes of the dataset as input features of the ML models. The

second scenario (2) uses an initial feature selection phase before training the models. The goal was to test the automatic feature selection option provided by TransmogrifAI. Under this scenario, and for H2O, we used the features that were considered more relevant by the best H2O performing ML model for the first scenario. The obtained results are presented in Table 9, where the computational execution time is presented in the *minutes:seconds* notation.

Table 9: Results for the churn data (best values in **bold**).

Tool	Scenario	Execution Time	Best Algorithm	Test Data MAE	Test Data NMAE	Test Data RMSE
H2O AutoML	1	**00:28**	SA	**0.112**	**11.2%**	**0.189**
	2	**00:26**	GLM	0.126	12.6%	0.186
TransmogrifAI	1	03:44	RF	0.210	21.0%	0.283
	2	03:36	GBT	**0.109**	**10.9%**	**0.136**

For the Event Forecasting dataset, we performed a transformation on the dataset by using a set of time lags to create the regression inputs since both H2O AutoML and TransmogrifAI do not have native univariate time series forecasting algorithms (e.g., ARIMA, Holt-Winters), These lags were created using the `CaseSeries` function of the rminer R package (Cortez, 2010), under three input, lagged scenarios: 1 – with time lags $t-1$, $t-24$ and $t-25$, where t is the current time (corresponding to the previous hour, day and hour before that day); 2 – with all time lags from the last 24 hours (from $t-1$ to $t-24$); and 3 – with the time lags $t-12$, $t-24$, $t-36$ and $t-48$. The results for each scenario are presented in Table 10.

Table 10: Results for the event forecasting data (best values in **bold**).

Tool	Scenario	Execution Time	Best Algorithm	Test data MAE	Test data NMAE	Test data RMSE
H2O AutoML	1	**02:32**	GBM	**2673**	**5.08%**	**4032**
	2	**02:53**	GBM	2138	4.07%	3535
	3	**01:50**	GBM	**2589**	**4.92%**	**4079**
TransmogrifAI	1	05:16	GBT	2725	5.18%	4332
	2	05:00	RF	**2101**	**4.00%**	**3441**
	3	03:47	RF	3468	6.60%	5212

Given that the Fraud dataset is highly unbalanced (with only around 0.01% of the calls being illegitimate), three scenarios were analyzed during the training phase to test the effect of balancing methods. The first scenario (1) used a simple oversampling which used "fraud" records and a random selection (with replacement) of "normal" cases. The second (2) and third (3) scenarios use the **Synthetic Minority Oversampling Technique (SMOTE)**, which is a more sophisticated balancing method that generates synthetic examples for the minority class, such that the training data gets more balanced (Chawla et al., 2002). The SMOTE was used to generate 100% of new fraud cases in the second scenario and 200% of

extra fraud examples in the third scenario. The test phase also considers three scenarios of unseen data, with different normal to fraud ratios: A – 50%/50%, thus balanced; B – 75%/24%; and C – 80%/20%. Table 11 shows the obtained results.

Table 11: Results for the fraud detection data (best values in **bold**).

Tool	Train Scenario	Execution Time	Best Algorithm	Test Scenario	AUC	Precision	Recall
H2O AutoML	1	05:20	GBM	A	0.97	**0.99**	**0.98**
				B	**0.95**	**0.99**	**0.97**
				C	**0.95**	**0.98**	**0.99**
	2	07:18	GBM	A	**0.99**	1	**0.99**
				B	**0.97**	**0.99**	**0.98**
				C	**0.97**	**0.99**	**0.96**
	3	08:54	GBM	A	**0.99**	**1**	**0.99**
				B	**0.99**	0.99	**0.98**
				C	**0.98**	**0.99**	**0.98**
TransmogrifAI	1	**01:19**	RF	A	**0.98**	**0.99**	0.96
				B	0.93	0.98	**0.97**
				C	0.92	**0.98**	0.93
	2	**01:45**	RF	A	0.98	**1**	**0.99**
				B	0.96	**0.99**	**0.98**
				C	0.96	0.97	0.94
	3	**02:13**	GBT	A	**0.99**	**1**	**0.99**
				B	0.98	**1**	**0.98**
				C	0.97	**0.99**	0.96

3.2.4.3 Discussion

In terms of computational processing time, the results show that in general a small effort is needed by the AutoML tools. The highest execution time is related with the fraud detection data for the H2O tool and it corresponds to just around 9 minutes.

This small computational effort is explained by several factors, including: usage of a distributed ML and multi-core machine; usage of benchmark telecommunications datasets that are either have a small number of examples (churn, event forecasting) or inputs (fraud detection); and the disabling of the DL algorithm in the H2O tool. Nevertheless, the execution time results confirm that the AutoML can perform an automatic ML selection in a reasonable time. And if larger datasets were analyzed, the computational effort could be reduced by adding more processing elements to the computational cluster. As for the tool comparison, H2O AutoML required less time to process the regression tasks (churn and event forecasting), while TransmogrifAI was faster for the classification task.

In terms of the predictive performance, H2O AutoML obtained better results for three regression comparisons (for both MAE and RMSE: scenario 1 for churn and scenarios 1 and 3 for event forecasting),

and seven classification comparisons (when using the AUC measure). TransmogrifAI obtained the best results in two regression scenarios and two classification ones. Overall, the AutoML predictive results are of high quality and the tools do not present substantial predictive differences. For example, all H2O AUC test results are equal to or higher than 95%, which corresponds to an excellent discrimination level. And the largest AUC classification difference when compared with TransmogrifAI is just 3 percentage points. Similarly, the best churn prediction models present a NMAE value that corresponds to an interesting value of around 10%. The tool NMAE differences are small for scenario 2 (1.7 percentage points) but larger for scenario 1 (9.8 percentage points). As for the event forecasting, the predictions are of high quality, with NMAE values ranging from 4.0% to 6.6%. The tool NMAE differences are very small for scenarios 1 and 2, with percentage point differences of 0.10 and 0.07, while the difference is larger for scenario 3 (1.68 percentage points).

The predictive results confirm the potential of the distributed AutoML technologies, which are capable of achieving high-quality predictive results in a reasonable amount of time and with a minimum human intervention. The obtained results were shown to the risk management software and analytics company, which opted to select the H2O AutoML tool for several reasons. First, it provided better predictive results for the majority of the tested scenarios. In particular, when the AutoML tools presented the largest metric differences, the best results were achieved by H2O. Second, the company classified the tool as "more mature" software, since as shown in Table 4, it is available in different programming languages and it can be integrated with more platforms (other than Spark). Also, the H2O provided an easy to use Graphical User Interface.

3.2.5 Tecnological Architecture

After the comparative ML experiments, the analytics company selected the H2O AutoML tool for the model training component. The remaining technological modules were then designed in cooperation with the company. Since one of the prerequisites of the architecture is that it is distributed, we tried to identify technologies with distributed capabilities. Given that H2O can be integrated with Apache Spark (using the Sparkling Water module) and that Spark provides functions for data processing, we relied on Spark's API functions to implement the remaining components of the architecture. The updated architecture, with references to the technologies used, is illustrated in Fig. 8.

3.2.5.1 Components

This subsection describes the current implementation of each module of the architecture. The updated technological architecture changed some of the modules initially described in Section 3.2.2. These changes were related to feedback received from the analytics company or due to technological restrictions.

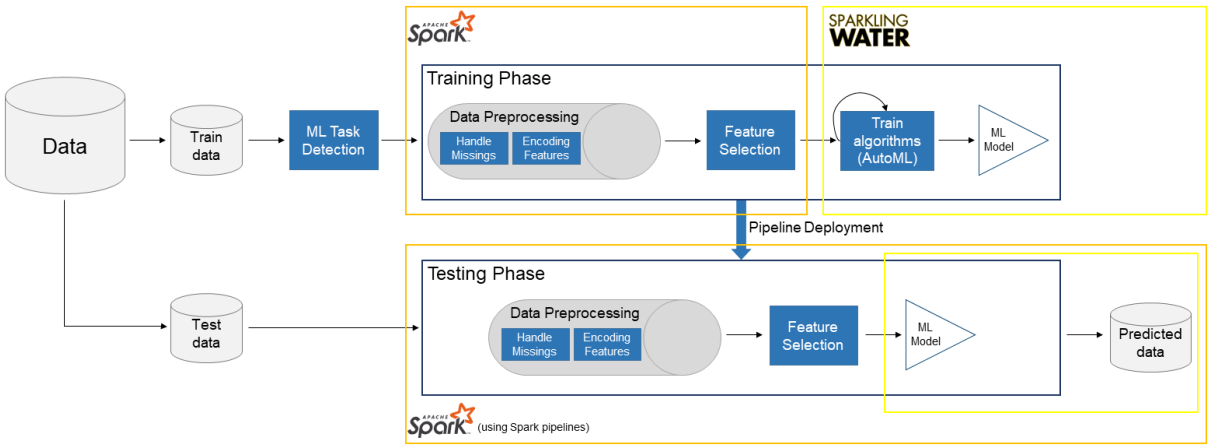


Figure 8: The technological automated and scalable ML architecture.

- **Machine Learning Task Detection:** currently set to detect if the ML pipeline should be considered a binary classification, multi-class classification, pure regression, or a univariate time series task since these are the typical telecommunications risk management ML tasks used by the company. The detection of the ML task can be overridden by the user. This is due to the fact that it could be useful to consider an ML task different than the one suggested by the module. For example, the end-user might want to consider a regression task, although the target column of the dataset only has a few number of levels, which could be automatically considered a multi-class classification. If the user specifies an ML task before running the pipeline, this component is skipped. The type of supervised tasks handled will be expanded according to feedback provided by the software company clients and the AutoML tools capabilities. Interesting future possibilities of tasks to be addressed are multivariate time series, ordinal classification, or multi-target regression.

- **Data Preprocessing:** currently, the preprocessing transformations (e.g., dealing with missing data, the encoding of categorical features, standardization of numerical features) are done using Apache Spark's functions for extracting, transforming and selecting features (Apache Spark, 2020a). To deal with missing data in numerical columns we use the `Imputer` function from Spark. This function replaces the unknown values of a column with its mean value. For categorical columns, we replace the unknown fields with a predefined tag (e.g., "Unknown"). The encoding of categorical features is done by default using Spark's one-hot encoding function. If the categorical column has a high cardinality (a vast number of levels), instead of the one-hot encoding we apply the `String Indexer` function. This function replaces the values of the column by numerical indices. The standardization of numerical features uses the `Standard Scaler` function from Spark. This function normalizes the column to have mean zero and standard deviation one.

- **Feature Selection:** currently, this module uses the Chi-Squared feature selection function from Apache Spark. This method decides what features to keep based on Chi-Squared statistical test.

Depending on the dataset and the ML task, we filter a fixed number of features or a percentage of features with the most correlation. Additionally, we added the possibility for the user (usually a domain expert) to influence this step. Thus, the user can specify beforehand the features that will be used as inputs by the model training module. Such features cannot be removed by the feature selection step, although other features can be added to the ones that the user selected. If no features were chosen by the user, this component works without restrictions. Also by request of the company, we created an auxiliary pipeline that performs a simple feature filtering, outputting a list of the most relevant features for a particular supervised learning dataset but without fitting an ML model (e.g., usage of the simple correlation statistic).

- **Model Training:** currently, this module uses one of two AutoML approaches we implemented, depending on the ML task that is being considered. For classification (binary or multi-class) and regression tasks, we use H2O AutoML to automatically find and tune the best model. Since none of the AutoML tools we analyzed support native univariate time series forecasting algorithms, we implemented our own AutoML for the time series task. In order to create the AutoML for time series, we used the algorithms implemented by the GitHub repository scalaTS[1] as a base. The repository includes a set of time series algorithms, such as autoregressive integrated moving average (ARIMA), autoregressive moving average (ARMA), autoregression (AR), and moving average (MA). Also, the package includes hyperparameter optimization capabilities, with the algorithms Auto ARIMA, Auto ARMA, Auto AR, and Auto MA, which pick the best parameters for each algorithm. The repository is built on top of Apache Spark using the distributed DataFrames objects, allowing distributed training and forecasting. In order to select the best algorithm for a time series task, we run each Auto algorithm with the training data and select the one that performs best on the validation data by using a rolling window validation (Oliveira et al., 2017).

- **Pipeline Deployment:** currently, the pipeline management module uses an Apache Spark API related to ML pipelines (Apache Spark, 2020b). To create a Spark ML pipeline it is necessary to detail a list of stages and then fit the pipeline to the training data. After fitting the pipeline to the training data, the Spark API allows the export of the pipeline to the disk. This process is applied during the training phase of the architecture. To apply a pipeline to test data it is necessary to load the model from a file. Then, using the `Transform` function, it is possible to apply the pipeline to previously unseen data. This process is applied during the test phase of the architecture, generating a set of predictions.

3.2.5.2 API

In order to facilitate the execution of the architecture, we also created a REST API to mediate the communication between the end-users and the pipelines. The development of the API resulted in two main

[1] https://github.com/liao-iu/scalaTS/

endpoints: one to run the train pipeline and the other to run the test pipeline.

Since the execution of each request consists of one Apache Spark job (using H2O's capabilities through the Sparkling Water module), the API works as an intermediary between the end-user and the execution of the code inside Spark. This way, the API server receives the client's requests and uses the parameters of the body of the request to initiate a Spark job inside the server (using the spark-submit command). After the execution of the application that was submitted to Spark, the server receives the output of the job (e.g., metrics of training, predictions). The server formats the response to the appropriate format (e.g., XML, JSON) and sends the response to the client interface. Fig. 9 depicts this process.

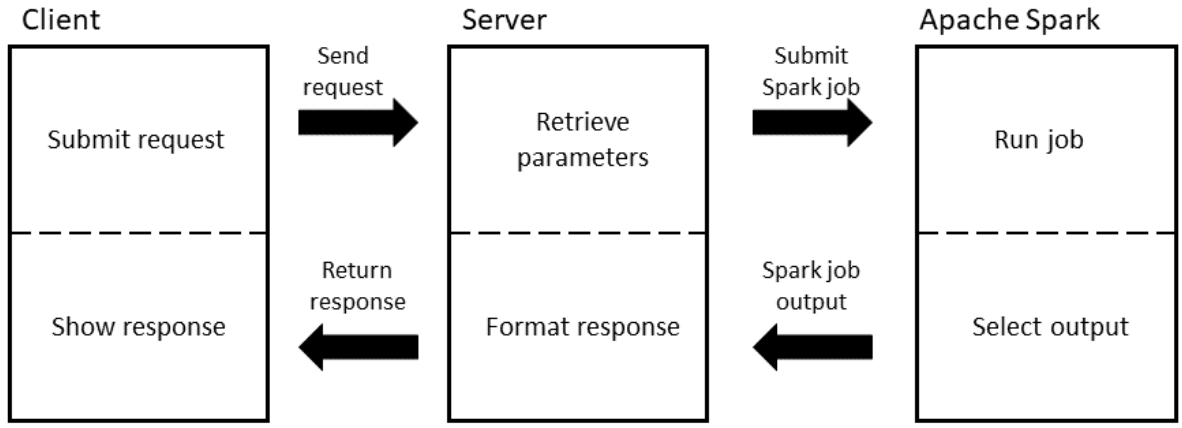


Figure 9: Adopted scheme for handing of requests and responses.

We highlight that the current version of the overall architecture, which received positive feedback from the Portuguese software company of the IRMDA project, is expected to be incrementally improved in future research. In particular, we intend to evolve and test the non AutoML components by using more real-world datasets and feedback from the analytics company clients.

3.3 A Comparison of AutoML Tools for Machine Learning, Deep Learning and XGBoost

3.3.1 Introduction

A ML application includes typically several steps: data preparation, feature engineering, algorithm selection and hyperparameter tuning. Most of these steps require trial and error approaches, especially for non-ML experts. More experienced practitioners often use heuristics to exploit the vast dimensional space of parameters (Lin et al., 2018). With the increasing number of non-specialists working with ML (Thornton et al., 2013), in the last years there has been an attempt to automate several components of the ML workflow, giving rise to the concepts of AutoML (Guyon et al., 2019) and AutoDL (Y. Chen et al., 2020).

This section focuses on the selection of the best supervised ML algorithm and its hyperparameter tuning. The comparison study considers eight recent open-source AutoML technologies: Auto-Keras, Auto-PyTorch, Auto-Sklearn, AutoGluon, H2O AutoML, rminer, TPOT, and TransmogrifAI. To evaluate these tools, we use twelve popular datasets retrieved from the OpenML platform, divided into regression, binary and multi-class classification tasks. In particular, we design three main scenarios for the benchmark study: General Machine Learning (GML) algorithm selection; DL selection (also known as AutoDL) and XGB hyperparameter tuning. Each tool is measured in terms of its predictive performance (using an external 10-fold cross-validation) and computational cost (measured in terms of time elapsed). Moreover, the best AutoML tools are further compared with the best public OpenML predictive results (which are assumed as the "gold standard").

The section is organized as follows. Section 3.3.2 describes the AutoML tools and datasets. Next, Section 3.3.3 details the benchmark design. Finally, Section 3.3.4 presents the obtained results.

3.3.2 Materials and Methods

3.3.2.1 AutoML Tools

This study compares eight recent open-source AutoML tools. Whenever possible, all tools were executed with their default values, in order to prevent any bias towards a particular tool, while also corresponding to a natural non-ML-expert choice. When available in the tool documentation, we show the number of hyperparameters ($\mathcal{H}$) tuned by the AutoML. Additional information about the AutoML tools to is available in Section 2.2.2.

- **Auto-Keras**: in this work, we adopt Auto-Keras version 1.0.7, which is used in the DL scenario (Section 3.3.3).

- **Auto-PyTorch**: similarly to Auto-Keras, we use Auto-PyTorch (version 0.0.2) only in the second DL scenario.

- **Auto-Sklearn**: we use Auto-SkLearn version 0.7.0 in the first GML scenario, since it does not implement an automated DL or XGB. All ML algorithms (when available for the task type) were tested: AdaBoost ($\mathcal{H} = 4$), Bernoulli ($\mathcal{H} = 2$) and Multinomial NB ($\mathcal{H} = 2$), Gaussian NB ($\mathcal{H} = 0$), DT ($\mathcal{H} = 4$), Extremely Randomized Trees (XRT) ($\mathcal{H} = 5$), GBM ($\mathcal{H} = 6$), k-Nearest Neighbors (k-NN) ($\mathcal{H} = 3$), Linear Discriminant Analysis (LDA) ($\mathcal{H} = 4$), Linear SVM (LSVM) ($\mathcal{H} = 4$), Kernel based SVM (KSVM) ($\mathcal{H} = 7$), Passive Aggressive ($\mathcal{H} = 3$), Quadratic Discriminant Analysis (QDA) ($\mathcal{H} = 2$), RF ($\mathcal{H} = 5$) and a Multiple Linear Regression (MR) classifier ($\mathcal{H} = 10$).

- **AutoGluon**: in this work, we consider the tabular prediction feature of AutoGluon version 0.0.13. The tabular prediction executes several ML algorithms and then returns a Stacked Ensemble that uses the distinct ML models in multiple layers. In the GML scenario (Section 3.3.3), ensemble

includes all non DL algorithms: GBM, CatBoost Boosted Trees, RF, Extra Trees (XT), k-NN and MR. For the DL scenario, the AutoGluon uses a DL dense architecture that uses heuristics to set the hidden layer sizes, employing also ReLU activation functions, dropout regularization and batch normalization layers (AutoGluon, 2021).

- **H2O AutoML**: in this work, we use H2O AutoML version 3.30.1.2 for the three comparison scenarios: GML, DL and XGB. In GML, all ML algorithms were explored (except DL): GLM ($\mathcal{H} = 1$), GBM ($\mathcal{H} = 8$), RF ($\mathcal{H} = 0$), XRT ($\mathcal{H} = 0$), XGB ($\mathcal{H} = 9$) and two Stacked Ensembles: Best – with only the best models per ML family; and All – with all trained algorithms. For the DL scenario, the H2O tool uses a fully connected multi-layer perceptron trained with a stochastic gradient descent back-propagation algorithm. The searched $\mathcal{H} = 7$ hyperparameters include the number of hidden layers and hidden units per layer, the learning rate, the number of training epochs, activation functions and input and hidden layer dropout values. Finally, for the XGB scenario, the tool tunes the same $\mathcal{H} = 9$ hyperparameters of GML.

- **rminer**: similarly to H2O, we test this tool in the GML and XGB scenarios. In GML, we used the ``automl3'' template, which searches the best model among: GLM ($\mathcal{H} = 2$), Gaussian kernel SVM ($\mathcal{H} = 2$ for classification and $\mathcal{H} = 3$ for regression), shallow multilayer perceptron (with one hidden layer, $\mathcal{H} = 1$), RF ($\mathcal{H} = 1$), XGB ($\mathcal{H} = 1$) and a Stacked Ensemble ($\mathcal{H} = 2$, similar to H2O Stacked Best).

- **TPOT**: the GML scenario tested all TPOT version 0.11.5 algorithms: DT, RF, XGB, (multinomial) Logistic Regression and k-NN. TPOT was not included in the third comparison scenario (XGB, Section 3.3.3) because the tool does not allow the selection of a single algorithm, such as XGB.

- **TransmogrifAI**: version 0.7.0 uses a grid search to perform the search of the best ML model. In the GML scenario, the tool was tested with all its ML algorithms: NB, DT, Gradient-Boosted Tree (GBT), RF, MR, LR and LSVM.

Table 12 summarizes the AutoML tools that were used. For each tool, we detail the base ML **Framework**, available Application Programming Interface (**API**) programming **Language**, compatible **Operating Systems**, and if it supports **DL** (Auto-Keras and Auto-PyTorch only address DL).

3.3.2.2 Data

The analyzed datasets (Table 13) were retrieved from OpenML (Vanschoren et al., 2013). The data selection criterion was defined as selecting the most downloaded datasets that did not include missing data and that are related with three supervised learning tasks: regression, binary and multi-class classification. The datasets reflect different numbers of instances (**Rows**), input variables (**Cols.**) and output target response values (**Classes/levels**, from 2 to 257; the last column details the **Target** domain values).

Table 12: Description of the compared AutoML tools.

AutoML Tool	**Framework**	**API Lang.**	**Operating Systems**	**DL**	**Scenario**		
					GML	**DL**	**XGB**
Auto-Keras	Keras	Python	MacOs Linux Windows	Yes (only)		✓	
Auto-PyTorch	PyTorch	Python	MacOs Linux Windows	Yes (only)		✓	
Auto-Sklearn	Scikit-Learn	Python	Linux	No	✓		
AutoGluon	PyTorch	Python	MacOS (P.) Linux	Yes	✓	✓	
H2O AutoML	H2O	Java Python R	MacOs Linux Windows (P.)	Yes	✓	✓	✓
rminer	rminer	R	MacOs Linux Windows	No	✓		✓
TPOT	Scikit-Learn	Python	MacOs Linux Windows	No	✓		
TransmogrifAI	Spark (MLlib)	Scala	MacOs Linux Windows	No	✓		

P. - partially supported (with less capabilities).

3.3.3 Benchmark Design

The comparison study assumes three main scenarios (Table 12). The first **GML** scenario executes all ML algorithms from the AutoML tools except DL, aiming to perform a more horizontal ML family agnostic search. DL was discarded since: some of the tools do not implement DL (Table 12); the training of DL models often requires a higher computational effort; and the second scenario is exclusively devoted to DL. The second **DL** scenario focuses on NAS, as implemented by the Auto-Keras, Auto-PyTorch, AutoGluon and H2O AutoML tools. Finally, the third scenario is more vertical, considering only the **XGB** algorithm. XGB was selected since it is a recently proposed non DL algorithm that includes a large number of hyper-parameters (e.g., H2O documentation mentions 40 hyperparameters of which only $\mathcal{H} = 9$ are tuned). In this scenario, we test H2O and rminer, since they are AutoML tools that allow to run the single XGB algorithm.

For every predictive experiment, the datasets were equally divided into tens folds, used for the external

Table 13: Description of the selected OpenML datasets.

Dataset	Task	Rows	Cols.	Classes/ Levels	Target Values
Cholesterol	regression	303	14	152	[126, 564]
Churn	binary	5000	21	2	{0,1}
Cloud	regression	108	7	94	[0, 6]
Cmc	multi-class	1473	10	10	{0,1,...,9}
Credit	binary	1000	21	2	{0,1}
Diabetes	binary	768	9	2	{0,1}
Dmft	multi-class	797	5	6	{0,1,...,5}
Liver Disorders	regression	345	6	16	[0, 20]
Mfeat	multi-class	2000	7	10	{0,1,...,9}
Plasma	regression	315	14	257	[179, 1727]
Qsar	binary	1055	42	2	{0,1}
Vehicle	multi-class	846	19	4	{0,1,...,3}

cross-validation. In order to create validation sets (to select the best ML algorithms and hyperparameters), we adopted an internal 5-fold validation. For instance, if the data contains 100 instances, then in the first external 10-fold iteration 90 examples are used by the tool for fitting purposes (model selection and training), with the remaining 10 instances being used for the external testing. The 90 fit examples are further divided into 5 folds. In the first internal fold, each ML is trained with 72 instances and 18 are used for validation purposes (allowing to select the best model). Since neither Auto-Keras nor Auto-PyTorch natively support cross-validation during the fitting phase, we used a simpler holdout train (75%) and test (25%) set split to select and fit the models.

In all three scenarios the same measures are used to evaluate the performance of the external 10-fold test set predictions. Popular prediction measures were selected: regression - MAE ($\in [0.0,\infty[$, where 0.0 denotes a perfect predictor); binary classification - Area Under the receiver operating characteristic Curve (AUC) ($\in$ [0.0,1.0], where 1.0 denotes the ideal classifier); multi-class classification - Macro F1-score ($\in$ [0.0,1.0], where 1.0 denotes the perfect model). Whenever allowed by the AutoML tool, we adopted the same measures for the internal AutoML validation set model comparison. The exceptions were with multi-class datasets and the Auto-Keras and Auto-PyTorch tools, which did not allow to use a Macro F1-score validation, thus the default loss function was adopted for these tools.

All experiments were executed using an Intel Xeon 1.70GHz server with 56 cores and 2TB of disk space. For each external fold, we also recorded the computational effort (in terms of time elapsed) for the AutoML fit (model selection and training). When the AutoML tool allowed to specify a time limit for training, the chosen time was one hour (3,600 s). Also, for the tools that implement an early stopping AutoML parameter, we fixed the value to three rounds. To aggregate the distinct external 10-fold results, we compute the average values. We also provide the 10-fold average t-distribution 95% confidence intervals, which can

be used to attest if the tool differences are statistically significant (e.g., by checking if two confidence intervals do not overlap). Nevertheless, given that there is a very large number of comparisons, to select the best tool for each task, we adopt a lexicographic approach (Freitas, 2004), which considers first the best average predictive performance (with a precision up to 1% or 0.01 points) and then the average computational effort (precision in s). To facilitate the lexicographic regression analysis, we compute the Normalized MAE (NMAE) score, which is a scale independent measure, where $NMAE = MAE/(\max(y) - \min(y))$ and y denotes the output target (Oliveira et al., 2017).

3.3.4 Results

Fig. 10 and Fig. 11 summarize the main scenario (GML) results. In total, there were 12 (dataset) $\times$ 6 (tools) $\times$ 10 (folds) = 720 AutoML executions. Fig. 10 presents the average computational effort (in s) for each external 10-fold iteration. Fig. 11 shows the average external test scores (grouped in terms of the binary, multi-class and regression tasks). To facilitate the visualization of the regression scores, in the right of Figure 11 we use the NMAE score in the y-axis.

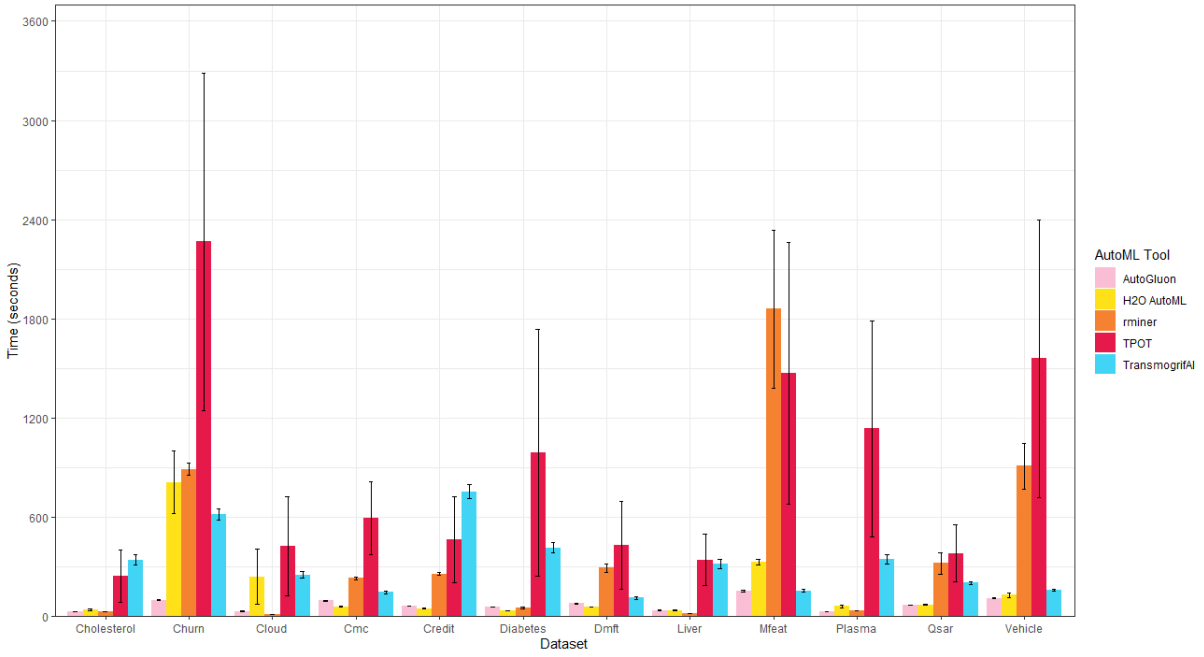


Figure 10: Execution time (y-axis) for the GML scenario (bars denote external 10-fold average values with 95% confidence intervals; the Auto-Sklearn values were omitted from the graph because they are always constant and equal to 3,600 s).

For GML, Auto-Sklearn always requires the maximum allowed computational effort (3,600 s), followed by TPOT (average of 858 s per external fold and dataset). The other tools are much faster: AutoGluon – lowest average value (70 s), best in 5 datasets; H2O – second average value (158 s), best in 5 datasets; TransmogrifAI – third best average (317 s); rminer – fourth best average (408 s), best in 2 datasets. Regarding the prediction performances, there is a high overall correlation between the validation and

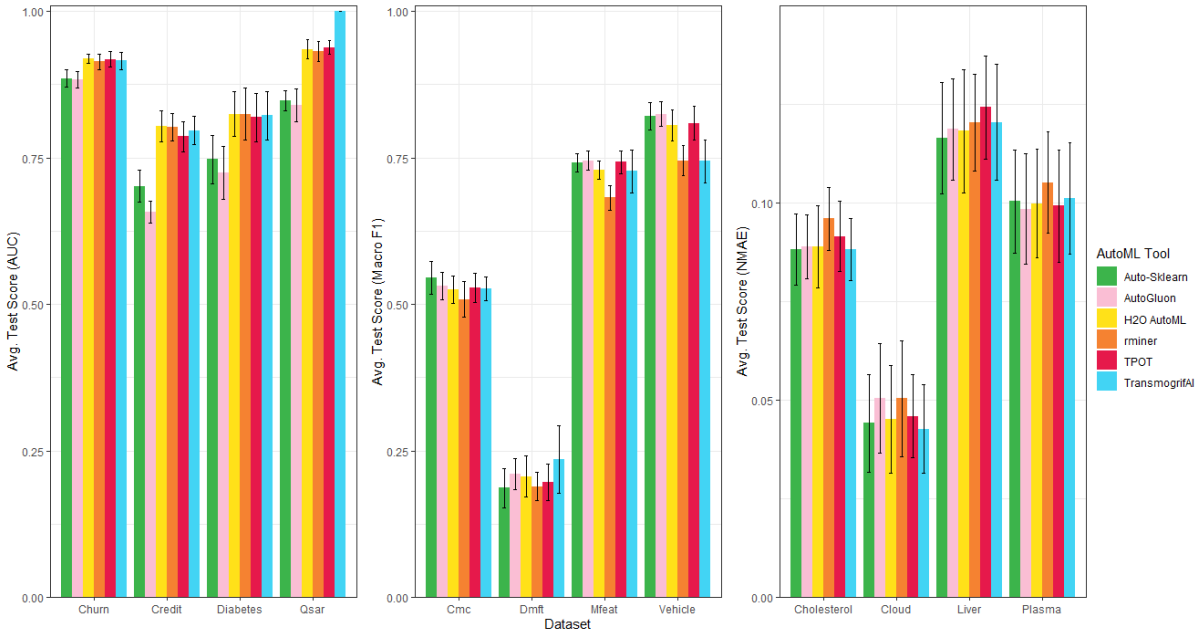


Figure 11: Predictive results for the GML scenario (bars denote external 10-fold average values with 95% confidence intervals).

test scores (not shown in Fig. 11, although the same effect is present in Table 14 and Table 15), when considering all tool execution values: 0.75 – binary; 0.90 – multi-class; and 0.92 – regression. For the binary classification, and when considering the test set results, the AutoML differences are smaller for churn (maximum difference of 3 *percentage points* - *pp*) and higher for the other datasets (10 *pp* for diabetes, 15 *pp* for credit and 16 *pp* for qsar). TransmogrifAI is the best tool in 3 of the datasets (churn, credit and qsar), also obtaining the best average AUC per dataset (88%). An almost identical average (87%) is achieved by H2O (best in churn and credit), rminer (best in diabetes) and TPOT (best in churn). AutoGluon and Auto-Sklearn produced the worst overall results (average AUCs per dataset of 78% and 80%). Turning to multi-class tasks, the AutoML differences (best tool test result minus the worst one) are smaller when compared with the binary task: 4 *pp* – Cmc; 5 *pp* – Dmft; 6 *pp* – Mfeat; and 8 *pp* – Vehicle. The best test dataset average is obtained by AutoGluon (Macro F1-Score 58%), followed by Auto-Sklearn, H2O and TPOT (Macro F1-Score of 57%), then TransmogrifAI (56%) and finally rminer (53%). In terms of datasets, the best results were: Cmc – Auto-Sklearn (54%); Dmft – TransmogrifAI (24%); Mfeat – AutoGluon, Auto-Sklearn and TPOT (74%); and vehicle - AutoGluon and Auto-Sklearn (82%). As for the regression tasks, the AutoML tool differences for each dataset are very small, corresponding to 1 *pp* in terms of NMAE for all three datasets. In effect, all tools obtain the same average NMAE per dataset (9%). Using the lexicographic selection (Section 3.3.3), the GML tool recommendation is: binary - TransmogrifAI; multi-class - AutoGluon; regression – rminer. The DL benchmark consisted of 12 (dataset) $\times$ 4 (tools) $\times$ 10 (folds) = 480 AutoML executions. Table 14 shows the average DL 10-fold results ($\pm$ the 95% confidence intervals) in terms of the external computational effort (**Time**), internal validation (**Val.**) and test scores

(**Test**). The Auto-Keras and Auto-PyTorch validation scores are omitted, since they are not disclosed by the tools. Regarding execution time, AutoGluon is much faster than the other tools, requiring an average fit time of just 24 s. The second fastest DL tool is Auto-Keras (average of 984 s), followed by H2O (3,458 s) and then Auto-PyTorch (3,600 s). As for the prediction performances, the average test values per dataset are: binary (AUC) - H2O (85%), Auto-PyTorch (77%); AutoGluon (72%) and Auto-Keras (69%); multi-class (Macro F1-score) - AutoGluon (57%), Auto-PyTorch (56%); H2O (50%) and Auto-Keras (43%); regression (NMAE) - H2O (10%); Auto-PyTorch and AutoGluon (11%); Auto-Keras (13%). While only four tools are compared, larger differences among the tools were obtained for the DL scenario when compared with GML: binary - ranging from 11 *pp* (Qsar) to 24 *pp* (credit); multi-class - from 4 *pp* to 30 *pp*; and regression – from 2 *pp* to 10 *pp*. The lexicographic recommendation for DL is: binary - H2O; multi-class - AutoGluon; regression – H2O. However, when considering both GML and DL scenarios, the lexicographic selection favors GML tools.

Table 15 presents the XGB scenario results (total of 240 AutoML executions). Both tools require a low computational effort, with rminer presenting the faster fit times (average of 5 s), while H2O requires around 16 times more computation (average of 80 s). Both tools also present similar predictive performances, with the tool differences ranging from 0 to 2 *pp*. The lexicographic selection for XGB favors: binary - rminer (average AUC of 86%); multi-class - H2O (average Macro F1-score of 55%); regression - rminer (average NMAE of 9%). When considering both DL and XGB scenarios, the lexicographic choice favors rminer XGB for the binary classification and regression tasks, while AutoGluon DL is the selected tool for multi-class. When analyzing all three scenarios, the overall lexicographic selection is: binary – TransmogrifAI GML; multi-class – AutoGluon GML; regression - rminer XGB.

Finally, we contrast the best main GML scenario results (which consider more ML algorithms and AutoML tools) with the best public OpenML results (Table 16). For each dataset, we show in rounded brackets the best GML AutoML tool and the type of OpenML modeling (the algorithm name or "Pipeline", with the latter denoting a ML workflow that includes a data preparation step). While the best OpenML result includes predictions for all external 10-fold instances, we do not know the exact validation and testing procedures that were adopted. Thus, rather than assuming a "correct" comparison, we use here the best OpenML results as a "gold standard", denoting a proxy to the best results that can be achieved when using a human expert ML modeling. The column **Attempts** from Table 16 denotes the number of human ML attempts, termed as a "run" in OpenML. The higher the number of attempts, the stronger is our assumption that the gold standard was reached. While all 12 datasets have high download numbers, the attempts distribution is highly unbalanced towards the classification tasks, particularly the binary ones (e.g., Credit has more than 419,000 attempts). The results from Table 16 confirm the quality of the AutoML. In effect, the tools obtained prediction scores that are close to the best OpenML results in seven datasets (e.g., the maximum difference is 2 and 5 *pp* for the binary and multi-task classification tasks). More importantly, the AutoML outperformed the best OpenML for three regression tasks and for two highly modeled binary datasets.

Table 14: Results for the DL scenario (best values in **bold**).

Dataset	Tool	Time	Measure	Val.	Test
Churn	Auto-Keras	2294±309	AUC	-	0.74±0.04
	Auto-PyTorch	3600±000		-	0.81±0.08
	AutoGluon	**0041**±003		0.90±0.00	0.80±0.02
	H2O AutoML	3600±000		**0.92**±0.00	**0.92**±0.02
Credit	Auto-Keras	1498±333		-	0.52±0.02
	Auto-PyTorch	3600±000		-	0.68±0.04
	AutoGluon	**0021**±002		0.77±0.01	0.57±0.03
	H2O AutoML	3600±000		**0.78**±0.00	**0.76**±0.04
Diabetes	Auto-Keras	0828±179		-	0.70±0.04
	Auto-PyTorch	3600±000		-	0.73±0.03
	AutoGluon	**0022**±001		0.79±0.01	0.65±0.05
	H2O AutoML	3600±000		**0.84**±0.01	**0.82**±0.03
Qsar	Auto-Keras	1154±336		-	0.82±0.03
	Auto-PyTorch	3600±000		-	0.87±0.02
	AutoGluon	**0026**±003		0.91±0.00	0.84±0.03
	H2O AutoML	3600±000		**0.92**±0.00	**0.92**±0.02
Cmc	Auto-Keras	0884±139	Macro F1	-	0.45±0.04
	Auto-PyTorch	3600±000		-	0.51±0.04
	AutoGluon	**0027**±003		**0.55**±0.00	**0.53**±0.02
	H2O AutoML	3600±000		**0.55**±0.00	0.48±0.06
Dmft	Auto-Keras	0478±045		-	0.16±0.04
	Auto-PyTorch	3600±000		-	**0.18**±0.04
	AutoGluon	**0018**±002		0.21±0.01	0.17±0.02
	H2O AutoML	3600±000		**0.23**±0.01	0.14±0.04
Mfeat	Auto-Keras	0713±095		-	0.44±0.04
	Auto-PyTorch	3600±000		-	0.73±0.02
	AutoGluon	**0032**±002		**0.76**±0.00	**0.74**±0.01
	H2O AutoML	3600±000		0.69±0.01	0.58±0.06
Vehicle	Auto-Keras	0708±053		-	0.65±0.05
	Auto-PyTorch	3600±000		-	**0.85**±0.02
	AutoGluon	**0038**±003		**0.85**±0.01	0.82±0.03
	H2O AutoML	3600±000		0.82±0.01	0.78±0.03
Cholesterol	Auto-Keras	0665±059	MAE	38.4±3.2	57.7±29.3
	Auto-PyTorch	3600±000		59.5±3.9	60.5±24.8
	AutoGluon	**0013**±002		39.0±0.5	**39.4**±4.0
	H2O AutoML	2509±279		**37.3**±0.4	39.9±4.1
Cloud	Auto-Keras	0549±572		**0.00**±0.00	0.88±0.36
	Auto-PyTorch	3600±000		0.20±0.04	**0.30**±0.10
	AutoGluon	**0013**±001		0.39±0.03	0.43±0.12
	H2O AutoML	3256±437		0.25±0.01	0.32±0.11
Liver Disorders	Auto-Keras	0879±797		2.67±0.29	2.60±0.37
	Auto-PyTorch	3600±000		2.54±0.14	**2.41**±0.35
	AutoGluon	**0018**±003		3.39±0.20	3.44±0.33
	H2O AutoML	3326±546		**2.23**±0.04	2.68±0.33
Plasma	Auto-Keras	1156±111		**127**±006	170±020
	Auto-PyTorch	3600±000		175±091	191±097
	AutoGluon	**0014**±002		156±004	155±022
	H2O AutoML	3600±000		151±003	**160**±017

Table 15: Results for the XGB scenario (best values in **bold**).

Dataset	Tool	Time	Measure	Val.	Test
Churn	H2O AutoML	356±182	AUC	**0.93**±0.00	**0.92**±0.01
	rminer	**006**±000		0.92±0.00	**0.92**±0.01
Credit	H2O AutoML	021±002		**0.79**±0.01	**0.79**±0.02
	rminer	**004**±000		0.77±0.01	**0.79**±0.02
Diabetes	H2O AutoML	013±002		**0.83**±0.01	**0.82**±0.05
	rminer	**003**±001		0.81±0.01	**0.82**±0.04
Qsar	H2O AutoML	037±004		**0.93**±0.00	**0.93**±0.01
	rminer	**004**±000		**0.93**±0.00	**0.93**±0.02
Cmc	H2O AutoML	035±002	Macro F1	**0.53**±0.01	**0.53**±0.02
	rminer	**006**±000		**0.53**±0.01	0.52±0.03
Dmft	H2O AutoML	034±002		**0.23**±0.01	**0.21**±0.04
	rminer	**009**±000		0.20±0.01	0.19±0.02
Mfeat	H2O AutoML	121±009		**0.72**±0.06	**0.71**±0.02
	rminer	**015**±001		**0.72**±0.01	**0.71**±0.02
Vehicle	H2O AutoML	088±012		**0.77**±0.00	**0.77**±0.04
	rminer	**007**±000		0.76±0.01	0.75±0.03
Cholesterol	H2O AutoML	025±005	MAE	**39.1**±0.5	**39.2**±4.2
	rminer	**002**±000		42.7±1.0	42.4±5.1
Cloud	H2O AutoML	167±120		**0.28**±0.02	**0.30**±0.08
	rminer	**002**±000		0.31±0.01	**0.30**±0.08
Liver Disorders	H2O AutoML	020±004		**2.29**±0.04	**2.35**±0.32
	rminer	**002**±000		2.49±0.05	2.45±0.30
Plasma	H2O AutoML	042±008		**154**±003	**154**±022
	rminer	**003**±001		171±004	170±021

3.4 Conclusions

Applying ML to real-world business problems can be time-consuming, resource-intensive, and challenging. AutoML facilitates this process by running systematic search processes with minimum input from the user. Another important ML aspect is scalability, which can be achieved by using a divide-and-conquer approach, in which smaller datasets are analyzed by different processing elements. However, automation and scalability are two of the main current challenges of ML. On one hand, there is a need for applications that are capable of distribution in order to add computation scalability to their processes. On the other hand, ML applications must be capable of automating their typical steps (e.g., feature selection, model training) to allow non-expert users to develop ML models and use them to make predictions.

Table 16: Comparison between best GML scenario and best OpenML results (best values in **bold**).

Dataset	Measure	Best		OpenML
		AutoML	OpenML	Attempts
Churn	AUC	0.919 (H2O AutoML)	**0.930** (Pipeline)	5,132
Credit		**0.803** (H2O AutoML)	0.800 (Ranger)	419,021
Diabetes		0.825 (rminer)	**0.842** (XGB)	132,164
Qsar		**1.000** (TransmogrifAI)	0.938 (XGB)	147,659
Cmc	Macro F1	0.545 (Auto-Sklearn)	**0.572** (Pipeline)	21,446
Dmft		0.236 (TransmogrifAI)	**0.262** (Pipeline)	19,445
Mfeat		0.745 (AutoGluon)	**0.772** (Pipeline)	22,136
Vehicle		0.820 (AutoGluon)	**0.870** (Pipeline)	23,532
Cholesterol	MAE	**38.59** (TransmogrifAI)	38.60 (Cforest)	160
Cloud		**0.256** (TransmogrifAI)	0.268 (Pipeline)	5
Liver Disorders		2.329 (Auto-Sklearn)	**2.309** (Pipeline)	77
Plasma		**152.3** (AutoGluon)	152.6 (Pipeline)	15

In this study, we address these challenges by demonstrating the initial supervised AutoML experiments, aiming to gain insights into the application of AutoML for supervised learning tasks, as well as to identify a robust and reliable evaluation method for the proposed AutoML framework. For the first experiment (Section 3.2), we propose a ML framework to automate the typical workflow of supervised ML applications without the need for human input. The framework includes the modules of task detection, data preprocessing, feature selection, model training, and pipeline deployment. The framework was developed within project IRMDA, a R&D project developed by a leading Portuguese software and analytics company that provides services for the domain of telecommunications risk management. The company clients work with datasets of variable sizes (large or small) and are mostly non-ML experts. Thus, the proposed framework uses distributed ML to add computational scalability to the process and AutoML to

automate the search for the best algorithm and hyperparameters.

In order to assess the most appropriate AutoML tools for this model training module, we initially conducted a benchmark experiment. First, we analyzed the features of eight open-source AutoML tools (Auto-Gluon, Auto-Keras, Auto-Sklearn, Auto-Weka, H2O AutoML, Rminer, TPOT, and TransmogrifAI). Then, we selected the tools that allowed a distributed execution for the experiments (H2O AutoML and TransmogrifAI). The benchmark study used three real-world datasets provided by the software company from the domain of telecommunications risk management. The proposed framework was positively evaluated by the analytics company, which selected H2O AutoML as the best tool for the model training module. After the selection of H2O AutoML for the model training module, we developed the technological architecture. We selected technologies with distributed capabilities for the remaining modules of the initially proposed framework. Most of the remaining modules were implemented using Apache Spark's API functions. Then, we describe the current implementation of each module of the architecture. Finally, we describe the REST API that was created to facilitate communication between the end-users (the company clients) and the implemented pipelines.

In the second study (Section 3.3), we benchmark eight recent open-source supervised learning AutoML tools: Auto-Keras, Auto-PyTorch, Auto-Sklearn, AutoGluon, H2O AutoML, rminer, TPOT and TransmogrifAI. A large set of computational experiments was held by considering an external 10-fold cross-validation, twelve datasets and three tool comparison scenarios. Each tool was benchmarked by measuring its computational effort and predictive scores. We retrieved popular datasets from the OpenML platform, which were equally grouped into regression, binary, and multi-class classification tasks. The three comparison scenarios were: General Machine Learning (GML) - with a large range of classical ML algorithms; Deep Learning (DL) - focusing on tools with DL Neural Architecture Search (NAS) capabilities; and XGBoost (XGB) - considering a single XGB algorithm hyperparameter tuning.

To select the best tools for each scenario, we adopted a lexicographic approach, which considers first, for each task, the best average predictive score and then the lowest computational effort. For GML, the lexicographic selection favors TransmogrifAI for binary classification, AutoGluon for multi-class classification and rminer for regression. For DL, the selection is H2O for the binary and regression tasks and AutoGluon for regression. As for the XGB scenario, rminer is the best overall option for binary and regression tasks, while H2O is recommended for multi-class tasks.

A global overall analysis, considering all three scenarios, favors the GML approach, which produced the best predictive scores. This result should be taken with some caution since GML explored more ML algorithms and AutoML tools. Nevertheless, the slightly lower AutoML DL predictive performances might be explained by two factors. Firstly, the analyzed datasets are relatively "small", with the largest dataset containing only 5,000 instances. And it is known that DL tends to produce better results (when compared with shallow methods) when modeling big data Ng, 2020. Secondly, the AutoML tools with DL capabilities are more recent and thus might be still immature when compared with GML tools. For instance, the tested Auto-PyTorch and AutoGluon versions are still in their zero dot something versions (e.g., 0.0.2). To further measure the quality of the GML AutoML modeling, we compared the best GML results with the best

predictions publicly available on the OpenML platform. The OpenML comparison confirmed that current GML AutoML tools provide competitive results, producing close predictions in seven datasets and even outperforming the human ML modeling in five datasets.

Chapter 4

Using Supervised and One-Class Automated Machine Learning for Predictive Maintenance

4.1 Research Context

This chapter summarizes the PhD work carried out under the CMMS R&D project, which explored the application of ML for Predictive Maintenance, following the contributions highlighted in Chapter 3. Similar to the IRMDA project, CMMS also had a relatively short execution time (approximately one year), corresponding to the third year of this PhD project. As part of the CMMS project, we conducted preliminary experiments to develop a novel AutoML framework by employing a GE to evolve OCC ML algorithms using single and multi-objective optimization. First, we applied state-of-the-art AutoML tools using a real-world PdM dataset. Then, we proposed AutoOneClass, an AutoML framework that focuses on three OCC algorithms: deep AE, IF, and OC-SVM. The GE was employed to optimize the search for the best OCC ML algorithm and its hyperparameters, assuming a single or multi-objective optimization search. The framework also allows for unsupervised or supervised validation setups. The unsupervised setup used unlabeled data during validation and anomaly scores to evaluate the ML models, while the supervised setup employed a labeled validation set to assess model performance. This R&D project contributed to advancing the state-of-the-art regarding the application of AutoML in the PdM industry and provided valuable practical knowledge about the usage of OCC algorithms within AutoML. Further details about the work conducted under the CMMS project are presented in this chapter, which resulted in two published papers: a Best Paper Award conference paper (Ferreira, Pilastri, Sousa, et al., 2021) and a Q1 journal paper (Ferreira et al., 2022).

4.2 Introduction

The Industry 4.0 phenomenon allowed companies to focus on analyzing historical data to obtain valuable insights. In particular, PdM is a crucial application area that emerged from this context, where the goal is to optimize the maintenance and repair process of equipments through the usage of ML algorithms (Silva et al., 2021). Indeed, some ML studies try to anticipate the failure of equipments (typically, manufacturing machines), aiming to reduce the costs of repairs (Carvalho et al., 2019; Cline et al., 2017; Kanawaday & Sane, 2017; Paolanti et al., 2018). Other approaches (Ayvaz & Alpay, 2021; Benedetti et al., 2018; Butte et al., 2018; Çınar et al., 2020) use ML algorithms to predict the behavior of the manufacturing process.

Despite all potential Industry 4.0 benefits, many organizations do not currently apply ML to enhance maintenance activities. Furthermore, for those who rely primarily on Data Science experts, the ML models are tuned manually, often requiring several trial-and-error experiments. In contrast with the human ML design approach, in this study we focus on an AutoML, aiming to automate the ML modeling phase and thus reduce the data to maintenance insights process cycle. Moreover, we apply AutoML using real-world data collected from the client of a Portuguese software company in the area of maintenance management.

The AutoML was explored for two specific prediction tasks: the number of days until an equipment fails and if the equipments will fail in a fixed number of days. We designed a large set of computational experiments to assess the AutoML predictive performance of ten open-source tools focused on a supervised learning. Additionally, we propose AutoOneClass, a novel AutoML approach for OCC that uses a GE optimization. Finally, to provide a baseline comparison, we compare the best AutoML and AutoOneClass results with two manual ML analyses, based on a non-expert ML modeling made previously by one of the company's professionals and an external ML expert design. The comparison favors the supervised AutoML and AutoOneClass results, thus attesting to the potential of the AutoML approach for the PdM application domain.

The main contributions of this work are summarized as follows:

(i) We propose AutoOneClass, an AutoML framework that focuses on OCC using three algorithms: deep AE, IF, and OC-SVM. AutoOneClass uses GE to optimize the search for the best OCC ML algorithm and its associated hyperparameters for a given dataset;

(ii) For the AutoOneClass method, we assume a single or multi-objective search. The single-objective approach only uses the predictive performance to select the best ML model, while the multi-objective variant considers two objectives simultaneously, predictive performance and training time;

(iii) We use two validation setups for AutoOneClass: unsupervised and supervised validation. The purely unsupervised method uses unlabeled data during validation and anomaly scores to evaluate the ML models. The supervised validation (using a labeled validation set) uses the AUC of the ROC to assess model performance;

(iv) We conduct a large set of experiments, predicting equipment failures in different time windows (e.g., 3 days, 5 days) and compare the results from the new AutoOneClass with ten AutoML tools, focused on classical ML and DL.

The chapter is organized as follows. Section 4.3 describes the supervised AutoML tools, the proposed AutoOneClass approach, and the analyzed PdM dataset. Then, Section 4.4 shows and discusses the experimental results. Finally, Section 4.5 presents the main conclusions and future work directions.

4.3 Materials and Methods

4.3.1 Supervised AutoML Tools

In this study, we compare ten recent open-source AutoML tools for supervised classification and regression tasks. Most of the selected tools were explored on a recent benchmark study (Ferreira, Pilastri, Martins, et al., 2021). In order to achieve a more fair comparison, we did not tune the hyperparameters of the AutoML tools. Table 17 summarizes the main characteristics of the ten supervised AutoML tools, namely the base framework (**Framework**), the available API languages (**API**), if the tool uses DL algorithms (**DL**), if the tool supports GPU usage (**GPU**), and the version that we used in our experiments (**Version**). Additional details about the AutoML tools are provided in Section 2.2.2.

Table 17: Description of the supervised AutoML tools.

Tool	Framework	API	DL	GPU	Version
Auto-Keras	Keras	Python	✓(only)	✓	1.0.18
Auto-PyTorch	PyTorch	Python	✓(only)	✓	0.1.1
Auto-Sklearn	Scikit-Learn	Python	-	-	0.14.6
AutoGluon	Gluon	Python	✓	✓	0.2.0
H2O AutoML	H2O	Java, Python, R	✓	✓(partial)	3.32.1.3
MLJar	CatBoost, Keras, Scikit-Learn, XGB	Python	✓	-	0.11.2
PyCaret	Scikit-Learn	Python	-	✓(partial)	2.3.10
rminer	rminer	R	-	-	1.4.6
TPOT	Scikit-Learn	Python	-	✓(partial)	0.11.7
TransmogrifAI	Spark (MLlib)	Scala	-	-	0.7.0

4.3.2 AutoOneClass: Automated One-Class Learning

All the AutoML tools described in Section 4.3.1 apply supervised learning techniques (e.g., binary classification, regression). However, as explained in Section 2.3.3, there are PdM studies that focus on an unsupervised learning (Amruthnath & Gupta, 2018b; Cho et al., 2018). In particular, we focus on an OCC,

such as adopted by Makridis et al. (2020) and Straus et al. (2018). OCC is often employed in anomaly detection tasks, where the ML algorithms are only trained with normal examples, producing learning models that tend to trigger high anomaly scores when faced with abnormal records outside the learned normal input space (Ribeiro et al., 2022). OCC is valuable for PdM, since the associated classification tasks are often extremely unbalanced. Indeed, a large majority of the PdM records are related with a normal equipment functioning, thus these records can be more easily collected without a high data labeling cost.

While there is currently a large list of AutoML frameworks, these solutions typically only focus on supervised learning. Indeed, as argued by Bahri et al. (2022) and He et al. (2021), very few works have explored AutoML outside the supervised learning domain, thus this is still a current research challenge for AutoML. Following this research gap, in this study we propose AutoOneClass, an AutoML framework that focuses on a OCC. AutoOneClass uses GE to optimize the search for the OCC ML algorithm and its associated hyperparameters for a given dataset.

Furthermore, AutoOneClass can assume a single or multi-objective search. The single-objective approach only uses the predictive performance to select the best ML model, while the multi-objective variant considers two objectives, predictive performance and training efficiency (measured by computational training time). For the multi-objective setup, we adopt a Pareto optimization that performs a simultaneous optimization of both objectives, resulting in a final Pareto front that contains a set of non-dominated solutions, where each solution constitutes a different predictive performance vs training efficiency trade-off. As such, there is no *a priori* definition of fixed weights between the two objectives. We note that the AutoOneClass multi-objective variant was developed in order to allow the selection of lighter ML models, even if they have a slightly lower performance, since these types of models are valuable when handling big data. However, the dataset analyzed in this work is relatively small.

AutoOneClass is mainly designed for anomaly detection tasks, where there is a distinction between "normal" instances and "anomalies", in particular when "normal" records represent most of the dataset. Given that AutoOneClass implements OCC algorithms, the ML models are trained using only data from one of the classes (typically, the "normal" class). In all our experiments related to AutoOneClass, we only used normal data for the learning (i.e., training) phase.

However, the AutoOneClass validation (which will impact the GE optimization) can be executed using two setups: unsupervised validation, where the model performance is evaluated only using unlabeled data (e.g., through an anomaly score); or supervised validation, where there is access to a labeled validation set to assess the model performance using supervised learning metrics (e.g., AUC). This means that, in our AutoOneClass experiments, for the unsupervised validation setup, the validation set was composed only of normal data; for the supervised validation setup, we used validation sets comprised of labeled data (with normal and abnormal records). Nevertheless, independently of the validation setup, the AutoOneClass method only uses normal data during training, thus only dealing with an OCC training.

Given the different validation strategies, we use distinct fitness functions as the predictive objectives for the GE optimization. In the cases where supervised validation is used, we consider the maximization of the validation AUC as our predictive objective. For the predictive objective of the unsupervised validation,

we minimize an anomaly score, which was set to vary within the range [0,1] for all OCC methods, thus allowing its interpretation as an anomaly score probability.

We note that under the unsupervised validation assumption, there is no access to labeled data (i.e., abnormal examples) to perform a model selection, thus the AUC computation is not feasible in this scenario. Since a model selection criterion is needed (e.g., to select the best AE configuration), we assume the anomaly score minimization as a proxy for the AUC. The rationale is that if a model provides a low anomaly score when trained with a large set of normal data, then it should be capable of triggering high anomaly scores for abnormal data, which should reflect on a good enough ROC curve. Nevertheless, to correctly benchmark the unsupervised validation scenario, we used labeled data on the test set, allowing us to compute the ROC curves and their AUC measures, which are then compared with the ones obtained when using the supervised validation scenario.

4.3.2.1 One-Class Learning Algorithms

AutoOneClass uses three popular OCC algorithms: AE, IF, and OC-SVM. This means that, for a given dataset, AutoOneClass selects one of these three algorithms at the end of the GE optimization. The AEs were implemented through the TensorFlow library using the Keras submodule (Martín Abadi et al., 2015), while both IF and OC-SVM were implemented using the Scikit-Learn framework (Pedregosa et al., 2011). Additional details about the base learners are provided in Section 2.2.3.

4.3.2.2 Grammatical Evolution

In this work, we built AutoOneClass using PonyGE2, an open source implementation of GE in Python (Fenton et al., 2017) that allows the usage of Python BNF (PyBNF), in which the production rules can include Python code. For the AutoOneClass framework, we developed a PyBNF grammar that can tune the hyperparameters of the OCC algorithms described in Section 4.3.2.1. The grammar was then adapted to allow two types of optimization: All - in which the GE execution generates one of the three algorithms for each solution (individual); and separate mode, in which the GE only generates one family of algorithms for all individuals (e.g., AEs). The PyBNF grammar we used in this work is shown in Fig. 12.

In practice, the usage of PyBNF allowed us to generate snippets of Python code that allow GE to generate different types of ML models. For example, the IF and OC-SVM grammars were implemented by creating the respective Scikit-Learn class and adding the hyperparameters as terminals and non-terminals.

This process was more complex for the AEs, since the TensorFlow API requires the definition of a variable number of layers. To achieve this, we defined the grammar to generate only the encoder: first, generate an input layer with the same number of nodes as the number of attributes of the dataset and then add a variable number of hidden layers. Since the decoder is symmetrical to the encoder, this component is not included in the grammar. Also, given that in a typical AE, the subsequent encoder layers have fewer nodes than the previous layer, we defined the layer nodes as a percentage (between 0% and 100%) of nodes of the previous layer instead of a fixed number. Finally, we defined an auxiliary

```
<response> ::= <autoencoder> | <iforest> | <ocsvm>

<autoencoder> ::=  encoder = Sequential(){::}
                   encoder.add(Input(shape=(input_shape,), name="'input'")){::}
                   <hidden_layers>{::}
                   <latent_space>{::}
                   model = get_model_from_encoder(encoder){::}
                   model.add(Dense(input_shape, activation=<activation>, name="'output'")){::}
                   model.compile(<optimizer>, "'mae'")
<hidden_layers> ::= <Dense>{::} | <Dense>{::}<Dense>{::} | <hidden_layers><Dense>{::} | <Dense>{::}<extra>{::}
<Dense> ::= encoder.add(Dense(units = <percentage>, activation = <activation>))
<activation> ::= "'relu'" | "'sigmoid'" | "'softmax'" | "'softplus’" | "'tanh'" | "'selu'" | "'elu'" | “’exponential'"
<latent_space> ::= encoder.add(Dense(units = <percentage>, activation = <activation>, name="'latent'"))
<extra> ::= encoder.add(Dropout(rate=0.<dropout_digit>)){::} | encoder.add(BatchNormalization()){::}
<dropout_digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<percentage> ::= 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100
<optimizer> ::= "'RMSprop'" | "'Adam'"


<iforest> ::= model = IsolationForest(n_estimators=<estimators>, contamination=<contamination>, bootstrap=<bootstrap>)
<estimators> ::= <digit><estimators> | <digit>
<estimators_digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<contamination> ::= "'auto'" | 0.<contamination_digits>
<contamination_digits> ::= 1 | 2 | 3 | 4 | 5
<bootstrap> ::= "True" | "False"


<ocsvm> ::= model = OneClassSVM(kernel=<kernel>, degree=<degree>, gamma=<gamma>, shrinking=<shrinking>)
<kernel> ::= "'linear'" | "'poly'" | "'rbf'" | "'sigmoid'"
<degree> ::= <digit><degree_digit> | <digit>
<degree_digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<gamma> ::= "'scale'" | "'auto'"
<shrinking> ::= "True" | "False"
```

Figure 12: The PyBNF grammar used in this work.

function `get_model_from_encoder`, which translates the generated phenotype to a functional Keras AE.

4.3.3 Data

The data used in this work was provided by a Portuguese software company focused on maintenance management and presents a real historical record from one of the company's clients. The company has many PdM datasets, detailed in Fig. 13.

For the context of this work, we assume a tabular dataset composed of the aggregation of several attributes from each entity. Overall, the data includes 2,608 records and 21 input attributes. Each record represents an action (e.g., a work order) related to one of the company's equipments (e.g., an industrial machine). In addition, each record includes diverse input attributes, such as the machine's tasks, material consumption, and meter readings.

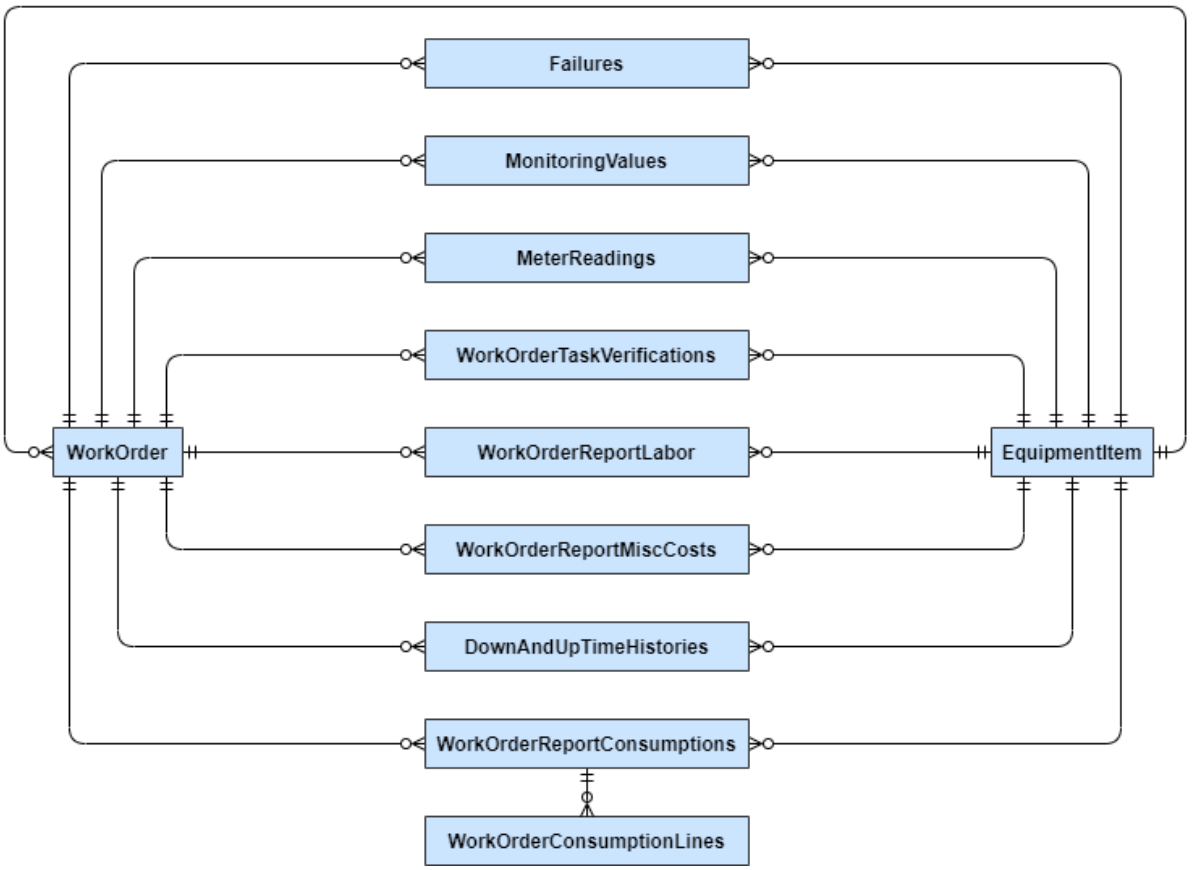


Figure 13: Entities and relationships between the datasets.

Table 18 details the input and output variables (**Attribute**), their description (**Description**), data type (**Type**), number of levels (**Levels**), domain values (**Domain**), and example values from one of the records (**Example**).

Half (12) of the 21 input attributes are categorical. Among these, most present a low cardinality (e.g., RecordType, Brand). However, some attributes present a very high cardinality (e.g., Part). The dataset includes five target variables for regression or binary classification tasks. The regression task target (attribute DaysToNextFailure) describes the number of days between that record and the failure of the respective equipment. As for the binary classification targets (attributes FailOnxDays), these describe if the equipment will fail or not in a certain amount of days (e.g., in three days).

Fig. 14 shows the histogram for the regression target and the balancing of classes for the binary classification targets. Regarding the regression target (attribute DaysToNextFailure), the available equipments will fail between 0 and 1550 days. However, many records (956) present a value between 0 and 155 days until the next failure. On the other hand, only a small number of records present a number of days until failure (e.g., only 89 records present a value larger than 1240). Fig. 14 also shows that all four binary classification targets present highly imbalanced classes, with the majority of the records corresponding to "normal" situations. Only a tiny percentage of the equipments will fail on the respective interval (between

Table 18: Description of the equipment maintenance dataset attributes.

Attribute	**Description**	**Type**	**Levels**	**Domain**	**Example**
RecordType	Type of record	String	5	-	Failure
Brand	Brand of the equipment	String	2	-	Rossi
WOType	Type of work order	String	4	-	Corrective
PriorityLevel	Priority Level of the work order	String	4	-	Urgent
Responsible	Responsible for the work order	String	3	-	R4
Employee	Employee that performed the action	String	12	-	E100
TotalTime	Duration of the action (in hours)	Float	17	[0, 8]	8
Quantity	Consumption quantity	Float	32	[0, 300]	90
Part	Part that was consumed	String	161	-	T-1073
Meter	Meter associated to meter reading	String	11	-	L-0002
MeterCumulative Reading	Cumulative reading of meter	Float	1477	[0, 73636]	22767
IncrementValue	Increment compared to last reading	Float	475	[0, 54570]	168
MaintenancePlan	Maintenance Plan associated to task	String	5	-	P-000011
Task	Executed task	String	5	-	T-0001
AssetWithFailure	Identification of the equipment	String	15	-	A577
ParentAsset	Parent equipment of *AssetWithFailure*	String	11	-	LINHA2
Day	Day of the month of the record	Integer	31	[1, 31]	4
DayOfWeek	Day of the week of the record	Integer	7	[1, 7]	6
Month	Month of the record	Integer	12	[1, 12]	2
Year	Year of the record	Integer	6	[2015, 2019]	2019
DaysAfterPurchase	Age of the equipment (in days)	Integer	852	[0, 6309]	4479
DaysToNext Failure	Number of Days until the next failure of the equipment	Integer	1015	[0, 1550]	3
FailOn3Days	Indication whether the equipment will fail in the next 3 days	Integer	2	{0,1}	1
FailOn5Days	Indication whether the equipment will fail in the next 5 days	Integer	2	{0,1}	1
FailOn7Days	Indication whether the equipment will fail in the next 7 days	Integer	2	{0,1}	1
FailOn10Days	Indication whether the equipment will fail in the next 10 days	Integer	2	{0,1}	1

3 and 10 days, depending on the target). The most imbalanced target column is FailOn3Days, with only 2.53% of records that will present failures in 3 days. As expected, the larger the interval being considered, the larger the percentage of the "failure" class. However, even the least unbalanced target column (FailOn10Days) presents 7.82% of records that will have failures. As other studies show (e.g., (Dangut et al., 2022)), imbalanced datasets are very prevalent in the PdM domain since failures are frequently sporadic compared to health situations.

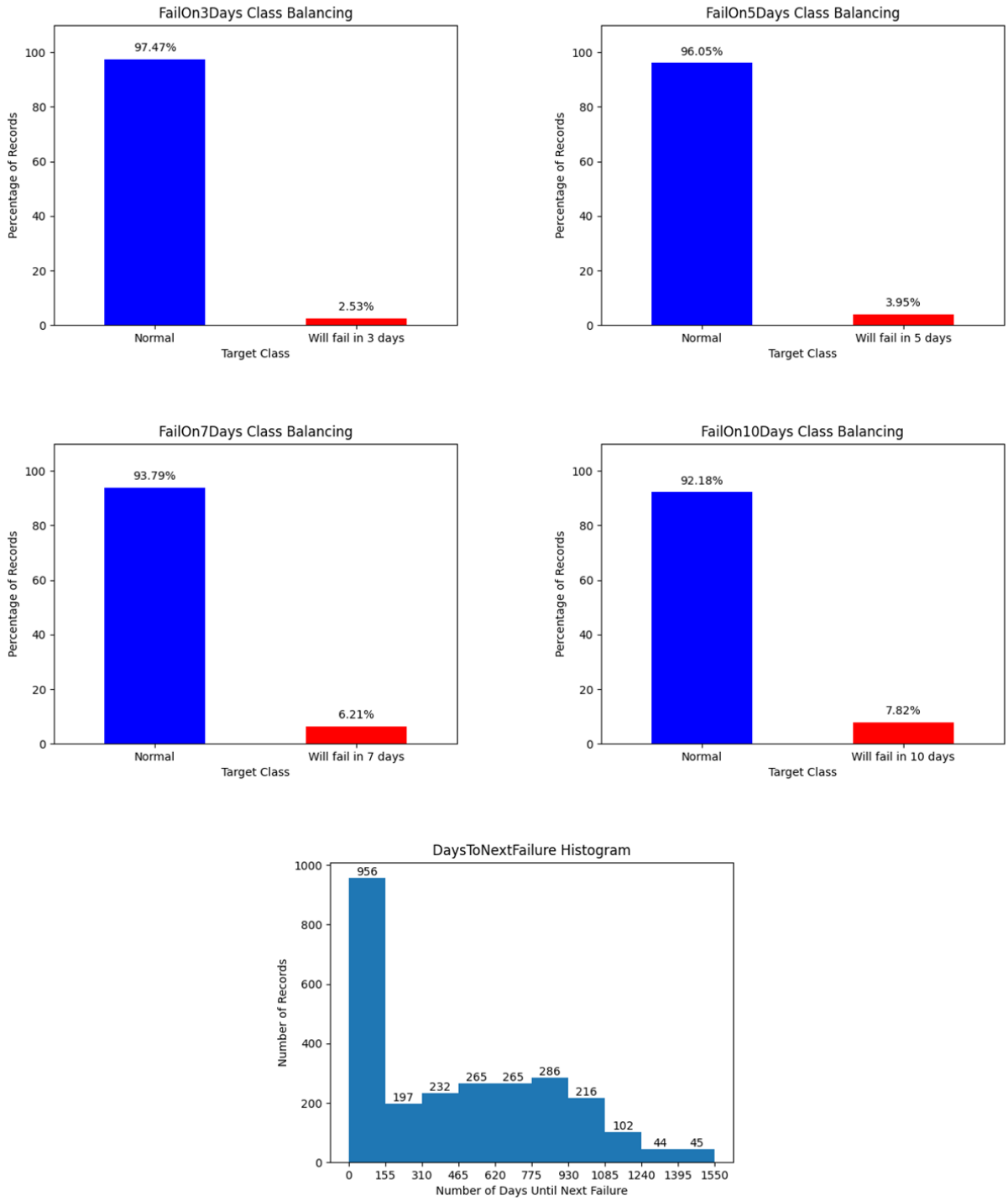


Figure 14: Balancing of the binary classification targets and histogram of the regression target.

4.3.4 Data Preprocessing

Since several data attributes are of the type String (as shown in Table 18), which is not accepted by some AutoML tools, we opted to encode all String attributes into numerical types. To decide the most appropriate techniques to transform the textual attributes into numerical, we first analyzed the number of records and

corresponding percentage for missing and unique values, which are presented in Table 19.

Table 19: Missing values and unique values of the datasets.

Attribute	**Missing Values**		**Unique Values**	
	No.	%	No.	%
RecordType	0	0	5	<1
Brand	203	8	2	<1
WOType	1801	69	4	<1
PriorityLevel	1801	69	4	<1
Employee	2228	85	12	<1
TotalTime	0	0	-	-
Quantity	32	1	-	-
Part	2338	90	161	6
Meter	0	0	11	<1
MeterCumulative	0	0	-	-
IncrementValue	0	0	-	-
MaintenancePlan	2228	85	5	<1
Task	2228	85	5	<1
AssetWithFailure	0	0	15	<1
ParentAsset	0	0	11	<1
Day	0	0	31	1
DayOfWeek	0	0	7	<1
Month	0	0	12	<1
Year	0	0	6	<1
DaysAfterPurchase	0	0	-	-
DaysToNextFailure	0	0	-	-
FailOn3Days	0	0	2	<1
FailOn5Days	0	0	2	<1
FailOn7Days	0	0	2	<1
FailOn10Days	0	0	2	<1

For the String attributes that presented a low cardinality (five levels or less), we applied the known One-Hot encoding. For the columns that had missing values, we replaced the missing value with zero, which is assumed as a numeric code value for the "unknown" level. Since the One-Hot encoding method creates one binary column for each level of the original attribute, we applied a different transformation for the columns with a higher cardinality.

Indeed, for the categorical variables with more than five levels, we used the **Inverse Document Frequency (IDF)** technique, available on the Python CANE module (Matos et al., 2022). This method converts a categorical column into a numerical column of positive values based on the frequency of each attribute level. IDF uses the function $f(x) = log(n/fx)$, where n is the length of x and fx is the frequency of x. The benefit of IDF, compared with One-Hot Encoding, is that the IDF technique does not generate new columns, which is useful for attributes with high cardinality (e.g., the attribute Part has 161 levels).

The remaining attributes (of Integer and Float types) were not altered because most AutoML tools already apply preprocessing techniques to the numerical columns (e.g., normalization, standardization). Furthermore, we did not replace the missing values for the only numerical column that presented missing values (Quantity), since the AutoML tools usually perform an imputation task before running the algorithms. After applying the transformations, the final dataset had 42 inputs and five target columns.

4.3.5 Evaluation

In order to evaluate the results from the AutoML tools and AutoOneClass, we adopted a similar approach to the benchmark developed by Ferreira, Pilastri, Martins, et al. (2021). For every predictive experiment, we divided the dataset into 10 folds for an external cross-validation and adopted an internal 5-fold cross-validation (i.e., over the training data) for the AutoML tools, to select the best algorithm and hyperparameters (executed automatically by the AutoML tools). To evaluate the test set (from external 10-fold validation) predictions we used the MAE ($\in [0.0,\infty[$, where 0.0 represents a perfect model) for the regression task and the AUC analysis ($\in [0.0,1.0]$, where 1.0 indicates an ideal classifier) for the binary classification targets. We also used MAE and AUC for the internal validation, responsible for choosing the best ML model.

For all ten AutoML tools, we defined a maximum training time of one hour (3,600 seconds) and an early stopping of three rounds, when available. The maximum time of one hour was chosen since it is the default value for most of the AutoML tools. We computed the average of the evaluation measures on the test sets of the 10 external folds to provide an aggregated value. Additionally, we use confidence intervals based on the t-distribution with 95% confidence to verify the statistical significance of the experiments. In order to identify the best results for each target, we chose the AutoML tool with the best average predictive performance (with maximum precision of 0.01). All experiments were executed using an Intel Xeon 1.70GHz server with 56 cores and 64GB of RAM, without a GPU.

4.4 Results

4.4.1 AutoML Results

The first comparison focused on the supervised AutoML tools detailed in Section 4.3.1. For each AutoML tool, we executed five experiments, one for each target variable (DaysToNextFailure and FailOnxDays). Table 20 shows the average external test scores for all 10 folds and the respective confidence intervals (near the $\pm$ symbol). For the best models of each target, we also apply the nonparametric Wilcoxon test for measuring statistical significance (Hollander et al., 2013).

The best tool for the regression task (DaysToNextFailure) was AutoGluon, which produced the lowest average MAE. Besides AutoGluon, the two best tools were H2O AutoML and Auto-Sklearn. For this task, the maximum predictive difference among all tools was 79.07 points (days). On the other hand, the worst

Table 20: Average predictive results obtained by the AutoML tools (best values for each target in **bold**).

		Targets				
		Days Until Next Failure	**Fail In 3 Days**	**Fail In 5 Days**	**Fail In 7 Days**	**Fail In 10 Days**
		MAE	**AUC**	**AUC**	**AUC**	**AUC**
AutoDL Tools	Auto-Keras	84.02±37.55	0.72±0.12	0.74±0.05	0.79±0.05	0.79±0.03
	Auto-PyTorch	12.75±6.45	0.76±0.10	0.74±0.07	0.78±0.13	0.79±0.13
AutoML Tools	Auto-Sklearn	6.20±0.50	0.82±0.10	0.84±0.08	0.90±0.05	0.91±0.04
	AutoGluon	**4.95**[a]±0.57	**0.98**[b]±0.02	**0.97**[c]±0.02	**0.98**[c]±0.01	**0.99**[c]±0.01
	H2O AutoML	5.53±0.62	**0.98**[b]±0.01	0.96±0.03	**0.98**[c]±0.01	0.98±0.01
	MLJar	8.32±0.62	0.77±0.12	0.82±0.06	0.85±0.07	0.89±0.05
	PyCaret	7.91±1.20	0.77±0.11	0.80±0.07	0.86±0.05	0.89±0.04
	rminer	8.89±0.75	0.95±0.05	0.93±0.04	0.97±0.03	0.98±0.02
	TPOT	7.05±0.57	0.97±0.03	0.96±0.02	**0.98**[c]±0.01	**0.99**[c]±0.01
	TransmogrifAI	17.34±1.23	0.92±0.04	0.94±0.03	0.96±0.02	0.98±0.01

[a]Statistically significant (p-value < 0.05) under a pairwise comparison when compared with the tools: Auto-Keras, Auto-PyTorch, Auto-Sklearn, MLJar, PyCaret, rminer, TPOT, and TransmogrifAI.
[b]Statistically significant (p-value < 0.05) under a pairwise comparison when compared with the tools: Auto-Keras, Auto-PyTorch, MLJar, PyCaret, and TransmogrifAI.
[c]Statistically significant (p-value < 0.05) under a pairwise comparison when compared with the tools: Auto-Keras, Auto-PyTorch, MLJar, and PyCaret.

tool was Auto-Keras, which produced an average MAE of 84.02 days, a significantly higher value when compared to the remaining AutoML and AutoDL tools.

As for the binary classification, AutoGluon was the best tool for all four binary classification targets, followed by H2O AutoML and TPOT (best in two targets each). The binary classification results show that the AutoDL tools (Auto-Keras and Auto-PyTorch) performed significantly worse than the AutoML tools, obtaining lower AUC results than all these tools. Nonetheless, the predictive test set results also present significant discrepancies between tools: maximum difference of 26 *percentage points (pp)* for FailOn3Days, 23 *pp* for FailOn5Days, 20 *pp* for FailOn7Days, and 20 *pp* for FailOn10Days. However, when excluding the AutoDL tools, these differences are smaller: maximum difference of 21 *pp* for FailOn3Days, 17 *pp* for FailOn5Days, 13 *pp* for FailOn7Days, and 10 *pp* for FailOn10Days. Even though the AutoDL tools show, in general, worse results, they obtained similar results between each other, with the maximum predictive difference of 4 *pp* (for the target FailOn3Days) between Auto-Keras and Auto-PyTorch.

Additionally, we analyzed the training times (average of the external 10 folds) and respective confidence intervals of the AutoML tools, shown in Table 21. The slowest tool was Auto-Sklearn, which always required the maximum allowed training time (3,600 s), followed by Auto-Keras (average of 2,550 s per external fold and dataset) and MLJar (average of 2,015 s). On the other hand, PyCaret presented the lowest average value (206 s), best in two datasets; AutoGluon - second best average value (396 s), best in three datasets; rminer - third best average (440 s).

Table 21: Average training times (in seconds) obtained by the AutoML tools (best values for each target in **bold**).

		Targets				
		Days Unitl Next Failure	**Fail In 3 Days**	**Fail In 5 Days**	**Fail In 7 Days**	**Fail In 10 Days**
AutoDL Tools	Auto-Keras	2532±1137	2579±516	3374±2135	3209±1233	1055±291
	Auto-PyTorch	1514±116	1450±161	1262±107	1524±111	1334±155
AutoML Tools	Auto-Sklearn	3600±0	3600±0	3600±0	3600±0	3600±0
	AutoGluon	**130**±9	**143**±12	**146**±17	264±243	1296±291
	H2O AutoML	643±573	495±180	635±310	831±596	2764±613
	MLJar	1519±57	1607±42	1653±46	2066±413	3232±770
	PyCaret	178±9	193±4	200±5	**208**±19	**253**±41
	rminer	329±10	355±4	361±6	378±22	776±721
	TPOT	1552±770	1936±1020	2032±774	1839±804	1903±1206
	TransmogrifAI	656±10	688±6	710±16	739±7	777±3

The overall results suggest that AutoML tools that focus on classical ML algorithms (e.g., DTs, RF) are best suited to help the Portuguese company to predict failures for their equipments. Nonetheless, the AutoDL predictive results might be justified by the small size of the analyzed dataset (which contains only 2,608 records) since it is generally accepted that DL tends to produce better results with large datasets (Ng, 2020). Also, since the experiments did not use GPU, the maximum training time of one hour might have not allowed the AutoDL tools to perform enough computation to achieve competitive results.

4.4.2 AutoOneClass Results

The second predictive comparison considers the AutoOneClass method, proposed and described in Section 4.3.2. Given that the method only works for binary classification tasks, the regression task was not considered in these predictive tests. Instead, we performed several experiments with different parameters, such as the type of validation, the used algorithms, and the type of optimization (single or multi-objective). We executed all the AutoOneClass experiments with an initial population of 10 individuals and 10 generations (GE parameters). The summary of the different parameters used in the experimental evaluation is shown in Table 22. We note that we adopted the default PonyGE2 values for crossover and mutation, namely: Variable Onepoint crossover (selection of a different point on each parent genome for crossover to occur) with a crossover probability of 75%; and Int Flip Per Codon mutation (random mutation of every individual codon in the genome) with a mutation probability of 100%.

Table 23 shows the average test results of the 10 folds and the respective confidence intervals. The table also shows the type of validation (**Validation**) that was used, which algorithms were considered (**Alg.**), and which of the two available optimization modes (single-objective or multi-objective) was chosen (**Opt.**). For comparison reasons, the table also shows, for each binary classification target, the best AutoDL

Table 22: Parameters used for the AutoOneClass experiments and respective values.

Parameter	**Used Values**
Population Size	10
Number of Generations	10
Crossover	Variable Onepoint with 75% crossover probability (PonyGE2 default)
Mutation	Int Flip Per Codon with 100% mutation probability (PonyGE2 default)
Validation Type	Supervised
	Unsupervised
Algorithm	Autoencoder
	Isolation Forest
	One-Class SVM
	All (the three algorithms simultaneously)
Optimization Type	Single-objective
	Multi-objective
Predictive Objective	Maximize Validation AUC (for supervised validation)
	Minimize Reconstruction Error (for AE unsupervised validation)
	Minimize Anomaly Score (for IF and OC-SVM unsupervised validation)
Effiency Objective	None (for single-objective)
	Training Time (for multi-objective)
Targets	FailOn3Days
	FailOn5Days
	FailOn7Days
	FailOn10Days

and AutoML results (from Table 20). For the best models of each target, we apply the nonparametric Wilcoxon test for measuring statistical significance.

It is worth mentioning that, for the single-objective executions, the average test results shown on the table represent the average of the best models (one model per fold) since this type of optimization only considers the predictive performance of the ML models and is able to identify one "leader" model. On the other hand, for the multi-objective optimization, the average results include several models per fold (all that belong to the Pareto front), since it considers two objectives (predictive performance and training time). Therefore it generates more than one optimal model per fold.

The results show that, on average, the executions that used a supervised validation (using a labeled validation set) achieved better results than those with unsupervised validation sets (using unlabeled validation data). While the supervised validation achieved an average 0.73 of AUC (across all algorithms and optimization types), the unsupervised validation obtained 0.66 points, on average. Regarding the previously discussed topic related to the usage of the anomaly scores as a proxy for the AUC for the unsupervised validation (mentioned in Section 4.3.2), we note that these experimental results have shown that the improvement of the supervised validation is relatively small (average of 7 percentage points), thus backing the usage of the anomaly score minimization criterion for the unsupervised validation scenario.

When comparing the types of algorithms considered in these experiments (AEs, IF, OC-SVM, or the

Table 23: Average predictive results (AUC) obtained by the proposed AutoOneClass method (best values obtained by AutoOneClass for each target in **bold**).

				Targets			
	Validation	**Alg.**	**Opt.***	**Fail In 3 Days**	**Fail In 5 Days**	**Fail In 7 Days**	**Fail In 10 Days**
AutoOneClass	Supervised	AE	SO	0.73±0.01	0.67±0.00	0.72±0.01	0.71±0.01
	Supervised	AE	MO	0.67±0.05	0.64±0.01	0.71±0.01	0.69±0.01
	Supervised	IF	SO	0.79±0.01	**0.80**[b]±0.02	**0.80**[b]±0.01	**0.80**[c]±0.02
	Supervised	IF	MO	0.77±0.02	0.77±0.02	0.79±0.01	0.77±0.01
	Supervised	OC-SVM	SO	0.70±0.01	0.67±0.01	0.70±0.01	0.67±0.01
	Supervised	OC-SVM	MO	0.68±0.02	0.66±0.01	0.67±0.02	0.67±0.02
	Supervised	All	SO	**0.80**[a]±0.05	0.76±0.06	0.77±0.04	0.77±0.03
	Supervised	All	MO	0.76±0.06	0.77±0.06	0.77±0.05	0.75±0.03
	Unsupervised	AE	SO	0.71±0.02	0.64±0.01	0.71±0.01	0.69±0.01
	Unsupervised	AE	MO	0.67±0.05	0.63±0.02	0.71±0.01	0.69±0.01
	Unsupervised	IF	SO	0.70±0.05	0.68±0.04	0.71±0.03	0.69±0.02
	Unsupervised	IF	MO	0.66±0.06	0.69±0.04	0.71±0.04	0.67±0.07
	Unsupervised	OC-SVM	SO	0.62±0.06	0.60±0.07	0.55±0.06	0.65±0.06
	Unsupervised	OC-SVM	MO	0.60±0.08	0.57±0.06	0.55±0.06	0.63±0.08
Best NAS/AutoDL result				0.76±0.10	0.74±0.07	0.79±0.05	0.79±0.03
Best AutoML result				0.98±0.01	0.97±0.02	0.98±0.01	0.99±0.01

*SO - Single-objective; MO - Multi-objective.
[a]Statistically significant (p-value < 0.05) under a pairwise comparison when compared with all the other setups except: Supervised/IF/SO and Supervised/IF/MO.
[b]Statistically significant (p-value < 0.05) under a pairwise comparison when compared with all the other setups except: Supervised/All/MO.
[c]Statistically significant (p-value < 0.05) under a pairwise comparison when compared with all the other setups except: Supervised/All/SO.

three simultaneously), the mode with all three algorithms simultaneously generated the best results, with an average AUC of 0.77. Next, the second best algorithm was IF (average of 0.74 AUC), followed by AE (average of 0.69 AUC), and the OC-SVM algorithm obtained the worst results, with 0.64 of average AUC.

Another interesting result was that the single-objective executions only achieved slightly better predictive results than the multi-objective ones. Indeed, grouping the results by type of validation and algorithm, the average difference between the single-objective and multi-objective results was 0.02 *pp*. These differences can be further analyzed in Fig. 15.

Similar to the previous experiment, we also analyze the average training times for the AutoOneClass results, shown in Table 24. The results show that the average training times of AutoOneClass when using AEs were much higher than the other algorithms (average training time of 2,732 s across all folds and datasets). On the other hand, OC-SVM presented the lowest average training time (85 s), followed by IF (194 s) and lastly the setup which uses all algorithms (538 s).

A comparison between the predictive results achieved by the proposed AutoOneClass method (shown in Table 23) and the AutoML results (shown in Table 20) shows that none of the AutoOneClass executions

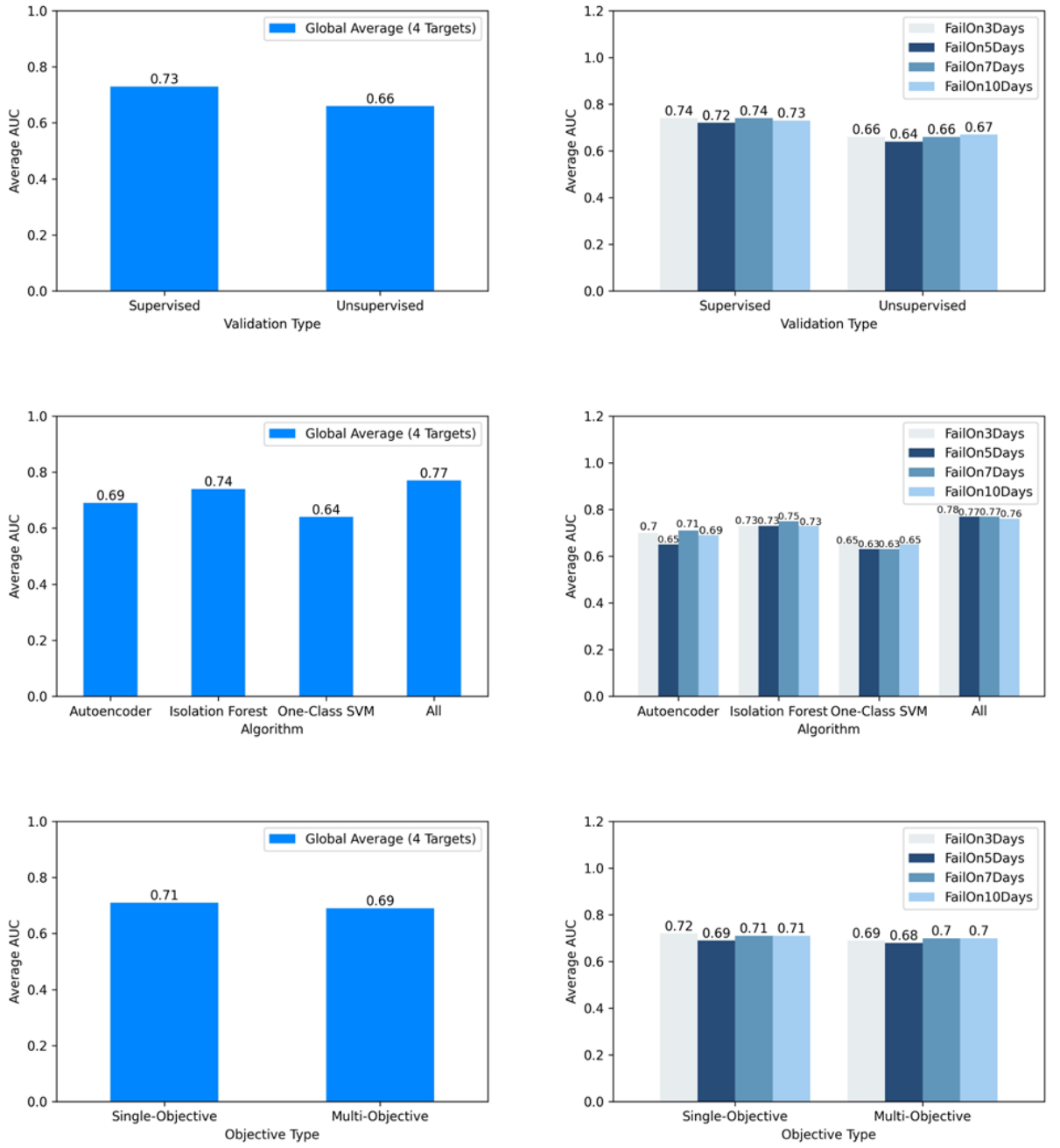


Figure 15: AutoOneClass results aggregated by validation type, algorithm, and optimization type, both globally (left) and per target (right).

outperformed the best AutoML tools on all four binary classification targets. However, when comparing the AutoOneClass results only with AutoDL tools, AutoOneClass generated at least one result better than all of the AutoDL tools (Auto-Keras and Auto-PyTorch).

It should be stressed that the AutoOneClass method requires much less labeled data to train the ML models (only uses labeled data for the supervised validation), when compared with the supervised AutoML tools, which typically require a labeled dataset with a balanced ratio of normal and abnormal records. In

Table 24: Average training times (in seconds) obtained by the proposed AutoOneClass method (best values obtained by AutoOneClass for each target in **bold**).

				Targets			
	Validation	**Alg.**	**Opt.***	**Fail In 3 Days**	**Fail In 5 Days**	**Fail In 7 Days**	**Fail In 10 Days**
AutoOneClass	Supervised	AE	SO	2349±945	3151±1250	3215±1566	2352±928
	Supervised	AE	MO	4747±2265	3053±1186	2242±888	3109±1220
	Supervised	IF	SO	157±49	177±71	191±86	255±100
	Supervised	IF	MO	123±45	213±75	87±32	159±64
	Supervised	OC-SVM	SO	136±53	115±45	106±42	93±37
	Supervised	OC-SVM	MO	108±43	116±45	96±39	84±33
	Supervised	All	SO	571±209	464±188	598±239	609±238
	Supervised	All	MO	729±243	491±189	416±161	422±151
	Unsupervised	AE	SO	2700±1073	2563±1005	2352±918	2430±921
	Unsupervised	AE	MO	2418±931	2396±965	2282±874	2348±911
	Unsupervised	IF	SO	468±207	365±153	229±104	146±56
	Unsupervised	IF	MO	94±42	168±62	198±75	71±30
	Unsupervised	OC-SVM	SO	74±28	67±26	59±22	66±26
	Unsupervised	OC-SVM	MO	**63**±24	**63**±23	**48**±21	**64**±25
Best NAS/AutoDL result				1514±116	1450±161	1262±107	1334±155
Best AutoML result				130±9	143±12	146±17	208±19

*SO - Single-objective; MO - Multi-objective.

many real-world PdM scenarios, there is a huge number of normal records and anomaly records might not always be available. Thus, AutoOneClass could be valuable in PdM use cases, when most of the data is comprised by normal data and where anomaly records are costly to be collected and labeled (e.g., equipment condition monitoring, failure detection).

We note that these experiments had some limitations that might present disadvantages for the AutoOneClass method. First, the training time of one hour might have been insufficient for tools that rely on DL algorithms (e.g., AEs, AutoDL tools), in particular since no GPU is used. Second, the usage of a larger dataset could have improved both AutoOneClass and AutoDL predictive results. Third, GE optimization used fixed values (PonyGE2 default) for some of the parameters, such as the crossover and mutation operators.

4.4.3 Comparison With a Human ML Modeling

Finally, we compare the best AutoML results for each target with the best result achieved by two examples of a human ML modeling, as performed by a non-ML expert belonging to the analyzed Portuguese software company and an external ML expert. Table 25 compares the prediction results achieved using the manual ML design and best AutoML tools. For each AutoML tool, Table 25 includes the algorithm (**Alg.**) that was most often the leader across the external folds (in rounded brackets). For the human modeling, Table 25

shows the best obtained result and the used algorithm.

Table 25: Comparison between the best AutoML results, AutoOneClass results, and human ML modeling results (expert and non-expert) for each target (best values in **bold**).

Target	**Measure**	**Best Results**							
		AutoML		**AutoOneClass**		**Human (Non-expert)**		**Human (Expert)**	
		Score	**Tool (Alg.)**	**Score**	**Alg.**	**Score**	**Alg.**	**Score**	**Alg.**
DaysToNextFailure	MAE	**4.948**	AutoGluon (Ensemble)	-	-	68.361	RF	6.510	RDT
FailOn3Days	AUC	**0.979**	H2O AutoML (GBM)	0.795	IF	0.500	RF	0.764	DT
FailOn5Days	AUC	**0.971**	AutoGluon (Ensemble)	0.800	IF	0.529	RF	0.794	RF
FailOn7Days	AUC	**0.982**	TPOT (RF)	0.804	IF	0.581	RF	0.830	KNN
FailOn10Days	AUC	**0.988**	AutoGluon (Ensemble)	0.797	IF	0.563	RF	0.865	RF

DT - Decision Tree; IF - Isolation Forest; KNN - K-Nearest Neighbors; GBM - Gradient Boosting Machine; RF - Random Forest; RDT - Randomized Decision Trees

It should be noted that the human non-ML expert used a distinct preprocessing procedure, since it applied the One-Hot encoding to all categorical attributes (and not IDF for the high cardinality ones, as we adopted for the AutoML tools). However, the external ML expert used the same preprocessing adopted by the AutoML tools.

The comparison clearly favors the AutoML results for all predicted target variables. For regression, the non-expert modeling achieved an average error of 68.36 days, which was only better than Auto-Keras (which obtained an average MAE of 84.02). On the other hand, the best expert modeling result was an MAE of 6.51, which was only surpassed by three AutoML tools (AutoGluon, H2O AutoML, and Auto-Sklearn). As mentioned in Section 4.3.2, the AutoOneClass method was not applied to the regression target since it is only performs a binary classification.

For the binary classification task, all AutoML tools achieved results that can be considered excellent (AUC higher than 0.90). On the other hand, the non-expert modeling achieved slightly better results than a random model, while the expert's modeling achieved good results, with AUCs between 0.764 and 0.865. The AutoOneClass method also produced good predictive results, surpassing the human expert modeling in two of the four binary classification tasks (targets FailOn3Days and FailOn5Days).

These results suggest that supervised AutoML can be a valuable to automate the modeling phase when applying ML to PdM tasks. The usage of AutoML has several benefits, such as the ability to surpass human ML modeling, accelerate the creation of good ML models, and free the ML expert to focus on other essential ML phases, such as Data Understanding and Data Preparation. As for the proposed AutoOneClass method, the results demonstrate that OCC can also be used for binary PdM tasks, being

particularly valuable the labeling anomaly data is costly. While outperformed by some of the supervised AutoML tools, AutoOneClass has shown competitive results when compared with using human experts or AutoML tools focused only on DL.

4.5 Conclusions

Predictive Maintenance is a crucial industrial application that is being increasingly enhanced by the adoption of ML. However, most ML related works assume an expert ML model design that requires manual effort and time. In this chapter, we explore the potential of AutoML to automate PdM ML modeling. We used real-world data provided by a Portuguese software company within the domain of maintenance management to predict equipment malfunctions.

Our goal was to anticipate failures from several types of equipments (e.g., industrial machines), using two ML tasks: regression - to predict the number of days until the next failure of the equipment; and binary classification - to predict if the equipment will fail in a fixed amount of days (e.g, in three days).

For the ML modeling and training, we relied on two main approaches. First, we explored ten recent state-of-the-art supervised AutoML and AutoDL tools: Auto-Keras, Auto-PyTorch, Auto-Sklearn, AutoGluon, H2O AutoML, MLJar, PyCaret, rminer, TPOT, and TransmogrifAI. Second, we propose AutoOneClass, a novel AutoML method focused on an OCC that uses a GE optimization.

Several computational experiments were held, assuming five predictive tasks (one regression and four binary classifications). When comparing the supervised learning results, AutoGluon presented the best average results among the AutoML tools. The AutoOneClass results were also satisfactory, surpassing the AutoML tools focused on DL. The AutoML and AutoOneClass results were further compared with two human ML designs, performed by a non-expert and an ML expert. The comparison favored all AutoML tools, which provided better average results than both manual approaches. Overall, the best results were achieved by the supervised AutoML tools. However, the AutoOneClass surpassed the expert human modeling in two predictive targets and the performed OCC is quite useful when anomalous PdM labeling is costly. These results confirm the potential of the supervised AutoML modeling and the proposed AutoOneClass approach, which can automatically provide high-quality predictive models. This is particularly valuable for the PdM domain since industrial data can arise with a high velocity. Thus, the predictive models can be dynamically updated through time, reducing the data analysis effort.

Chapter 5

AutoOC: Automated Multi-objective Design of Deep Autoencoders and One-Class Classifiers using Grammatical Evolution

5.1 Research Context

This chapter outlines the research work developed during the fourth and final year of this doctoral program, whose focus was to further develop the initial version of the AutoML framework proposed in Chapter 4. This work resulted in AutoOC, an improved version of the AutoOneClass method. This new version of the framework addressed a specific gap in the previous research work by focusing exclusively on OCC ML algorithms. It was observed in the state-of-the-art that most AutoML tools target supervised learning tasks (e.g., classification, regression) and do not handle OCC. Additionally, the new version of the framework targeted multi-objective optimization using the NSGA-II algorithm, which maximizes the predictive performance of the OCC learners while minimizing their training time. The objective of this work was to address the efficiency objectives of the PhD project by generating lightweight ML models, a crucial aspect when dealing with real-world Big Data. The proposed AutoOC adopts two computationally efficient mechanisms to speed up the overall execution time: continuous sampling of training data and parallel fitness evaluation using multi-core processors. We carried out several computational experiments to evaluate the effectiveness of AutoOC on eight public datasets from various domains using two distinct validation modes (unsupervised and supervised). We compared the obtained results with a baseline state-of-the-art OCC algorithm and also with public human predictive results. This work is further described in the chapter and resulted in a publication in an international Q1 journal (Ferreira & Cortez, 2023).

5.2 Introduction

In recent years, NE has gained increasing attention as an interesting approach to optimize ANN models (Stanley et al., 2009). By adopting an EC method as the main search engine, NE automates the design of ANNs (e.g., hyperparameters, structure, weights), often finding good solutions in complex and high-dimensional neural modeling spaces while using a reasonable amount of computational resources. Indeed, NE has been successfully applied to a variety of tasks, including (Baymurzina et al., 2022; Cortez et al., 2020; Floreano et al., 2008): reinforcement learning, unsupervised learning, optimization, time series forecasting, supervised learning, and DL NAS.

With the worldwide growth of ML applications, there has been a growing interest in the usage of AutoML tools (Ferreira, Pilastri, Martins, et al., 2021). AutoML alleviates the modeling effort of non-ML experts by automating the search for the best ML algorithm and its hyperparameters. Several recently proposed AutoML tools are based on NE approaches (e.g., (Cetto et al., 2019; Miranda et al., 2022)). However, the vast majority of AutoML tools target a supervised learning (e.g., classification, regression) and do not handle an OCC.

Also known as unary classification, OCC can be viewed as a subclass of unsupervised learning, where the ML model only learns using training examples from a single class (Moya & Hush, 1996; Zola et al., 2021). This type of learning is valuable in diverse real-world scenarios where labeled data is non-existent, infeasible, or difficult (e.g., requiring a costly and slow manual class assignment), such as fraud detection (Seliya et al., 2021), cybersecurity (Arregoces et al., 2022), Predictive Maintenance (Ferreira et al., 2022) or industrial quality assessment (Ribeiro et al., 2022).

This work presents a novel application of NE to the field of OCC (as shown in Section 2.3.4) and contributes to the growing body of research on the use of GE for optimizing ML models. In particular, we present AutoOC, an AutoML method for OCC that is based on a GE. GE has been shown to be effective at optimizing the hyperparameters of ML models (Ryan et al., 2018). AutoOC performs a multi-objective optimization, using the NSGA-II algorithm to maximize the predictive performance of the OCC learners while minimizing their training time. The goal is to generate lightweight ML models, an important aspect when working with real-world Big Data that are common in One-Class Classification (OCC) tasks. Furthermore, AutoOC adopts two computationally efficient mechanisms to speed up the overall execution time (Pereira et al., 2021): a continuous sampling of training data and a parallel fitness evaluation by adopting multi-core processors. Moreover, the adopted grammar allows a flexible definition of which OCC learners are optimized. In this work, we particularly explore two AutoML grammar variants:

- NE - a pure evolutionary NAS approach that searches for the best model using two types of deep AEs, standard dense AE and VAE; and

- ALL - a more general AutoML that selects the best of five OCC learners, namely IF, LOF, OC-SVM, AE, and VAE.

Several computational experiments are held to evaluate the effectiveness of the two AutoOC variants, using eight public datasets and two distinct validation modes (unsupervised and supervised). The results are compared with a baseline IF and also with the best public supervised learning results from the OpenML platform (Vanschoren et al., 2013).

This study is organized as follows. Section 5.3 describes the problem formulation of the OCC optimization task. Next, Section 5.4 describes the proposed AutoOC method. Then, Section 5.5 presents the experimental results, including the datasets used, the experimental setup and the obtained results. Finally, Section 5.6 presents the main conclusions and discusses future work directions.

5.3 Problem Formulation

In this work, we address the CASH problem for OCC ML tasks. The CASH problem was first proposed in Thornton et al. (2013) and defines the problem of, given a search space of ML algorithms and its associated hyperparameters, selecting the best algorithm and fine-tuning its hyperparameters by using an optimization method (e.g., Bayesian optimization). The original proposal of the CASH focused on a supervised learning task, in particular classification algorithms. Similarly, most of the recent research works that approach the CASH problem are focused on a supervised learning (as described in Section 2.3.4).

Let $\mathcal{D}_{\text{train}} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ denote a training dataset with n unlabeled (the normal) examples, where $\mathbf{x}_i$ denotes a vector with several input attribute values. There is also a disjoint validation set $\mathcal{D}_{\text{valid}}$ with a length of m examples and that can assume two variants: unsupervised, $\mathcal{D}_{\text{valid}_U} = \{\mathbf{x}_{n+1}, ..., \mathbf{x}_{n+m}\}$; or supervised, $\mathcal{D}_{\text{valid}_S} = \{(\mathbf{x}_{n+1}, y_{n+1}), ..., (\mathbf{x}_{n+m}, y_{n+m})\}$, where y_i denotes a binary labeled output class (e.g., $y_i \in \{0, 1\}$). Let $\mathcal{A} = \{A^1, ..., A^k\}$ define a finite set of k OCC algorithms and $\Lambda = \{\Lambda^1, ..., \Lambda^k\}$ the respective hyperparameter search spaces. The CASH search space is defined by $\mathcal{S} = A^i_\lambda$, where A^i_λ denotes the usage of algorithm A^i with the hyperparameter values $\lambda \in \Lambda^i$ and $i \in \{1, ..., k\}$. A particular A^i_λ OCC algorithm is trained using the unlabeled training examples, namely the $\mathcal{D}_{\text{train}}$ dataset, generating the learning model $\mathcal{M}^i_\lambda$.

In this work, we assume multi-objective OCC CASH task, where an $\mathcal{O}$ optimization algorithm searches for the best A^*_λ combination that satisfies:

$$A^*_\lambda \in \underset{A^j \in \mathcal{A}, \lambda \in \Lambda^j}{argmin} \; (\mathcal{L}_1(A^i_\lambda, \mathcal{D}_{\text{valid}}), \mathcal{L}_2(A^i_\lambda, \mathcal{D}_{\text{train}})) \tag{5.1}$$

where $\mathcal{L}_1$ denotes a generalization error measured using the validation set and $\mathcal{L}_2$ represents the computational effort required to train the learning model ($\mathcal{M}^i_\lambda$).

5.4 Proposed Method: AutoOC

In this work, we propose the AutoOC method to solve the multi-objective CASH problem for OCC ML tasks. The algorithm search space $\mathcal{A}$ is composed of a maximum of five ($k = 5$) OCC learners, namely

$A = \{\text{IF,LOF,OC-SVM,AE,VAE}\}$ (see Section 5.4.3). The search for the best OCC algorithm and hyperparameters (O) is performed by a computationally efficient multi-objective that uses a GE and the NSGA-II algorithm that returns a set of best Pareto solutions $\mathcal{B} = \{A^1_\lambda, ..., A^p_\lambda\}$, where each A^i_λ combination is a non-dominated solution in terms of the $\mathcal{L}_1$ and $\mathcal{L}_2$ minimization objectives. As for the hyperparameter search spaces (Λ), they are defined by the adopted GE grammar (as detailed in Section 5.4.4).

GE is a biologically inspired evolutionary algorithm for generating computer programs. The algorithm was proposed by O'Neill and Ryan in 2001 (O'Neill & Ryan, 2001) and has been widely used in both optimization and ML tasks. GE can handle complex optimization problems with a large number of objectives and constraints. It can also handle continuous and discrete optimization problems, as well as problems with mixed variables. Indeed, GE has been shown to be effective in finding high-quality solutions in a relatively short time, compared to other optimization methods (Nyathi & Pillay, 2018). In GE, a set of programs is represented as strings of characters, known as chromosomes. The chromosomes are encoded using a formal grammar, which defines the syntax and structure of the programs. The grammar is used to parse the chromosomes and generate the corresponding programs, which are then evaluated using a fitness function. The fitness function measures the quality of the programs and is used to guide the evolution process toward better solutions.

There are two main reasons that make GE a suitable choice for our AutoML OCC search. Firstly, it can handle variable-length solution representations, which is useful when handling different types of OCC algorithms, where each algorithm contains its own hyperparameters. Secondly, and in contrast with other variable-length EC methods, such as Genetic Programming or Gene Expression Programming, it allows an easy customization of the OCC search space, since it is defined by a human-readable grammar. The AutoOC grammar employs up to $k = 5$ OCC methods and directly generates Python code. If needed, the grammar can be adapted to include any combination of the five base learners, additional hyperparameters, or even new OCC algorithms.

AutoOC assumes a multi-objective optimization by adopting the popular NSGA-II algorithm that was proposed in 2002 (Deb et al., 2002). The algorithm is based on the concept of non-dominance, which means that a solution is considered superior to another solution if it is not worse than the other solution in any objective and strictly better in at least one objective. The goal of NSGA-II is to find a set of non-dominated solutions, known as the Pareto front, which represents the trade-off between the different objectives. NSGA-II includes a crowding distance measure, which is used to preserve diversity among the solutions and avoid premature convergence. The algorithm has been widely used in various fields, including engineering, economics, and biology, and has shown promising results in a variety of multi-objective optimization problems (Coello et al., 2007).

In this study, we implemented a Pareto optimization approach to simultaneously minimize two objectives: generalization discrimination error ($\mathcal{L}_1$) and training time ($\mathcal{L}_2$). The resulting Pareto front contains a set of non-dominated solutions, each representing a trade-off between the two objectives. The rationale of this multi-objective approach is to allow for the selection of lightweight OCC models, even if they are

associated with a slightly lower performance. Indeed, reducing the computational training time is particularly valuable within the OCC domain, since most of the analyzed datasets are unlabeled and thus often rather large.

5.4.1 Acceleration Mechanisms and Objective Functions

As explained in Section 5.3, the training data $\mathcal{D}_{\text{train}}$ is composed only of data from one class ("normal"data). OCC typically involves a large set of unlabeled data, thus performing an evolutionary optimization in this domain is a computationally demanding task. In order to speed up the GE execution time, AutoOC adopts two recently proposed computationally efficient mechanisms (Pereira et al., 2021).

Firstly, AutoOC uses a periodic sampling mechanism, where each g generation of the GE optimization uses the random sample $\mathcal{D}^{g}_{train}$ that includes $s < n$ examples from the entire training dataset. For an example dataset with n =10,000 records and a sample size of s=2,500, each generation of the GE optimization will use s=2,500 randomly sampled records to train the OCC models. The sampling is applied to the entire dataset at the beginning of each generation and it is performed with replacement (similarly to the bagging ML ensemble method), meaning that a specific record can be chosen more than once. The reason for this approach is related to an acceleration of the total optimization time, since training the models on a small sample of a dataset will be faster than training all the individuals on an entire dataset, especially if the dataset has a huge number of records (e.g., millions of records). On the other hand, the fact that each generation uses a different set of examples will allow the optimization to avoid overfitting the training set since the training data is always different. Secondly, each A^{i}_{λ} solution is trained in a parallel manner, where a $\mathcal{M}^{i}_{\lambda}$ model is obtained by applying the A^{i}_{λ} algorithm to the $\mathcal{D}^{g}_{train}$ dataset. This means that, for each generation, more than one individual can be trained at the same time using different cores (processors). In practice, when the used machine has more cores than the population size, it is possible to train all the individuals at the same time. For each trained $\mathcal{M}^{i}_{\lambda}$ model, AutoOC stores the value of $\mathcal{L}_2(A^{i}_{\lambda}, \mathcal{D}^{g}_{train})$, which corresponds to the time elapsed to obtain $\mathcal{M}^{i}_{\lambda}$ when using a single core, in seconds. This $\mathcal{L}_2$ value corresponds to the second objective function, which guides the O search in terms of minimizing the OCC training computational effort.

AutoOC is primarily designed for anomaly detection tasks, where most examples are "normal"records. While the training only uses normal examples, the OCC predictive performance validation can be performed using two distinct setups (Ferreira et al., 2022): unsupervised validation, where the model performance is evaluated using only $\mathcal{D}_{\text{valid}_U}$ unlabeled data (e.g., through an anomaly score), or supervised validation, where there is access to a (often smaller) $\mathcal{D}_{\text{valid}_S}$ labeled validation set to assess the model performance by using supervised learning metrics, such as the popular AUC of the ROC curve classification measure (Fawcett, 2006).

All OCC models produce an anomaly score (S_j) for a particular $\mathbf{x}_j$ data example. The $\mathcal{M}^{i}_{\lambda}$ validation or test anomaly scores are first normalized within the $S_i \in [0, 1]$ range by applying a min-max normalization using the training data. Two relevant performance measures adopted in this work are the average anomaly

score ($\overline{S}$) and AUC:

$$\begin{aligned} \overline{S} &= \frac{1}{l}\sum_{j=1}^{l} S_j \\ AUC &= \int_0^1 ROC\, dTh \end{aligned} \tag{5.2}$$

where l denotes the length of the predicted data (e.g., $l = m$ for a validation set) and $Th \in [0, 1]$ is a threshold decision value, allowing to interpret the predicted anomaly class as positive if $S_i > Th$. The ROC curve plots the False Positive Rate (FPR) versus the True Positive Rate (TPR) for all threshold values. AUC is a popular binary classification measure of performance, providing two main advantages (Coelho et al., 2022). Firstly, quality values are not influenced by the presence of unbalanced data, which occurs in OCC tasks. Secondly, the AUC values can be easily interpreted as follows: 50% – performance of a random classifier; 60% - reasonable; 70% - good; 80% - very good; 90% - excellent; and 100% - perfect.

In this study, we explore the two OCC validation modes (supervised and unsupervised), which lead to two distinct fitness functions that measure OCC generalization error performance ($\mathcal{L}_1$, the first objective function). For the supervised validation, the generalization error performance (to be minimized), is defined as $\mathcal{L}_1(A^i_\lambda, \mathcal{D}_{\text{valid}_S}) = 1-$AUC. The lower the $\mathcal{L}_1$ value, the better will be the OCC AUC predictive performance. Under the unsupervised validation mode, labeled data (i.e., abnormal examples) is not available, making the computation of the AUC infeasible. Therefore, to select the best ML models, the average anomaly score ($\overline{S}$) is used as a proxy for the 1-AUC computation: $\mathcal{L}_1(A^i_\lambda, \mathcal{D}_{\text{valid}_U}) = \overline{S}$. The idea is that if a model produces a low anomaly score when trained on a large set of normal data, it should be capable of generating high anomaly scores for abnormal data, resulting in a satisfactory ROC curve. Nevertheless, it is important to note that to accurately benchmark the unsupervised validation scenario, labeled data was used in the test set, allowing the computation of ROC curves and AUC measures, which were then compared to those obtained using the supervised validation scenario. Table 26 summarizes the type of data used for each validation setup.

Table 26: Validation modes for AutoOC.

Validation Mode	**Training Set**	**Validation Set**	**Test Set**
Supervised	Unlabeled Data	Labeled Data	Labeled Data
Unsupervised	Unlabeled Data	Unlabeled Data	Labeled Data

5.4.2 Pseudo-code

The pseudo-code for our proposed AutoOC is illustrated in Algorithm 1. There are four main inputs, the training and validation sets ($\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{valid}}$, the sampling size ($s$), the maximum number of generations (G) and the population size (N_P). The search algorithm ($\mathcal{O}$) combines a GE with a multi-objective NSGA-II optimization. The GE elements are used to generate the initial population and breed new individuals (through crossover and mutation operators). As for the NSGA-II procedures, they enforce a simultaneous multi-objective search in terms of selecting interesting new population individuals and the best set of Pareto

solutions. Moreover, the two AutoOC acceleration mechanisms are implemented in lines 6 (periodic random sampling) and 7 (parallel execution of the training algorithm and computation of its validation measures). After G generations (the termination criteria), the search returns the best searched Pareto front of solutions ($\mathcal{B}$).

Algorithm 1 AutoOC pseudo-code.

1: **Inputs:** $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}}, \mathcal{S}, s, G, N_P$ ▷ Training and validation sets, search space, training sample size, maximum number of generations and population size
2: $\mathcal{B} \leftarrow \emptyset$ ▷ Initialize $\mathcal{B}$
3: $\mathcal{P}^0 \leftarrow create(\mathcal{S}, N_P)$ ▷ Initial GE population with $\{A_\lambda^1, ..., A_\lambda^{N_P}\}$ solutions
4: $g \leftarrow 0$
5: **while** $g < G$ **do** ▷ Cycle up to G generations
6: $\mathcal{D}_{\text{train}}^g \leftarrow sample(\mathcal{D}_{\text{train}}, s)$ ▷ Random sample of size s
7: $F^g \leftarrow evaluate(\mathcal{P}^g, \mathcal{D}_{\text{train}}^g, \mathcal{D}_{\text{valid}})$ ▷ Fitness values ($\mathcal{L}_1$ and $\mathcal{L}_2$) for $\mathcal{P}^g$
8: $\mathcal{P}^g \leftarrow \mathcal{P}^g \cup \mathcal{B}$ ▷ Add $\mathcal{B}$ to current population
9: $\mathcal{B} \leftarrow \emptyset$ ▷ Reinitialize $\mathcal{B}$
10: **if** $A_\lambda^i \in \mathcal{P}^g$ is a NSGA-II interesting (e.g., non-dominated) solution **then** ▷ Apply NSGA-II
11: $\mathcal{B} \leftarrow \mathcal{B} \cup A_\lambda^i$ ▷ Add A_λ^i to the set of best Pareto solutions
12: **end if**
13: $\mathcal{P}^{g+1} \leftarrow$ evolve($\mathcal{P}^g$) ▷ Apply GE crossover and mutation operators and NSGA-II selection
14: $g \leftarrow g + 1$ ▷ Increment g
15: **end while**
16:
17: **return** $\mathcal{B}$ ▷ Return best solutions (Pareto front of OCC models)

5.4.3 Base Learners

AutoOC uses up to five popular OCC algorithms: AEs, IF, LOF, OC-SVM, and VAEs. The AEs and VAEs were implemented through the Keras module of TensorFlow library (Martín Abadi et al., 2015), while IF, LOF, and OC-SVM used the Scikit-Learn framework (Pedregosa et al., 2011). Table 27 summarizes the five adopted base learners in terms of: the name of the (**Algorithm**), the base (**Framework**), used (**Version**), and (**API**) documentation reference. Additional details about the base learners are provided in Section 2.2.3.

Table 27: Characteristics of the base learners used by AutoOC.

Algorithm	Framework	Version	API
Isolation Forest (IF)	Scikit-Learn	1.2.0	(Scikit-Learn, 2022a)
Local Outlier Factor (LOF)	Scikit-Learn	1.2.0	(Scikit-Learn, 2022b)
One-Class SVM (OC-SVM)	Scikit-Learn	1.2.0	(Scikit-Learn, 2022c)
Autoencoder (AE)	TensorFlow	2.6.0	(TensorFlow, 2022a)
Variational Autoencoder (VAE)	TensorFlow	2.6.0	(TensorFlow, 2022b)

5.4.4 AutoOC Grammar

In this study, we used an open-source implementation of GE in Python (PonyGE2) to develop AutoOC. PonyGE2 (Fenton et al., 2017) allows for the use of Python BNF (PyBNF), which enables the inclusion of Python code in the production rules. To build AutoOC, a PyBNF grammar was developed to tune the hyperparameters of the OCC algorithms described in Section 5.4.3. The use of PyBNF allowed for the generation of Python code snippets that enabled GE to produce various types of ML models. For example, the IF, LOF, and OC-SVM grammars were implemented by creating the corresponding Scikit-Learn class and adding the hyperparameters as terminals and non-terminals.

We note that the proposed grammar includes all IF, LOF, and OC-SVM hyperparameters that were available in the consulted Scikit-Learn documentation (see Table 27). The process was different for the AEs and VAEs, as the TensorFlow API requires the definition of a variable number of layers. To address this, the grammar was designed to generate only the encoder: first, an input layer with the same number of nodes as the number of attributes in the dataset is generated, followed by a variable number of hidden layers. Given that in a typical AE or VAE the subsequent encoder layers have fewer nodes than the previous layer, the layer nodes were defined as a percentage (between 0% and 100%) of nodes in the previous layer rather than a fixed number. Two auxiliary functions, (`get_ae_from_encoder` and `get_vae_from_encoder`), were also defined to translate the generated phenotype into functional TensorFlow AEs and VAEs. The decoder, which is symmetrical to the encoder, was not included in the grammar. Besides the ANN structure, DL architectures include a large number of additional hyperparameters. In order to reduce the search space, using modeling knowledge from previous OCC works (e.g., (Coelho et al., 2022; Ribeiro et al., 2022)) we fixed some choices, such as the usage of the MAE measure as the loss function for both AE and VAE and usage of Batch Normalization layers for AE. We also restricted the search space for some hyperparameters. For instance, only two optimizers are explored to adjust the AE weights (RMSprop and Adam). Moreover, while the analyzed TensorFlow version provides up to 16 activation functions, the proposed grammar only searches for the best of eight of these functions (e.g., ReLU). Nevertheless, in future works and if needed, the grammar can be easily adapted to include other DL hyperparameter choices.

The proposed grammar of AutoOC defines the OCC search space ($\mathcal{S}$) and is flexible enough to allow the usage of any combination of the five base learners (Section 5.4.3) or even include other OCC algorithms. In this work, we empirically study the effect of two AutoOC variants: NE – assuming only the deep AE and VAE base learners (thus $k = 2$), working as a pure NAS optimization; ALL – where all $k = 5$ five base learners are used during the optimization, working as a more general AutoML OCC search. The developed PyBNF grammar for the "ALL" mode is shown in Fig. 16. The grammar for the "NE" mode follows a similar logic, using only the AEs and VAEs entries.

```
<response> ::= <autoencoder> | <iforest> | <lof> | <ocsvm> | <vae>

<autoencoder> ::=  encoder = Sequential(){::}
                   encoder.add(Input(shape=(input_shape,), name="'input'")){::}
                   <hidden_layers>{::}
                   <latent_space>{::}
                   model = get_ae_from_encoder(encoder){::}
                   model.add(Dense(input_shape, activation=<activation>, name="'output'")){::}
                   model.compile(<optimizer>, "'mae'")
<hidden_layers> ::= <Dense>{::} | <Dense>{::}<Dense>{::} | <hidden_layers><Dense>{::} | <Dense>{::}<extra>{::}
<Dense> ::= encoder.add(Dense(units = <percentage>, activation = <activation>))
<activation> ::= "'relu'" | "'sigmoid'" | "'softmax'" | "'softplus’" | "'tanh'" | "'selu'" | "'elu'" | “’exponential'"
<latent_space> ::= encoder.add(Dense(units = <percentage>, activation = <activation>, name="'latent'"))
<extra> ::= encoder.add(Dropout(rate=0.<dropout_digit>)){::} | encoder.add(BatchNormalization()){::}
<dropout_digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<percentage> ::= 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100
<optimizer> ::= "'RMSprop'" | "'Adam'"


<iforest> ::= model = IsolationForest(n_estimators=<estimators>, contamination=<contamination>, bootstrap=<bootstrap>, n_jobs =-1)
<estimators> ::= <digit><estimators> | <digit>
<estimators_digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<contamination> ::= "'auto'" | 0.<contamination_digits>
<contamination_digits> ::= 1 | 2 | 3 | 4 | 5
<bootstrap> ::= "True" | "False"


<ocsvm> ::= model = OneClassSVM(kernel=<kernel>, degree=<degree>, gamma=<gamma>, shrinking=<shrinking>)
<kernel> ::= "'linear'" | "'poly'" | "'rbf'" | "'sigmoid'"
<degree> ::= <digit><degree_digit> | <digit>
<degree_digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<gamma> ::= "'scale'" | "'auto'"
<shrinking> ::= "True" | "False"


<lof> ::= model = LocalOutlierFactor(n_neighbors = <n_neighbors>, algorithm = <algorithm>, leaf_size = <leaf_size>,
                   metric = <metric>, contamination = <contamination>, novelty = "True", n_jobs =-1)
n_neighbors> ::= <single_digit> | <single_digit><digit> | <single_digit><digit><digit>
<algorithm> ::= "'ball_tree'" | "'kd_tree'" | "'brute'" | "'auto'"
<leaf_size> ::= <single_digit> | <single_digit><digit>
<metric> ::= "'minkowski'" | "'cityblock'" | "'chebyshev'" | "'euclidean'" | "'l1'" | "'l2'"


<vae> ::=     encoder = Sequential(){::}
                   encoder.add(Input(shape=(input_shape,), name="'input'")){::}
                   <hidden_layers>{::}
                   <z_mean>{::}
                   <z_log_var>{::}
                   <z>{::}
                   model = get_vae_from_encoder(encoder){::}
                   model.add(Dense(input_shape, activation=<activation>, name="'output'")){::}
                   model.compile(<optimizer>, "'mae'")
<z_mean> ::= encoder.add(Dense(units = <percentage>, activation = <activation>, name="'z_mean'"))
<z_log_var> ::= encoder.add(Dense(units = <percentage>, activation = <activation>, name="'z_log_var’"))
<z> ::= Lambda(sample, output_shape=(<percentage>,), name='z')
```

Figure 16: The adopted PyBNF grammar (for the full “ALL” search space representation mode).

5.5 Experimental Results

5.5.1 Datasets

A total of eight public domain datasets (Table 28) were retrieved from OpenML (Vanschoren et al., 2013), an open platform for sharing datasets and ML experiments. As selection criteria, we opted to select binary classification tasks from distinct application domains (e.g., banking, telecommunications) and reflecting different numbers of instances (**Rows**), categorical (**Categorical Columns**) and numerical attributes (**Numerical Columns**), and output target class balancing (**Class Balancing**). We particularly selected datasets with a clear distinction between the two classes, where the majority class could be considered as "normal" and the minority class as an "anomaly" state. Table 28 also details the name of the dateset (**Dataset**) and the unique OpenML identifier (**OpenML ID**).

Table 28: Description of the selected OpenML datasets.

Dataset	OpenML ID*	Rows	Categorical Columns	Numerical Columns	Class Balancing ("normal"/"anomaly")
Bank Marketing	1461	45,211	16	9	88%/12%
Churn	40701	5,000	4	16	86%/14%
Credit Card	44235	284,807	0	30	99%/1%
EEG	1471	14,980	0	14	55%/45%
Mushroom	24	8,124	23	0	52%/48%
Nomao	1486	34,465	30	89	71%/29%
Phoneme	1489	5,404	0	5	70%/30%
Spambase	44	4,601	0	57	60%/40%

*The datasets can be retrieved by entering their OpenML unique identifier (ID) at the following URL: http://www.openml.org/search?type=data&id=ID.

Since AutoOC focuses on algorithm selection and hyperparameters, the datasets need to be preprocessed before feeding them into the OCC base learners. In order to achieve a fair comparison, the same fixed data preprocessing is applied to all datasets.

Since none of the base learners deals with data attributes of type String, we encoded all String attributes into numerical types. For categorical attributes with low cardinality (ten levels or fewer), we applied the popular one-hot encoding. For categorical columns with missing values, we replaced the missing values with zero, which is treated as a numeric code value for the "unknown" level. As for high cardinality categorical attributes, the one-hot transform produces a large number of binary inputs, which highly affects the computational performance (in terms of both memory and processing time). Thus, for these attributes, we employed instead the IDF technique, available in the Python CANE module (Matos et al., 2022), which converts a categorical column into a numerical column of positive values based on the frequency of each attribute level. IDF uses the function $f(x) = log(n/fx)$, where n is the length of x and $f(x)$ is the frequency of x. This technique has the advantage of generating just one numeric column

for each attribute, thus reducing the ML computational effort. For the remaining attributes of Integer and Float types, we applied a z-score standardization (Hastie et al., 2009), which results in a new scale with a mean of zero and standard deviation of one. The missing values in numerical columns were also replaced with the mean value for that column (mean imputation).

5.5.2 Experimental Setup

All experiments were run on an Intel Xeon 1.70 GHz server with 56 cores and 64 GB of RAM, without a GPU. When running AutoOC, we stored two types of time elapsed times (in seconds), the overall GE execution time and the training time required by the OCC algorithms. To assess the performance of AutoOC, we followed an approach based on the benchmark in (Ferreira, Pilastri, Martins, et al., 2021). We divided the datasets into 10 folds to obtain an external cross-validation, which is used to get test (unseen) data that allows measuring the predictive generalization performance of the selected OCC model. As for the training data, it is further split by applying an internal and random holdout split, where 75% of the data is used for fitting purposes ($\mathcal{D}^{g}_{\text{train}}$) and the remaining 25% is used for validation purposes ($\mathcal{D}_{\text{valid}}$).

To evaluate the predictions on the test set from the external 10-fold validation, we employed the AUC analysis of the ROC curve (Fawcett, 2006). The obtained results are aggregated by computing the median of the evaluation measures across the 10 external folds and their respective 95% confidence intervals based on the nonparametric Wilcoxon test (Hollander et al., 2013), to determine the statistical significance of the experiments.

5.5.3 AutoOC Results

For each dataset, we executed four AutoOC experiments, with two base learner configurations (NE and ALL) and both validation modes (supervised and unsupervised). Since it is unfeasible to evaluate every possible combination of the GE optimization parameters, we fixed some of these values using reasonable assumptions and some preliminary experiments performed using other OCC datasets. The summary of the different parameters used in the experimental evaluation is shown in Table 29. All experiments were executed with an initial random generated population of 20 individuals and 100 generations. Also, for the GE parameters of crossover and mutation, we adopted the default PonyGE2 values: Variable One-point crossover (selection of a different point on each parent genome for crossover to occur) with a crossover probability of 75%; and Int Flip Per Codon mutation (random mutation of every individual codon in the genome) with a mutation probability of 100%. Additionally, we applied both acceleration mechanisms described in Section 5.4.1, using a periodic random sampling, performed in each generation and applied to all the population individuals, of n=2,500 records and parallel training.

Table 30 presents the results obtained by AutoOC on the eight open-source datasets described in Section 5.5.1. The table shows the median test set results of the external 10 folds and the respective confidence intervals for the predictions (**Median AUC**) and the efficiency (**Median Training Time**, in

Table 29: GE parameters used for the experiments.

Parameter	**Used Values**
Population Size (N_P)	20
Number of Generations (G)	100
Crossover	Variable One-point with 75% crossover probability (PonyGE2 default)
Mutation	Int Flip Per Codon with 100% mutation probability (PonyGE2 default)
Base Learners Setup	ALL (used algorithms: AE, IF, LOF, OC-SVM, VAE) NE (used algorithms: AE, VAE)
Optimization Type	Multi-objective (NSGA-II)
Predictive Objective	Minimize validation 1-AUC ($\mathcal{L}_1$ for supervised validation) Minimize validation average anomaly score $\overline{S}$ ($\mathcal{L}_1$ for unsupervised validation)
Efficiency Objective	Minimize training time ($\mathcal{L}_2$)
Validation Type	Supervised Unsupervised
Sampling size	s=2,500
Parallel Training	True

seconds). It is worth noting that, since these experiments apply a multi-objective approach, each external fold generates more than one optimal model per fold (all that belong to the Pareto front). Thus, we divided the AUC and training time median results into three columns each. The predictions (**Pred.**) column only considers the individuals from the Pareto front with the best predictive objective score; the **Speed** column considers the Pareto front individuals with the least training time (efficiency objective, in seconds); the column **Pareto** considers all the individuals belonging to the Pareto front. Table 30 also shows the median time needed for the GE optimization (**Median GE Time**) with confidence intervals, the type of validation (**V**) that was used, and which base learner setup was considered (**BL**). For the best results of each dataset (AUC, training time, and GE time; values highlighted using a **boldface** font), we apply the nonparametric Wilcoxon test for measuring statistical significance.

Regarding the predictive performance, the ALL mode with supervised validation achieved the best median AUC on the test set for: seven of the eight datasets when considering predictive power; three datasets when considering the training speed; and six datasets when considering the entire Pareto front. In these scenarios, the supervised ALL achieved a median of 11.0 AUC **Percentage Points (pp)** higher than the respective second-best configuration for predictive mode, 2.0 pp for speed mode, and 5.5 pp for the Pareto mode. An interesting result was obtained by the Credit Card dataset, achieving the best predictive results exclusively with NE approaches, namely with the supervised validation mode. This setup obtained, on median, 1.0 AUC pp higher than the second-best setup for predictive mode, 9.0 pp for speed mode, and 6.0 pp for the Pareto mode.

When considering the total GE execution time, the unsupervised ALL approach required a median value of 702 s across all datasets, followed by supervised ALL (716 s), supervised NE (2,009 s), and unsupervised NE (2,025 s). These results can be explained by the training time required by the deep ANNs

Table 30: AutoOC experimental results (best values for each measure in **bold**).

Dataset	BL	V*	Median AUC			Median Training Time			Median GE Time
			Pred.	Speed	Pareto	Pred.	Speed	Pareto	
Bank	ALL	S	**0.72**[a]±0.01	0.52±0.02	**0.63**[a]±0.01	**0.29**[c]±0.20	**0.01**[c]±0.00	**0.05**[a]±0.03	**740**[c]±57
Bank	ALL	U	0.62±0.03	0.55±0.04	0.58±0.01	0.41±0.02	0.01±0.00	0.11±0.03	758±34
Bank	NE	S	0.62±0.01	0.56±0.00	0.59±0.00	8.48±1.59	1.55±0.09	3.98±0.74	2,002±102
Bank	NE	U	0.65±0.02	**0.58**[b]±0.03	0.60±0.02	9.46±0.86	3.27±0.65	6.14±0.43	1,505±44
Churn	ALL	S	**0.75**[a]±0.01	**0.56**[b]±0.02	**0.65**[a]±0.01	**0.01**[a]±0.02	**0.01**[c]±0.00	**0.01**[a]±0.00	691±81
Churn	ALL	U	0.62±0.03	0.55±0.01	0.59±0.01	0.27±0.02	0.01±0.00	0.08±0.01	**645**[c]±38
Churn	NE	S	0.63±0.01	0.55±0.00	0.60±0.01	7.13±1.24	1.33±0.10	3.47±0.25	1,657±29
Churn	NE	U	0.53±0.01	0.52±0.02	0.53±0.00	7.50±0.82	2.12±0.39	3.92±0.22	1,389±24
Credit	ALL	S	0.92±0.00	0.80±0.08	0.88±0.01	**0.13**[a]±0.04	**0.01**[c]±0.00	**0.04**[a]±0.02	**949**[c]±125
Credit	ALL	U	0.97±0.09	0.84±0.01	0.89±0.04	0.48±0.04	0.01±0.00	0.15±0.02	1,191±448
Credit	NE	S	**0.98**[e]±0.00	**0.93**[a]±0.00	**0.95**[a]±0.00	7.25±0.81	1.19±0.08	3.56±0.54	2,211±202
Credit	NE	U	0.91±0.10	0.89±0.01	0.90±0.02	10.79±1.28	2.54±0.58	5.31±0.47	4,208±1430
EEG	ALL	S	**0.68**[a]±0.03	0.51±0.01	**0.59**[a]±0.02	**0.22**[c]±0.43	0.01±0.00	**0.05**[c]±0.15	**617**[c]±230
EEG	ALL	U	0.56±0.01	**0.52**[b]±0.02	0.53±0.03	0.27±0.12	**0.01**[c]±0.00	0.06±0.02	645±41
EEG	NE	S	0.52±0.02	0.49±0.02	0.51±0.01	5.53±2.45	1.36±0.12	2.82±0.81	3,798±965
EEG	NE	U	0.51±0.00	0.51±0.00	0.51±0.01	7.99±1.32	1.88±0.13	3.95±0.21	1,985±56
Mushroom	ALL	S	**1.00**[d]±0.00	0.62±0.13	0.81±0.05	0.27±0.06	**0.01**[c]±0.00	0.06±0.02	1,057±113
Mushroom	ALL	U	0.58±0.06	0.50±0.10	0.58±0.02	**0.24**[c]±0.04	0.01±0.00	**0.03**[a]±0.01	**998**[c]±90
Mushroom	NE	S	0.99±0.04	0.82±0.20	0.87±0.02	9.22±1.46	2.08±0.15	3.99±0.23	2,016±1508
Mushroom	NE	U	0.99±0.04	**0.83**[f]±0.02	**0.91**[a]±0.02	9.46±1.35	2.62±0.19	5.31±0.49	2,041±31
Nomao	ALL	S	**0.83**[a]±0.02	0.62±0.06	**0.73**[a]±0.02	**0.01**[a]±0.00	**0.01**[c]±0.00	**0.01**[a]±0.00	**885**[c]±115
Nomao	ALL	U	0.63±0.05	0.59±0.02	0.62±0.09	0.46±0.09	0.01±0.00	0.07±0.02	905±58
Nomao	NE	S	0.70±0.01	0.50±0.01	0.63±0.01	5.62±1.58	1.89±0.14	3.08±0.63	1,901±67
Nomao	NE	U	0.68±0.02	**0.67**[a]±0.02	0.68±0.03	7.48±0.48	3.07±0.37	5.29±0.27	2,370±266
Phoneme	ALL	S	**0.74**[a]±0.01	**0.57**[b]±0.06	**0.68**[a]±0.01	**0.01**[a]±0.01	**0.01**[c]±0.00	**0.01**[a]±0.00	654±84
Phoneme	ALL	U	0.63±0.02	0.53±0.03	0.60±0.02	0.22±0.05	0.01±0.00	0.07±0.02	**634**[c]±81
Phoneme	NE	S	0.62±0.01	0.52±0.01	0.58±0.01	13.98±5.47	1.27±0.02	4.24±0.78	2212±88
Phoneme	NE	U	0.56±0.01	0.51±0.02	0.53±0.02	6.73±1.63	1.38±0.11	3.26±0.39	2192±134
Spambase	ALL	S	**0.81**[a]±0.01	**0.62**[b]±0.07	**0.73**[a]±0.01	**0.12**[a]±0.13	0.01±0.00	**0.03**[a]±0.03	**608**[c]±111
Spambase	ALL	U	0.68±0.04	0.59±0.00	0.63±0.04	0.29±0.02	**0.01**[c]±0.00	0.07±0.01	618±51
Spambase	NE	S	0.62±0.01	0.50±0.01	0.57±0.00	9.76±0.92	1.49±0.06	4.47±0.28	1,979±40
Spambase	NE	U	0.73±0.01	0.60±0.04	0.65±0.01	13.15±1.43	3.29±0.57	7.28±0.55	2,009±51

* Validation mode: S - Supervised; U - Unsupervised.
[a]Statistically significant (p-value < 0.05) under a pairwise comparison when compared with all the other setups.
[b]Statistically significant (p-value < 0.05) under a pairwise comparison when compared with none of the other setups.
[c]Statistically significant (p-value < 0.05) under a pairwise comparison when compared with the setups: Supervised NE and Unsupervised NE.
[d]Statistically significant (p-value < 0.05) under a pairwise comparison when compared with the setups: Unsupervised ALL.
[e]Statistically significant (p-value < 0.05) under a pairwise comparison when compared with the setups: Supervised ALL.
[f]Statistically significant (p-value < 0.05) under a pairwise comparison when compared with the setups: Supervised ALL and Unsupervised ALL.

(either a traditional AE or a VAE), which is higher when compared with the other base learners. In effect, both ALL setups (supervised and unsupervised) tend to produce lightweight OCC models, presenting median training time values always lower than one second, and most of the times being only 0.01 s. In contrast, the setups with the NE variant present median training times ranging from 1.19 s and 13.98 s,

with a median value of 3.99 s. Nevertheless, the total GE execution time results back the proposed AutoOC as a computationally efficient tool to model large OCC datasets. For instance, for the largest dataset (Credit Card, with around 285,000 examples), and when adopting the supervised validation mode, the ALL and NE variants only require a median GE optimization time of 949 s (around 16 minutes) and 2,211 s (around 37 minutes). In Section 5.5.4, we further compare these Credit execution time results (using the sampling mechanism) with a GE that uses all training data (no sampling).

To further compare the obtained AutoOC results, we analyzed the Pareto fronts from the test set results. Given that each experiment is composed of ten test sets (one for each external fold), we aggregate the distinct Pareto fronts from each experiment. Inspired by the ROC curve vertical aggregation (Fawcett, 2006), we aggregate the results vertically. To facilitate the visual analysis, in all Pareto front graphs shown in this study, we assume the -AUC minimization objective on the x-axis and the training time minimization objective on the y-axis. Thus, the ideal point corresponds to the bottom left corner of the Pareto graphs. For different values of -AUC, we estimate the Wilcoxon median training time and the respective 95% confidence intervals. The obtained median curves are presented in Fig. 17. The figure shows that for the ALL setups, the results are usually close in terms of training time, presenting differences that depend on the dataset but that tend to be small. As for the predictive performance, the supervised ALL tends to produce better AUC scores (e.g., Churn, EEG, Mushroom, Nomao, Phoneme, Spambase). As for the NE setups, the results usually present higher training times than the ALL setups. Moreover, the 95% confidence intervals usually do not overlap with the ALL setups, showing statistically significant differences.

5.5.4 Credit Card Dataset Results

For more detailed results, we present in this section additional analyses of one of the datasets used in the experiments. We chose the Credit Card dataset to perform these analyses for two main reasons. Firstly, this dataset is a very accurate representation of a typical OCC learning scenario, since it has a large number of examples (284,807) and presents a huge unbalance between classes (with more than 99% of examples belonging to the “normal” class). Second, it is among the datasets that obtained the best experimental predictive results in Table 30.

The first Credit Card analysis is related to the hypervolume, assuming a reference point of (AUC=0; maximum training time = 15 s). As a demonstration, we selected the first fold for each of the four experiments performed on the Credit Card dataset to evaluate the hypervolume evolution across the GE generations. For each generation, we computed the median hypervolume value of the current Pareto-optimal front. Fig. 18 presents the evolution of the hypervolume measure (in percentage, y-axis) through the 100 generations of the GE optimization (x-axis). The figure includes two plots, one for each validation mode, for a better comparison since different predictive objectives are being considered in each validation mode (AUC for supervised mode and anomaly score for unsupervised mode). The figure shows a fast hypervolume growth in the first 10 generations, even though it continues to increase until the end of the optimization, but at a lower rate. Three of the four curves present a period without significant improvements

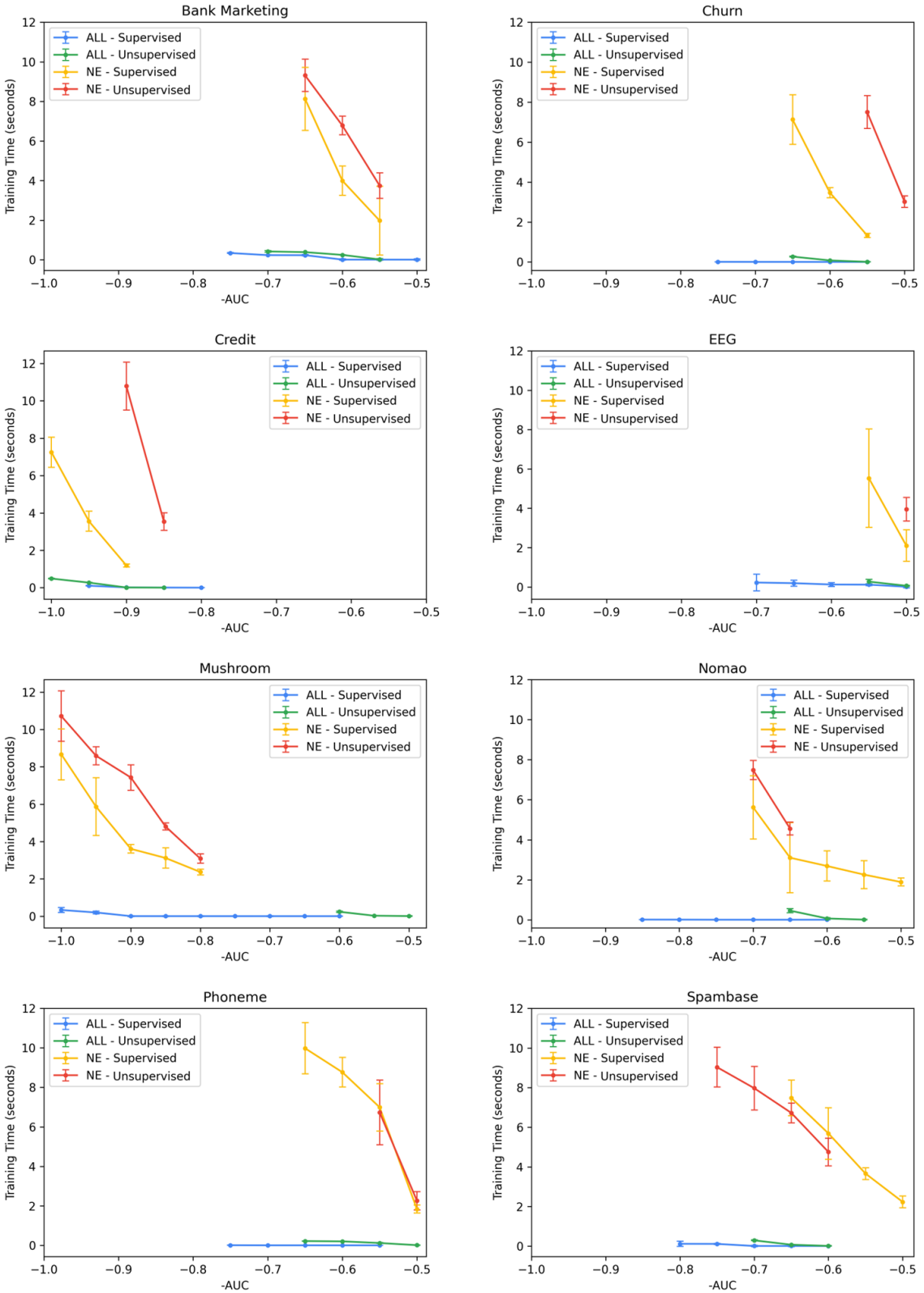


Figure 17: AutoOC experimental results (points denote the Wilcoxon median values and whiskers represent the respective 95% confidence intervals).

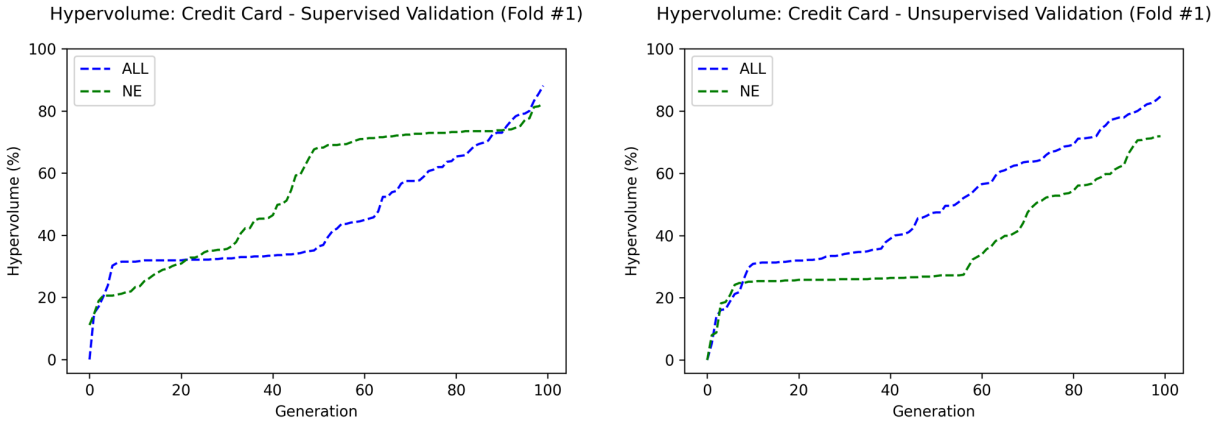


Figure 18: Hypervolume (y-axis, in %) generation evolution (x-axis) for one fold of the Credit Card experiments.

in the hypervolume in the first half of the optimization process (until generation 50). It is also worth noting that the experiments that used the ALL mode achieved a better final hypervolume percentage than the respective NE experiment. This can be explained by the fact that, even though the ALL mode presented a lower predictive performance than NE, it was able to generate individuals with much lower training time.

Table 31 provides an additional analysis of the Credit Card dataset experiments regarding the composition of the Pareto front. For each of the four setups (two base learner setups and validation modes), the table details the median number of individuals on the Pareto front by each type of base learner and in total. The table shows that the ALL setups presented a median number of Pareto front individuals lower than the

Table 31: Median number of individuals per base learner on the Pareto Front of the Credit Card dataset experiments.

Dataset	**Base Learner Setup**	**Validation Mode**	**Median Number of Individuals**					
			Pareto Front	**IF**	**LOF**	**OC-SVM**	**AE**	**VAE**
Credit Card	ALL	Supervised	6	4	3	1	1	0.5
	ALL	Unsupervised	6.5	3.5	1	1.5	1	1
	NE	Supervised	9	-	-	-	6	4.5
	NE	Unsupervised	12.5	-	-	-	6	7

NE setup. For the ALL setup, the most common base learner was IF, followed by LOF, and OC-SVM. Both AEs and VAEs are represented in the Pareto curve with a median number of one or fewer individuals. Both NE setups present a larger median number of individuals on the Pareto front (9 and 12.5). Regarding the presence of the base learners, the division between AE and VAE is relatively balanced. As an example, Fig. 19 shows the Pareto front of one of the folds of each of the four Credit Card experiments, detailing the type of base learner from each point, represented by the initials of the respective base learner.

To study the effect of the periodic sampling mechanism, we replicated the AutoOC experiments for

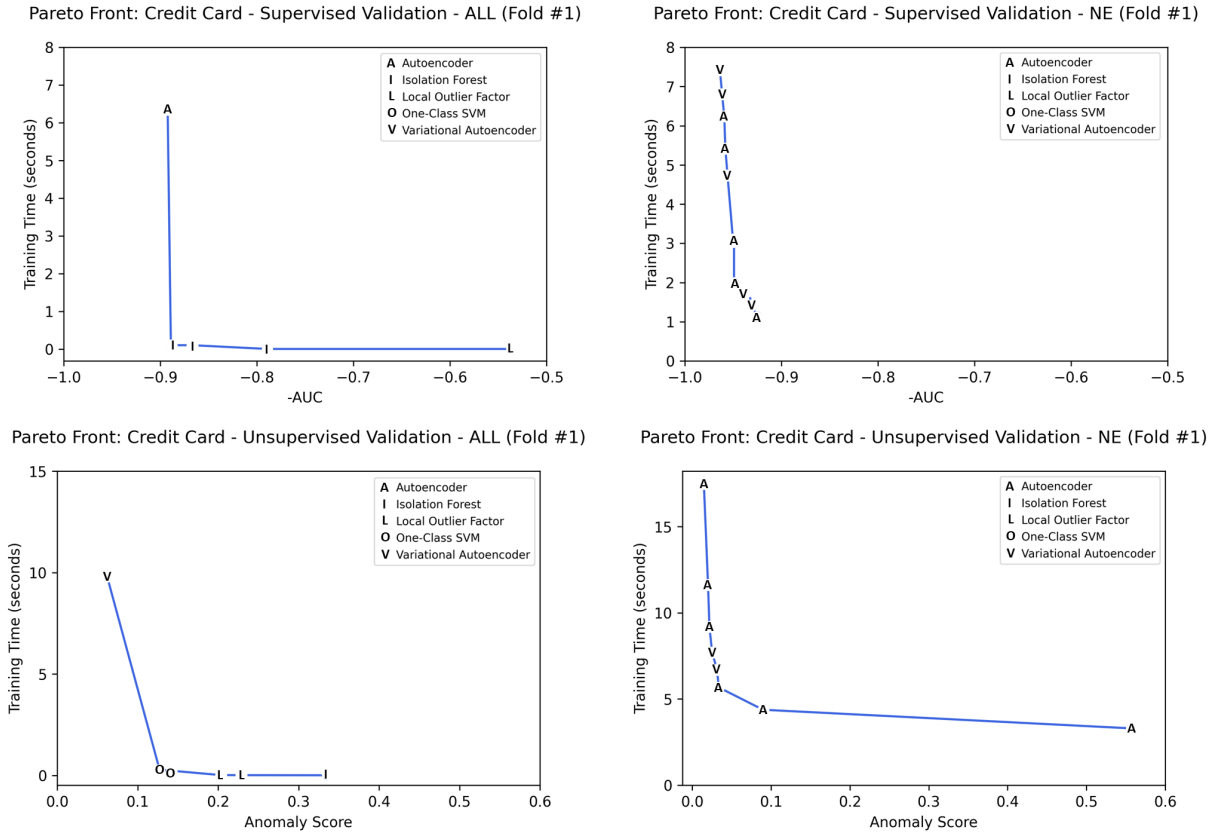


Figure 19: Pareto curves for one fold of the Credit Card experiments. Each Pareto front point is denoted by the initial of the respective base learner.

the Credit Card dataset using the full dataset (284,807 rows). Table 32 shows the results obtained on the Credit Card dataset using sampling with n=2,500 (also shown in Table 30) and using the full dataset (without sampling). Regarding the optimization time, the results clearly show that without sampling a

Table 32: Comparison of AutoOC results for the Credit Card dataset using the sampling mechanism and the full dataset (best values for each measure in **bold**).

Sampling Mode	BL	V*	Median AUC			Median Training Time			Median GE Time
			Pred.	Speed	Pareto	Pred.	Speed	Pareto	
Sampling	All	S	0.92±0.00	0.80±0.08	0.88±0.01	**0.13**±0.04	**0.01**±0.00	**0.04**±0.02	**949**±125
Sampling	All	U	0.97±0.09	0.84±0.01	0.89±0.04	0.48±0.04	0.01±0.00	0.15±0.02	1,191±448
Sampling	NE	S	**0.98**±0.00	**0.93**±0.00	**0.95**±0.00	7.25±0.81	1.19±0.08	3.56±0.54	2,211±202
Sampling	NE	U	0.91±0.10	0.89±0.01	0.90±0.02	10.79±1.28	2.54±0.58	5.31±0.47	4,208±1430
No Sampling	All	S	0.95±0.01	0.87±0.02	0.89±0.02	**38.88**±3.12	2.04±0.41	**23.75**±1.50	17,274±1,026
No Sampling	All	U	0.98±0.01	0.89±0.03	0.91±0.01	45.41±5.95	**1.97**±0.35	27.28±3.78	**16,756**±970
No Sampling	NE	S	**0.99**±0.00	0.94±0.01	**0.96**±0.00	421.57±62.24	240.98±22.37	335.81±39.81	38,391±2,489
No Sampling	NE	U	0.98±0.01	**0.95**±0.00	0.96±0.01	393.70±44.92	256.01±15.70	355.65±35.43	36,431±980

* Validation mode: S - Supervised; U - Unsupervised.

substantially higher computational effort is required (the increase is between ×8 and ×18). Similarly, the OCC training time of the individuals was also much higher when using the full dataset. In some cases, it

was 200 times higher than the respective experiment with sampling. As for the predictive results, there is only a rather small improvement when adopting the full dataset (e.g., around 3 pp for the Pareto individuals). Given that OCC tasks are often associated with Big Data and several real-world applications tend to require lightweight ML models, the results from Table 32 do value the proposed sampling mechanism.

5.5.5 Comparison with a Baseline Method and a Supervised Gold Standard

In a last empirical comparison, we contrast the best AutoOC results with a default (not tuned) IF (also trained as AutoOC with n=2,500 random samples) and the best public OpenML results. For each dataset, we show the best median AutoOC AUC score (column **Pred.** from Table 30), the median score obtained by the baseline IF, and the best result published in OpenML (including the AUC score, the used algorithm name, and the number of human ML attempts, described as "runs" in OpenML). It is worth mentioning that this comparison should be viewed with some caution. Firstly, we only compare the predictive performance AUC results and not other measures targeted by AutoOC, such as the total execution time or training time of the ML models. Secondly, AutoOC is fully automated and the best OpenML results were obtained after a large number of ML human expert modeling trials (ranging from 5,463 to 416,606). Thirdly, the OpenML results adopt a particular data preprocessing method and a supervised learning using the complete training datasets. Fourthly, we do not know the exact validation and testing procedures adopted by the OpenML modeling attempts. Thus, rather than assuming an ideal ML comparison, we use the best OpenML results as a "gold standard", denoting a proxy to the upper limit of the best empirical predictive results that can be achieved when using a human expert supervised learning modeling. The results are shown in Table 33.

Table 33: Comparison of the best AutoOC results with a baseline IF and best OpenML public results.

Dataset	**Best Results**				
	AutoOC	**IF**	**OpenML**		
	Score	**Score**	**Score**	**Algorithm**	**Runs**
Bank Marketing	0.723	0.546	0.938	XGB	40,465
Churn	0.746	0.603	0.932	GBM*	5,463
Credit Card	0.976	0.883	0.942	GBM*	416,606
EEG	0.675	0.505	0.998	SVM*	97,277
Mushroom	1.000	0.782	0.998	SVM	12,556
Nomao	0.830	0.668	0.953	DT	32,749
Phoneme	0.743	0.510	0.971	AdaBoost*	113,799
Spambase	0.806	0.524	0.989	XGB	58,350

*Algorithm used in a pipeline (with one or more preprocessing steps).

When comparing the AutoOC results with the baseline IF, it is possible to verify that the best AutoOC results always achieved a median AUC higher than the IF. The differences between the best AutoOC and IF results ranged from 9 pp and 28 pp, with a median difference of 17 pp. As for the comparison with the best public OpenML results, the supervised human modeling obtains a median overall AUC of 0.96, which is around 0.18 pp higher when compared with the AutoOC (median AUC of 0.78). While these results were expected, it should be highlighted that most best AutoOC results are of quality, obtaining a good discrimination (AUC>70%) for three datasets (Bank Marketing, Churn, and Phoneme), a very good predictive performance (AUC>80%) in two cases (Nomao and Spambase), and an excellent discrimination (AUC>90%) in two datasets (Credit Card and Mushroom). We particularly highlight the two excellent AUC results that even outperformed the best OpenML public results. Indeed, this corresponds to a high-quality AutoOC performance behavior, the best OpenML supervised results were obtained after 12,556 (Mushroom) and 416,606 (Credit Card) human attempts.

5.6 Conclusions

In this work, we presented AutoOC, which consists of a computationally efficient GE to automate the design of lightweight One-Class Classification Machine Learning models. We particularly explore two AutoOC variants: a pure Neuroevolution that evolves two types of Deep Learning Autoencoder, standard dense Autoencoder and Variational Autoencoder; and a general Automated Machine Learning version termed ALL and that searches for the best of five OCC algorithms, namely Isolation Forest, Local Outlier Factor, One-Class SVM, AE and VAE. The proposed GE adopts an evolutionary multi-objective optimization approach, aiming to maximize the predictive performance of the OCC learners while minimizing their training time. Moreover, it includes two mechanisms to speed up the execution time, a periodic sampling of the training data and a fitness evaluation parallelization by using a multi-core processing. To the best of our knowledge, this is the first time that GE has been applied as a NE and AutoML for OCC tasks.

A large set of empirical experiments was held, considering eight public domain datasets retrieved from the OpenML platform, two GE variants (NE and ALL) and two validation scenarios (unsupervised and supervised). Overall, competitive results were achieved by the proposed AutoOC, which is capable of modeling large datasets using a reasonable amount of computational resources. For instance, for the largest analyzed dataset (Credit Card, which contains around 285 thousand examples) and supervised validation mode, the median execution time of AutoOC was around 16 minutes for the general ALL AutoML and around 37 minutes for the NE. Moreover, the optimized One-Class Classification models require a reduced training time. For example, when assuming the sampled n=2,500 training examples and the best Pareto predictive performance results, the ALL setup requires a median training time that is lower than 1 s, while the NE variant optimizes AEs that need a median training time of 8.2 s. As for the predictive performance AutoOC results, quality Area Under the Curve of the Receiver Operating Characteristic curve values (e.g., >70%) were obtained for seven of the eight analyzed datasets. The AutoOC tool clearly

outperformed a baseline IF and even managed to surpass the best public OpenML human modeling approach for two datasets (Credit and Mushroom).

Chapter 6

Conclusions

This chapter presents the conclusion of this PhD thesis. First, Section 6.1 presents a summary of the overall PhD work. Then, Section 6.2 discusses the obtained results and identifies the limitations. Lastly, Section 6.3 presents future research directions.

6.1 Overview

The current availability of an enormous amount of data, algorithms and powerful computing hardware, allows companies and ML practitioners to develop ML applications that can produce fast and accurate predictions (Darwiche, 2018). Currently, automation and efficiency are highlighted as two of the most important features of real-world ML applications. Automation is crucial to enable ML practitioners to manage tasks of the ML workflow effectively. The rise of non-specialists working with ML has increased the focus on automating ML tasks, leading to the emergence of AutoML. Efficiency is particularly beneficial for ML applications in the context of Big Data or when there are hardware limitations. Distributed or parallel learning, which involve using multiple machines or processors to process portions of the ML algorithm or the data, are often used to address these efficiency concerns. The main objective of this thesis is to contribute to the body of knowledge in the area of ML by designing and implementing an automated and efficient ML framework. Specifically, the PhD proposes a computationally efficient AutoML tool that uses a GE to automate the design of lightweight OCC ML models.

The initial part of this PhD work was carried out under the IRMDA R&D project (Chapter 3). This project had an execution time of two years and it was aimed at developing a ML application for a Portuguese analytics company, in order to assist the company telecommunications clients. The main requirements of the ML system were automation and scalability. The proposed ML technological architecture was developed to be used by non-ML experts, by automating all typical tasks of a common supervised ML application and was designed to work within a computational cluster with several processing nodes. The training module of the proposed framework was developed using the H2O AutoML tool, given that it was the best performing tool that also allowed an distributed exectuion. The remaining modules were developed

using Apache Spark distributed data processing capabilities. The AutoML benchmark also showed that the analyzed AutoML frameworks produced close results. These results can even outperform human ML modeling, in particular when considering GML AutoML tools. These first experiments aimed to explore existing AutoML frameworks, with particular focus on automated and distributed ML, and aimed to gain insights into the application of AutoML for supervised learning tasks. Furthermore, we sought to identify a reliable and robust evaluation method for the proposed AutoML framework.

Following the initial experiments, the next part of this PhD was developed under the CMMS project (Chapter 4). CMMS focused on the application of ML algorithms for PdM, in particular to predict equipment failures. This project an execution time of approximately one year, which corresponded to the third year of this PhD project. Within this scope, we conducted the first preliminary experiments related to a novel AutoML framework by applying a GE to design and evolve different OCC ML algorithms using both single and multi-objective optimization. The first step was to apply state-of-the-art AutoML tools using a real-world PdM dataset related to maintenance equipment failures. Based on our findings, we developed and proposed AutoOneClass, a novel AutoML framework that specializes in OCC and employs three algorithms: deep AE, IF, and OC-SVM. The AutoOneClass framework uses a GE to optimize the search for the best OCC ML algorithm and its hyperparameters, allowing for single or multi-objective search. We also designed two validation setups for the framework: unsupervised validation, which employs unlabeled data during validation and anomaly scores to evaluate the ML models; and supervised validation, which uses a labeled validation set to assess model performance. The computational experiments assumed five predictive tasks: one regression and four binary classifications. Among the existing AutoML tools, AutoGluon presented the best average results. The AutoOneClass results were also good, surpassing the AutoML tools focused on DL. In general, the results favored the AutoML tools when compared with AutoOneClass and with human ML designs. Nevertheless, our proposed approach surpassed the expert human modeling in two predictive targets, showing that OCC can be valuable for the PdM domain since industrial data can arise with a high velocity. Our research on this project provided valuable insights into the use of OCC algorithms within AutoML and contributed to advancing the state-of-the-art regarding AutoML approaches for the PdM industry.

In the final part of this PhD thesis (Chapter 5), we describe the work developed during the fourth and final year of this doctoral program. In this final contribution, we aimed to further develop the initial version of the AutoML framework proposed in Chapter 4. This led to the creation of AutoOC, an enhanced version of the AutoOneClass method. The new contribution focused entirely on OCC ML algorithms, which were identified as a research gap in the previous work. State-of-the-art research showed that the vast majority of AutoML tools focused on supervised learning (e.g., classification, regression) and did not handle an OCC. In addition, AutoOC focused solely on multi-objective optimization, using the NSGA-II algorithm to maximize the predictive performance of the OCC learners while minimizing their training time. The goal was to address the efficiency aspect of the PhD objectives by generating lightweight ML models, which is crucial when working with real-world Big Data. Moreover, the proposed AutoOC adopts two computationally efficient mechanisms to speed up the overall execution time: continuous

sampling of training data and parallel fitness evaluation by adopting multi-core processors. To evaluate the effectiveness of AutoOC, we conducted several computational experiments using eight public datasets from various domains and two distinct validation modes (unsupervised and supervised). Overall, the experiments showed that AutoOC achieved competitive results, even when considering large datasets (e.g., with more than 285,000 examples). By using the two acceleration mechanisms, AutoOC is able to perform the GE search in less than 40 minutes for the larger datasets. Additionally, the optimized OCC models require a reduced training time, in most cases lower than 1 s. The AutoOC outperformed a baseline IF in all datasets and even surpassed the best public human modeling approaches for two datasets (Credit and Mushroom).

6.2 Discussion

Overall, the final version of the proposed automated and efficient Machine Learning framework (AutoOC, described in Chapter 5) provides a novel approach to the field of AutoML, offering a range of capabilities that were rarely addressed in previous research works on the topic.

First, this framework addresses an OCC task, in contrast to most AutoML frameworks. While there is currently a large availability of AutoML works, these solutions typically only focus on supervised learning. Identified as a current research challenge for AutoML, we apply AutoML techniques outside the supervised learning domain, using five OCC algorithms, namely Autoencoder, Isolation Forest, Local Outlier Factor, One-Class SVM, and Variational Autoencoder. The proposed framework is able to optimize the search for the OCC ML algorithm and its associated hyperparameters for a given dataset using Grammatical Evolution.

Second, we address efficiency of the AutoML search by adopting an evolutionary multi-objective optimization approach, aiming to maximize the predictive performance of the OCC learners while minimizing their training time. Most existing AutoML solutions only focus on a single-objective search, typically predictive performance. This often results in solutions that take too long to obtain useful ML models (e.g., in the Section 3.3 study, one of the tools always takes the maximum allowed training time) or in very complex models (e.g., the best models of H2O AutoML are usually Stacked Ensembles with dozens of models). By considering the training time in our proposed framework, we intend to generate lightweight OCC Machine Learning models, an important aspect when working with real-world Big Data. Additionally, to speed up the training time we also use a periodic sampling of the training data and a fitness evaluation parallelization by using a multi-core processing.

Third, we adopt a flexible and extensible approach in which more learners can be easily added to the framework. In most AutoML tools, the set of ML algorithms and range of hyperparameters are mostly rigid, where the user can only limit the AutoML execution to a subset of the available learners. Since we use Grammatical Evolution, it assumes a variable-length solution representation (allowing to cope with different learners) and easy customization of the search space by means of an explicit grammar (allowing

to adjust more or fewer base learners and their respective hyperparameters, if needed).

Despite the positive results obtained by the distinct experiments, it is possible to identify limitations to this PhD project as a whole. One of the main difficulties was the fact that this PhD was developed under more than one R&D project (e.g., project IRMDA, project CMMS), all with short-term duration (no more than two years). This meant that the experiments had to address a reduced scope, given the time restrictions. The availability of real-world data to use in the experiments was also smaller since the process of making the real data available usually takes several months (e.g., for bureaucratic reasons). We addressed this limitation by dividing the PhD work into two phases: a first phase (described in Chapter 3), where the focus was to explore existing AutoML frameworks within the domain of supervised learning tasks and identify research gaps; and a second phase (described in Chapter 4 and Chapter 5), where we focused on developing a novel automated and efficient ML framework for OCC.

An additional challenge we faced was the fact that AutoML is currently a very hot research topic, with hundreds of research works being published and dozens of new frameworks being proposed every month. This was a challenge that had implications in the earlier stages of this PhD, since initially the plan was to address a novel AutoML framework within supervised learning. After the work described in Chapter 3 and by regularly revisiting the AutoML state-of-the-art, it was identified that the was a heavy focus on supervised learning AutoML and a research gap in other ML tasks (e.g., One-Class Classification).

Finally, another limitation of this PhD work was the fact that the last stage was not associated with any R&D project, thus it was not possible to use real-world data from a specific application domain and that involved at least a research collaboration with one business client for the last experiment (Chapter 5). We addressed this limitation by considering a large number of public domain datasets (eight datasets retrieved from OpenML). We selected popular datasets from distinct application domains that reflected different numbers of instances, attributes, and output target class balancing, in order to have a varied set of datasets to evaluate our proposed AutoML framework. Also, the fact that the datasets were retrieved from OpenML, allowed us to compare our obtained results with the best public OpenML results. This allowed us to use these OpenML results as a "gold standard", denoting a proxy to the best empirical predictive results that can be achieved when using an human expert modeling.

6.3 Future Work

Regarding this PhD work, there are several research directions that could be addressed in future work. For instance, we wish to test the proposed framework on a wider range of datasets and add more OCC algorithms to the grammar to further validate its effectiveness. We plan to develop a benchmark with a more significant number of OCC algorithms, such as Gaussian Mixture Model or recently proposed OCC algorithms (Blázquez-Garcia et al., 2021; Golan & El-Yaniv, 2018; Hayashi et al., 2021; Lenz et al., 2021; Mauceri et al., 2020). We aim to experiment different values for crossover, mutation and training time to assess their impact on the GE optimization. Another interesting future experimentation is to study

the usage of other multi-objective algorithms, such as R-NSGA-II (Abouhawwash & Deb, 2021). Also, we intend to analyze in more depth the correlation between the AUC and the anomaly scores when using a supervised validation. We wish to test the proposed framework with more real-world datasets, to verify further consistency with the obtained results and verify if the proposed method could be implemented in real-world scenarios.

Furthermore, it could be valuable to check the possibility of further enhancing the efficiency of the framework, adding other elements besides the already implemented acceleration mechanisms (random sampling and parallel training). This could be achieved by using a Selective Sampling (Afonso et al., 2023), a distributed computation engine for data processing (e.g., Apache Spark), or by using algorithms that can be run on GPUs.

Additionally, we aim to add more functionalities to the proposed framework. These could include the possibility of using other efficiency objectives apart from training time, such as inference time, model complexity (e.g., number of parameters), or even the model size. Other new features could include an early stopping to the GE optimization via a monitoring of the evolution of the hypervolume measure or a maximum optimization time. We also intend to experiment with AutoML technologies that can automatically perform other ML phases apart from modeling, such as feature engineering and selection.

Finally, we intend to study the possibility of extending the current framework by adding Federated Learning capabilities. This could be implemented in a decentralized manner, where the different models do not share data (e.g., for confidentiality reasons). At the end of each generation, these "local" models could be unified into a "global" model using distinct aggregation techniques, such as Federated Averaging (Ferreira et al., 2023).

Bibliography

Abouhawwash, M., & Deb, K. (2021). Reference Point Based Evolutionary Multi-objective Optimization Algorithms with Convergence Properties Using KKTPM and ASF Netrics. *Journal of Heuristics*, *27*(4), 575–614. https://doi.org/10.1007/s10732-021-09470-4 (cit. on p. 98)

Afonso, C., Matta, A., Matos, L. M., Gomes, M. B., Santos, A., Pilastri, A. L., & Cortez, P. (2023). Machine Learning for Predicting Production Disruptions in the Wood-Based Panels Industry: A Demonstration Case. *Artificial Intelligence Applications and Innovations - 19th IFIP WG 12.5 International Conference, AIAI 2023, Hersonissos, Léon, Spain, June 14-17, 2023* (cit. on p. 98).

Aggarwal, C. C. (2018). *Neural Networks and Deep Learning - A Textbook*. Springer. https://doi.org/10.1007/978-3-319-94463-0. (Cit. on p. 14)

Ali, M. (2022). *PyCaret: An Open Source, Low-code Machine Learning Library in Python* [PyCaret version 2.3.10]. https://www.pycaret.org. (Cit. on p. 16)

Amruthnath, N., & Gupta, T. (2018a). A Research Study on Unsupervised Machine Learning Algorithms for Early Fault Detection in Predictive maintenance. *2018 5th International Conference on Industrial Engineering and Applications (ICIEA)*, 355–361. https://doi.org/10.1109/IEA.2018.8387124 (cit. on pp. 26, 27)

Amruthnath, N., & Gupta, T. (2018b). Fault Class Prediction in Unsupervised Learning Using Model-based Clustering Approach. *2018 International Conference on Information and Computer Technologies (ICICT)*, 5–12. https://doi.org/10.1109/INFOCT.2018.8356831 (cit. on p. 57)

Anjum, M. S., & Ryan, C. (2021). Seeding Grammars in Grammatical Evolution to Improve Search-Based Software Testing. *SN Computer Science*, *2*(4), 280. https://doi.org/10.1007/s42979-021-00631-7 (cit. on pp. xi, 20)

Apache Spark. (2020a). Extracting, Transforming and Selecting Features - Spark 2.4.5 Documentation. https://spark.apache.org/docs/latest/ml-features. (Cit. on p. 40)

Apache Spark. (2020b). ML Pipelines - Spark 2.4.5 Documentation. https://spark.apache.org/docs/latest/ml-pipeline.html. (Cit. on p. 41)

Arena, S., Florian, E., Zennaro, I., Orrù, P., & Sgarbossa, F. (2022). A Novel Decision Support System for Managing Predictive Maintenance Strategies Based on Machine Learning Approaches. *Safety Science*, *146*, 105529. https://doi.org/10.1016/j.ssci.2021.105529 (cit. on pp. 26, 27)

Arregoces, P., Vergara, J., Gutierrez, S. A., & Botero, J. F. (2022). Network-based Intrusion Detection: A One-class Classification Approach. *2022 IEEE/IFIP Network Operations and Management Symposium, NOMS 2022, Budapest, Hungary, April 25-29, 2022*, 1–6. https://doi.org/10.1109/NOMS54207.2022.9789927 (cit. on pp. 17, 75)

Assunção, F., Lourenço, N., Ribeiro, B., & Machado, P. (2020). Evolution of Scikit-Learn Pipelines with Dynamic Structured Grammatical Evolution. In P. A. Castillo, J. L. J. Laredo, & F. F. de Vega (Eds.), *Applications of Evolutionary Computation - 23rd European Conference, EvoApplications 2020, Held as Part of EvoStar 2020, Seville, Spain, April 15-17, 2020, Proceedings* (pp. 530–545). Springer. https://doi.org/10.1007/978-3-030-43722-0_34. (Cit. on p. 28)

AutoGluon. (2021). AutoGluon: AutoML Toolkit for Deep Learning — AutoGluon Documentation 0.2.0 documentation. https://auto.gluon.ai/. (Cit. on pp. 24, 44)

Ayvaz, S., & Alpay, K. (2021). Predictive Maintenance System for Production Lines in Manufacturing: A Machine Learning Approach Using IoT Data in Real-time. *Expert Systems with Applications*, *173*, 114598. https://doi.org/10.1016/j.eswa.2021.114598 (cit. on pp. 27, 56)

Bahri, M., Salutari, F., Putina, A., & Sozio, M. (2022). AutoML: State of the Art with a Focus on Anomaly Detection, Challenges, and Research Directions. *International Journal of Data Science and Analytics*, 1–14. https://doi.org/10.1007/s41060-022-00309-0 (cit. on p. 58)

Balaprakash, P., Tiwari, A., Wild, S. M., Carrington, L., & Hovland, P. D. (2016). AutoMOMML: Automatic Multi-objective Modeling with Machine Learning. In J. M. Kunkel, P. Balaji, & J. J. Dongarra (Eds.), *High Performance Computing - 31st International Conference, ISC High Performance 2016, Frankfurt, Germany, June 19-23, 2016, Proceedings* (pp. 219–239). Springer. https://doi.org/10.1007/978-3-319-41321-1_12. (Cit. on p. 28)

Basheer, I. A., & Hajmeer, M. (2000). Artificial Neural Networks: Fundamentals, Computing, Design, and Application. *Journal of Microbiological Methods*, *43*(1), 3–31. https://doi.org/10.1016/S0167-7012(00)00201-3 (cit. on p. 14)

Baymurzina, D., Golikov, E. A., & Burtsev, M. S. (2022). A Review of Neural Architecture Search. *Neurocomputing*, *474*, 82–93. https://doi.org/10.1016/j.neucom.2021.12.014 (cit. on pp. 19, 75)

Benedetti, M. D., Leonardi, F., Messina, F., Santoro, C., & Vasilakos, A. V. (2018). Anomaly Detection and Predictive Maintenance for Photovoltaic Systems. *Neurocomputing*, *310*, 59–68. https://doi.org/10.1016/j.neucom.2018.05.017 (cit. on p. 56)

Biau, G. (2012). Analysis of a Random Forests Model. *Journal of Machine Learning Research*, *13*, 1063–1095. http://dl.acm.org/citation.cfm?id=2343682 (cit. on p. 13)

Bishop, C. M. (2007). *Pattern Recognition and Machine Learning, 5th Edition*. Springer. https://www.worldcat.org/oclc/71008143. (Cit. on p. 13)

Blázquez-Garcia, A., Conde, A., Mori, U., & Lozano, J. A. (2021). Water Leak Detection Using Self-supervised Time Series Classification. *Information Sciences*, *574*, 528–541. https://doi.org/10.1016/j.ins.2021.06.015 (cit. on p. 97)

Blum, A. L., & Langley, P. (1997). Selection of Relevant Features and Examples in Machine Learning. *Artificial Intelligence*, *97*(1-2), 245–271. https://doi.org/10.1016/s0004-3702(97)00063-5 (cit. on p. 33)

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324 (cit. on p. 13)

Breunig, M. M., Kriegel, H., Ng, R. T., & Sander, J. (2000). LOF: Identifying Density-Based Local Outliers. In W. Chen, J. F. Naughton, & P. A. Bernstein (Eds.), *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA* (pp. 93–104). ACM. https://doi.org/10.1145/342009.335388. (Cit. on p. 18)

Brockwell, P. J., & Davis, R. A. (2016). *Introduction to Time Series and Forecasting*. Springer. https://doi.org/https://doi.org/10.1007/b97391. (Cit. on p. 12)

Butte, S., Prashanth, A., & Patil, S. (2018). Machine Learning Based Predictive Maintenance Strategy: a Super Learning Approach with Deep Neural Networks. *2018 IEEE Workshop on Microelectronics and Electron Devices (WMED)*, 1–5. https://doi.org/10.1109/WMED.2018.8360836 (cit. on pp. 27, 56)

Çakir, M., Güvenç, M. A., & Mistikoglu, S. (2021). The Experimental Application of Popular Machine Learning Algorithms on Predictive Maintenance and the Design of IIoT Based Condition Monitoring System. *Computers & Industrial Engineering*, *151*, 106948. https://doi.org/10.1016/j.cie.2020.106948 (cit. on p. 27)

Calabrese, M., Cimmino, M., Fiume, F., Manfrin, M., Romeo, L., Ceccacci, S., Paolanti, M., Toscano, G., Ciandrini, G., Carrotta, A., Mengoni, M., Frontoni, E., & Kapetis, D. (2020). SOPHIA: An Event-Based IoT and Machine Learning Architecture for Predictive Maintenance in Industry 4.0. *Information*, *11*(4), 202. https://doi.org/10.3390/info11040202 (cit. on p. 27)

Carvalho, T. P., Soares, F. A. A. M. N., Vita, R., da Piedade Francisco, R., Basto, J. P. T. V., & Alcalá, S. G. S. (2019). A Systematic Literature Review of Machine Learning Methods Applied to Predictive Maintenance. *Computers & Industrial Engineering*, *137*. https://doi.org/10.1016/j.cie.2019.106024 (cit. on p. 56)

Cetto, T., Byrne, J., Xu, X., & Moloney, D. (2019). Size/Accuracy Trade-Off in Convolutional Neural Networks: An Evolutionary Approach. In L. Oneto, N. Navarin, A. Sperduti, & D. Anguita (Eds.), *Recent Advances in Big Data and Deep Learning, Proceedings of the INNS Big Data and Deep Learning Conference INNSBDDL 2019, Genova, Italy 16-18 April 2019* (pp. 17–26). Springer. https://doi.org/10.1007/978-3-030-16841-4_3. (Cit. on pp. 28, 75)

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, *16*, 321–357. https://doi.org/10.1613/jair.953 (cit. on p. 37)

Chen, Y., Song, Q., & Hu, X. (2020). Techniques for Automated Machine Learning. *SIGKDD Explorations*, *22*(2), 35–50. https://doi.org/10.1145/3447556.3447567 (cit. on pp. 25, 26, 42)

Chen, Y., Lin, Y., Kung, C., Chung, M., & Yen, I. (2019). Design and Implementation of Cloud Analytics-Assisted Smart Power Meters Considering Advanced Artificial Intelligence as Edge Analytics in Demand-Side Management for Smart Homes. *Sensors*, *19*(9), 2047. https://doi.org/10.3390/s19092047 (cit. on p. 14)

Chen, Z., Yeo, C. K., Lee, B., Lau, C. T., & Jin, Y. (2018). Evolutionary Multi-objective Optimization Based Ensemble Autoencoders for Image Outlier Detection. *Neurocomputing*, *309*, 192–200. https://doi.org/10.1016/j.neucom.2018.05.012 (cit. on p. 28)

Cheng, B., & Titterington, D. M. (1994). Neural Networks: A Review From a Statistical Perspective. *Statistical Science*, 2–30 (cit. on p. 14).

Cheng, J. C., Chen, W., Chen, K., & Wang, Q. (2020). Data-driven Predictive Maintenance Planning Framework for MEP Components based on BIM and IoT using Machine Learning Algorithms. *Automation in Construction*, *112*, 103087. https://doi.org/10.1016/j.autcon.2020.103087 (cit. on pp. 26, 27)

Cho, S., May, G., Tourkogiorgis, I., Perez, R., Lázaro, Ó., de la Maza, B., & Kiritsis, D. (2018). A Hybrid Machine Learning Approach for Predictive Maintenance in Smart Factories of the Future. *Advances in Production Management Systems. Smart Manufacturing for Industry 4.0 - IFIP WG 5.7 International Conference, APMS 2018, Seoul, Korea, August 26-30, 2018, Proceedings, Part II*, *536*, 311–317. https://doi.org/10.1007/978-3-319-99707-0_39 (cit. on pp. 26, 27, 57)

Çınar, Z. M., Abdussalam Nuhu, A., Zeeshan, Q., Korhan, O., Asmael, M., & Safaei, B. (2020). Machine Learning in Predictive Maintenance Towards Sustainable Smart Manufacturing in Industry 4.0. *Sustainability*, *12*(19), 8211. https://doi.org/10.3390/su12198211 (cit. on p. 56)

Cline, B., Niculescu, R. S., Huffman, D., & Deckel, B. (2017). Predictive Maintenance Applications for Machine Learning. *2017 Annual Reliability and Maintainability Symposium (RAMS)*, 1–7. https://doi.org/10.1109/RAM.2017.7889679 (cit. on pp. 27, 56)

Coelho, G., Matos, L. M., Pereira, P. J., Ferreira, A. L., Pilastri, A. L., & Cortez, P. (2022). Deep Autoencoders for Acoustic Anomaly Detection: Experiments with Working Machine and In-vehicle Audio. *Neural Computing and Applications*, *34*(22), 19485–19499. https://doi.org/10.1007/s00521-022-07375-2 (cit. on pp. 23, 79, 81)

Coello, C. A. C., Lamont, G. B., & van Veldhuizen, D. A. (2007). *Evolutionary Algorithms for Solving Multi-objective Problems, Second Edition*. Springer. (Cit. on pp. 21, 77).

Cook, D. (2016). *Practical Machine Learning with H2O: Powerful, Scalable Techniques for Deep Learning and AI*. O'Reilly Media, Inc. (Cit. on p. 16).

Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. *Machine Learning*, *20*(3), 273–297. https://doi.org/10.1007/BF00994018 (cit. on p. 14)

Cortez, P. (2010). Data Mining with Neural Networks and Support Vector Machines Using the R/rminer Tool. *Advances in Data Mining. Applications and Theoretical Aspects, 10th Industrial Conference, ICDM 2010, Berlin, Germany, July 12-14, 2010. Proceedings*, *6171*, 572–583. https://doi.org/10.1007/978-3-642-14400-4_44 (cit. on pp. 16, 37)

Cortez, P. (2020). Package 'rminer'. https://cran.r-project.org/web/packages/rminer/rminer.pdf. (Cit. on p. 24)

Cortez, P. (2021). *Modern optimization with R*. Springer. (Cit. on p. 21).

Cortez, P., Pereira, P. J., & Mendes, R. (2020). Multi-step Time Series Prediction Intervals Using Neuroevolution. *Neural Computing and Applications*, *32*(13), 8939–8953. https://doi.org/10.1007/s00521-019-04387-3 (cit. on pp. 19, 75)

Cristianini, N., & Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press. https://doi.org/10.1017/CBO9780511801389. (Cit. on p. 14)

Dalianis, H. (2018). Evaluation Metrics and Evaluation. In *Clinical Text Mining: Secondary Use of Electronic Patient Records* (pp. 45–53). Springer International Publishing. https://doi.org/10.1007/978-3-319-78503-5_6. (Cit. on p. 22)

Dangut, M. D., Skaf, Z., & Jennions, I. K. (2022). Handling Imbalanced Data for Aircraft Predictive Maintenance Using the BACHE Algorithm. *Applied Soft Computing*, 108924. https://doi.org/10.1016/j.asoc.2022.108924 (cit. on p. 62)

Darwiche, A. (2018). Human-level Intelligence or Animal-like Abilities? *Communications of the ACM*, *61*(10), 56–67. https://doi.org/10.1145/3271625 (cit. on pp. 1, 31, 94)

Das, P., Ivkin, N., Bansal, T., Rouesnel, L., Gautier, P., Karnin, Z. S., Dirac, L., Ramakrishnan, L., Perunicic, A., Shcherbatyi, I., Wu, W., Zolic, A., Shen, H., Ahmed, A., Winkelmolen, F., Miladinovic, M., Archembeau, C., Tang, A., Dutt, B., ... Venkateswar, K. (2020). Amazon SageMaker Autopilot: a White Box AutoML Solution at Scale. In S. Schelter, S. Whang, & J. Stoyanovich (Eds.), *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning* (2:1–2:7). ACM. https://doi.org/10.1145/3399579.3399870. (Cit. on pp. 25, 26)

Deb, K., Agrawal, S., Pratap, A., & Meyarivan, T. (2002). A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182–197. https://doi.org/10.1109/4235.996017 (cit. on pp. 21, 77)

de Lima Thomaz, R., Carneiro, P. C., Bonin, J. E., Macedo, T. A. A., Patrocinio, A. C., & Soares, A. B. (2018). Novel Mahalanobis-based Feature Selection Improves One-Class Classification of Early Hepatocellular Carcinoma. *Medical & Biological Engineering & Computing*, *56*(5), 817–832. https://doi.org/10.1007/s11517-017-1736-5 (cit. on pp. 28, 29)

de Sá, A. G. C., Pinto, W. J. G. S., Oliveira, L. O. V. B., & Pappa, G. L. (2017). RECIPE: A Grammar-Based Framework for Automatically Evolving Classification Pipelines. In J. McDermott, M. Castelli, L. Sekanina, E. Haasdijk, & P. García-Sánchez (Eds.), *Genetic Programming - 20th European Conference, EuroGP 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings* (pp. 246–261). https://doi.org/10.1007/978-3-319-55696-3_16. (Cit. on p. 28)

Dhir, N., Rudny, T., Zilli, D., & Tosi, A. (2020). An Automatic Type-Inferential General Latent Feature Model. *2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom,*

July 19-24, 2020, 1–8. https://doi.org/10.1109/IJCNN48605.2020.9207720 (cit. on pp. 25, 26)

Domingos, P. (2018). *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. Basic Books, Inc. (Cit. on p. 12).

Drori, I., Liu, L., Nian, Y., Koorathota, S. C., Li, J. S., Moretti, A. K., Freire, J., & Udell, M. (2019). AutoML Using Metadata Language Embeddings. *arXiv preprint arXiv:1910.03698* (cit. on p. 26).

Edun, A. S., LaFlamme, C., Kingston, S. R., Furse, C. M., Scarpulla, M. A., & Harley, J. B. (2022). Anomaly Detection of Disconnects Using SSTDR and Variational Autoencoders. *IEEE Sensors Journal*, *22*(4), 3484–3492. https://doi.org/10.1109/JSEN.2022.3140922 (cit. on p. 19)

Elshawi, R., Maher, M., & Sakr, S. (2019). Automated Machine Learning: State-of-the-art and Open Challenges. *arXiv preprint arXiv:1906.02287* (cit. on p. 24).

Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural Architecture Search: A Survey. *The Journal of Machine Learning Research*, *20*, 55:1–55:21. http://jmlr.org/papers/v20/18-598.html (cit. on pp. 15, 16)

Erickson, N., Mueller, J., Shirkov, A., Zhang, H., Larroy, P., Li, M., & Smola, A. (2020). AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. *arXiv preprint arXiv:2003.06505* (cit. on pp. 25, 26).

Estevez-Velarde, S., Gutiérrez, Y., Almeida-Cruz, Y., & Montoyo, A. (2021). General-purpose Hierarchical Optimisation of Machine Learning Pipelines with Grammatical Evolution. *Information Sciences*, *543*, 58–71. https://doi.org/10.1016/j.ins.2020.07.035 (cit. on p. 28)

Estevez-Velarde, S., Gutiérrez, Y., Montoyo, A., & Almeida-Cruz, Y. (2019). AutoML Strategy Based on Grammatical Evolution: A Case Study about Knowledge Discovery from Text. In A. Korhonen, D. R. Traum, & L. Màrquez (Eds.), *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers* (pp. 4356–4365). Association for Computational Linguistics. https://doi.org/10.18653/v1/p19-1428. (Cit. on p. 28)

Fawcett, T. (2006). An Introduction to ROC Analysis. *Pattern Recognition Letters*, *27*(8), 861–874. https://doi.org/10.1016/j.patrec.2005.10.010 (cit. on pp. 22, 78, 84, 87, 139)

Fenton, M., McDermott, J., Fagan, D., Forstenlechner, S., Hemberg, E., & O'Neill, M. (2017). PonyGE2: Grammatical Evolution in Python. In P. A. N. Bosman (Ed.), *Genetic and Evolutionary Computation Conference, Berlin, Germany, July 15-19, 2017, Companion Material Proceedings* (pp. 1194–1201). ACM. https://doi.org/10.1145/3067695.3082469. (Cit. on pp. 59, 81)

Ferreira, L., & Cortez, P. (2023). AutoOC: Automated Multi-objective Design of Deep Autoencoders and One-class Classifiers Using Grammatical Evolution. *Applied Soft Computing*, *144*, 110496. https://doi.org/10.1016/j.asoc.2023.110496 (cit. on pp. xiv, 7, 74, 139, 144)

Ferreira, L., Pilastri, A., Romano, F., & Cortez, P. (2022). Using Supervised and One-Class Automated Machine Learning for Predictive Maintenance. *Applied Soft Computing, Elsevier*, *131*, 109820. https://doi.org/10.1016/j.asoc.2022.109820 (cit. on pp. 6, 17, 27, 28, 55, 75, 78, 144)

Ferreira, L., Pilastri, A. L., Martins, C., Santos, P., & Cortez, P. (2020a). A Scalable and Automated Machine Learning Framework to Support Risk Management. In A. P. Rocha, L. Steels, & H. J. van den Herik (Eds.), *Agents and Artificial Intelligence, 12th International Conference, ICAART 2020, Valletta, Malta, February 22-24, 2020, Revised Selected Papers* (pp. 291–307). Springer. https://doi.org/10.1007/978-3-030-71158-0_14. (Cit. on pp. 5, 31)

Ferreira, L., Pilastri, A. L., Martins, C., Santos, P., & Cortez, P. (2020b). An Automated and Distributed Machine Learning Framework for Telecommunications Risk Management. In A. P. Rocha, L. Steels, & H. J. van den Herik (Eds.), *Proceedings of the 12th International Conference on Agents and Artificial Intelligence, ICAART 2020, Volume 2, Valletta, Malta, February 22-24, 2020* (pp. 99–107). SCITEPRESS. https://doi.org/10.5220/0008952800990107. (Cit. on pp. 5, 25, 26, 31)

Ferreira, L., Pilastri, A. L., Martins, C. M., Pires, P. M., & Cortez, P. (2021). A Comparison of AutoML Tools for Machine Learning, Deep Learning and XGBoost. *International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021*, 1–8. https://doi.org/10.1109/IJCNN52387.2021.9534091 (cit. on pp. 6, 25, 26, 31, 57, 65, 75, 84, 115, 135)

Ferreira, L., Pilastri, A. L., Sousa, V., Romano, F., & Cortez, P. (2021). Prediction of Maintenance Equipment Failures Using Automated Machine Learning. *Intelligent Data Engineering and Automated Learning - IDEAL 2021 - 22nd International Conference, IDEAL 2021, Manchester, UK, November 25-27, 2021, Proceedings*, *13113*, 259–267. https://doi.org/10.1007/978-3-030-91608-4_26 (cit. on pp. 6, 55)

Ferreira, L., Silva, L., Morais, F., Martins, C. M., Pires, P. M., Rodrigues, H., Cortez, P., & Pilastri, A. L. (2023). International Revenue Share Fraud Prediction on the 5G Edge Using Federated Learning. *Computing*. https://doi.org/10.1007/s00607-023-01174-w (cit. on p. 98)

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., & Hutter, F. (2020). Auto-Sklearn 2.0: The Next Generation. *arXiv preprint arXiv:2007.04074* (cit. on p. 26).

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., & Hutter, F. (2015). Efficient and Robust Automated Machine Learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada* (pp. 2962–2970). http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning. (Cit. on pp. 15, 24)

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., & Hutter, F. (2019). Auto-sklearn: Efficient and Robust Automated Machine Learning. In F. Hutter, L. Kotthoff, & J. Vanschoren (Eds.), *Automated Machine Learning - Methods, Systems, Challenges* (pp. 113–134). Springer. https://doi.org/10.1007/978-3-030-05318-5_6. (Cit. on p. 16)

Floreano, D., Dürr, P., & Mattiussi, C. (2008). Neuroevolution: From Architectures to Learning. *Evolutionary Intelligence*, *1*(1), 47–62. https://doi.org/10.1007/s12065-007-0002-4 (cit. on pp. 19, 75)

Freitas, A. A. (2004). A Critical Review of Multi-objective Optimization in Data Mining: a Position Paper. *SIGKDD Explorations*, *6*(2), 77–86. https://doi.org/10.1145/1046456.1046467 (cit. on p. 47)

Gama, J., Rocha, R., & Medas, P. (2003). Accurate Decision Trees for Mining High-speed Data Streams. In L. Getoor, T. E. Senator, P. M. Domingos, & C. Faloutsos (Eds.), *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003* (pp. 523–528). ACM. https://doi.org/10.1145/956750.956813. (Cit. on p. 13)

Gardner, S., Golovidov, O., Griffin, J., Koch, P., Shi, R., Wujek, B., & Xu, Y. (2021). Fair AutoML Through Multi-objective Optimization (cit. on p. 28).

Gardner, S., Golovidov, O., Griffin, J., Koch, P., Thompson, W., Wujek, B., & Xu, Y. (2019). Constrained Multi-Objective Optimization for Automated Machine Learning. In L. Singh, R. D. D. Veaux, G. Karypis, F. Bonchi, & J. Hill (Eds.), *2019 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2019, Washington, DC, USA, October 5-8, 2019* (pp. 364–373). IEEE. https://doi.org/10.1109/DSAA.2019.00051. (Cit. on p. 28)

Gijsbers, P., LeDell, E., Thomas, J., Poirier, S., Bischl, B., & Vanschoren, J. (2019). An Open Source AutoML Benchmark. *arXiv preprint arXiv:1907.00909* (cit. on pp. 24, 26).

Gimeno Saborit, R. (2019). *Benchmark Auto Machine Learning* (tech. rep.). Universitat Autonoma de Barcelona, Spain. (Cit. on p. 26).

Gohel, H. A., Upadhyay, H., Lagos, L., Cooper, K., & Sanzetenea, A. (2020). Predictive Maintenance Architecture Development for Nuclear Infrastructure using Machine Learning. *Nuclear Engineering and Technology*, *52*(7), 1436–1442. https://doi.org/10.1016/j.net.2019.12.029 (cit. on p. 27)

Golan, I., & El-Yaniv, R. (2018). Deep Anomaly Detection Using Geometric Transformations. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada* (pp. 9781–9791). https://proceedings.neurips.cc/paper/2018/hash/5e62d03aec0d17facfc5355dd90d441c-Abstract.html. (Cit. on p. 97)

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press. (Cit. on p. 14).

Guyon, I., Bennett, K. P., Cawley, G. C., Escalante, H. J., Escalera, S., Ho, T. K., Macià, N., Ray, B., Saeed, M., Statnikov, A. R., & Viegas, E. (2015). Design of the 2015 ChaLearn AutoML Challenge. *2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, July 12-17, 2015*, 1–8. https://doi.org/10.1109/IJCNN.2015.7280767 (cit. on pp. 15, 24)

Guyon, I., Chaabane, I., Escalante, H. J., Escalera, S., Jajetic, D., Lloyd, J. R., Macià, N., Ray, B., Romaszko, L., Sebag, M., Statnikov, A. R., Treguer, S., & Viegas, E. (2016). A Brief Review of the ChaLearn AutoML Challenge: Any-time Any-dataset Learning without Human Intervention. In F. Hutter, L. Kotthoff, & J. Vanschoren (Eds.), *Proceedings of the 2016 Workshop on Automatic Machine Learning, AutoML 2016, co-located with 33rd International Conference on Machine Learning (ICML 2016), New York City, NY, USA, June 24, 2016* (pp. 21–30). JMLR.org. http://proceedings.mlr.press/v64/guyon_review_2016.html. (Cit. on pp. 15, 24)

Guyon, I., Sun-Hosoya, L., Boullé, M., Escalante, H. J., Escalera, S., Liu, Z., Jajetic, D., Ray, B., Saeed, M., Sebag, M., Statnikov, A. R., Tu, W., & Viegas, E. (2019). Analysis of the AutoML Challenge Series 2015-2018. In F. Hutter, L. Kotthoff, & J. Vanschoren (Eds.), *Automated Machine Learning - Methods, Systems, Challenges* (pp. 177–219). Springer. https://doi.org/10.1007/978-3-030-05318-5_10. (Cit. on pp. 1, 15, 24, 26, 42)

H2O.ai. (2021). *H2O AutoML* [H2O version 3.32.1.3]. http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html. (Cit. on pp. 16, 24)

Han, J., Kamber, M., & Pei, J. (2011). *Data Mining: Concepts and Techniques, 3rd edition*. Morgan Kaufmann. (Cit. on p. 11).

Hanussek, M., Blohm, M., & Kintz, M. (2021). Can AutoML Outperform Humans? An Evaluation on Popular OpenML Datasets Using AutoML Benchmark, 29–32. https://doi.org/10.1145/3448326.3448353 (cit. on pp. 25, 26)

Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer. https://doi.org/10.1007/978-0-387-84858-7. (Cit. on pp. 12, 13, 22, 84)

Hayashi, T., Fujita, H., & Hernandez-Matamoros, A. (2021). Less Complexity One-class Classification Approach Using Construction Error of Convolutional Image Transformation Network. *Information Sciences*, *560*, 217–234. https://doi.org/10.1016/j.ins.2021.01.069 (cit. on p. 97)

He, X., Zhao, K., & Chu, X. (2021). AutoML: A Survey of the State-of-the-art. *Knowledge-Based Systems*, *212*, 106622. https://doi.org/10.1016/j.knosys.2020.106622 (cit. on pp. 24, 58)

Hirzel, M., Kate, K., Ram, P., Shinnar, A., & Tsay, J. (2022). Gradual AutoML using Lale. In A. Zhang & H. Rangwala (Eds.), *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022* (pp. 4794–4795). ACM. https://doi.org/10.1145/3534678.3542630. (Cit. on p. 28)

Ho, T. K. (1995). Random Decision Forests. *Third International Conference on Document Analysis and Recognition, ICDAR 1995, August 14 - 15, 1995, Montreal, Canada. Volume I*, 278–282. https://doi.org/10.1109/ICDAR.1995.598994 (cit. on p. 13)

Hodge, V. J., & Austin, J. (2004). A Survey of Outlier Detection Methodologies. *Artificial Intelligence Review*, *22*(2), 85–126. https://doi.org/10.1023/B:AIRE.0000045502.10941.a9 (cit. on p. 13)

Hollander, M., Wolfe, D. A., & Chicken, E. (2013). *Nonparametric Statistical Methods*. John Wiley & Sons. (Cit. on pp. 65, 84).

Jain, A. K. (2010). Data Clustering: 50 Years Beyond K-means. *Pattern Recognition Letters*, *31*(8), 651–666. https://doi.org/10.1016/j.patrec.2009.09.011 (cit. on p. 12)

Jesson, J., Matheson, L., & Lacey, F. M. (2011). *Doing Your Literature Review: Traditional and Systematic Techniques*. Sage. (Cit. on p. 10).

Jin, H., Song, Q., & Hu, X. (2019). Auto-Keras: An Efficient Neural Architecture Search System. In A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, & G. Karypis (Eds.), *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage,*

AK, USA, August 4-8, 2019 (pp. 1946–1956). ACM. https://doi.org/10.1145/3292500.3330648. (Cit. on pp. 15, 16, 24)

Jing, K., Xu, J., & Xu, H. (2020). NASABN: A Neural Architecture Search Framework for Attention-Based Networks. *2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020*, 1–7. https://doi.org/10.1109/IJCNN48605.2020.9207600 (cit. on p. 26)

Jr., C. H. N. L., & Barbosa, H. J. C. (2019). Auto-CVE: a Coevolutionary Approach to Evolve Ensembles in Automated Machine Learning. In A. Auger & T. Stützle (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019* (pp. 392–400). ACM. https://doi.org/10.1145/3321707.3321844. (Cit. on p. 28)

Kanawaday, A., & Sane, A. (2017). Machine Learning for Predictive Maintenance of Industrial Machines Using IoT Sensor Data. *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 87–90. https://doi.org/10.1109/ICSESS.2017.8342870 (cit. on pp. 27, 56)

Karunasingha, D. S. K. (2022). Root Mean Square Error or Mean Absolute Error? Use Their Ratio as Well. *Information Sciences*, *585*, 609–629. https://doi.org/10.1016/j.ins.2021.11.036 (cit. on p. 22)

Kingma, D. P., & Welling, M. (2019). An Introduction to Variational Autoencoders. *Foundations and Trends in Machine Learning*, *12*(4), 307–392. https://doi.org/10.1561/2200000056 (cit. on p. 19)

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., & Leyton-Brown, K. (2017). Auto-WEKA 2.0: Automatic Model Selection and Hyperparameter Optimization in WEKA. *The Journal of Machine Learning Research*, *18*, 25:1–25:5. http://jmlr.org/papers/v18/16-261.html (cit. on p. 24)

Larocque-Villiers, J., Dumond, P., & Knox, D. (2021). Automating Predictive Maintenance Using State-Based Transfer Learning and Ensemble Methods. *14th IEEE International Symposium on Robotic and Sensors Environments, ROSE 2021, Virtual Event, October 28-29, 2021*, 1–7. https://doi.org/10.1109/ROSE52750.2021.9611768 (cit. on pp. 26, 27)

Le, T. T., Fu, W., & Moore, J. H. (2020). Scaling Tree-based Automated Machine Learning to Biomedical Big Data with a Feature Set Selector. *Bioinformatics*, *36*(1), 250–256. https://doi.org/10.1093/bioinformatics/btz470 (cit. on p. 17)

Lenz, O. U., Peralta, D., & Cornelis, C. (2021). Average Localised Proximity: A New Data Descriptor with Good Default One-class Classification Performance. *Pattern Recognition*, *118*, 107991. https://doi.org/10.1016/j.patcog.2021.107991 (cit. on p. 97)

Li, W., & Moore, A. W. (2007). A Machine Learning Approach for Efficient Traffic Classification. *15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2007), October 24-26, 2007, Istanbul, Turkey*, 310–317. https://doi.org/10.1109/MASCOTS.2007.2 (cit. on p. 15)

Liang, J. Z., Meyerson, E., Hodjat, B., Fink, D., Mutch, K., & Miikkulainen, R. (2019). Evolutionary Neural AutoML for Deep Learning. In A. Auger & T. Stützle (Eds.), *Proceedings of the Genetic and*

Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019 (pp. 401–409). ACM. https://doi.org/10.1145/3321707.3321721. (Cit. on pp. 25, 26)

Lin, J., Liu, Z., Wang, H., Li, L., & Han, S. (2018). AMC: AutoML for Model Compression and Acceleration on Mobile Devices. In V. Ferrari, M. Hebert, C. Sminchisescu, & Y. Weiss (Eds.), *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII* (pp. 815–832). Springer. https://doi.org/10.1007/978-3-030-01234-2_48. (Cit. on pp. 1, 15, 42)

Liu, F. T., Ting, K. M., & Zhou, Z. (2008). Isolation Forest. *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, 413–422. https://doi.org/10.1109/ICDM.2008.17 (cit. on pp. xi, 17, 18)

Mahjoubi, S., Barhemat, R., Guo, P., Meng, W., & Bao, Y. (2021). Prediction and Multi-objective Optimization of Mechanical, Economical, and Environmental Properties for Strain-hardening Cementitious Composites (SHCC) Based on Automated Machine Learning and Metaheuristic Algorithms. *Journal of Cleaner Production*, *329*, 129665. https://doi.org/10.1016/j.jclepro.2021.129665 (cit. on p. 28)

Makridis, G., Kyriazis, D., & Plitsos, S. (2020). Predictive Maintenance Leveraging Machine Learning for Time-series Forecasting in the Maritime Industry. *23rd IEEE International Conference on Intelligent Transportation Systems, ITSC 2020, Rhodes, Greece, September 20-23, 2020*, 1–8. https://doi.org/10.1109/ITSC45102.2020.9294450 (cit. on pp. 26, 27, 58)

Maleki, S., Maleki, S., & Jennings, N. R. (2021). Unsupervised Anomaly Detection with LSTM Autoencoders Using Statistical Data-filtering. *Applied Soft Computing*, *108*, 107443. https://doi.org/10.1016/j.asoc.2021.107443 (cit. on p. 17)

Marinescu, R., Kishimoto, A., Ram, P., Rawat, A., Wistuba, M., Palmes, P. P., & Botea, A. (2021). Searching for Machine Learning Pipelines Using a Context-Free Grammar. *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, 8902–8911. https://doi.org/10.1609/aaai.v35i10.17077 (cit. on p. 28)

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, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, ... Xiaoqiang Zheng. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/. (Cit. on pp. 59, 80)

Matos, L. M., Azevedo, J., Matta, A., Pilastri, A. L., Cortez, P., & Mendes, R. (2022). Categorical Attribute traNsformation Environment (CANE): A Python Module for Categorical to Numeric Data Preprocessing. *Software Impacts*, *13*, 100359. https://doi.org/10.1016/j.simpa.2022.100359 (cit. on pp. 64, 83)

Mauceri, S., Sweeney, J., & McDermott, J. (2020). Dissimilarity-based Representations for One-class Classification on Time Series. *Pattern Recognition*, *100*, 107122. https://doi.org/10.1016/j.patcog.2019.107122 (cit. on p. 97)

Miikkulainen, R. (2010). Topology of a Neural Network. In C. Sammut & G. I. Webb (Eds.), *Encyclopedia of Machine Learning* (pp. 988–989). Springer. https://doi.org/10.1007/978-0-387-30164-8_837. (Cit. on p. 14)

Miranda, T. Z., Sardinha, D. B., Basgalupp, M. P., & Cerri, R. (2022). A New Grammatical Evolution Method for Generating Deep Convolutional Neural Networks with Novel Topologies. In J. E. Fieldsend & M. Wagner (Eds.), *GECCO '22: Genetic and Evolutionary Computation Conference, Companion Volume, Boston, Massachusetts, USA, July 9 - 13, 2022* (pp. 663–666). ACM. https://doi.org/10.1145/3520304.3529025. (Cit. on pp. 28, 75)

MLFlow. (2023). MLflow - A platform for the machine learning lifecycle. https://mlflow.org/. (Cit. on p. 140)

Moctezuma, L. A., & Molinas, M. (2020). Multi-objective Optimization for EEG Channel Selection and Accurate Intruder Detection in an EEG-based Subject Identification System. *Scientific Reports*, *10*(1), 1–12. https://doi.org/10.1038/s41598-020-62712-6 (cit. on pp. 28, 29)

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2012). *Foundations of Machine Learning*. MIT Press. http://mitpress.mit.edu/books/foundations-machine-learning-0. (Cit. on p. 12)

Moya, M. M., & Hush, D. R. (1996). Network Constraints and Multi-objective Optimization for One-Class Classification. *Neural Networks*, *9*(3), 463–474. https://doi.org/10.1016/0893-6080(95)00120-4 (cit. on pp. 17, 75)

Moyano, J. M., & Ventura, S. (2022). Auto-adaptive Grammar-Guided Genetic Programming Algorithm to Build Ensembles of Multi-Label Classifiers. *Information Fusion*, *78*, 1–19. https://doi.org/10.1016/j.inffus.2021.07.005 (cit. on p. 28)

Neto, H. A., Alves, R. C., & Campos, S. V. (2020). NASirt: AutoML Based Learning with Instance-level Complexity Information. *arXiv preprint arXiv:2008.11846* (cit. on p. 26).

Ng, A. (2020). *Machine Learning Yearning*. deeplearning.ai. https://www.deeplearning.ai/machine-learning-yearning/. (Cit. on pp. 53, 67)

Nyathi, T., & Pillay, N. (2018). Comparison of a Genetic Algorithm to Grammatical Evolution for Automated Design of Genetic Programming Classification Algorithms. *Expert Systems with Applications*, *104*, 213–234. https://doi.org/10.1016/j.eswa.2018.03.030 (cit. on pp. 19, 77)

Oliveira, N., Cortez, P., & Areal, N. (2017). The Impact of Microblogging Data for Stock Market Prediction: Using Twitter to Predict Returns, Volatility, Trading Volume and Survey Sentiment Indices. *Expert Systems with applications*, *73*, 125–144. https://doi.org/10.1016/j.eswa.2016.12.036 (cit. on pp. 41, 47)

Olson, R. S., Urbanowicz, R. J., Andrews, P. C., Lavender, N. A., Kidd, L. C., & Moore, J. H. (2016). Automating Biomedical Data Science Through Tree-Based Pipeline Optimization. In G. Squillero &

P. Burelli (Eds.), *Applications of Evolutionary Computation - 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 - April 1, 2016, Proceedings, Part I* (pp. 123–137). Springer. https://doi.org/10.1007/978-3-319-31204-0_9. (Cit. on p. 24)

O'Neill, M., & Ryan, C. (2001). Grammatical Evolution. *IEEE Transactions on Evolutionary Computation*, *5*(4), 349–358. https://doi.org/10.1109/4235.942529 (cit. on pp. 19, 77)

Paolanti, M., Romeo, L., Felicetti, A., Mancini, A., Frontoni, E., & Loncarski, J. (2018). Machine Learning Approach for Predictive Maintenance in Industry 4.0. *14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, MESA 2018, Oulu, Finland, July 2-4, 2018*, 1–6. https://doi.org/10.1109/MESA.2018.8449150 (cit. on pp. 27, 56)

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., VanderPlas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830. https://doi.org/10.5555/1953048.2078195 (cit. on pp. 16, 59, 80)

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2008). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *24*(3), 45–77. http://www.jmis-web.org/articles/765 (cit. on pp. xi, 3)

Pereira, P. J., Cortez, P., & Mendes, R. (2021). Multi-objective Grammatical Evolution of Decision Trees for Mobile Marketing User Conversion Prediction. *Expert Systems with Applications*, *168*, 114287. https://doi.org/10.1016/j.eswa.2020.114287 (cit. on pp. 75, 78)

Perera, P., Oza, P., & Patel, V. M. (2021). One-class classification: A survey. *arXiv preprint arXiv:2101.03064* (cit. on p. 17).

Peteiro-Barral, D., & Guijarro-Berdiñas, B. (2013). A Survey of Methods for Distributed Machine Learning. *Progress in Artificial Intelligence*, *2*(1), 1–11. https://doi.org/10.1007/s13748-012-0035-5 (cit. on pp. 1, 4, 24, 31)

Pfisterer, F. (2022). *Democratizing Machine Learning*. Ludwig-Maximilians-Universität München. https://doi.org/10.5282/edoc.30947. (Cit. on p. 28)

Płonska, A., & Płonski, P. (2022). MLJAR: State-of-the-art Automated Machine Learning Framework for Tabular Data. Version 0.10.3. https://github.com/mljar/mljar-supervised. (Cit. on p. 16)

Poole, D., Mackworth, A. K., & Goebel, R. (1998). *Computational Intelligence - a Logical Approach*. Oxford University Press. (Cit. on p. 11).

Ribeiro, D., Matos, L. M., Moreira, G., Pilastri, A. L., & Cortez, P. (2022). Isolation Forests and Deep Autoencoders for Industrial Screw Tightening Anomaly Detection. *Computers*, *11*(4), 54. https://doi.org/10.3390/computers11040054 (cit. on pp. 17, 58, 75, 81)

Risi, S., & Togelius, J. (2017). Neuroevolution in Games: State of the Art and Open Challenges. *IEEE Trans. Comput. Intell. AI Games*, *9*(1), 25–41. https://doi.org/10.1109/TCIAIG.2015.2494596 (cit. on p. 19)

Rokach, L., & Maimon, O. (2007). *Data Mining with Decision Trees - Theory and Applications* (Vol. 69). WorldScientific. https://doi.org/10.1142/6604. (Cit. on p. 13)

Russell, S. J., & Norvig, P. (2003). *Artificial Intelligence - A Modern Approach, 2nd Edition*. Prentice Hall. http://www.worldcat.org/oclc/314283679. (Cit. on p. 12)

Ryan, C., O'Neill, M., & Collins, J. (2018). *Handbook of Grammatical Evolution* (Vol. 1). Springer. (Cit. on p. 75).

Salesforce. (2022). TransmogrifAI - AutoML Library for Building Modular, Reusable, Strongly Typed Machine Learning Workflows on Spark from Salesforce Engineering. https://transmogrif.ai/. (Cit. on pp. 17, 24)

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, *3*(3), 210–229 (cit. on p. 11).

Samuel, A. L. (1967). Some Studies in Machine Learning Using the Game of Checkers. II—Recent Progress. *IBM Journal of Research and Development*, *11*(6), 601–617 (cit. on p. 11).

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., & Williamson, R. C. (2001). Estimating the Support of a High-Dimensional Distribution. *Neural Computation*, *13*(7), 1443–1471. https://doi.org/10.1162/089976601750264965 (cit. on p. 19)

Schwarz, G. (1978). Estimating the Dimension of a Model. *The Annals of Statistics*, *6*(2), 461–464. https://doi.org/10.1214/aos/1176344136 (cit. on p. 139)

Scikit-Learn. (2022a). Isolation Forest. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html. (Cit. on pp. 80, 138)

Scikit-Learn. (2022b). Local Outlier Factor. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html. (Cit. on pp. 80, 138)

Scikit-Learn. (2022c). One-Class SVM. https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html. (Cit. on pp. 80, 138)

Seliya, N., Zadeh, A. A., & Khoshgoftaar, T. M. (2021). A Literature Review on One-Class Classification and its Potential Applications in Big Data. *Journal of Big Data*, *8*(1), 122. https://doi.org/10.1186/s40537-021-00514-x (cit. on pp. 17, 75)

Silva, A. J., Cortez, P., Pereira, C., & Pilastri, A. (2021). Business Analytics in Industry 4.0: A Systematic Review. *Expert Systems*, e12741. https://doi.org/https://doi.org/10.1111/exsy.12741 (cit. on p. 56)

Stanley, K. O., D'Ambrosio, D. B., & Gauci, J. (2009). A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Artificial Life*, *15*(2), 185–212. https://doi.org/10.1162/artl.2009.15.2.15202 (cit. on p. 75)

Straus, P., Schmitz, M., Wöstmann, R., & Deuse, J. (2018). Enabling of Predictive Maintenance in the Brownfield through Low-Cost Sensors, an IIoT-Architecture and Machine Learning. *IEEE International Conference on Big Data (IEEE BigData 2018), Seattle, WA, USA, December 10-13, 2018*, 1474–1483. https://doi.org/10.1109/BigData.2018.8622076 (cit. on pp. 26, 27, 58)

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., & Fergus, R. (2014). Intriguing Properties of Neural Networks (Y. Bengio & Y. LeCun, Eds.). http://arxiv.org/abs/1312.6199 (cit. on p. 14)

TensorFlow. (2022a). Convolutional Variational Autoencoder. https://www.tensorflow.org/tutorials/generative/cvae. (Cit. on pp. 80, 138)

TensorFlow. (2022b). Intro to Autoencoders. https://www.tensorflow.org/tutorials/generative/autoencoder. (Cit. on pp. 80, 138)

Thornton, C., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2013). Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In I. S. Dhillon, Y. Koren, R. Ghani, T. E. Senator, P. Bradley, R. Parekh, J. He, R. L. Grossman, & R. Uthurusamy (Eds.), *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013* (pp. 847–855). ACM. https://doi.org/10.1145/2487575.2487629. (Cit. on pp. 1, 3, 15, 16, 31, 42, 76)

Tornede, T., Tornede, A., Wever, M., Mohr, F., & Hüllermeier, E. (2020). AutoML for Predictive Maintenance: One Tool to RUL Them All. *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning, ITEM 2020, Belgium, September 14-18, 2020, Revised Selected Papers*, *1325*, 106–118. https://doi.org/10.1007/978-3-030-66770-2_8 (cit. on pp. 26, 27)

Tranmer, M., & Elliot, M. (2008). Multiple Linear Regression. *The Cathie Marsh Centre for Census and Survey Research (CCSR)*, *5*, 30–35 (cit. on p. 13).

Truong, A., Walters, A., Goodsitt, J., Hines, K. E., Bruss, C. B., & Farivar, R. (2019). Towards Automated Machine Learning: Evaluation and Comparison of AutoML Approaches and Tools. *31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019, Portland, OR, USA, November 4-6, 2019*, 1471–1479. https://doi.org/10.1109/ICTAI.2019.00209 (cit. on pp. 24–26)

Turing, A. (1950). Computing Machinery and Intelligence. *Mind*, *59*(236), 433 (cit. on p. 11).

Vanschoren, J., van Rijn, J. N., Bischl, B., & Torgo, L. (2013). OpenML: Networked Science in Machine Learning. *ACM SIGKDD Explorations Newsletter*, *15*(2), 49–60. https://doi.org/10.1145/2641190.2641198 (cit. on pp. 24, 44, 76, 83)

Vapnik, V. (1998). *Statistical Learning Theory*. New York: Wiley. (Cit. on p. 14).

Wang, C., & Wu, Q. (2019). FLO: Fast and Lightweight Hyperparameter Optimization for AutoML. *arXiv preprint arXiv:1911.04706* (cit. on p. 26).

Wang, L. (2005). *Support Vector Machines: Theory and Applications*. Springer Science & Business Media. (Cit. on p. 14).

Wang, S., Liu, Q., Zhu, E., Porikli, F., & Yin, J. (2018). Hyperparameter Selection of One-class Support Vector Machine by Self-adaptive Data Shifting. *Pattern Recognition*, *74*, 198–211. https://doi.org/10.1016/j.patcog.2017.09.012 (cit. on p. 19)

Waring, J., Lindvall, C., & Umeton, R. (2020). Automated Machine Learning: Review of the State-of-the-art and Opportunities for Healthcare. *Artificial Intelligence in Medicine*, *104*, 101822. https://doi.org/10.1016/j.artmed.2020.101822 (cit. on pp. 25, 26)

Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. (2016). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann. (Cit. on pp. 16, 23).

Xanthopoulos, I., Tsamardinos, I., Christophides, V., Simon, E., & Salinger, A. (2020). Putting the Human Back in the AutoML Loop. In A. Poulovassilis, D. Auber, N. Bikakis, P. K. Chrysanthis, G. Papastefanatos, M. A. Sharaf, N. Pelekis, C. Renso, Y. Theodoridis, K. Zeitouni, T. Cerquitelli, S. Chiusano, G. Vargas-Solar, B. Omidvar-Tehrani, K. Morik, J. Renders, D. Firmani, L. Tanca, D. Mottin, ... Y. Velegrakis (Eds.), *Proceedings of the Workshops of the EDBT/ICDT 2020 Joint Conference, Copenhagen, Denmark, March 30, 2020*. CEUR-WS.org. http://ceur-ws.org/Vol-2578/ETMLP5.pdf. (Cit. on p. 26)

Yakovlev, A., Moghadam, H. F., Moharrer, A., Cai, J., Chavoshi, N., Varadarajan, V., Agrawal, S. R., Karnagel, T., Idicula, S., Jinturkar, S., & Agarwal, N. (2020). Oracle AutoML: A Fast and Predictive AutoML Pipeline. *Proceedings of the VLDB Endowment*, *13*(12), 3166–3180. https://doi.org/10.14778/3415478.3415542 (cit. on p. 26)

Yao, Q., Wang, M., Chen, Y., Dai, W., Yi-Qi, H., Yu-Feng, L., Wei-Wei, T., Qiang, Y., & Yang, Y. (2018). Taking Human out of Learning Applications: A Survey on Automated Machine Learning. *arXiv preprint arXiv:1810.13306* (cit. on p. 24).

Zimmer, L., Lindauer, M., & Hutter, F. (2021). Auto-Pytorch: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *43*(9), 3079–3090. https://doi.org/10.1109/TPAMI.2021.3067763 (cit. on pp. 16, 25, 26)

Zola, P., Cortez, P., & Brentari, E. (2021). Twitter Alloy Steel Disambiguation and User Relevance via One-Class and Two-Class News Titles Classifiers. *Neural Computing and Applications*, *33*(4), 1245–1260. https://doi.org/10.1007/s00521-020-04991-8 (cit. on pp. 17, 75)

Zöller, M., & Huber, M. F. (2021). Benchmark and Survey of Automated Machine Learning Frameworks. *Journal of Artificial Intelligence Research*, *70*, 409–472. https://doi.org/10.1613/jair.1.11854 (cit. on pp. 24–26)

Appendix A

Computational Experiments and Code of Section 3.3 Benchmark Study

A.1 Overview

The benchmark study presented in Section 3.3 was applied in the paper "A Comparison of AutoML Tools for Machine Learning, Deep Learning and XGBoost" (Ferreira, Pilastri, Martins, et al., 2021). The code used in these experiments is publicly available at the following GitHub repository: luisferreira97/autoautoml. This repository includes all the code used in a benchmark of eight open-source AutoML tools: Auto-Keras, Auto-PyTorch, Auto-Sklearn, AutoGluon, H2O AutoML, rminer, TPOT and TransmogrifAI. This comparison study includes hundreds of computational experiments based on three scenarios: General Machine Learning (GML), Deep Learning (DL), and XGBoost (XGB).

A.2 Folder Description and Structure

The GitHub repository is organized as follows:

1. The code that was used to generate all the benchmark models is inside the data folder and its subfolders.
2. Inside the data folder, there is a subfolder for each of the datasets used for the benchmark.
3. Inside the datasets subfolders, there is one subfolder for each AutoML tool used for that dataset.
4. Inside the tools subfolders, there is the script used to generate the ML models and the resulting metadata (e.g., model leaderboards, performance metrics)

The folder structure can also be represented by the following tree:

```
project
```

```
├── aux-functions: scripts to divide the original datasets into folds
│   ├── join-data
│   └── split-data
├── docs: PDF of the IJCNN paper and other documentation
└── data
    ├── dataset A
    │   ├── AutoML tool A
    │   │   ├── run: script to run the experiment
    │   │   ├── fold 1
    │   │   │   ├── model leaderboard
    │   │   │   ├── performance metrics
    │   │   │   └── other metadata files
    │   │   ├── fold 2
    │   │   ├── fold 3
    │   │   └── other folds
    │   ├── AutoML tool B
    │   ├── AutoML tool C
    │   └── other AutoML tools
    ├── dataset B
    ├── dataset C
    └── other datasets
```

A.3 Code Examples

The scripts that execute the AutoML experiments follow a similar structure for all datasets. The variable part of the scripts are the training functions, that are specific for each AutoML API. This section presents an example of the code used for each AutoML tool.

A.3.1 Auto-Keras

```
import autokeras as ak

for x in range(0, 10):
    fold_folder = "./data/cholesterol/autokeras/fold"+str(x+1)
    folds = [fold1, fold2, fold3, fold4, fold5,
             fold6, fold7, fold8, fold9, fold10]
    test_df = folds[x]
    X_test = test_df.drop(columns=[target]).to_numpy()
```

```
    y_test = test_df[target].to_numpy()

    del folds[x]
    train_df = pd.concat(folds)
    X_train = train_df.drop(columns=[target]).to_numpy()
    y_train = train_df[target].to_numpy()

    classifier = ak.StructuredDataRegressor(
        loss="mean_absolute_error",
        objective="val_loss",
        directory=fold_folder,
        project_name="results",
    )

    start = datetime.now().strftime("%H:%M:%S")
    classifier.fit(x=X_train, y=y_train, validation_split=0.25)
    end = datetime.now().strftime("%H:%M:%S")

    try:
        classifier.evaluate(x=X_test, y=y_test)
    except Exception as e:
        error = str(e)

    best_trial = error[error.find("trial"):
        error.find("/checkpoints")]

    trials = os.listdir(fold_folder + "/results")

    for trial in trials:
        if trial == best_trial:
            checkpoints = os.listdir(
                fold_folder+"/results/"+trial+"/checkpoints")
            checkpoints = [int(checkpoint[6:])
                for checkpoint in checkpoints]
            checkpoints.sort()
            if checkpoints[0] == 0:
                del checkpoints[0]
```

```
            best = min(checkpoints)
            os.rename(
                fold_folder + "/results/" +
                    trial + "/checkpoints/epoch_0",
                fold_folder
                + "/results/"
                + trial
                + "/checkpoints/epoch_"
                + str(best - 1),
            )
        elif "trial" in trial:
            rmtree(fold_folder + "/results/" + trial)

    preds = classifier.predict(x=X_test)

    perf = {}
    perf["start"] = start
    perf["end"] = end
    perf["test_score"] = sklearn.metrics.mean_absolute_error(
        y_test, preds.reshape((len(preds),))
    )

    perf = json.dumps(perf)
    f = open(fold_folder + "/perf.json", "w")
    f.write(perf)
    f.close()

    model = classifier.export_model()

    try:
        model.save(fold_folder+"/model_autokeras",
            save_format="tf")
    except:
        model.save(fold_folder + "/model_autokeras.h5")
```

A.3.2 Auto-Sklearn

```
from autosklearn import classification, regression

for x in range(0, 10):
    fold_folder = "./data/cholesterol/autosklearn/fold"
        + str(x + 1)
    folds = [fold1, fold2, fold3, fold4, fold5,
             fold6, fold7, fold8, fold9, fold10]
    test_df = folds[x]
    X_test = test_df.drop(columns=[target]).to_numpy()
    y_test = test_df[target].to_numpy()

    del folds[x]
    train_df = pd.concat(folds)
    X_train = train_df.drop(columns=[target]).to_numpy()
    y_train = train_df[target].to_numpy()

    automl = autosklearn.regression.AutoSklearnRegressor(
        resampling_strategy="cv",
        resampling_strategy_arguments={"folds": 5}
    )

    start = datetime.now().strftime("%H:%M:%S")
    automl.fit(
        X_train.copy(), y_train.copy(),
        metric=autosklearn.metrics.mean_absolute_error
    )
    automl.refit(X_train.copy(), y_train.copy())
    end = datetime.now().strftime("%H:%M:%S")

    predictions = automl.predict(X_test)

    perf = {}
    perf["start"] = start
    perf["end"] = end
    perf["statistics"] = automl.sprint_statistics()
    perf["pred_score"] = sklearn.metrics.mean_absolute_error(
        y_test, predictions)
```

```
    perf = json.dumps(perf)
    f = open(fold_folder + "/perf.json", "w")
    f.write(perf)
    f.close()

    from joblib import dump

    dump(automl, fold_folder + "/model.joblib")
```

A.3.3 Auto-Pytorch

```
from autoPyTorch import AutoNetClassification, AutoNetRegression

for x in range(0, 10):
    fold_folder = "./data/cholesterol/autopytorch/fold"
        + str(x + 1)
    folds = [fold1, fold2, fold3, fold4, fold5,
             fold6, fold7, fold8, fold9, fold10]
    test_df = folds[x]
    X_test = test_df.drop(columns=[target]).to_numpy()
    y_test = test_df[target].to_numpy()

    del folds[x]
    train_df = pd.concat(folds)
    X_train = train_df.drop(columns=[target]).to_numpy()
    y_train = train_df[target].to_numpy()

    autonet = AutoNetRegression( # "tiny_cs", # config preset
        log_level="info", budget_type="time",
        max_runtime=300, min_budget=1, max_budget=100
    )

    start = datetime.now().strftime("%H:%M:%S")
    perf = autonet.fit(
        X_train=X_train, Y_train=y_train,
        validation_split=0.25,
```

```
        early_stopping_patience=3
    )
    end = datetime.now().strftime("%H:%M:%S")

    preds = autonet.predict(X=X_test)

    perf["start"] = start
    perf["end"] = end
    perf["test_score"] = sklearn.metrics.mean_absolute_error(
        y_test, preds)

    perf = json.dumps(perf)
    f = open(fold_folder + "/perf.json", "w")
    f.write(perf)
    f.close()

    torch.save(autonet.get_pytorch_model(),
        fold_folder + "/model")
```

A.3.4 AutoGluon

```
from autogluon import TabularPrediction as task

for x in range(0, 10):
    fold_folder = "./data/cholesterol/autogluon/fold"+str(x+1)
    folds = [fold1, fold2, fold3, fold4, fold5,
             fold6, fold7, fold8, fold9, fold10]
    test_df = folds[x]
    del folds[x]
    train_df = pd.concat(folds)
    train_data = task.Dataset(train_df)
    start = datetime.now().strftime("%H:%M:%S")
    predictor = task.fit(
        train_data=train_data,
        label=label_column,
        eval_metric="mean_absolute_error",
        num_bagging_folds=5,
```

```
        output_directory=fold_folder,
    )
    end = datetime.now().strftime("%H:%M:%S")
    predictor.leaderboard().to_csv(fold_folder +
        "/leaderboard.csv")
    test_data = task.Dataset(test_df)
    y_test = test_data[label_column]
    y_pred = predictor.predict(test_data)
    perf = predictor.evaluate_predictions(
        y_true=y_test.to_numpy(), y_pred=y_pred,
        auxiliary_metrics=True
    )
    perf = dict(perf)
    perf["start"] = start
    perf["end"] = end
    perf = json.dumps(perf)
    f = open(fold_folder + "/perf.json", "w")
    f.write(perf)
    f.close()
```

A.3.5 H2O AutoML (Deep Learning)

```
import h2o
from h2o.automl import H2OAutoML

for x in range(0, 10):
    h2o.init()

    fold_folder = "./data/cholesterol/h2o-DL/fold" + str(x + 1)
    folds = [fold1, fold2, fold3, fold4, fold5,
             fold6, fold7, fold8, fold9, fold10]
    test_df = folds[x]
    test = h2o.H2OFrame(test_df)

    del folds[x]
    train_df = pd.concat(folds)
    train = h2o.H2OFrame(train_df)
```

```
x = train.columns
y = target
x.remove(y)

aml = H2OAutoML(
    seed=42,
    sort_metric="mae",
    nfolds=5,
    include_algos=["DeepLearning"],
    max_runtime_secs=3600,
)

start = datetime.now().strftime("%H:%M:%S")
aml.train(x=x, y=y, training_frame=train)
end = datetime.now().strftime("%H:%M:%S")

lb = aml.leaderboard.as_data_frame()
lb.to_csv(fold_folder + "/leaderboard.csv", index=False)

perf = aml.leader.model_performance(test)

perf = aml.training_info
perf["start"] = start
perf["end"] = end
perf["metric"] = aml.leader.model_performance(test).mae()

perf = json.dumps(perf)
f = open(fold_folder + "/perf.json", "w")
f.write(perf)
f.close()

my_local_model = h2o.download_model(aml.leader,
    path=fold_folder)

h2o.shutdown()
```

```
    import time

    time.sleep(5)
```

A.3.6 H2O AutoML (Without Deep Learning)

```
import h2o
from h2o.automl import H2OAutoML

for x in range(0, 10):
    h2o.init()

    fold_folder = "./data/cholesterol/h2o-no-DL/fold"+str(x+1)
    folds = [fold1, fold2, fold3, fold4, fold5,
             fold6, fold7, fold8, fold9, fold10]
    test_df = folds[x]
    test = h2o.H2OFrame(test_df)

    del folds[x]
    train_df = pd.concat(folds)
    train = h2o.H2OFrame(train_df)

    x = train.columns
    y = target
    x.remove(y)

    aml = H2OAutoML(
        seed=42,
        sort_metric="mae",
        nfolds=5,
        exclude_algos=["DeepLearning"],
        max_runtime_secs=3600,
    )

    start = datetime.now().strftime("%H:%M:%S")
    aml.train(x=x, y=y, training_frame=train)
    end = datetime.now().strftime("%H:%M:%S")
```

```
    lb = aml.leaderboard.as_data_frame()
    lb.to_csv(fold_folder + "/leaderboard.csv", index=False)

    perf = aml.leader.model_performance(test)

    perf = aml.training_info
    perf["start"] = start
    perf["end"] = end
    perf["metric"] = aml.leader.model_performance(test).mae()

    perf = json.dumps(perf)
    f = open(fold_folder + "/perf.json", "w")
    f.write(perf)
    f.close()

    my_local_model = h2o.download_model(aml.leader,
        path=fold_folder)

    h2o.shutdown()

    import time

    time.sleep(5)
```

A.3.7 H2O AutoML (XGBoost)

```
import h2o
from h2o.automl import H2OAutoML

for x in range(0, 10):
    h2o.init(port=54322)

    fold_folder = "./data/cholesterol/h2o-xgboost/fold"
        + str(x + 1)
    folds = [fold1, fold2, fold3, fold4, fold5,
             fold6, fold7, fold8, fold9, fold10]
```

```
    test_df = folds[x]
    test = h2o.H2OFrame(test_df)

    del folds[x]
    train_df = pd.concat(folds)
    train = h2o.H2OFrame(train_df)

    x = train.columns
    y = target
    x.remove(y)

    aml = H2OAutoML(
        seed=42,
        sort_metric="mae",
        nfolds=5,
        include_algos=["XGBoost"],
        max_runtime_secs=3600,
    )

    start = datetime.now().strftime("%H:%M:%S")
    aml.train(x=x, y=y, training_frame=train)
    end = datetime.now().strftime("%H:%M:%S")

    lb = aml.leaderboard.as_data_frame()
    lb.to_csv(fold_folder + "/leaderboard.csv", index=False)

    perf = aml.leader.model_performance(test)

    perf = aml.training_info
    perf["start"] = start
    perf["end"] = end
    perf["metric"] = aml.leader.model_performance(test).mae()

    perf = json.dumps(perf)
    f = open(fold_folder + "/perf.json", "w")
    f.write(perf)
    f.close()
```

```
    my_local_model = h2o.download_model(aml.leader,
        path=fold_folder)

    h2o.shutdown()

    import time

    time.sleep(5)
```

A.3.8 rminer

```
library(rminer)

for (x in 0:9) {
    fold_folder = paste("./data/cholesterol/rminer/fold",
        as.character(x+1), sep="")
    folds = list(fold1, fold2, fold3, fold4, fold5,
        fold6, fold7, fold8, fold9, fold10)
    test_df = folds[[(x+1)]]
    folds = folds[-(x+1)]
    train_df <- do.call(rbind, folds)

    inputs=ncol(train_df)-1


    sm=mparheuristic(model="automl3",n=NA,task=task,
        inputs=inputs)
    method=c("kfold",5,123)
    search=list(search=sm,smethod="auto",
        method=method,metric=metric,convex=0)

    M=fit(chol~.,data=train_df,model="auto",
        search=search,fdebug=TRUE)

    P=predict(M,test_df)
```

```
    test_metric = round(
        mmetric(test_df$chol,P,metric=metric),2
        )
    best_model = M@model
    best_model_index = match(best_model,M@mpar$LB$model)
    validation_metric = round(
        M@mpar$LB$eval,4)[best_model_index]

    results = paste('{"time":',
                    M@time,
                    ',"best_model":"',
                    best_model,
                    '","validation_metric":',
                    validation_metric,
                    ',"test_metric":',
                    test_metric,
                    '}',
                    sep = "")

    write(results, paste(fold_folder, "/perf.json", sep= ""))
    save(M, file = paste(fold_folder, "/model.RData", sep= ""))
}
```

A.3.9 rminer (XGBoost)

```
library(rminer)

for (x in 0:9) {
    fold_folder=paste("./data/cholesterol/rminer-xgboost/fold",
    as.character(x+1), sep="")
    folds = list(fold1, fold2, fold3, fold4,
        fold5, fold6, fold7, fold8, fold9, fold10)
    test_df = folds[[(x+1)]]
    folds = folds[-(x+1)]
    train_df <- do.call(rbind, folds)

    inputs=ncol(train_df)-1
```

```
    sm=mparheuristic(model="xgboost",n="heuristic10",
        task=task, inputs=inputs)
    method=c("kfold",5,123)
    search=list(search=sm,smethod="grid",
        method=method,metric=metric,convex=0)

    M=fit(chol~.,data=train_df,model="xgboost",
        search=search,fdebug=TRUE)

    P=predict(M,test_df)

    results = paste('{"time":',
                    M@time,
                    ',"validation_metric":',
                    M@error,
                    ',"test_metric":',
                    round(
                    mmetric(test_df$chol,P,metric=metric),
                    2),'}',
                    sep = "")

    write(results, paste(fold_folder, "/perf.json", sep= ""))
    save(M, file = paste(fold_folder, "/model.RData", sep= ""))
}
```

A.3.10 TPOT

```
from tpot import TPOTClassifier, TPOTRegressor

for x in range(0, 10):
    fold_folder = "./data/cholesterol/tpot/fold" + str(x + 1)
    folds = [fold1, fold2, fold3, fold4, fold5,
             fold6, fold7, fold8, fold9, fold10]
    test_df = folds[x]
    X_test = test_df.drop(columns=[target]).to_numpy()
    y_test = test_df[target].to_numpy()
```

```
    del folds[x]
    train_df = pd.concat(folds)
    X_train = train_df.drop(columns=[target]).to_numpy()
    y_train = train_df[target].to_numpy()

    tpot = TPOTRegressor(
        max_time_mins=60,
        verbosity=3,
        random_state=42,
        scoring="neg_mean_absolute_error",
        cv=5,
        n_jobs=-1,
        early_stop=3,
    )

    start = datetime.now().strftime("%H:%M:%S")
    tpot.fit(X_train, y_train)
    end = datetime.now().strftime("%H:%M:%S")

    tpot.export(fold_folder + "/pipeline.py")
    perf = {"score": tpot.score(X_test, y_test),
        "start": start, "end": end}
    perf = json.dumps(perf)
    f = open(fold_folder + "/perf.json", "w")
    f.write(perf)
    f.close()
```

A.3.11 TransmogrifAI

```
import com.salesforce.op._

object AzureBlobAnalysisv2 {
  def main(args: Array[String]) {
    LogManager.getLogger("com.salesforce.op")
        .setLevel(Level.ERROR)
    val conf = new SparkConf()
```

```
conf.setAppName("AutoMLForAll")

implicit val spark = SparkSession.builder.config(
    conf).getOrCreate()
val confh=new org.apache.hadoop.conf.Configuration()

val targetColumn = "chol"

for(fold <- 1 to 10){
  println("Fold: " + fold);

  var fold_folder="./data/cholesterol/transmografai/fold"+
    fold.toString()

  var train_df = spark.sqlContext.read.format("csv")
  .option(
    "header", "true").option(
    "inferSchema", "true").load(
    fold_folder + "/train.csv")

  var toBechanged = train_df.schema.fields.filter(
    x => x.dataType == IntegerType ||
    x.dataType == LongType)
  toBechanged.foreach({ row =>
  train_df = train_df.withColumn(row.name.concat("tmp"),
  train_df.col(row.name).cast(DoubleType))
    .drop(row.name)
    .withColumnRenamed(row.name.concat("tmp"), row.name)
  })

  var (saleprice, features) = FeatureBuilder.
    fromDataFrame[RealNN](
    train_df, response = targetColumn)
  var featureVector = features.toSeq.autoTransform()
  var checkedFeatures = saleprice.sanityCheck(
    featureVector, checkSample = 1.0,
    removeBadFeatures = true)
```

```
var pred = RegressionModelSelector.withCrossValidation(
  numFolds = 5,
  validationMetric = Evaluators.Regression.mae).setInput(
  saleprice, checkedFeatures).getOutput()

var wf = new OpWorkflow()

var start = Calendar.getInstance.getTime
var model = wf.setInputDataset(
  train_df).setResultFeatures(pred).train()
var end = Calendar.getInstance.getTime

print(model.summaryPretty())

var summary = model.summaryPretty()

val evaluator = Evaluators.Regression(
  ).setLabelCol(saleprice).setPredictionCol(pred)

var testData = spark.sqlContext.read.format(
  "csv").option("header", "true").option(
  "inferSchema", "true").load(fold_folder + "/test.csv")
var toBechanged2 = testData.schema.fields.filter(
  x => x.dataType == IntegerType ||
  x.dataType == LongType)
toBechanged2.foreach({ row =>
  testData = testData.withColumn(row.name.concat("tmp"),
  testData.col(row.name).cast(DoubleType))
    .drop(row.name)
    .withColumnRenamed(row.name.concat("tmp"), row.name)
})

var preds = model.setInputDataset(
testData).scoreAndEvaluate(evaluator).toString()

var w = new BufferedWriter(new FileWriter(
fold_folder + "/perf.txt"))
```

```
    w.write("START\n")
    w.write(start.toString())
    w.write("\n\n\n\nEND\n")
    w.write(end.toString())
    w.write("\n\n\n\nPREDS\n")
    w.write(preds)
    w.write("\n\n\n\nSUMMARY\n")
    w.write(summary)
    w.close()

    model.save(fold_folder + "/model")

  }

  // Read data as a DataFrame
  var passengersData = spark.sqlContext.read.format("csv")
    .option("header", "true")
    .option("inferSchema", "true")
    .load("cholesterol-train.csv")

  val passengersData = DataReaders.Simple.csvCase[Liver]
  (Option(
      "liver-disorders-train.csv")).readDataset().toDF()

  val targetColumn = spark.sparkContext.wholeTextFiles(
  "drinks").take(1)(0)._2
  val targetColumn = "class"
  //Convert Int and Long to Double to avoid
  //Feature Builder exception with Integer / Long Types
  val toBechanged = passengersData.schema.fields.filter(
  x => x.dataType == IntegerType || x.dataType == LongType)
  toBechanged.foreach({ row =>
    passengersData = passengersData.withColumn(
    row.name.concat("tmp"),
    passengersData.col(row.name).cast(DoubleType))
      .drop(row.name)
      .withColumnRenamed(row.name.concat("tmp"), row.name)
```

```
    })
    // Let's try to understand from the target variable
    // which ML problem we want to solve
    val view = passengersData.createOrReplaceTempView("myview")
    val countTarget = spark.sql(
    "SELECT COUNT(DISTINCT " + targetColumn + ") FROM myview")
    .take(
    1)(0).get(0).toString().toInt
    val targetType = passengersData.schema.fields.filter(
    x => x.name == targetColumn).take(1)(0).dataType
    // Max Distinct Values for Binary Classification
    // is 2 and for multi class is 30
    val binaryL: Int = 2
    val multiL: Int = 30

    //If the target variable has 2 distinct values
    // and it is numeric can be a binary classification
    if (countTarget == binaryL && targetType == DoubleType) {
      val (saleprice, features) = FeatureBuilder.
      fromDataFrame[RealNN](
      passengersData, response = targetColumn)
      val featureVector = features.toSeq.autoTransform()
      val checkedFeatures = saleprice.sanityCheck(
      featureVector,
      checkSample = 1.0, removeBadFeatures = true)
      val pred = BinaryClassificationModelSelector.
      withCrossValidation(
      numFolds = 5,
      validationMetric = Evaluators.
      BinaryClassification.auROC).setInput(
      saleprice, checkedFeatures).getOutput()
      val wf = new OpWorkflow()

      val start = Calendar.getInstance.getTime
      val model = wf.setInputDataset(
      passengersData).setResultFeatures(pred).train()
      val end = Calendar.getInstance.getTime
```

```
      print(model.summaryPretty())

      val evaluator = Evaluators.BinaryClassification(

      ).setLabelCol(saleprice).setPredictionCol(pred)

      model.setInputDataset(testData)
        .scoreAndEvaluate(evaluator)

      model.save("transmogrifai")
    }

    var testData = spark.sqlContext.read.format("csv").option(
    "header", "true").option(
    "inferSchema", "true").load(
    "/home/lferreira/autoautoml/data/mfeat/mfeat-test.csv")
    val toBechanged = testData.schema.fields.filter(
    x => x.dataType == IntegerType || x.dataType == LongType)
    toBechanged.foreach(
    { row =>
      testData = testData.withColumn(row.name.concat("tmp"),
      testData.col(row.name).cast(DoubleType))
        .drop(row.name)
        .withColumnRenamed(row.name.concat("tmp"), row.name)
    })
    model.setInputDataset(testData).scoreAndEvaluate(evaluator)
    spark.close()
  }
}
```

A.4 Citation

If you use the GitHub repository for your research, please cite the following paper:

- Ferreira, L., Pilastri, A. L., Martins, C. M., Pires, P. M., & Cortez, P. (2021). A Comparison of AutoML Tools for Machine Learning, Deep Learning and XGBoost. *International Joint Conference*

on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021, 1–8. https://doi.org/10.1109/IJCNN52387.2021.9534091

```
@inproceedings{DBLP:conf/ijcnn/FerreiraPMPC21,
  author    = {Lu{\'{\i}}s Ferreira and
               Andr{\'{e}} Luiz Pilastri and
               Carlos Manuel Martins and
               Pedro Miguel Pires and
               Paulo Cortez},
  title     = {A Comparison of AutoML Tools for Machine Learning,
  Deep Learning and XGBoost},
  booktitle = {International Joint Conference on Neural Networks,
  {IJCNN} 2021, Shenzhen, China, July 18-22, 2021},
  pages     = {1--8},
  publisher = {{IEEE}},
  year      = {2021},
  url       = {https://doi.org/10.1109/IJCNN52387.2021.9534091},
  doi       = {10.1109/IJCNN52387.2021.9534091}
}
```

Appendix B

AutoOC: A Python Module for Automated Multi-objective One-Class Classification

B.1 AutoOC: Automated One-Class Classification

This appendix presents the AutoOC open-source Python module, an automated and computationally efficient Grammatical Evolution (GE) approach that optimizes the hyperparameters of One-Class Classification (OCC) learners. AutoOC includes a multi-objective mode, optimizing both the OCC predictive performance and an efficiency measure (e.g., training time). By using GE, a biologically inspired evolutionary algorithm, AutoOC uses a formal grammar to select and tune the hyperparameters of five OCC base learners, namely Isolation Forest (IF), Local Outlier Factor (LOF), One-Class SVM (OC-SVM), Autoencoder (AE), and Variational Autoencoder (VAE). AutoOC also applies the NSGA-II algorithm to perform a multi-objective optimization, maximizing the predictive performance of the OCC learners while minimizing their training time or other of the provided efficiency objectives (e.g., prediction time). Figure 20 shows a high-level overview of the AutoOC method.

By instantiating its main Python class, AutoOC provides a set of functions to easily find the best and most efficient OCC models for a given dataset. The main steps carried out by AutoOC are problem formulation, data loading, model optimization, test set predictions, and model evaluation.

B.1.1 Problem Definition

The first step when using AutoOC is to provide information about the dataset and ML problem context (e.g., performance metrics). The possible options for problem definition are:

- Class definition: the adopted dataset encoding for the "anomaly" and "normal" classes (parameters `anomaly_class` and `normal_class`).
- Algorithm: which OCC learners can be created during the AutoOC optimizations. Even though there is currently a fixed number of setups, it is possible to easily add new OCC learners or use any

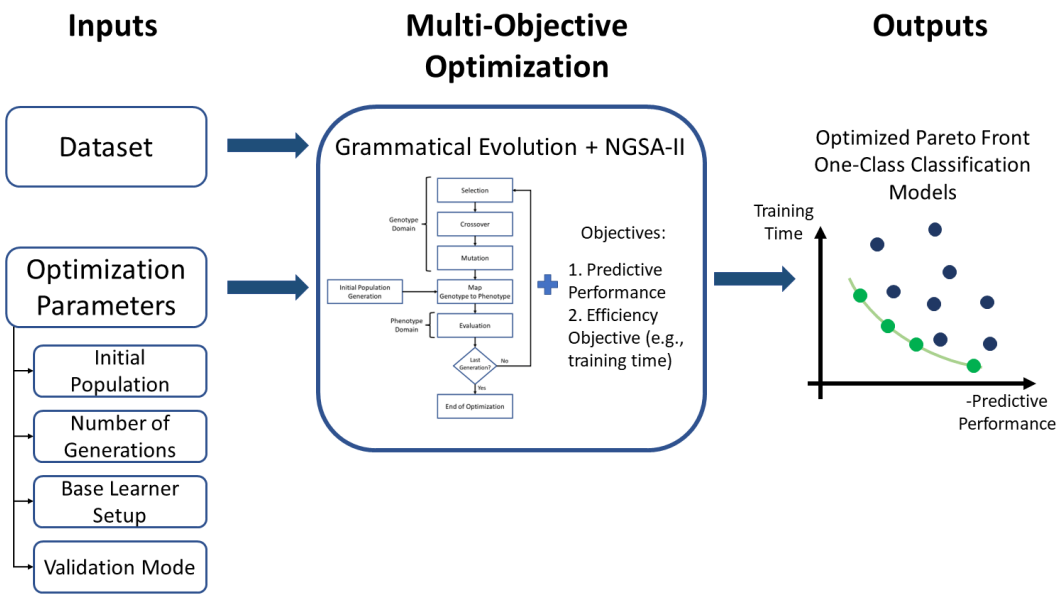


Figure 20: High-level overview of the AutoOC tool.

combination of the existing algorithms. The current learner setups (as of AutoOC version 0.0.14) are:

- `autoencoders`: uses Deep AE, from the TensorFlow library (TensorFlow, 2022a).
- `var`: uses VAE, from the TensorFlow library (TensorFlow, 2022b).
- `iforest`: uses IF, from the Scikit-Learn library (Scikit-Learn, 2022a).
- `lof`: uses LOF, from the Scikit-Learn library (Scikit-Learn, 2022b).
- `svm`: uses OC-SVM, from the Scikit-Learn library (Scikit-Learn, 2022c).
- `nas`: uses both AEs and VAEs, working as a NAS approach.
- `all`: uses all five OCC base learners.

- Multi-objective optimization: using the `multiobjective` boolean parameter, it is possible to choose a single-objective or multi-objective optimization. In a single-objective mode, only the predictive performance will be optimized (the adopted metric will depend on the usage of labeled or unlabeled validation data); for the multi-objective mode, an additional objective will be optimized, related to computational efficiency.

- Efficiency metric: when choosing the multi-objective optimization mode, it is possible to select which measure will be used as the efficiency objective. The possible options for the parameter `performance_metric` are:

  1. Training time: minimizes the training time of the individuals (OCC models).
  2. Predict time: minimizes the time it takes to predict one record from the validation data.

3. Number of parameters: minimizes the number of parameters of the ANN model, given by Keras `count_params()` function. Only available when using Deep Learning (DL) OCC learners (algorithm setups `autoencoders`, `var`, or `nas`).
4. **Bayesian Information Criterion (BIC)**: minimizes the value of the BIC (also known as Schwarz Information Criterion), a criterion used for model selection of a finite set of models (Schwarz, 1978).

- Multicore: when set to True, the optimization will run in a multicore manner, using all available processors. When set to False, AutoOC will only use a single core.

B.1.2 Data Loading

AutoOC is designed to optimize OCC learners, where the training data is composed only of "normal"records. Depending on the type of the provided validation data, the AutoOC optimization can work with two validation manners: unsupervised validation, where the model performance is evaluated using only unlabeled data (e.g., through an anomaly score); or supervised validation, where there is access to a labeled validation set to assess the model performance using supervised learning metrics, such as the AUC of the ROC curve classification measure (Fawcett, 2006). Table 34 summarizes the type of data used for each AutoOC validation setup.

Table 34: Required data for each AutoOC validation mode. Adapted from Ferreira and Cortez (2023).

Validation Mode	Training Data	Validation Data	Test Data
Supervised	Unlabeled Data (`X_train`)	Labeled Data (`X_val`, `y_val`)	Labeled Data (`X_test`, `y_test`)
Unsupervised	Unlabeled Data (`X_train`)	Unlabeled Data (`X_val`)	Labeled Data (`X_test`, `y_test`)

AutoOC provides the method `load_example_data()` to load the popular ECG dataset, provided by the Google API[1]. This method returns a dataset already preprocessed and split into training, validation, and test data that can be used to run AutoOC optimization.

B.1.3 Model Optimization

The execution of the AutoOC GE optimization is performed by the `fit()` function. This function accepts the following parameters:

- Supervised or unsupervised validation: the `fit()` function requires the user to provide the training data (`X_train`) and validation data. If the user only provides the validation set features (`X_val`), AutoOC will adopt the unsupervised mode; when the user provides a labeled set of targets (`y_val`), the supervised mode will be adopted by the optimization.

[1] http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv

- Population size: the `pop` parameter allows the definition of the GE initial population size (defaults to 100). In general terms, the population size represents the number of individuals (or candidate solutions) that will be created in each generation. The higher the value of `pop`, the more OCC models will be trained in each generation. For example, a `pop` equal to 100 means that AutoOC will optimize 100 OCC models in each GE generation.

- Number of generations: the `gen` parameter allows the definition of the GE number of generations (defaults to 100). A generation represents an iteration of the GE, where the individuals are replaced by new solutions (also named offspring). The higher the number of generations, the more iterations there will be during the AutoOC optimization execution. For example, if `pop` is equal to 100 and `gen` equal to 50, it means that the GE optimization will run during 50 iterations, training 100 OCC in each one (total of 5,000 models).

- Epochs: internal Keras parameter, representing the maximum number of epochs that each model based on DL learners (AE or VAE) will be trained on (defaults to 100).

- Early stopping: AutoOC provides two parameters related to early stopping of the GE optimization. The `early_stopping_tolerance` allows the definition of the tolerance value t used for early stopping (defaults to 0.01). The `early_stopping_rounds` parameter, when set to an integer value r, will stop the optimization if the performance does not improve by value t after r consecutive GE generations (defaults to False). We note that, for single-objective optimization, the early stopping value will be based on the chosen predictive metric (e.g., AUC); for multi-objective, the performance will be based on the hypervolume.

- "Always at hand": The boolean parameter `always_at_hand`, when set to True, keeps the current best OCC model (or the best Pareto front models for multi-objective optimization) during the execution of the GE optimization. This feature allows the usage of the current best OCC models, even if the GE optimization is still running. Defaults to False.

- Results path: all the metadata related to each AutoOC run is exported to the path related to parameter `results_path`. The metadata that is saved includes all the individual models generated during the optimization (all generations), leaderboards files including the evaluation of each generated solution, PDF reports with the evolution of predictive performance and efficiency across generations, and other metadata associated with the OCC models (e.g., images with the AEs and VAEs architectures).

- MLFlow integration: AutoOC is integrated with MLFlow (MLFlow, 2023), an open-source solution for experiment tracking and registry of ML models. AutoOC provides three parameters, to personalize the tracking URI (`mlflow_tracking_uri`), experiment name (`mlflow_experiment_name`), and run name (`mlflow_run_name`).

B.1.4 Test Set Predictions

The AutoOC `predict()` function uses the generated execution results to predict the labels on test data. The only required parameter is the data containing the test inputs (`X_test`). The optional `mode` parameter can be changed to select which individuals from the last generation are used to predict. The default value for this parameter is "all", which uses all individuals (OCC models) from the last GE generation; "best" – uses the model from the last generation that achieved the best predictive performance metric (e.g., highest validation AUC); "simplest" – uses the model from the last generation with the best efficiency metrics (e.g., lowest training time), only available when using multi-objective optimization; "pareto" – uses the non-dominated solutions (Pareto front) from the last generation (these are the models that achieved simultaneously the best predictive metric and efficiency metric), only for multi-objective optimization.

Additionally, the `threshold` parameter (only used for AEs and VAEs) can be used to set the threshold used for the prediction. The possible values for this parameter are: "mean": for each individual (DL model), the threshold value is the sum of the mean reconstruction error obtained on the validation data and one standard deviation (this is the default value for the `threshold` parameter); "percentile": for each model, the threshold value is the 95^{th} percentile of the reconstruction error obtained on the validation data (there is an additional `percentile` parameter to change the percentile); "max": for each model, the threshold value is maximum reconstruction error obtained on the validation data; it is also possible to use a fixed value for all models, by passing an Integer or Float value to the `threshold` parameter. In this case, the threshold value will be the same for all the models.

B.1.5 Model Evaluation

After making the predictions with the `predict()` function it is possible to manually calculate performance measures (e.g., AUC, accuracy). However, AutoOC provides the `evaluate()` function as a more convenient way to do it. It is possible to use the `mode` and `threshold` parameters (similarly to the `predict()` function) to directly predict and evaluate the performance of the AutoOC execution best models. Currently, the `evaluate()` function supports five predictive metrics from the Scikit-Learn library: "roc_auc", "accuracy", "precision", "recall", and "f1".

B.2 Code Example

A full example using the AutoOC Python package is shown below[2].

```
from autooc.autooc import AutoOC

"""
```

[2]This example code is also presented at the CodeOcean reproducible capsule: https://codeocean.com/capsule/7689106/tree/v3

```
Define the problem

In this example, the problem will be considered a
multiobjective problem, where the performance metric
is 'Training Time'. The 'anomaly' class is encoded as 0
and the 'normal' class is encoded as 1.
In this example, the AutoOC library will only use
Deep Autoencoders during the optimization search.
"""

aoc = AutoOC(anomaly_class=0,
             normal_class=1,
             multiobjective=True,
             performance_metric="training_time",
             algorithm="autoencoder"
             )


"""
Load the Data

AutoOC provides an example dataset, already preprocessed
and splitted.
The dataset is based on ECG data, provided by Google API:
storage.googleapis.com/download.tensorflow.org/data/ecg.csv
"""

X_train, X_val, X_test, y_test = aoc.load_example_data()


"""
Run AutoOC optimization

In this example, the .fit() method will run in the
'unsupervised' mode, as the y_val parameter was
not provided (otherwise, it would run in the
'supervised' mode). The example runs through 3
```

```
generations and initial population of 3 individuals,
with a maximum of 100 epochs per individual
(in this example, the optimization only generates
eep Autoencoders). The MLFlow and remaining metadata
are stored in the "results" folder.
"""
run = aoc.fit(
    X=X_train,
    X_val=X_val,
    pop=3,
    gen=3,
    epochs=100,
    mlflow_tracking_uri="../results",
    mlflow_experiment_name="test_experiment",
    mlflow_run_name="test_run",
    results_path="../results"
)
print(aoc.get_leaderboard())


"""
Make Predictions

In this example, the .predict() method is configured as "all",
meaning that all individuals from the last generation will be
used to predict on the test data.
The used threshold for the Autoencoders will be the associated
default value (in this case, the "mean" reconstruction error
obtained on validation data).
The predictions can later be used to evaluate the models
(e.g., through an AUC calculation).
"""

predictions = aoc.predict(X_test,
                          mode="all",
                          threshold="default")
```

```
"""
Evaluate Predictions

In this example, the .evaluate() method is configured
similarly to the .predict() method.
The function is using the ROC AUC metric, meaning that,
in this example, it will generate an AUC value for each of
the individuals from the last generation.
"""

score = aoc.evaluate(X_test,
                     y_test,
                     mode="all",
                     metric="roc_auc",
                     threshold="default")
print(f"Scores: {score}")
```

B.3 Impact on Academic Research

Two versions of AutoOC have been used in previous research works, namely two journal articles related to ML applications. Additionally, AutoOC has been published as a Python module[3] in June 2023, having more than 5,000 downloads[4].

First, a preliminary version of the software (named AutoOneClass) was proposed, associated with an Industry 4.0 PdM project (Ferreira et al., 2022). The goal was to predict the number of days until the next failure of an equipment and also determine if the equipments will fail in a fixed amount of days. In this first version, AutoOneClass adopted three OCC learners (AEs, IF, and OC-SVM) and it was applied to a recently collected dataset from a Portuguese software company. The results from AutoOneClass were compared with ten recent open-source AutoML technologies focused on a Supervised Learning and with two manual ML approaches. The AutoOneClass proposed method revealed competitive results, especially when compared with gloing.

Then, a second version of the software (named AutoOC) solution was developed and a robust benchmark with eight public OpenML datasets was performed to assess the performance of several scenarios, all focused on a multi-objective optimization (Ferreira & Cortez, 2023). In this new work, AutoOC already considered five distinct OCC base learners (AEs, IF, LOF, OC-SVM, and VAEs) and included includes two

[3] https://pypi.org/project/autooc
[4] https://pepy.tech/project/autooc

execution speedup mechanisms: a periodic training sampling and a multi-core fitness evaluation. AutoOC provided predictive results with high quality, outperforming a baseline IF for all the studied datasets and surpassing the best supervised public human modeling for two datasets.

It is worth noting that the current version of AutoOC already implements new features that were not available in the two previous versions that were used in both research works (e.g., distinct efficiency objectives, early stopping). Nevertheless, in the future, we aim to further extend the AutoOC capabilities by providing additional features, such as enhance the efficiency capabilities of AutoOC (e.g., by using Apache Spark or GPUs), add even more efficiency objectives (e.g., model size), and explore AutoML technologies to automate the phases of feature engineering and feature selection.