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

Luís Ferreira[a,b,*], André Pilastri[a], Filipe Romano[c], Paulo Cortez[b]

[a] *EPMQ - IT Engineering Maturity and Quality Lab, CCG ZGDV Institute, Guimaraes, Portugal*
[b] *ALGORITMI/LASI, Department of Information Systems, University of Minho, Guimaraes, Portugal*
[c] *Valuekeep, Braga, Portugal*

Abstract

Predictive Maintenance (PdM) is a critical area that is benefiting from the Industry 4.0 advent. Recently, several attempts have been made to apply Machine Learning (ML) to PdM, with the majority of the research studies assuming an expert-based ML modeling. In contrast with these works, this paper explores a purely Automated Machine Learning (AutoML) modeling for PdM under two main approaches. Firstly, we adapt and compare ten recent open-source AutoML technologies focused on a Supervised Learning. Secondly, we propose a novel AutoML approach focused on a One-Class (OC) Learning (AutoOneClass) that employs a Grammatical Evolution (GE) to search for the best PdM model using three types of learners (OC Support Vector Machines, Isolation Forests and deep Autoencoders). Using recently collected data from a Portuguese software company client, we performed a benchmark comparison study with the Supervised AutoML tools and the proposed AutoOneClass method 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. Overall, the results were close among the compared AutoML tools, with supervised AutoGluon obtaining the best results for all ML tasks. Moreover, the best supervised AutoML and AutoOneClass predictive results were compared with two manual ML modeling approaches

*Corresponding Author
Email addresses: luis.ferreira@dsi.uminho.pt (Luís Ferreira), andre.pilastri@ccg.pt (André Pilastri), filipe.romano@valuekeep.com (Filipe Romano), pcortez@dsi.uminho.pt (Paulo Cortez)

(using a ML expert and a non-ML expert), revealing competitive results.

Keywords: Automated Machine Learning, Predictive Maintenance, Supervised Learning, One-Class Learning.

1. Introduction

The Industry 4.0 phenomenon allowed companies to focus on analyzing historical data to obtain valuable insights. In particular, Predictive Maintenance (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 Machine Learning (ML) algorithms [1]. Indeed, some ML studies try to anticipate the failure of equipments (typically, manufacturing machines), aiming to reduce the costs of repairs [2, 3, 4, 5]. Other approaches [6, 7, 8, 9] 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 paper we focus on an Automated Machine Learning (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 One-Class (OC) Learning that uses a Grammatical Evolution (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 (non paper author) 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.

This work comprises a rather extended version of our previous work [10], thus including several new elements. Firstly, we perform an updated survey of the state-of-the-art works regarding the application of ML in PdM (Section 2). Secondly, we introduce and describe the new AutoOneClass framework (Section 3.2). Thirdly, the PdM dataset is presented with more depth (Sections 3.3 and 3.4), including information about missing and unique values and output target distributions. Fourthly, we perform additional computational experiments with more supervised AutoML tools and the proposed AutoOneClass method (Sections 4.1 and 4.2).

The main contributions of our work are summarized as follows:

(i) We propose AutoOneClass, an AutoML framework that focuses on OC Learning using three algorithms: deep Autoencoders (AE), Isolation Forests (IF), and One-Class Support Vector Machines (OC-SVM). AutoOneClass uses GE to optimize the search for the best OC 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 Area Under the Curve (AUC) of the Receiver Operating Characteristic (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 Deep Learning.

The paper is organized as follows. In Section 2, we present the related work. Next, Section 3 describes the supervised AutoML tools, the proposed AutoOneClass approach, and the analyzed PdM dataset. Then, Section 4 shows and discusses the experimental results. Finally, Section 5 presents the main conclusions and future work directions.

2. Related Work

Table 1 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 paper 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 1 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., Linear Regression, Decision Trees) are employed. Only five of the studies use Deep Learning (DL), but none of these use this type of ML algorithm exclusively. Moreover, only four studies adopted unsupervised ML algorithms [11, 12, 13, 18]. 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: [16] tries to classify the condition of the equipment (e.g., excellent, good) and [21] uses ML to suggest the type of maintenance needed for an equipment.

It terms of the ML modeling, the majority of the studies rely on a manual algorithm selection and hyperparameter tuning (Expert-based). There are only two works apart from this study that use AutoML: [19] is based on an existing AutoML framework (Auto-Sklearn), while [14] proposed an adaptation of the ML-Plan framework for PdM. In contrast with this paper, none of these studies compared more than one supervised AutoML tool. Moreover, the two works did not approach an unsupervised OC Learning, which is valuable for the PdM domain and that is here handled by using the proposed AutoOneClass method.

Table 1: Summary of the related work (ML applied to PdM).

Year	Ref.	ML Algorithms	FP	Real Data	ML Design
2017	[2]	ARIMA	✓	✓	Expert-based
2017	[3]	LR, LoR, DL, DT, RF, GBM	✓	✓	Expert-based
2018	[7]	GLM, RF GBM, DL	✓	n.d.	Expert-based
2018	[11]	K-means	✓	✓	Expert-based
2018	[4]	RF	✓	✓	Expert-based
2018	[12]	EM	✓	✓	Expert-based
2018	[13]	IF, LOF, OC-SVM	✓	✓	Expert-based
2020	[14]	AdaBoost RF	✓		AutoML
2020	[15]	SVM LogR	✓		Expert-based
2020	[16]	DL SVM		✓	Expert-based
2020	[17]	GBM RF	✓	✓	Expert-based
2020	[18]	DL, OC-SVM, XGB	✓	✓	Expert-based
2021	[19]	Classical ML DL Ensembles	✓	✓*	AutoML
2021	[20]	SVM, LDA, RF, DT, KNN	✓	✓	Expert-based
2021	[9]	RF, XGB, GBM, MLP, SVM, Adaboost	✓	✓	Expert-based
2022	[21]	DT		✓	Expert-based
2022	This work	Supervised and OC Learning	✓	✓	AutoML

ARIMA - Autoregressive Integrated Moving Average; DL - Deep Learning; DT - Decision Tree; EM - Expectation-Maximization; IF - Isolation Forest; GBM - Gradient Boosting Machine; 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).

3. Materials and Methods

3.1. Supervised AutoML Tools

In this article, 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 performed in [22]. In order to achieve a more fair comparison, we did not tune the hyperparameters of the AutoML tools. Table 2 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 are provided here:

Table 2: Description of the supervised AutoML tools (adapted from [10]).

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, XGBoost	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

- **Auto-Keras** is an AutoML Python library based on Keras [23]. It is designed to automate the construction on DL algorithms, usually named Automated Deep Learning (AutoDL) or Neural Architecture Search (NAS). Auto-Keras automatically tunes hyperparameters of Neural Networks, 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 [24].

- **Auto-Sklearn** is an AutoML library based on the popular Scikit-Learn framework. It uses Bayesian optimization, meta-learning, and Ensemble Learning modules to automate algorithm selection and hyperparameter tuning [25].

- **AutoGluon** is an AutoML toolkit based on the Gluon framework [26]. In this work, we only considered the tabular data module, which runs several algorithms and returns a Stacked Ensemble with multiple layers [27].

- **H2O AutoML** is the AutoML module from the H2O framework. H2O AutoML runs several algorithms from H2O and several Stacked Ensembles, with subsets of the trained ML models [28].

- **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, we used the "Perform" mode , since it is considered the most appropriate mode for a real-world usage [29].

- **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 [30].

- **rminer** is a library for the R programming language, focused on facilitating the usage of ML algorithms [31]. Rminer also provides AutoML functions that can be highly customized. This paper used the `"automl3"` template, which runs several ML algorithms and one Stacked Ensemble.

- **TPOT** is a Python AutoML tool that uses Genetic Programming to automate several phases of the ML workflow. It uses the Python Scikit-Learn framework to produce ML pipelines [32].

- **TransmogrifAI** is an end-to-end AutoML library written in Scala that runs on top of Apache Spark. It was created to increase ML efficiency through automation and an API that ensures compile-time type safety, modularity, and reuse [33].

3.2. AutoOneClass: Automated One-Class Learning

All the AutoML tools described in Section 3.1 apply Supervised Learning techniques (e.g., binary classification, regression). However, as explained in Section 2, there are PdM studies that focus on an Unsupervised Learning (e.g., [34, 12]). In particular, we focus on an OC Learning, such as adopted in [13, 18]. OC Learning 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) [35]. OC Learning 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 in [36, 37], 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 paper we propose AutoOneClass, an AutoML framework that focuses on a OC Learning. AutoOneClass uses GE to optimize the search for the OC 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 Au-

toOneClass implements OC Learning 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 OC Learning 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 OC Learning 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.

3.2.1. One-Class Learning Algorithms

AutoOneClass uses three popular OC Learning 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) [38], while both IF and OC-SVM were implemented using the Scikit-Learn framework [39].

AEs are a DL type that encode the input into a compressed representation (the latent space) and then decodes it back in order to reconstruct the original input as similar as possible to the original data [40]. AEs are used for several applications, such as dimensionality reduction or removing noise from data. AEs can be applied to OC Learning 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 [35]. Fig. 1 shows an example of an AE.

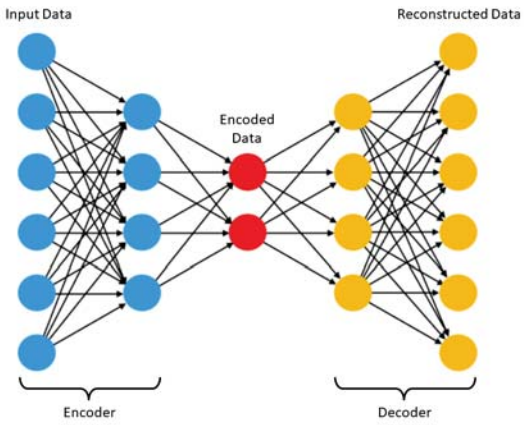


Figure 1: Example of an AE (the input data is encoded into a compressed representation and then it is decoded).

IF was proposed in 2008 [41] 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 "isolate" compared to normal points. Fig. 2 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.

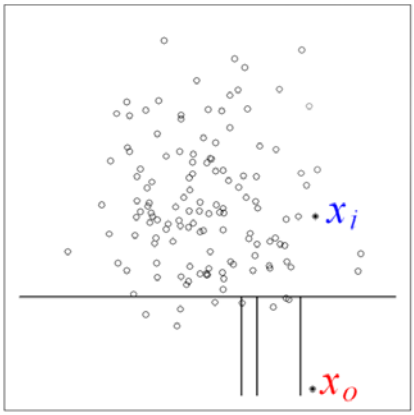


Figure 2: IF partitioning: x_0 is an anomaly (easily isolated) and x_i is a normal point (adapted from [41]).

OC-SVM is an extension of the Support Vector Machine (SVM) algorithm for unlabeled data [42, 43]. 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.

3.2.2. Grammatical Evolution

GE is an evolutionary algorithm proposed in 2001 [44]. Unlike Genetic Programming (GP), GE performs the evolutionary process on a provided grammar instead of on the actual programs. 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) [45].

For each generation, the evolutionary process of GE includes two main phases. The first phase is the evolution, where the algorithm generates new solutions using operations, such as crossovers and mutations. During the crossover operation, pairs of individuals are picked as parents and their genetic material is swapped to generate new individuals (children). The mu-

tation operation is applied after the crossover to the children individuals, usually consisting of randomly changing their genome to maintain genetic diversity. The second phase is the evaluation, where the population of individuals is evaluated using a fitness function. GE applies the evolution directly to the genome, while the evaluation is applied to the phenotype, which is obtained from the genome using a mapping process.

The GE mapping process uses the genome values to select production rules, usually in a 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. 3.

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

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

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

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

Figure 3: Example of a BNF grammar to generate strings.

In this paper, we built AutoOneClass using PonyGE2, an open source implementation of GE in Python [46] 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 One-Class Learning algorithms described in Section 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. 4.

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

```
<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 4: The PyBNF grammar used in this work.

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

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. 5.

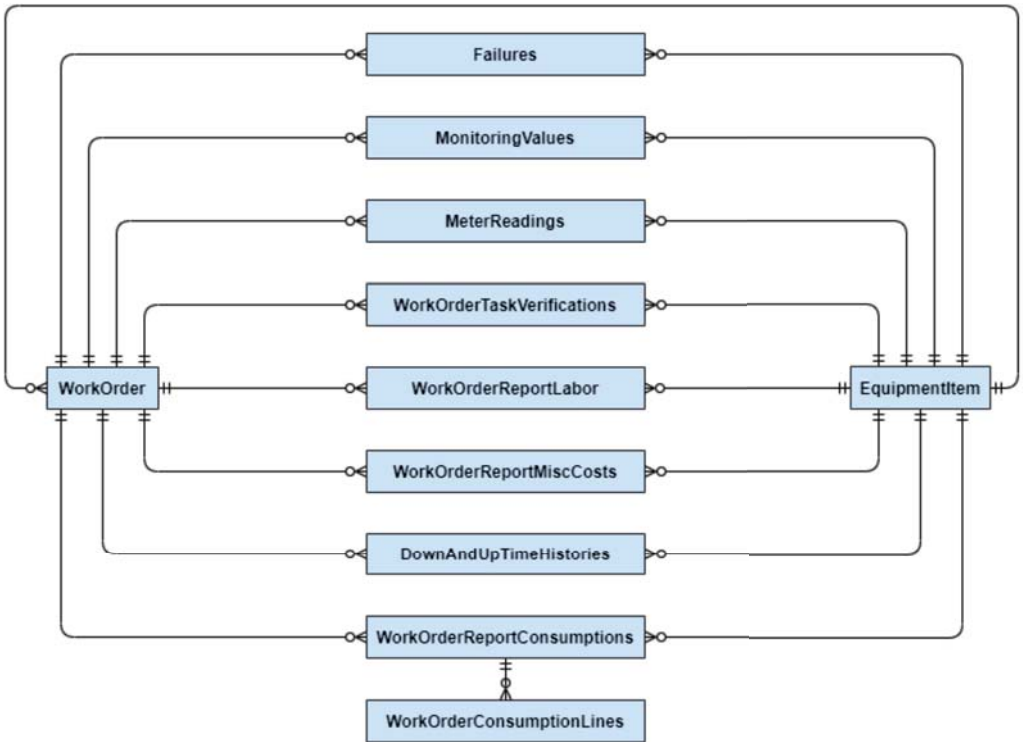


Figure 5: Entities and relationships between the datasets (adapted from [10]).

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.

Table 3 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**).

Table 3: Description of the equipment maintenance dataset attributes (adapted from [10]).

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

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. 6 shows the histogram for the regression target and the balancing of classes for the binary classification targets. Regarding the regression target

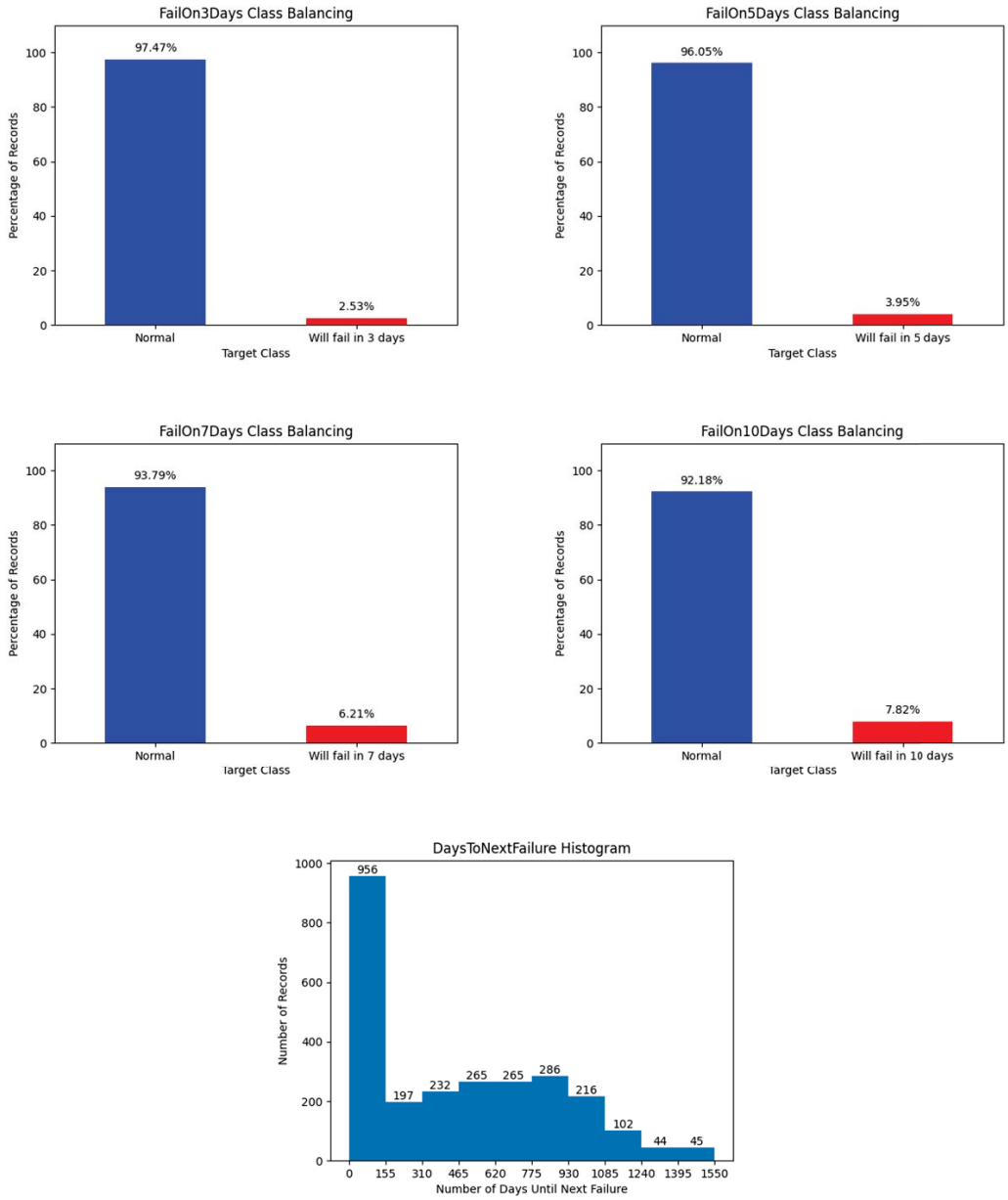


Figure 6: Balancing of the binary classification targets and histogram of the regression target.

(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. 6 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 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., [47]), imbalanced datasets are very prevalent in the PdM domain since failures are frequently sporadic compared to health situations.

3.4. Data Preprocessing

Since several data attributes are of the type String (as shown in Table 3), 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 4.

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 [48]. 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).

Table 4: 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

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.

3.5. Evaluation

In order to evaluate the results from the AutoML tools and AutoOneClass, we adopted a similar approach to the benchmark developed in [22]. 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 Mean Absolute Error (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. Results

4.1. AutoML Results

The first comparison focused on the supervised AutoML tools detailed in Section 3.1. For each AutoML tool, we executed five experiments, one for each target variable (DaysToNextFailure and FailOnxDays). Table 5 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 [49].

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

Table 5: Average predictive results obtained by the AutoML tools, best values for each target in **bold** (adapted from [10]).

		Targets				
		Days Unitl 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.

other hand, the worst 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 6. 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 6: 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., Decision Trees, Random Forest) 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 [50]. 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.

4.2. AutoOneClass Results

The second predictive comparison considers the AutoOneClass method, proposed and described in Section 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 7. We note that we adopted

Table 7: 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

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 8 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 and AutoML results (from Table 5). 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.

Table 8: 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.

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 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 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. 7.

Similar to the previous experiment, we also analyze the average training times for the AutoOneClass results, shown in Table 9. 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 8) and the AutoML results (shown in Table 5) shows that none of the AutoOneClass executions 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 super-

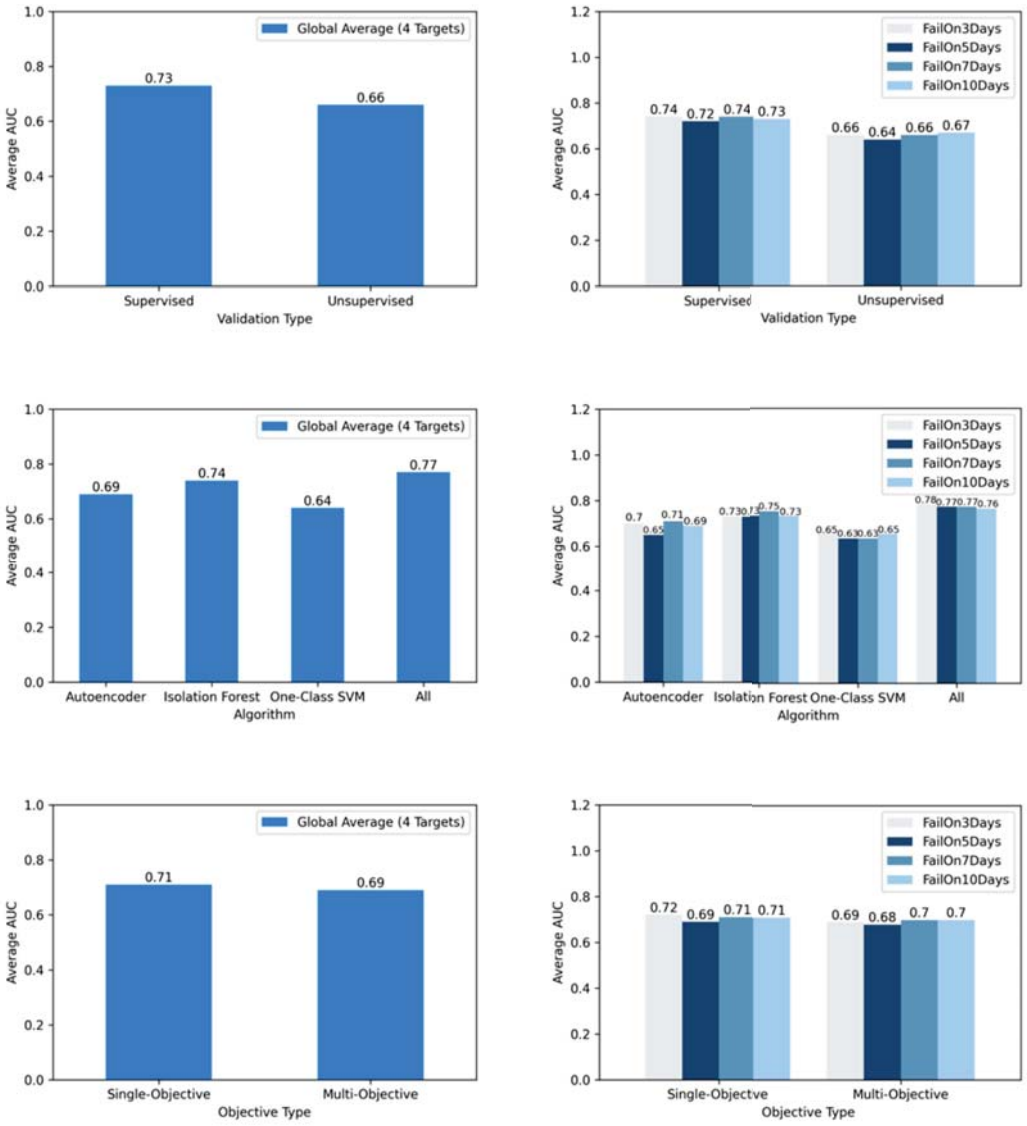


Figure 7: AutoOneClass results aggregated by validation type, algorithm, and optimization type, both globally (left) and per target (right).

vised validation), when compared with the supervised AutoML tools, which typically require a labeled dataset with a balanced ratio of normal and abnormal records. In many real-world PdM scenarios, there is a huge number of normal records and anomaly records might not always be available. Thus,

Table 9: 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.

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.

4.3. 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 10 compares the prediction results achieved

using the manual ML design and best AutoML tools. For each AutoML tool, Table 10 includes the algorithm (**Alg.**) that was most often the leader across the external folds (in rounded brackets). For the human modeling, Table 10 shows the best obtained result and the used algorithm.

Table 10: Comparison between the best AutoML results, AutoOneClass results, and human ML modeling results (expert and non-expert) for each target, best values in **bold** (adapted from [10]).

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 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 OC Learning 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.

5. Conclusions

PdM 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 paper, 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 OC Learning 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 iby the Supervised AutoML tools. However, the AutoOneClass surpassed the expert human modeling in two predictive targets and the performed OC Learning 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.

In future work, we intend to perform experiments with more datasets from the domain of PdM to verify further consistency with our results. We also intend to experiment with AutoML technologies that can automatically perform other ML phases apart from modeling, such as feature engineering and selection. Furthermore, regarding the AutoOneClass method, we plan to develop a benchmark with a more significant number of OC Learning algorithms (e.g., Local Outlier Factor, Gaussian Mixture Model, recently proposed OC Learning algorithms [51, 52, 53, 54, 55]) and datasets, including big data ones that should favor its multi-objective variant. Additionally, we aim to experiment different values for crossover, mutation and training time to assess their impact on the GE optimization. Also, for the multi-objective mode, we intend to analyze in more depth the correlation between the AUC and the anomaly scores when using a supervised validation. Finally, we wish to add more functionalities to AutoOneClass, such as the usage of other performance objectives (apart from training time) or the application of an early stopping to the GE optimization.

Acknowledgments

This work was executed under the project Cognitive CMMS - Cognitive Computerized Maintenance Management System, NUP: POCI-01-0247-FEDER-033574, co-funded by the Incentive System for Research and Technological Development, from the Thematic Operational Program Compet-

itiveness of the national framework program - Portugal2020. We wish to thank the anonymous reviewers for their helpful comments.

References

[1] A. J. Silva, P. Cortez, C. Pereira, A. Pilastri, Business analytics in industry 4.0: A systematic review, Expert Systems (2021) e12741doi:https://doi.org/10.1111/exsy.12741.

[2] A. Kanawaday, A. Sane, Machine learning for predictive maintenance of industrial machines using iot sensor data, in: 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), IEEE, 2017, pp. 87–90.

[3] B. Cline, R. S. Niculescu, D. Huffman, B. Deckel, Predictive maintenance applications for machine learning, in: 2017 annual reliability and maintainability symposium (RAMS), IEEE, 2017, pp. 1–7.

[4] M. Paolanti, L. Romeo, A. Felicetti, A. Mancini, E. Frontoni, J. Loncarski, Machine learning approach for predictive maintenance in industry 4.0, in: 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, MESA 2018, Oulu, Finland, July 2-4, 2018, IEEE, 2018, pp. 1–6. doi:10.1109/MESA.2018.8449150.

[5] T. P. Carvalho, F. A. A. M. N. Soares, R. Vita, R. da Piedade Francisco, J. P. T. V. Basto, S. G. S. Alcalá, A systematic literature review of machine learning methods applied to predictive maintenance, Comput. Ind. Eng. 137 (2019). doi:10.1016/j.cie.2019.106024.

[6] Massimiliano De Benedetti and Fabio Leonardi and Fabrizio Messina and Corrado Santoro and Athanasios V. Vasilakos, Anomaly detection and predictive maintenance for photovoltaic systems, Neurocomputing 310 (2018) 59–68. doi:10.1016/j.neucom.2018.05.017.

[7] S. Butte, A. Prashanth, S. Patil, Machine learning based predictive maintenance strategy: a super learning approach with deep neural networks, in: 2018 IEEE Workshop on Microelectronics and Electron Devices (WMED), IEEE, 2018, pp. 1–5.

[8] Z. M. Çınar, A. Abdussalam Nuhu, Q. Zeeshan, O. Korhan, M. Asmael, B. Safaei, Machine learning in predictive maintenance towards sustainable smart manufacturing in industry 4.0, Sustainability 12 (19) (2020) 8211.

[9] S. Ayvaz, K. Alpay, Predictive aintenance system for production lines in manufacturing: A machine learning approach using iot data in real-time, Expert Systems with Applications 173 (2021) 114598. doi:10.1016/j.eswa.2021.114598.

[10] L. Ferreira, A. L. Pilastri, V. Sousa, F. Romano, P. Cortez, Prediction of maintenance equipment failures using automated machine learning, in: Intelligent Data Engineering and Automated Learning - IDEAL 2021 - 22nd International Conference, IDEAL 2021, Manchester, UK, November 25-27, 2021, Proceedings, Vol. 13113 of Lecture Notes in Computer Science, Springer, 2021, pp. 259–267. doi:10.1007/978-3-030-91608-4_26.

[11] N. Amruthnath, T. Gupta, A research study on unsupervised machine learning algorithms for early fault detection in predictive maintenance, in: 2018 5th international conference on industrial engineering and applications (ICIEA), IEEE, 2018, pp. 355–361. doi:10.1109/IEA.2018.8387124.

[12] S. Cho, G. May, I. Tourkogiorgis, R. Perez, Ó. Lázaro, B. de la Maza, D. Kiritsis, A hybrid machine learning approach for predictive maintenance in smart factories of the future, in: 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, Vol. 536 of IFIP Advances in Information and Communication Technology, Springer, 2018, pp. 311–317. doi:10.1007/978-3-319-99707-0_39.

[13] P. Straus, M. Schmitz, R. Wöstmann, J. Deuse, Enabling of predictive maintenance in the brownfield through low-cost sensors, an iiot-architecture and machine learning, in: IEEE International Conference on Big Data (IEEE BigData 2018), Seattle, WA, USA, December 10-13, 2018, IEEE, 2018, pp. 1474–1483. doi:10.1109/BigData.2018.8622076.

[14] T. Tornede, A. Tornede, M. Wever, F. Mohr, E. Hüllermeier, Automl for predictive maintenance: One tool to RUL them all, in: 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, Vol. 1325 of Communications in Computer and Information Science, Springer, 2020, pp. 106–118. doi:10.1007/978-3-030-66770-2_8.

[15] H. A. Gohel, H. Upadhyay, L. Lagos, K. Cooper, A. Sanzetenea, Predictive maintenance architecture development for nuclear infrastructure using machine learning, Nuclear Engineering and Technology 52 (7) (2020) 1436–1442. doi:10.1016/j.net.2019.12.029.

[16] J. C. Cheng, W. Chen, K. Chen, Q. Wang, Data-driven predictive maintenance planning framework for mep components based on bim and iot using machine learning algorithms, Automation in Construction 112 (2020) 103087. doi:10.1016/j.autcon.2020.103087.

[17] M. Calabrese, M. Cimmino, F. Fiume, M. Manfrin, L. Romeo, S. Ceccacci, M. Paolanti, G. Toscano, G. Ciandrini, A. Carrotta, M. Mengoni, E. Frontoni, D. Kapetis, SOPHIA: an event-based iot and machine learning architecture for predictive maintenance in industry 4.0, Inf. 11 (4) (2020) 202. doi:10.3390/info11040202.

[18] G. Makridis, D. Kyriazis, S. Plitsos, Predictive maintenance leveraging machine learning for time-series forecasting in the maritime industry, in: 23rd IEEE International Conference on Intelligent Transportation Systems, ITSC 2020, Rhodes, Greece, September 20-23, 2020, IEEE, 2020, pp. 1–8. doi:10.1109/ITSC45102.2020.9294450.

[19] J. Larocque-Villiers, P. Dumond, D. Knox, Automating predictive maintenance using state-based transfer learning and ensemble methods, in: 14th IEEE International Symposium on Robotic and Sensors Environments, ROSE 2021, Virtual Event, October 28-29, 2021, IEEE, 2021, pp. 1–7. doi:10.1109/ROSE52750.2021.9611768.

[20] M. Çakir, M. A. Güvenç, S. Mistikoglu, The experimental application of popular machine learning algorithms on predictive maintenance and the design of iiot based condition monitoring system, Comput. Ind. Eng. 151 (2021) 106948. doi:10.1016/j.cie.2020.106948.

[21] S. Arena, E. Florian, I. Zennaro, P. Orrù, F. Sgarbossa, A novel decision support system for managing predictive maintenance strategies based on machine learning approaches, Safety science 146 (2022) 105529. doi:10.1016/j.ssci.2021.105529.

[22] L. Ferreira, A. L. Pilastri, C. M. Martins, P. M. Pires, P. Cortez, A comparison of automl tools for machine learning, deep learning and xgboost, in: International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021, IEEE, 2021, pp. 1–8. doi:10.1109/IJCNN52387.2021.9534091.

[23] H. Jin, Q. Song, X. Hu, Auto-keras: An efficient neural architecture search system, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, 2019, pp. 1946–1956.

[24] L. Zimmer, M. Lindauer, F. Hutter, Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl, IEEE Trans. Pattern Anal. Mach. Intell. 43 (9) (2021) 3079–3090. doi:10.1109/TPAMI.2021.3067763.

[25] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, F. Hutter, Auto-sklearn: Efficient and robust automated machine learning, in: F. Hutter, L. Kotthoff, J. Vanschoren (Eds.), Automated Machine Learning - Methods, Systems, Challenges, The Springer Series on Challenges in Machine Learning, Springer, 2019, pp. 113–134. doi:10.1007/978-3-030-05318-5_6.

[26] Auto-Gluon, AutoGluon: AutoML Toolkit for Deep Learning — AutoGluon Documentation 0.2.0 documentation (2021).
URL `https://auto.gluon.ai/`

[27] Qi, Wenwen and Xu, Chong and Xu, Xiwei, AutoGluon: A revolutionary framework for landslide hazard analysis, Natural Hazards Research 1 (3) (2021) 103–108. doi:10.1016/j.nhres.2021.07.002.

[28] H2O.ai, H2O AutoML, h2O version 3.32.1.3 (2021).
URL `http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html`

[29] A. Płońska, P. Płoński, Mljar: State-of-the-art automated machine learning framework for tabular data. version 0.10.3 (2022).
URL https://github.com/mljar/mljar-supervised

[30] M. Ali, PyCaret: An open source, low-code machine learning library in Python, pyCaret version 2.3.10 (June 2022).
URL https://www.pycaret.org

[31] P. Cortez, Data mining with neural networks and support vector machines using the r/rminer tool, in: Industrial conference on data mining, Springer, 2010, pp. 572–583.

[32] T. T. Le, W. Fu, J. H. Moore, Scaling tree-based automated machine learning to biomedical big data with a feature set selector, Bioinformatics 36 (1) (2020) 250–256.

[33] Salesforce, Transmogrifai - automl library for building modular, reusable, strongly typed machine learning workflows on spark from salesforce engineering (2022).
URL https://transmogrif.ai/

[34] N. Amruthnath, T. Gupta, Fault class prediction in unsupervised learning using model-based clustering approach, in: 2018 International Conference on Information and Computer Technologies (ICICT), IEEE, 2018, pp. 5–12. doi:10.1109/INFOCT.2018.8356831.

[35] D. Ribeiro, L. M. Matos, G. Moreira, A. L. Pilastri, P. Cortez, Isolation forests and deep autoencoders for industrial screw tightening anomaly detection, Computers 11 (4) (2022) 54. doi:10.3390/computers11040054.

[36] X. He, K. Zhao, X. Chu, Automl: A survey of the state-of-the-art, Knowl. Based Syst. 212 (2021) 106622. doi:10.1016/j.knosys.2020.106622.

[37] M. Bahri, F. Salutari, A. Putina, M. Sozio, Automl: state of the art with a focus on anomaly detection, challenges, and research directions, International Journal of Data Science and Analytics (2022) 1–14doi:10.1007/s41060-022-00309-0.

[38] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow,

A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL `https://www.tensorflow.org/`

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

[40] Sepehr Maleki and Sasan Maleki and Nicholas R. Jennings, Unsupervised anomaly detection with LSTM autoencoders using statistical data-filtering, Appl. Soft Comput. 108 (2021) 107443. doi:10.1016/j.asoc.2021.107443.

[41] F. T. Liu, K. M. Ting, Z. Zhou, Isolation forest, in: Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy, IEEE Computer Society, 2008, pp. 413–422. doi:10.1109/ICDM.2008.17.

[42] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, R. C. Williamson, Estimating the support of a high-dimensional distribution, Neural Comput. 13 (7) (2001) 1443–1471. doi:10.1162/089976601750264965.

[43] Siqi Wang and Qiang Liu and En Zhu and Fatih Porikli and Jianping Yin, Hyperparameter selection of one-class support vector machine by self-adaptive data shifting, Pattern Recognit. 74 (2018) 198–211. doi:10.1016/j.patcog.2017.09.012.

[44] M. O'Neill, C. Ryan, Grammatical evolution, IEEE Trans. Evol. Comput. 5 (4) (2001) 349–358. doi:10.1109/4235.942529.

[45] P. J. Pereira, P. Cortez, R. Mendes, Multi-objective grammatical evolution of decision trees for mobile marketing user conversion prediction, Expert Syst. Appl. 168 (2021) 114287. doi:10.1016/j.eswa.2020.114287.

[46] M. Fenton, J. McDermott, D. Fagan, S. Forstenlechner, E. Hemberg, M. O'Neill, 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, ACM, 2017, pp. 1194–1201. doi:10.1145/3067695.3082469.

[47] M. D. Dangut, Z. Skaf, I. K. Jennions, Handling imbalanced data for aircraft predictive maintenance using the bache algorithm, Applied Soft Computing (2022) 108924doi:10.1016/j.asoc.2022.108924.

[48] L. M. Matos, J. Azevedo, A. Matta, A. Pilastri, P. Cortez, R. Mendes, Categorical attribute transformation environment (cane): A python module for categorical to numeric data preprocessing, Software Impacts 13 (2022) 100359. doi:https://doi.org/10.1016/j.simpa.2022.100359.

[49] Hollander, Myles and Wolfe, Douglas A and Chicken, Eric, Nonparametric statistical methods, John Wiley & Sons, 2013.

[50] A. Ng, Machine Learning Yearning, deeplearning.ai, 2020.
URL `https://www.deeplearning.ai/machine-learning-yearning/`

[51] Izhak Golan and Ran El-Yaniv, Deep Anomaly Detection Using Geometric Transformations, in: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, 2018, pp. 9781–9791.

[52] Stefano Mauceri and James Sweeney and James McDermott, Dissimilarity-based representations for one-class classification on time series, Pattern Recognit. 100 (2020) 107122. doi:10.1016/j.patcog.2019.107122.

[53] Oliver Urs Lenz and Daniel Peralta and Chris Cornelis, Average Localised Proximity: A new data descriptor with good default one-class classification performance, Pattern Recognit. 118 (2021) 107991. doi:10.1016/j.patcog.2021.107991.

[54] Toshitaka Hayashi and Hamido Fujita and Andres Hernandez-Matamoros, Less complexity one-class classification approach using construction error of convolutional image transformation network, Inf. Sci. 560 (2021) 217–234. doi:10.1016/j.ins.2021.01.069.

[55] Ane Blázquez-García and Angel Conde and Usue Mori and José Antonio Lozano, Water leak detection using self-supervised time series classification, Inf. Sci. 574 (2021) 528–541. doi:10.1016/j.ins.2021.06.015.

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

Luís Ferreira[a,b,*], André Pilastri[a], Filipe Romano[c], Paulo Cortez[b]

[a] *EPMQ - IT Engineering Maturity and Quality Lab, CCG ZGDV Institute, Guimaraes, Portugal*
[b] *ALGORITMI/LASI, Department of Information Systems, University of Minho, Guimaraes, Portugal*
[c] *Valuekeep, Braga, Portugal*

Abstract

Predictive Maintenance (PdM) is a critical area that is benefiting from the Industry 4.0 advent. Recently, several attempts have been made to apply Machine Learning (ML) to PdM, with the majority of the research studies assuming an expert-based ML modeling. In contrast with these works, this paper explores a purely Automated Machine Learning (AutoML) modeling for PdM under two main approaches. Firstly, we adapt and compare ten recent open-source AutoML technologies focused on a Supervised Learning. Secondly, we propose a novel AutoML approach focused on a One-Class (OC) Learning (AutoOneClass) that employs a Grammatical Evolution (GE) to search for the best PdM model using three types of learners (OC Support Vector Machines, Isolation Forests and deep Autoencoders). Using recently collected data from a Portuguese software company client, we performed a benchmark comparison study with the Supervised AutoML tools and the proposed AutoOneClass method 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. Overall, the results were close among the compared AutoML tools, with supervised AutoGluon obtaining the best results for all ML tasks. Moreover, the best supervised AutoML and AutoOneClass predictive results were compared with two manual ML modeling approaches

*Corresponding Author
Email addresses: luis.ferreira@dsi.uminho.pt (Luís Ferreira), andre.pilastri@ccg.pt (André Pilastri), filipe.romano@valuekeep.com (Filipe Romano), pcortez@dsi.uminho.pt (Paulo Cortez)

(using a ML expert and a non-ML expert), revealing competitive results.

Keywords: Automated Machine Learning, Predictive Maintenance, Supervised Learning, One-Class Learning.

---

1. Introduction

The Industry 4.0 phenomenon allowed companies to focus on analyzing historical data to obtain valuable insights. In particular, Predictive Maintenance (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 Machine Learning (ML) algorithms [1]. Indeed, some ML studies try to anticipate the failure of equipments (typically, manufacturing machines), aiming to reduce the costs of repairs [2, 3, 4, 5]. Other approaches [6, 7, 8, 9] 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 paper we focus on an Automated Machine Learning (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 One-Class (OC) Learning that uses a Grammatical Evolution (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 (non paper author) 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.

This work comprises a rather extended version of our previous work [10], thus including several new elements. Firstly, we perform an updated survey of the state-of-the-art works regarding the application of ML in PdM (Section 2). Secondly, we introduce and describe the new AutoOneClass framework (Section 3.2). Thirdly, the PdM dataset is presented with more depth (Sections 3.3 and 3.4), including information about missing and unique values and output target distributions. Fourthly, we perform additional computational experiments with more supervised AutoML tools and the proposed AutoOneClass method (Sections 4.1 and 4.2).

The main contributions of our work are summarized as follows:

(i) We propose AutoOneClass, an AutoML framework that focuses on OC Learning using three algorithms: deep Autoencoders (AE), Isolation Forests (IF), and One-Class Support Vector Machines (OC-SVM). AutoOneClass uses GE to optimize the search for the best OC 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 Area Under the Curve (AUC) of the Receiver Operating Characteristic (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 Deep Learning.

The paper is organized as follows. In Section 2, we present the related work. Next, Section 3 describes the supervised AutoML tools, the proposed AutoOneClass approach, and the analyzed PdM dataset. Then, Section 4 shows and discusses the experimental results. Finally, Section 5 presents the main conclusions and future work directions.

2. Related Work

Table 1 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 paper 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 1 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., Linear Regression, Decision Trees) are employed. Only five of the studies use Deep Learning (DL), but none of these use this type of ML algorithm exclusively. Moreover, only four studies adopted unsupervised ML algorithms [11, 12, 13, 18]. 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: [16] tries to classify the condition of the equipment (e.g., excellent, good) and [21] uses ML to suggest the type of maintenance needed for an equipment.

It terms of the ML modeling, the majority of the studies rely on a manual algorithm selection and hyperparameter tuning (Expert-based). There are only two works apart from this study that use AutoML: [19] is based on an existing AutoML framework (Auto-Sklearn), while [14] proposed an adaptation of the ML-Plan framework for PdM. In contrast with this paper, none of these studies compared more than one supervised AutoML tool. Moreover, the two works did not approach an unsupervised OC Learning, which is valuable for the PdM domain and that is here handled by using the proposed AutoOneClass method.

Table 1: Summary of the related work (ML applied to PdM).

Year	Ref.	ML Algorithms	FP	Real Data	ML Design
2017	[2]	ARIMA	✓	✓	Expert-based
2017	[3]	LR, LoR, DL, DT, RF, GBM	✓	✓	Expert-based
2018	[7]	GLM, RF GBM, DL	✓	n.d.	Expert-based
2018	[11]	K-means	✓	✓	Expert-based
2018	[4]	RF	✓	✓	Expert-based
2018	[12]	EM	✓	✓	Expert-based
2018	[13]	IF, LOF, OC-SVM	✓	✓	Expert-based
2020	[14]	AdaBoost RF	✓		AutoML
2020	[15]	SVM LogR	✓		Expert-based
2020	[16]	DL SVM		✓	Expert-based
2020	[17]	GBM RF	✓	✓	Expert-based
2020	[18]	DL, OC-SVM, XGB	✓	✓	Expert-based
2021	[19]	Classical ML DL Ensembles	✓	✓*	AutoML
2021	[20]	SVM, LDA, RF, DT, KNN	✓	✓	Expert-based
2021	[9]	RF, XGB, GBM, MLP, SVM, Adaboost	✓	✓	Expert-based
2022	[21]	DT		✓	Expert-based
2022	This work	Supervised and OC Learning	✓	✓	AutoML

ARIMA - Autoregressive Integrated Moving Average; DL - Deep Learning; DT - Decision Tree; EM - Expectation-Maximization; IF - Isolation Forest; GBM - Gradient Boosting Machine; 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).

3. Materials and Methods

3.1. Supervised AutoML Tools

In this article, 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 performed in [22]. In order to achieve a more fair comparison, we did not tune the hyperparameters of the AutoML tools. Table 2 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 are provided here:

Table 2: Description of the supervised AutoML tools (adapted from [10]).

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, XGBoost	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

- **Auto-Keras** is an AutoML Python library based on Keras [23]. It is designed to automate the construction on DL algorithms, usually named Automated Deep Learning (AutoDL) or Neural Architecture Search (NAS). Auto-Keras automatically tunes hyperparameters of Neural Networks, 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 [24].

- **Auto-Sklearn** is an AutoML library based on the popular Scikit-Learn framework. It uses Bayesian optimization, meta-learning, and Ensemble Learning modules to automate algorithm selection and hyperparameter tuning [25].

- **AutoGluon** is an AutoML toolkit based on the Gluon framework [26]. In this work, we only considered the tabular data module, which runs several algorithms and returns a Stacked Ensemble with multiple layers [27].

- **H2O AutoML** is the AutoML module from the H2O framework. H2O AutoML runs several algorithms from H2O and several Stacked Ensembles, with subsets of the trained ML models [28].

- **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, we used the "Perform" mode , since it is considered the most appropriate mode for a real-world usage [29].

- **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 [30].

- **rminer** is a library for the R programming language, focused on facilitating the usage of ML algorithms [31]. Rminer also provides AutoML functions that can be highly customized. This paper used the `"automl3"` template, which runs several ML algorithms and one Stacked Ensemble.

- **TPOT** is a Python AutoML tool that uses Genetic Programming to automate several phases of the ML workflow. It uses the Python Scikit-Learn framework to produce ML pipelines [32].

- **TransmogrifAI** is an end-to-end AutoML library written in Scala that runs on top of Apache Spark. It was created to increase ML efficiency through automation and an API that ensures compile-time type safety, modularity, and reuse [33].

3.2. AutoOneClass: Automated One-Class Learning

All the AutoML tools described in Section 3.1 apply Supervised Learning techniques (e.g., binary classification, regression). However, as explained in Section 2, there are PdM studies that focus on an Unsupervised Learning (e.g., [34, 12]). In particular, we focus on an OC Learning, such as adopted in [13, 18]. OC Learning 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) [35]. OC Learning 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 in [36, 37], 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 paper we propose AutoOneClass, an AutoML framework that focuses on a OC Learning. AutoOneClass uses GE to optimize the search for the OC 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 Au-

toOneClass implements OC Learning 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 OC Learning 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 OC Learning 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.

3.2.1. One-Class Learning Algorithms

AutoOneClass uses three popular OC Learning 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) [38], while both IF and OC-SVM were implemented using the Scikit-Learn framework [39].

AEs are a DL type that encode the input into a compressed representation (the latent space) and then decodes it back in order to reconstruct the original input as similar as possible to the original data [40]. AEs are used for several applications, such as dimensionality reduction or removing noise from data. AEs can be applied to OC Learning 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 [35]. Fig. 1 shows an example of an AE.

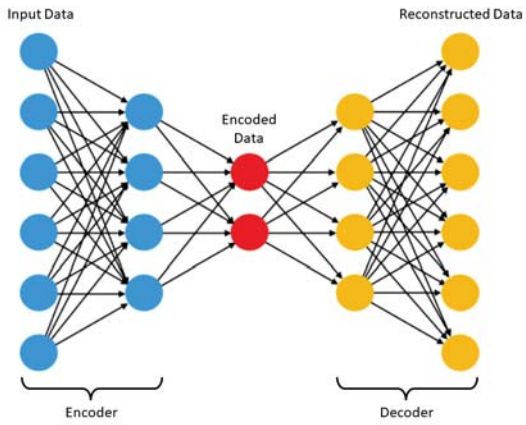


Figure 1: Example of an AE (the input data is encoded into a compressed representation and then it is decoded).

IF was proposed in 2008 [41] 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 "isolate" compared to normal points. Fig. 2 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.

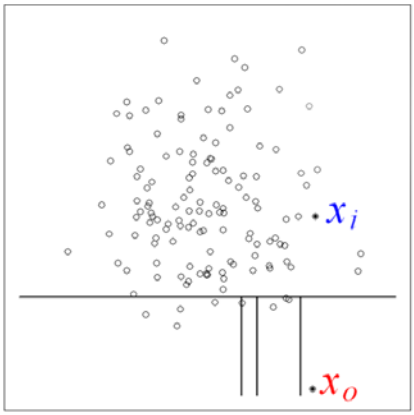


Figure 2: IF partitioning: x_0 is an anomaly (easily isolated) and x_i is a normal point (adapted from [41]).

OC-SVM is an extension of the Support Vector Machine (SVM) algorithm for unlabeled data [42, 43]. 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.

3.2.2. Grammatical Evolution

GE is an evolutionary algorithm proposed in 2001 [44]. Unlike Genetic Programming (GP), GE performs the evolutionary process on a provided grammar instead of on the actual programs. 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) [45].

For each generation, the evolutionary process of GE includes two main phases. The first phase is the evolution, where the algorithm generates new solutions using operations, such as crossovers and mutations. During the crossover operation, pairs of individuals are picked as parents and their genetic material is swapped to generate new individuals (children). The mu-

tation operation is applied after the crossover to the children individuals, usually consisting of randomly changing their genome to maintain genetic diversity. The second phase is the evaluation, where the population of individuals is evaluated using a fitness function. GE applies the evolution directly to the genome, while the evaluation is applied to the phenotype, which is obtained from the genome using a mapping process.

The GE mapping process uses the genome values to select production rules, usually in a 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. 3.

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

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

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

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

Figure 3: Example of a BNF grammar to generate strings.

In this paper, we built AutoOneClass using PonyGE2, an open source implementation of GE in Python [46] 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 One-Class Learning algorithms described in Section 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. 4.

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

```
<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 4: The PyBNF grammar used in this work.

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

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. 5.

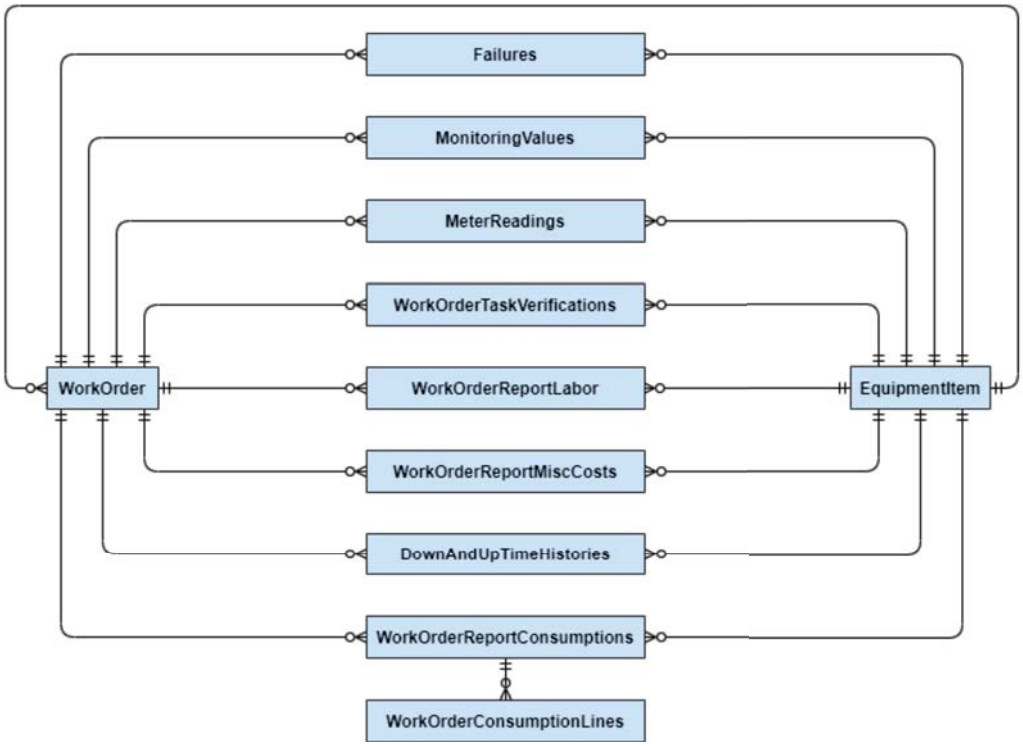


Figure 5: Entities and relationships between the datasets (adapted from [10]).

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.

Table 3 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**).

Table 3: Description of the equipment maintenance dataset attributes (adapted from [10]).

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

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. 6 shows the histogram for the regression target and the balancing of classes for the binary classification targets. Regarding the regression target

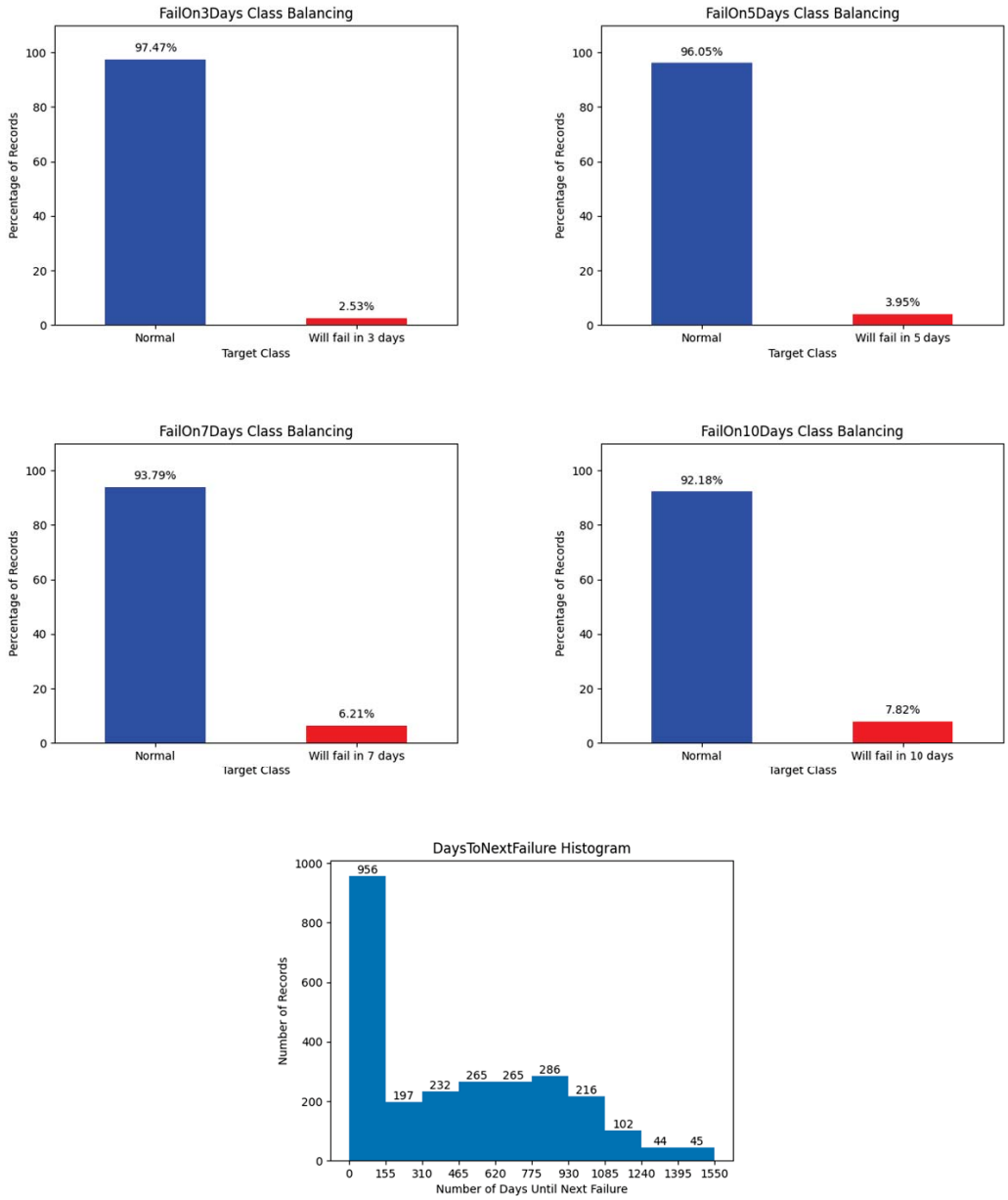


Figure 6: Balancing of the binary classification targets and histogram of the regression target.

(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. 6 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 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., [47]), imbalanced datasets are very prevalent in the PdM domain since failures are frequently sporadic compared to health situations.

3.4. Data Preprocessing

Since several data attributes are of the type String (as shown in Table 3), 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 4.

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 [48]. 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).

Table 4: 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

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.

3.5. Evaluation

In order to evaluate the results from the AutoML tools and AutoOneClass, we adopted a similar approach to the benchmark developed in [22]. 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 Mean Absolute Error (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. Results

4.1. AutoML Results

The first comparison focused on the supervised AutoML tools detailed in Section 3.1. For each AutoML tool, we executed five experiments, one for each target variable (DaysToNextFailure and FailOnxDays). Table 5 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 [49].

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

Table 5: Average predictive results obtained by the AutoML tools, best values for each target in **bold** (adapted from [10]).

		Targets				
		Days Unitl 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.

other hand, the worst 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 6. 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 6: 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	Auto-Keras	2532±1137	2579±516	3374±2135	3209±1233	1055±291
Tools	Auto-PyTorch	1514±116	1450±161	1262±107	1524±111	1334±155
	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
AutoML	MLJar	1519±57	1607±42	1653±46	2066±413	3232±770
Tools	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., Decision Trees, Random Forest) 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 [50]. 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.

4.2. AutoOneClass Results

The second predictive comparison considers the AutoOneClass method, proposed and described in Section 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 7. We note that we adopted

Table 7: 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

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 8 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 and AutoML results (from Table 5). 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.

Table 8: 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.

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 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 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. 7.

Similar to the previous experiment, we also analyze the average training times for the AutoOneClass results, shown in Table 9. 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 8) and the AutoML results (shown in Table 5) shows that none of the AutoOneClass executions 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 super-

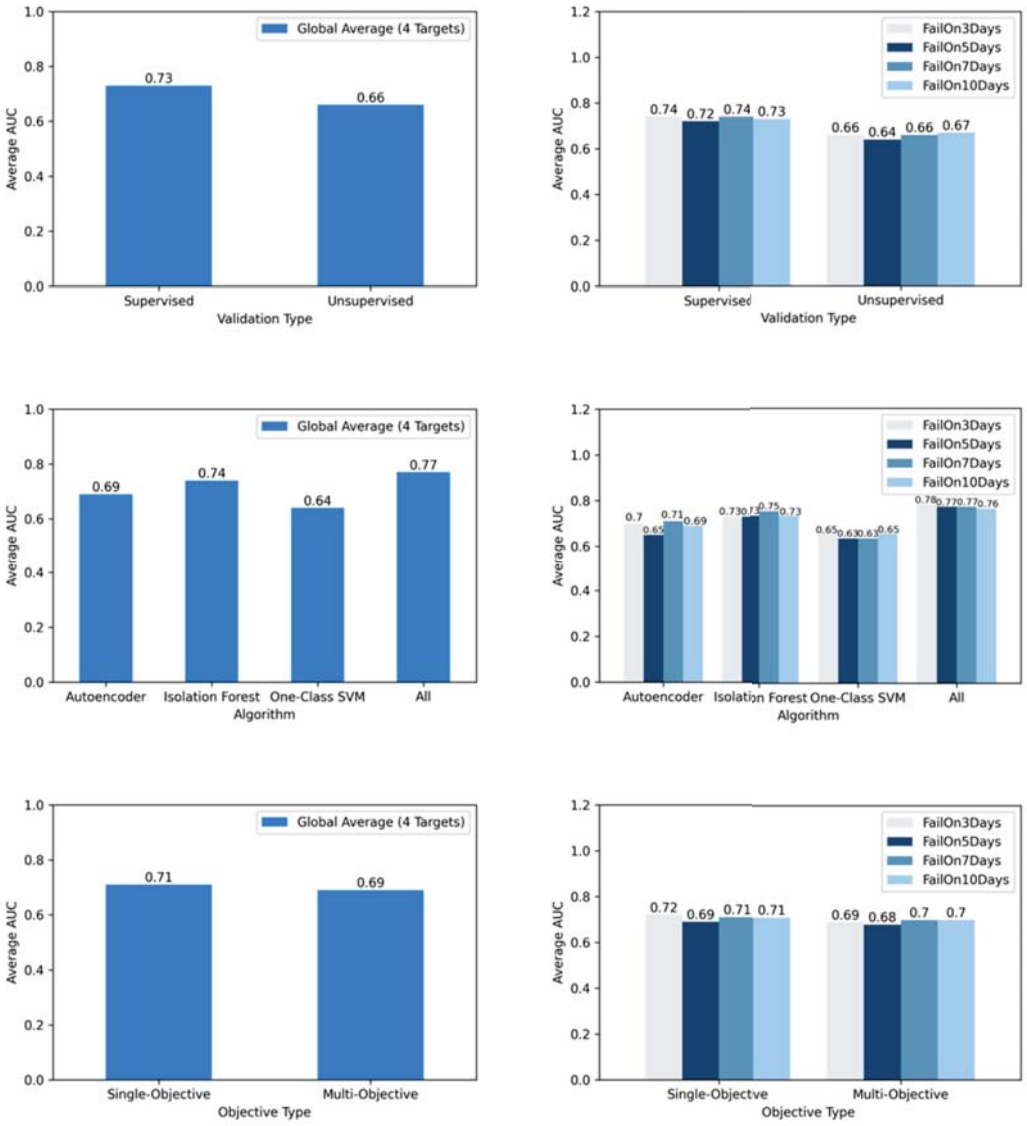


Figure 7: AutoOneClass results aggregated by validation type, algorithm, and optimization type, both globally (left) and per target (right).

vised validation), when compared with the supervised AutoML tools, which typically require a labeled dataset with a balanced ratio of normal and abnormal records. In many real-world PdM scenarios, there is a huge number of normal records and anomaly records might not always be available. Thus,

Table 9: 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.

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.

4.3. 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 10 compares the prediction results achieved

using the manual ML design and best AutoML tools. For each AutoML tool, Table 10 includes the algorithm (**Alg.**) that was most often the leader across the external folds (in rounded brackets). For the human modeling, Table 10 shows the best obtained result and the used algorithm.

Table 10: Comparison between the best AutoML results, AutoOneClass results, and human ML modeling results (expert and non-expert) for each target, best values in **bold** (adapted from [10]).

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 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 OC Learning 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.

5. Conclusions

PdM 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 paper, 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 OC Learning 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 iby the Supervised AutoML tools. However, the AutoOneClass surpassed the expert human modeling in two predictive targets and the performed OC Learning 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.

In future work, we intend to perform experiments with more datasets from the domain of PdM to verify further consistency with our results. We also intend to experiment with AutoML technologies that can automatically perform other ML phases apart from modeling, such as feature engineering and selection. Furthermore, regarding the AutoOneClass method, we plan to develop a benchmark with a more significant number of OC Learning algorithms (e.g., Local Outlier Factor, Gaussian Mixture Model, recently proposed OC Learning algorithms [51, 52, 53, 54, 55]) and datasets, including big data ones that should favor its multi-objective variant. Additionally, we aim to experiment different values for crossover, mutation and training time to assess their impact on the GE optimization. Also, for the multi-objective mode, we intend to analyze in more depth the correlation between the AUC and the anomaly scores when using a supervised validation. Finally, we wish to add more functionalities to AutoOneClass, such as the usage of other performance objectives (apart from training time) or the application of an early stopping to the GE optimization.

Acknowledgments

This work was executed under the project Cognitive CMMS - Cognitive Computerized Maintenance Management System, NUP: POCI-01-0247-FEDER-033574, co-funded by the Incentive System for Research and Technological Development, from the Thematic Operational Program Compet-

itiveness of the national framework program - Portugal2020. We wish to thank the anonymous reviewers for their helpful comments.

References

[1] A. J. Silva, P. Cortez, C. Pereira, A. Pilastri, Business analytics in industry 4.0: A systematic review, Expert Systems (2021) e12741doi:https://doi.org/10.1111/exsy.12741.

[2] A. Kanawaday, A. Sane, Machine learning for predictive maintenance of industrial machines using iot sensor data, in: 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), IEEE, 2017, pp. 87–90.

[3] B. Cline, R. S. Niculescu, D. Huffman, B. Deckel, Predictive maintenance applications for machine learning, in: 2017 annual reliability and maintainability symposium (RAMS), IEEE, 2017, pp. 1–7.

[4] M. Paolanti, L. Romeo, A. Felicetti, A. Mancini, E. Frontoni, J. Loncarski, Machine learning approach for predictive maintenance in industry 4.0, in: 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, MESA 2018, Oulu, Finland, July 2-4, 2018, IEEE, 2018, pp. 1–6. doi:10.1109/MESA.2018.8449150.

[5] T. P. Carvalho, F. A. A. M. N. Soares, R. Vita, R. da Piedade Francisco, J. P. T. V. Basto, S. G. S. Alcalá, A systematic literature review of machine learning methods applied to predictive maintenance, Comput. Ind. Eng. 137 (2019). doi:10.1016/j.cie.2019.106024.

[6] M. D. Benedetti, F. Leonardi, F. Messina, C. Santoro, A. V. Vasilakos, Anomaly detection and predictive maintenance for photovoltaic systems, Neurocomputing 310 (2018) 59–68. doi:10.1016/j.neucom.2018.05.017.

[7] S. Butte, A. Prashanth, S. Patil, Machine learning based predictive maintenance strategy: a super learning approach with deep neural networks, in: 2018 IEEE Workshop on Microelectronics and Electron Devices (WMED), IEEE, 2018, pp. 1–5.

[8] Z. M. Çınar, A. Abdussalam Nuhu, Q. Zeeshan, O. Korhan, M. Asmael, B. Safaei, Machine learning in predictive maintenance towards sustainable smart manufacturing in industry 4.0, Sustainability 12 (19) (2020) 8211.

[9] S. Ayvaz, K. Alpay, Predictive aintenance system for production lines in manufacturing: A machine learning approach using iot data in real-time, Expert Systems with Applications 173 (2021) 114598. doi:10.1016/j.eswa.2021.114598.

[10] L. Ferreira, A. L. Pilastri, V. Sousa, F. Romano, P. Cortez, Prediction of maintenance equipment failures using automated machine learning, in: Intelligent Data Engineering and Automated Learning - IDEAL 2021 - 22nd International Conference, IDEAL 2021, Manchester, UK, November 25-27, 2021, Proceedings, Vol. 13113 of Lecture Notes in Computer Science, Springer, 2021, pp. 259–267. doi:10.1007/978-3-030-91608-4_26.

[11] N. Amruthnath, T. Gupta, A research study on unsupervised machine learning algorithms for early fault detection in predictive maintenance, in: 2018 5th international conference on industrial engineering and applications (ICIEA), IEEE, 2018, pp. 355–361. doi:10.1109/IEA.2018.8387124.

[12] S. Cho, G. May, I. Tourkogiorgis, R. Perez, Ó. Lázaro, B. de la Maza, D. Kiritsis, A hybrid machine learning approach for predictive maintenance in smart factories of the future, in: 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, Vol. 536 of IFIP Advances in Information and Communication Technology, Springer, 2018, pp. 311–317. doi:10.1007/978-3-319-99707-0_39.

[13] P. Straus, M. Schmitz, R. Wöstmann, J. Deuse, Enabling of predictive maintenance in the brownfield through low-cost sensors, an iiot-architecture and machine learning, in: IEEE International Conference on Big Data (IEEE BigData 2018), Seattle, WA, USA, December 10-13, 2018, IEEE, 2018, pp. 1474–1483. doi:10.1109/BigData.2018.8622076.

[14] T. Tornede, A. Tornede, M. Wever, F. Mohr, E. Hüllermeier, Automl for predictive maintenance: One tool to RUL them all, in: 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, Vol. 1325 of Communications in Computer and Information Science, Springer, 2020, pp. 106–118. doi:10.1007/978-3-030-66770-2_8.

[15] H. A. Gohel, H. Upadhyay, L. Lagos, K. Cooper, A. Sanzetenea, Predictive maintenance architecture development for nuclear infrastructure using machine learning, Nuclear Engineering and Technology 52 (7) (2020) 1436–1442. doi:10.1016/j.net.2019.12.029.

[16] J. C. Cheng, W. Chen, K. Chen, Q. Wang, Data-driven predictive maintenance planning framework for mep components based on bim and iot using machine learning algorithms, Automation in Construction 112 (2020) 103087. doi:10.1016/j.autcon.2020.103087.

[17] M. Calabrese, M. Cimmino, F. Fiume, M. Manfrin, L. Romeo, S. Ceccacci, M. Paolanti, G. Toscano, G. Ciandrini, A. Carrotta, M. Mengoni, E. Frontoni, D. Kapetis, SOPHIA: an event-based iot and machine learning architecture for predictive maintenance in industry 4.0, Inf. 11 (4) (2020) 202. doi:10.3390/info11040202.

[18] G. Makridis, D. Kyriazis, S. Plitsos, Predictive maintenance leveraging machine learning for time-series forecasting in the maritime industry, in: 23rd IEEE International Conference on Intelligent Transportation Systems, ITSC 2020, Rhodes, Greece, September 20-23, 2020, IEEE, 2020, pp. 1–8. doi:10.1109/ITSC45102.2020.9294450.

[19] J. Larocque-Villiers, P. Dumond, D. Knox, Automating predictive maintenance using state-based transfer learning and ensemble methods, in: 14th IEEE International Symposium on Robotic and Sensors Environments, ROSE 2021, Virtual Event, October 28-29, 2021, IEEE, 2021, pp. 1–7. doi:10.1109/ROSE52750.2021.9611768.

[20] M. Çakir, M. A. Güvenç, S. Mistikoglu, The experimental application of popular machine learning algorithms on predictive maintenance and the design of iiot based condition monitoring system, Comput. Ind. Eng. 151 (2021) 106948. doi:10.1016/j.cie.2020.106948.

[21] S. Arena, E. Florian, I. Zennaro, P. Orrù, F. Sgarbossa, A novel decision support system for managing predictive maintenance strategies based on machine learning approaches, Safety science 146 (2022) 105529. doi:10.1016/j.ssci.2021.105529.

[22] L. Ferreira, A. L. Pilastri, C. M. Martins, P. M. Pires, P. Cortez, A comparison of automl tools for machine learning, deep learning and xgboost, in: International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021, IEEE, 2021, pp. 1–8. doi:10.1109/IJCNN52387.2021.9534091.

[23] H. Jin, Q. Song, X. Hu, Auto-keras: An efficient neural architecture search system, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, 2019, pp. 1946–1956.

[24] L. Zimmer, M. Lindauer, F. Hutter, Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl, IEEE Trans. Pattern Anal. Mach. Intell. 43 (9) (2021) 3079–3090. doi:10.1109/TPAMI.2021.3067763.

[25] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, F. Hutter, Auto-sklearn: Efficient and robust automated machine learning, in: F. Hutter, L. Kotthoff, J. Vanschoren (Eds.), Automated Machine Learning - Methods, Systems, Challenges, The Springer Series on Challenges in Machine Learning, Springer, 2019, pp. 113–134. doi:10.1007/978-3-030-05318-5_6.

[26] Auto-Gluon, AutoGluon: AutoML Toolkit for Deep Learning — AutoGluon Documentation 0.2.0 documentation (2021).
URL `https://auto.gluon.ai/`

[27] W. Qi, C. Xu, X. Xu, Autogluon: A revolutionary framework for landslide hazard analysis, Natural Hazards Research 1 (3) (2021) 103–108. doi:10.1016/j.nhres.2021.07.002.

[28] H2O.ai, H2O AutoML, h2O version 3.32.1.3 (2021).
URL `http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html`

[29] A. Płońska, P. Płoński, Mljar: State-of-the-art automated machine learning framework for tabular data. version 0.10.3 (2022).
URL https://github.com/mljar/mljar-supervised

[30] M. Ali, PyCaret: An open source, low-code machine learning library in Python, pyCaret version 2.3.10 (June 2022).
URL https://www.pycaret.org

[31] P. Cortez, Data mining with neural networks and support vector machines using the r/rminer tool, in: Industrial conference on data mining, Springer, 2010, pp. 572–583.

[32] T. T. Le, W. Fu, J. H. Moore, Scaling tree-based automated machine learning to biomedical big data with a feature set selector, Bioinformatics 36 (1) (2020) 250–256.

[33] Salesforce, Transmogrifai - automl library for building modular, reusable, strongly typed machine learning workflows on spark from salesforce engineering (2022).
URL https://transmogrif.ai/

[34] N. Amruthnath, T. Gupta, Fault class prediction in unsupervised learning using model-based clustering approach, in: 2018 International Conference on Information and Computer Technologies (ICICT), IEEE, 2018, pp. 5–12. doi:10.1109/INFOCT.2018.8356831.

[35] D. Ribeiro, L. M. Matos, G. Moreira, A. L. Pilastri, P. Cortez, Isolation forests and deep autoencoders for industrial screw tightening anomaly detection, Computers 11 (4) (2022) 54. doi:10.3390/computers11040054.

[36] X. He, K. Zhao, X. Chu, Automl: A survey of the state-of-the-art, Knowl. Based Syst. 212 (2021) 106622. doi:10.1016/j.knosys.2020.106622.

[37] M. Bahri, F. Salutari, A. Putina, M. Sozio, Automl: state of the art with a focus on anomaly detection, challenges, and research directions, International Journal of Data Science and Analytics (2022) 1–14doi:10.1007/s41060-022-00309-0.

[38] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow,

A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL `https://www.tensorflow.org/`

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

[40] S. Maleki, S. Maleki, N. R. Jennings, Unsupervised anomaly detection with LSTM autoencoders using statistical data-filtering, Appl. Soft Comput. 108 (2021) 107443. doi:10.1016/j.asoc.2021.107443.

[41] F. T. Liu, K. M. Ting, Z. Zhou, Isolation forest, in: Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy, IEEE Computer Society, 2008, pp. 413–422. doi:10.1109/ICDM.2008.17.

[42] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, R. C. Williamson, Estimating the support of a high-dimensional distribution, Neural Comput. 13 (7) (2001) 1443–1471. doi:10.1162/089976601750264965.

[43] S. Wang, Q. Liu, E. Zhu, F. Porikli, J. Yin, Hyperparameter selection of one-class support vector machine by self-adaptive data shifting, Pattern Recognit. 74 (2018) 198–211. doi:10.1016/j.patcog.2017.09.012.

[44] M. O'Neill, C. Ryan, Grammatical evolution, IEEE Trans. Evol. Comput. 5 (4) (2001) 349–358. doi:10.1109/4235.942529.

[45] P. J. Pereira, P. Cortez, R. Mendes, Multi-objective grammatical evolution of decision trees for mobile marketing user conversion prediction, Expert Syst. Appl. 168 (2021) 114287. doi:10.1016/j.eswa.2020.114287.

[46] M. Fenton, J. McDermott, D. Fagan, S. Forstenlechner, E. Hemberg, M. O'Neill, 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, ACM, 2017, pp. 1194–1201. doi:10.1145/3067695.3082469.

[47] M. D. Dangut, Z. Skaf, I. K. Jennions, Handling imbalanced data for aircraft predictive maintenance using the bache algorithm, Applied Soft Computing (2022) 108924doi:10.1016/j.asoc.2022.108924.

[48] L. M. Matos, J. Azevedo, A. Matta, A. Pilastri, P. Cortez, R. Mendes, Categorical attribute transformation environment (cane): A python module for categorical to numeric data preprocessing, Software Impacts 13 (2022) 100359. doi:https://doi.org/10.1016/j.simpa.2022.100359.

[49] M. Hollander, D. A. Wolfe, E. Chicken, Nonparametric statistical methods, John Wiley & Sons, 2013.

[50] A. Ng, Machine Learning Yearning, deeplearning.ai, 2020.
URL `https://www.deeplearning.ai/machine-learning-yearning/`

[51] I. Golan, R. El-Yaniv, Deep anomaly detection using geometric transformations, in: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, 2018, pp. 9781–9791.

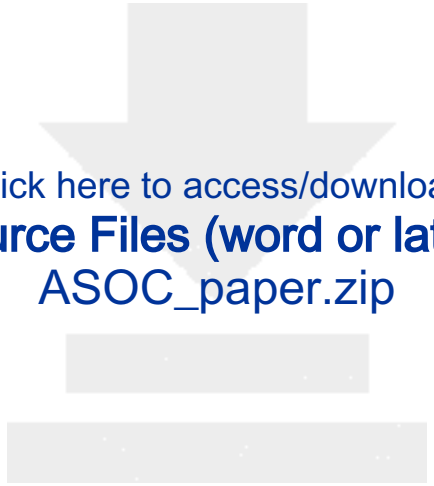
[52] S. Mauceri, J. Sweeney, J. McDermott, Dissimilarity-based representations for one-class classification on time series, Pattern Recognit. 100 (2020) 107122. doi:10.1016/j.patcog.2019.107122.

[53] O. U. Lenz, D. Peralta, C. Cornelis, Average localised proximity: A new data descriptor with good default one-class classification performance, Pattern Recognit. 118 (2021) 107991. doi:10.1016/j.patcog.2021.107991.

[54] T. Hayashi, H. Fujita, A. Hernandez-Matamoros, Less complexity one-class classification approach using construction error of convolutional image transformation network, Inf. Sci. 560 (2021) 217–234. doi:10.1016/j.ins.2021.01.069.

[55] A. Blázquez-García, A. Conde, U. Mori, J. A. Lozano, Water leak detection using self-supervised time series classification, Inf. Sci. 574 (2021) 528–541. doi:10.1016/j.ins.2021.06.015.

Source Files (word or latex)



Click here to access/download

Source Files (word or latex)

ASOC_paper.zip