Received: 15 December 2020 | Revised: 12 May 2021 | Accepted: 28 May 2021

DOI: 10.1002/int.22549

RESEARCH ARTICLE

WILEY

Meta-learning and the new challenges of machine learning

José Pedro Monteiro[1] | **Diogo Ramos**[2] | **Davide Carneiro**[1,2] | **Francisco Duarte**[1] | **João M. Fernandes**[1] | **Paulo Novais**[1]

[1]Department of Informatics, Algoritmi Centre, Universidade do Minho, Felgueiras, Portugal

[2]CIICESI, ESTG, Politécnico do Porto, Felgueiras, Portugal

Correspondence
Davide Carneiro, Escola Superior de Tecnologia e Gestão, Rua do Curral, Casa do Curral, Margaride, 4610-156 Felgueiras, Portugal.
Email: dcarneiro@estg.ipp.pt

Funding information
Fundação para a Ciência e a Tecnologia, Grant/Award Numbers: UIDB/04728/2020, UID/CEC/00319/2019; European Regional Development Fund, Grant/Award Number: 31/SI/2017 Proj. No. 39900

Abstract
In the last years, organizations and companies in general have found the true potential value of collecting and using data for supporting decision-making. As a consequence, data are being collected at an unprecedented rate. This poses several challenges, including, for example, regarding the storage and processing of these data. Machine Learning (ML) is also not an exception, in the sense that algorithms must now deal with novel challenges, such as learn from streaming data or deal with concept drift. ML engineers also have a harder task when it comes to selecting the most appropriate model, given the wealth of algorithms and possible configurations that exist nowadays. At the same time, training time is a stronger

1 | INTRODUCTION

Over the past years, the field of Machine Learning (ML) has been pushing for new approaches and methodologies to deal with the significant challenges that have been emerging. The main source of these challenges is the new properties of data.

First, data are now pervasive to virtually every company or organization. Organizations have understood the value of collecting, transforming, and using internal and external data for decision-support, as a way to gain a competitive edge over their peers. This was made possible, in part, through the general availability of the necessary resources (e.g., hardware and software) which were, up until recently, prohibitively expensive or simply nonexistent.

This shift led data to be more diverse than it traditionally used to be, and also to exist in a much higher volume than before.[1] This volume and diversity of data mean that it is nowadays common for a data set to be too large to be stored in the memory or even in the hard drive of a single computer. Thus, data sets are nowadays often distributed across large clusters, made up of dozens to thousands of computers, which act as storage and processing nodes.

Data are also increasingly dynamic. This derives from the velocity at which both companies and society move nowadays, with increasingly faster transactions and interactions. Organizations thus feel the need to adapt increasingly faster to these changes so that they are not left behind. When doing so and adapting their own response to external change, companies are, themselves, contributing to this change. The practical implication is that the statistical properties of data change in significant ways and faster than ever before.[2]

This new panorama, which results from a quick shift, poses significant challenges to many data-related fields, notably to ML. ML algorithms now need to learn from data that are not all in one same physical location, and are distributed across many machines.[3] There is also an increased pressure for organizations to take decisions faster. Thus, algorithms need not only to be able to learn from larger data sources, but also to do it in more efficient ways. Finally, models risk being outdated soon after they are trained, as the data change. Thus, models need to adapt dynamically to the data on an almost real-time basis, to ensure that decision-making performance is optimum.[4]

In this paper we look at meta-learning and how it can be used to address the challenges pointed out. Meta-learning is seen, in the context of this paper, as the ability of *learning to learn*. This rather broad definition includes, from our point of view, methods that rely on data about the data sets, about the models (e.g., configuration and performance metrics), among others. These types of data are often called meta-data. The so-called meta-data sets are also

This paper provides an analysis of the current state of meta-learning, with a focus on approaches that allow one to address the current challenges in the field of ML. Then, we describe the implementation of two novel meta-learning applications, one for algorithm selection and the other for streaming learning, as well as their validation and results.

The rest of the paper is organized as follows. Section 2 provides an overview of meta-learning applications, with the goal to cover the diverse and broad potential of these techniques. Section 3 establishes some relevant theoretic considerations, in regard to the approach proposed in this paper. Section 4 defines the problem being tackled: to use meta-learning to solve some of the current challenges in ML. Specifically, this section characterizes the two main problems being addressed: algorithm selection and continuous learning from streaming data. Section 5 provides the relevant details into the system that was developed to implement the envisaged solution. This system is then validated in Section 6, through its use with real publicly available data sets. Finally, the paper closes with Section 7, which presents the concluding remarks.

2 | RELATED WORK

Meta-learning is concerned with discovering patterns in the data and understanding their effect on the behavior of algorithms. Nonetheless, it can be approached in several different ways and, therefore, there are several potential applications[8,9] including selecting and recommending ML therefore, there are several potential applications, including selecting and recommending ML algorithms, knowledge discovery, combining base-level ML systems, controlling the learning process and bias management, or transferring meta-knowledge across domains.

In this section, we describe several representative contributions that are related to meta-learning in general, attempting to cover the main different application scenarios.

One of the challenges, in any ML project, is that of algorithm selection. On the one hand, there is an increasing wealth of algorithms, each with its particular characteristics, which although providing more alternatives and potentially better models, also makes it more difficult to select the best algorithm. Furthermore, newer algorithms can be configured and tweaked in many different ways, which further expands the problem of algorithm selection to algorithm selection and configuration. On the other hand, data sets are becoming increasingly complex, which makes it more difficult for a human expert to understand the properties of the data and to select the most appropriate algorithm, or at least to do it in an objective manner.

and the targets are the performance (meta-labels) of a set of algorithms when they are applied to the data set. Features may describe statistical information, information-theoretic characteristics, or "landmarkers,"[16] in which some properties (e.g., predictive performance) of a model learned by an algorithm are used as meta-features. One caveat is that, to be efficient, the landmarker-based features need to be computed relatively quickly otherwise one could simply run the candidate algorithms on the data set.

Given one such meta-data set, another algorithm, usually called a meta-learner, learns a model using the given meta-features. When given a new data set, first the meta-features are calculated, and the expected or relative predictive performances of different algorithms can then be predicted.

Another domain in which meta-learning is widely used nowadays is ensemble learning. Ensemble methods are an ML technique that combines several base models to produce one optimal predictive model. The way the base models are combined into a single larger model is what places ensembles under the meta-learning umbrella. The combination of these models can follow different approaches.

For instance, in the stacking generalization method[17] several base-level learning algorithms are combined using a meta-level learner to learn a linear function of the models produced by the base-level algorithms. Given a new data set, the predictions of the base-level algorithms are combined to provide the final prediction. Boosting[18] is another popular ensemble learning strategy in which the same base-level algorithm is used multiple times to train multiple models. In each iteration, a reweighted training set is used in which instances that were previously misclassified have a larger weight. The final boosting prediction is also a combination of base-level models. Bootstrap aggregation, also called bagging,[19] is another example of ensemble learning where multiple weak models are used. At prediction time, the predictions of the several weak models are aggregated to get the final prediction. Whenever the dependent variable is numeric, the prediction is usually the average value of the prediction of each weak model. When it is an enumeration, the prediction is taken from the most frequent class predicted among all the weak models. Extensive theoretical and empirical studies have been carried out in recent years in ensemble learning. Comprehensive reviews on ensemble techniques can be found in Reference [20].

Meta-learning has also been used as a variant of inductive transfer.[9] Inductive transfer refers to the ability of a learning mechanism to improve performance on the current or target task after having learned a different but related concept or skill on a previous source task.

TABLE 1 Sample train/test setting for binary functions over three Boolean variables (adaption from Reference [24])

Inputs	*f*0	*f*1	*f*2	*f*3	*f*4	*f*4	*f*4	*f*4	*f*4	*f*4	...
Training set											
000	0	0	0	0	0	0	0	0	0	0	...
001	0	0	0	0	0	0	0	0	0	0	
010	0	0	0	0	0	0	0	0	0	0	
011	0	0	0	0	0	0	0	0	0	0	
100	0	0	0	0	0	0	0	0	1	1	
101	0	0	0	0	1	1	1	1	0	0	
Test set											
110	0	0	1	1	0	0	1	1	0	1	...
111	0	1	0	1	0	1	0	1	0	0	

The NFL theorem in this table shows that the behavior on $\mathcal{T}_e$ of any learner $(f0, f1, ..., f255)$ trained on $\mathcal{T}_r$ is that of a random guesser (can be seen, e.g., while considering functions $f0$–$f3$ in Table 1). The NFL theorem in essence simply restates Hume's conclusion about induction having no rational basis[25]: "There can be no demonstrative argument to prove, that those instances, of which we have had no experience, resemble those, of which we have experience....". The crucial contribution of the NFL theorem is pointing out that whenever a learning algorithm performs well on some functions it must perform poorly on some others.

Taking that into consideration NFL theorem and Hume's conclusion, let us discuss the application of an Ultimate Learning Algorithm (ULA), therefore for a given learning algorithm, a model M is induced, which defines a class probability distribution p over the instances space. A ULA is a learning algorithm that induces a model M^*, such that

$$\forall M' \neq M^*, \quad E(\delta(p^*, p^\Omega)) \leq E(\delta(p', p^\Omega)), \tag{1}$$

where the expectation is computed for a given training/test set sample of the instance space, over the entire function space, δ is some appropriate distance measure and p^*, p^Ω, and p' are

characterized by challenges, such as concept drift, streaming data, or complex data sets. To this end we propose an architecture to support the implementation of meta-learning services. So far, this architecture has been used to address two key real-world problems: algorithm selection and continuous learning in streaming scenarios. This section describes the key aspects of each of these problems. Section 5, on the other hand, provides details regarding how a meta-learning system was implemented that allows the development of services that deal with these problems.

4.1 | Algorithm selection

In real ML problems, the task of algorithm selection is an inherent complex problem, with many variables and restrictions. The main goal of this task is to select the best algorithm and its configuration, that is, the one that minimizes the error function of its predictions when used on real test data. There is, nowadays, a plethora of algorithms to choose from, each of which with many parameters that can be tuned and will change the predictive performance of the resulting model.

Important aspects to be considered when selecting an algorithm include:

- the shape of the data set, that is, the number of rows, number of columns, and the ratio between them;
- the statistical properties of the variables;
- the type of data described by the data set (e.g., historical information, time-series, signals, and image);
- business restrictions or requirements, such as training time, degree of interpretability, target infrastructure (e.g., cloud vs. on premises), among others;
- business goals.

It is important to point out that there is no single best algorithm to be used in all problems, as results from the NFL theorem[22] described in Section 3. The brute-force approach is to try all the algorithms with different parameter settings (e.g., using Grid Search) for a given data set at hand.

In practice this is usually not feasible due to the computation time it takes to assess a large number of alternative algorithms available and their possible configuration. Thus, when in the phase of data preparation, the human expert usually gains some notions about the data set's

is to use a trial-and-error strategy. Taking algorithm selection as an example, the expert could get the performance estimations of the algorithms based on cross-validation, and then use performance measures to determine the best algorithm to use. Although feasible, this strategy may still require a reasonable amount of computing and human time, especially when there are many algorithms available.

From a meta-learning perspective, the main advantages of automated algorithm selection are twofold: to allow the human expert to save time by not having to configure and train different models manually, and to allow for a potentially better model than would be otherwise achieved by estimating which models are better suited for a data set, given their properties.

Moreover, the problem should be addressed as a ranking task instead of a classification task. This allows the system to provide the user with several alternatives and their respective performance scores. Thus, the user can select among several alternatives the one that is best suited to the problem's characteristics and constraints.

Meta-learning for algorithm ranking uses a general ML approach to generate meta-knowledge, mapping the characteristics of a data set, encoded in meta-features, to the expected performance of the available algorithms. This approach is particularly important for business domains that require the fast deployment of analytical techniques (e.g., stock market prediction and customer response prediction). Additional insights into the topic of meta-learning research and its applications, particularly for algorithm selection, can be found in References [8,9,27,28].

4.2 | Learning in streaming scenarios

The goal of ML algorithms is to find patterns in data that describe real-world phenomena. These patterns are, however, often not static. That is, they change over time. This change is known as *concept drift*. In streaming scenarios, more than in traditional ML, learning is as much about forgetting irrelevant and outdated details, while remembering relevant and new ones.[2] Not doing so would result in models that are too complex and also outdated.

A classic example in which concept drift happens is in the market basket analysis problem,[29] in which users' buying preferences change over time. Whenever these changes are periodic or seasonal, they can be incorporated into the model through new date-related variables (e.g., month of the year and season). However, some of these changes are more perennial and do not depend on cyclical factors; they just reflect changes in buying consumer that may be

time. Concept drift should not be confused with noise.[2] Noise represents more random variations in the distribution of the data, often difficult to predict. Concept drift, on the other hand, represents consistent and long-term changes, that can be predicted accurately if the model is updated. Excessive sensitivity to noise or even to outliers may lead the model into adapting to nonexisting changes, or to overfit.

There are two main ways for ML to deal with concept drift. In a batch learning approach, models are completely retrained to include the new data and, possibly, excluding some or all the old data. Given that the training of a model is potentially a computationally intensive and time-consuming task, this is often not feasible in practice, especially when concepts change too fast or too often.[31] An alternative is to use an incremental approach to learning, in which the model or part of it is updated, without the need to retrain the whole model. This is usually a less computationally intensive task.[32]

This section proposes an approach to deal with concept drift based on meta-learning that can be categorized as incremental. Specifically, we propose a bagging ensemble in which new base models are added over time, trained with more recent data. Once the pool model is full, older, and/or outdated, models are removed. The meta-learning task consists of finding the best weights to use in the bagging mechanism, and to decide what is the constitution and configuration of the ensemble at each moment. This meta-learning task is implemented through a genetic algorithm. The meta-model is thus able to continuously adapt with minimum effort.

5 | ARCHITECTURE

Section 4 described the two main ML problems addressed in this study: algorithm selection and continuous learning in streaming scenarios. To address these problems, a group of software components was devised and implemented, whose integration makes up a multifunctional meta-learning system. This section details this system in terms of its main components. First, an overview of the architecture of the system is provided. Then, a more in-depth description is provided, organized in terms of functionality rather than individual components.

The architecture of this system, illustrated in Figure 1, has two interfaces with external systems and eight main components. The first interface, represented on the left end of the figure, allows one to connect to data sources. There are two main types of data sources: batch and streaming. Batch data sources are usually static data sets, in the form of files, that are

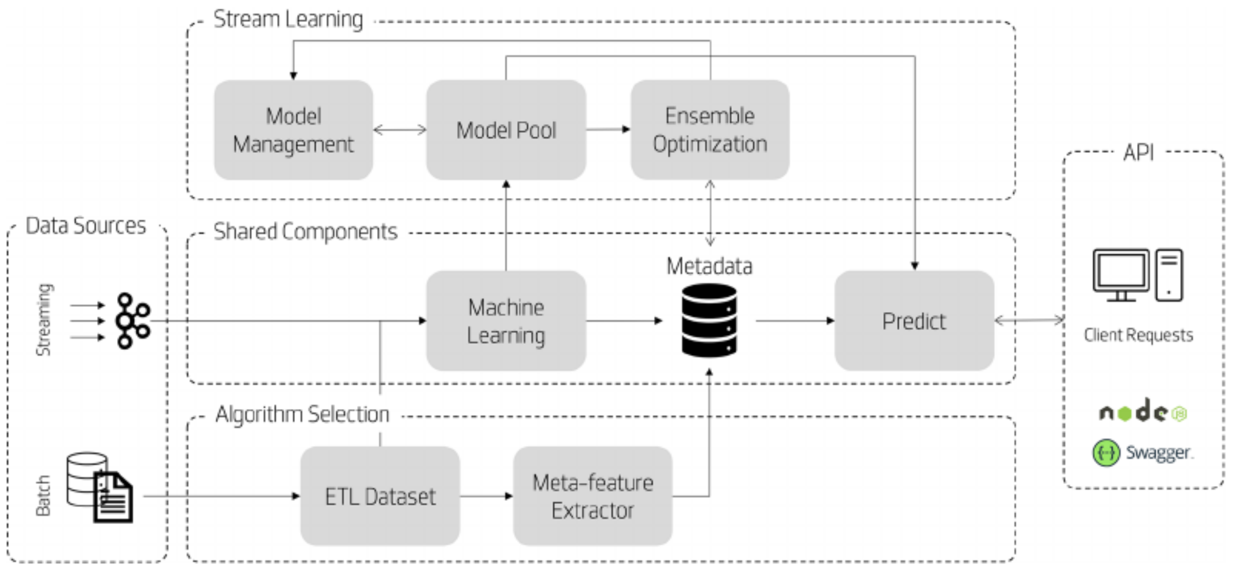


FIGURE 1 Main components of the proposed meta-learning system. API, application programming interface; ETL, extract, transform, and load [Color figure can be viewed at wileyonlinelibrary.com]

- Algorithm selection
  - *Extract, transform, and load (ETL) data set*—transformation of data sets into standard formats and representations;
  - *Meta-feature Extractor*—extraction of meta-features from data sets that describe the shape/structure of the data set, the statistical properties of the data, among others.
- Stream learning
  - *Model management*—services for managing models, such as loading/unloading models to/from memory;
  - *Model pool*—space in memory in which active models are kept. These models can be used to make predictions;
  - *Ensemble optimization*—implements a Genetic Algorithm to optimize the voting scheme of the base models in an ensemble.
- Common
  - *ML*—implements model training and assessment functionalities that are used by other components;

The ETL data set module is responsible for handling the data from the original data sets. This module first extracts the data from the sources of the different data sets (e.g., MySQL, Excel, and csv files) then transforms the data into a specific format and loads the data sets into a database. The most relevant task of this module is the transformation of the input data sets into an appropriate standard format, so that they can be used by the Meta-feature Extractor and ML modules. For instance, the name of the target variable is changed to y_target, creating a standard separation between the target and the features.

The Meta-feature Extractor module uses Python's library pymfe[33] (version 0.1.0) to extract the relevant meta-features from the input data sets. A total of 108 meta-features are extracted, which can be organized into six large groups: General, Statistical, Information-theoretic, Model-based, Landmarking, and Clustering. More details on this process are given in Section 5.1.1

Landmarking, and Clustering. More details on this process are given in Section 5.1.1.

The ML module is responsible for training ML models with the data set(s) provided by the ETL data set module, and to store the corresponding performance metrics in the database (e.g., AUC-ROC Score, Precision, Recall, F1 Score, Accuracy, and Root-Mean-Square Error [RMSE]). Models are trained using the algorithms provided by the Scikit-Learn package[34] (version 0.18.1). Seventeen algorithms of different categories are used, including Ensembles, Naive Bayes, Tree-based, or Support Vector Machines (SVM).

ML models are trained and evaluated using 10-fold cross-validation and, for some algorithms, hyperparameter tuning was implemented using Grid Search. The time required for model fitting is also stored as a meta-feature (e.g., for future comparisons between algorithms, namely, to use as an algorithm selection criteria).

Finally, the information received from the Meta-feature Extractor is combined with the results from the ML module and stored in the meta-data database. The final result is a matrix with 108 meta-features, with one instance for each combination of data set/algorithm used. After the concatenation, the final meta-data set is stored in an SQLite database. The process through which the meta-data set is generated is detailed next.

5.1.1 | Generation of meta-data sets

This section describes the process through which the meta-data set is built. Arbitrary data sets are received as input and their corresponding meta-features are extracted as described previously. As data sets are processed, this gradually builds the meta-data set that is then used to

TABLE 2 Example of a meta-data set with the sample of two data sets (Iris and Heart Disease), three meta-features (*mf*1, *mf*2, and *mf*3), the algorithm used (e.g., BernoulliNB), and the results of three different evaluation metrics (*em*1, *em*2, and *em*3)

Data set	*mf1*	*mf*2	*mf*3	Algorithms	*em*1	*em*2	*em*3
Iris	100	20	0.78	Random Forest	0.88	0.79	0.82
Iris	100	20	0.78	BernoulliNB	0.75	0.73	0.67
Iris	100	20	0.78	Ada Boost Classifier	0.90	0.80	0.85
Heart Disease	300	55	0.08	Random Forest	0.90	0.81	0.82
Heart Disease	300	55	0.08	BernoulliNB	0.87	0.83	0.83
Heart Disease	300	55	0.08	Ada Boost Classifier	0.92	0.84	0.83

According to their type, meta-features can also be organized into four categories[8,24,35,36]:

1. *Simple*: Number of instances, number of attributes, number of classes or targets, and so forth.[37]
2. *Statistical*: Mean kurtosis of attributes, mean skewness of attributes, and so forth.[38]
3. *Information-theoretic*: Class entropy, mean entropy of attributes, noise signal ratio, and so forth.[35]
4. *Landmarking*: Exploiting data set properties from the performance of a set of simple and fast learners with significantly different learning mechanisms.[16] Various landmarking methods are described in Reference [39].

An instance of this system was implemented and tested on 34 data sets, publicly available in well-known sources, such as kaggle or UCI (Table 3). These data sets were selected while bearing in mind key issues that might affect the performance of the meta-model for algorithm selection, including a number of data sets, diversity of the meta-features of the data sets, and diversity of the ML problems represented. That is, we looked for a relatively large number of data sets, from very different problems and domains. The degree of diversity of the resulting wealth of data can be assessed through visualizations, such as those depicted in Figure 2.

To have a common and unique framework for all data sets, independently of their type of ML problem, all data sets were transformed into binary classification problems. In multiclass data sets, a transformation was applied in which only the two most frequent target classes are

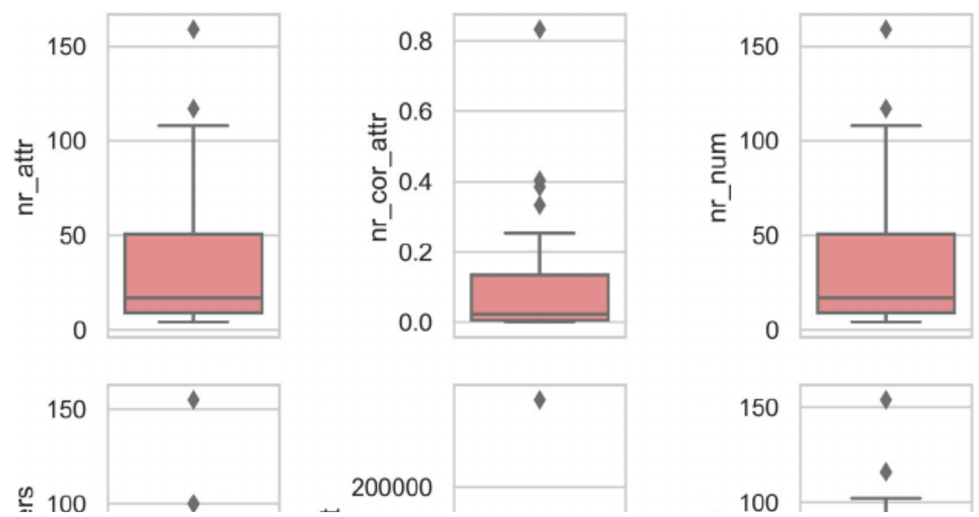
TABLE 3 Type and source of the 34 data sets used in this study

Data set	Source	Type
Adult income	https://www.kaggle.com/uciml/adult-census-income	Binary classification
Balance scale	https://archive.ics.uci.edu/ml/datasets/balance+scale	Multiclass classification
Balloons	https://archive.ics.uci.edu/ml/datasets/balloons	Binary classification
Bank	https://www.kaggle.com/lovelesh/bank-dataset	Binary classification
Bank marketing	https://www.kaggle.com/c/marketing-data	Binary classification
Breast cancer	https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin	Binary classification
Cars evaluation	https://archive.ics.uci.edu/ml/datasets/car+evaluation	Multiclass classification
Cereals	https://www.kaggle.com/crawford/80-cereals	Regression
Chess king rook	https://archive.ics.uci.edu/ml/machine-learning-databases/chess/king-rook-vs-king	Multiclass classification
Congressional voting	https://archive.ics.uci.edu/ml/datasets/congressional+voting+records	Binary classification
Credit card fraud	https://www.kaggle.com/mlg-ulb/creditcardfraud	Binary classification
Cryotherapy	https://archive.ics.uci.edu/ml/datasets/Immunotherapy+Dataset	Binary classification
Gender voice	https://www.kaggle.com/primaryobjects/voicegender	Binary classification
German credit data	https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)	Binary classification
Glass identification	https://archive.ics.uci.edu/ml/datasets/Glass+Identification	Multiclass classification

TABLE 3 (Continued)

Data set	Source	Type
Mushroom	https://archive.ics.uci.edu/ml/datasets/Mushroom	Multiclass classification
NBA logreg	https://data.world/exercises/logistic-regression-exercise-1	Binary classification
Pima native American diabetes	https://datahub.io/machine-learning/diabetes	Binary classification
School grades	https://archive.ics.uci.edu/ml/datasets/Student+Performance	Regression
Soccer international	https://www.kaggle.com/martj42/international-football-results-from-1872-to-2017	Multiclass classification
Student alcohol consumption—math grades	https://www.kaggle.com/uciml/student-alcohol-consumption	Regression
Student alcohol consumption—Portuguese grades	https://www.kaggle.com/uciml/student-alcohol-consumption	Regression
Tic tac toe	https://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame	Binary classification
Titanic	https://www.kaggle.com/c/titanic	Binary classification
Vehicle silhouette	https://archive.ics.uci.edu/ml/datasets/statlog+(vehicle+silhouettes)	Multiclass classification
Wine quality (red)	https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality	Regression
Wine quality (white)	https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality	Regression
Zoo animal	https://archive.ics.uci.edu/ml/datasets/Zoo	Multiclass classification

Figure 3 shows the average AUC score over the 34 data sets, in which Random Forest, with an



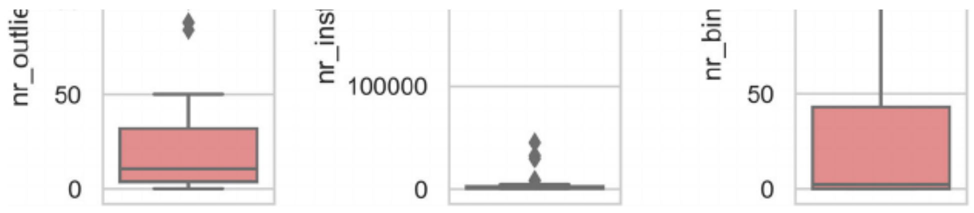
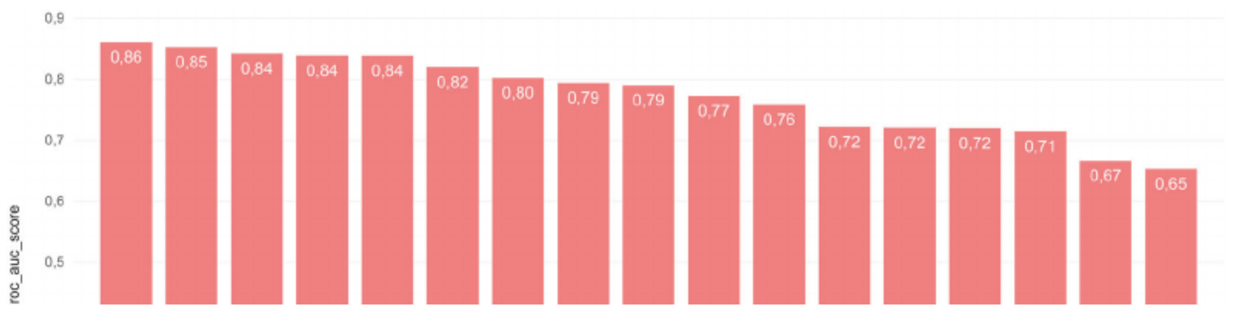


FIGURE 2 Some aggregated meta-features of the 34 data sets showing the diversity of the meta-data set. Number of attributes (nr_attr), number of attributes pairs with high correlation (nr_cor_attr), number of numeric attributes (nr_num), number of attributes with outliers values (nr_outliers), number of instances (nr_inst), number of binary attributes (nr_bin) [Color figure can be viewed at wileyonlinelibrary.com]



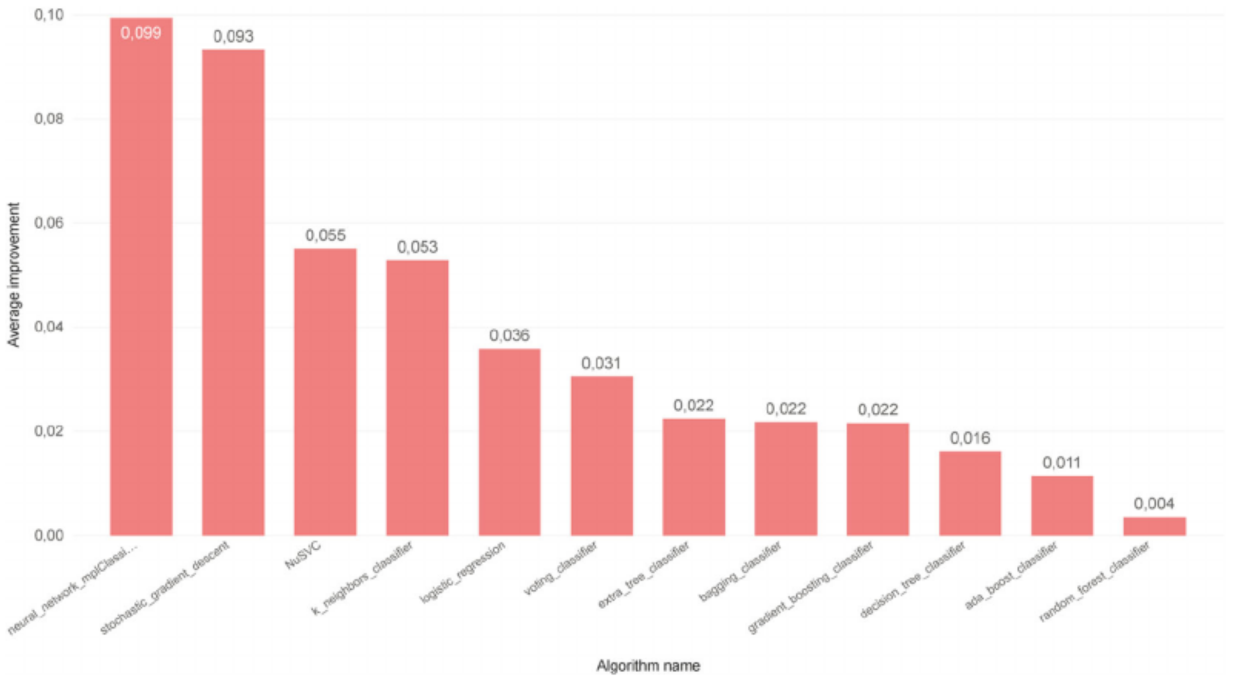


FIGURE 4 Average improvement of the algorithm performance (AUC) using Gridsearch. AUC, area under the receiver operating characteristic curve [Color figure can be viewed at wileyonlinelibrary.com]

The data sets that result from this process have a total of 118 columns: 108 meta-features, the algorithm name, five performance metrics, training time, a Boolean denoting whether a configuration resulting from hyperparameter optimization was used or not, the computation time of the meta data set, and the original data set name time of the meta-data set, and the original data set name.

5.1.2 | Algorithm ranking

In the era of streaming big data,[40] the task of algorithm ranking increases in complexity. In a batch scenario, algorithm ranking is a fairly easy task, based solely on accuracy metrics. While multiple metrics might be considered simultaneously (e.g., through a weighted sum), they are not conflicting. In a streaming scenario, on the other hand, training time may become an important factor

Heart Disease data set

e (s)	AUC (%)	Rank AUC	Rank time	Rank AUC St	Rank time St	β	β-Ranking
	81.86	1	6	0.00	0.31	0.09	1
	81.38	3	4	0.12	0.19	0.14	2
	80.59	5	1	0.25	0.00	0.17	3
	81.26	4	8	0.19	0.44	0.26	4
	79.01	8	2	0.44	0.06	0.32	5
	81.83	2	16	0.06	0.94	0.33	6
	79.28	7	12	0.38	0.69	0.47	7
	75.20	11	5	0.62	0.25	0.51	8
	80.22	6	17	0.31	1.00	0.52	9
	78.89	9	15	0.50	0.88	0.61	10
	78.59	10	13	0.56	0.75	0.62	11
	72.93	12	9	0.69	0.50	0.63	12
	65.16	13	7	0.75	0.38	0.64	13
	56.83	15	3	0.88	0.12	0.65	14
	64.86	14	14	0.81	0.81	0.81	15
	50.24	16	10	0.94	0.56	0.82	16
	49.39	17	17	1.00	0.62	0.89	17

The weights used are expected to represent the importance of each factor (accuracy and time) in the present ML problem. For instance, in a streaming scenario in which time is a very important restriction and models are required to adapt frequently and quickly even if their accuracy is not the best possible, time could have a higher weight than accuracy. In the case of this study, the weights of 0.7 and 0.3 for accuracy and time, respectively, were used.

β thus provides an overall performance metric for each algorithm, which combines prediction accuracy with training time, and reflects the relevance of both metrics. In this metric, 0 represents the best possible algorithm (an algorithm that ranked first in both AUC and time), while 1 represents the worst possible algorithm (an algorithm that ranked last in both AUC and time).

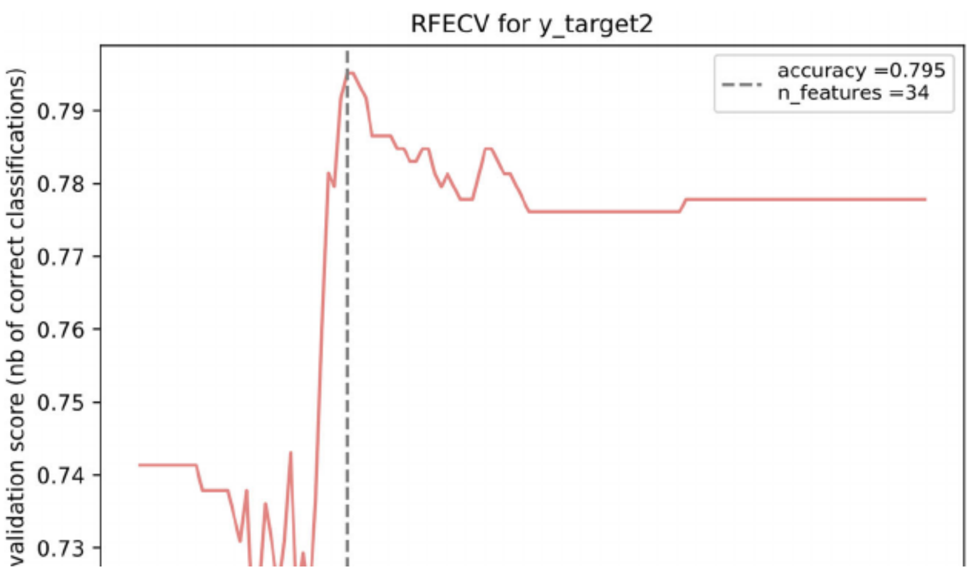
Table 4 shows the ranking of the 17 algorithms for the Heart Disease data set. The best algorithm for this data set, given the used weights, is the BernoulliNB, which ranks first in AUC and sixth in training time (although all the seven fastest algorithms have the same training time). The second best algorithm is LinearDistriminantAnalysis, which ranks third in AUC and fourth in time, and so on.

To address the challenge of predicting which algorithms are more suited to a given data set, the dependent variable y_target was created as follows: $y\_target = \beta - Ranking$ when, $\beta \leq 3$, otherwise $y\_target = others$. We thus address this problem as a multinomial classification problem with four classes.

The main disadvantage of this approach is that it produces highly unbalanced data sets as for every 17 lines of the data set, 14 of them have the value $others$. To deal with this issue the Synthetic Minority Oversampling Technique (SMOTE) technique was used.[41] SMOTE is a combination of undersampling the majority class and oversampling the minority class by creating "synthetic samples." To perform SMOTE, the library SMOTE from the imbalanced-learn package was used on the training data sets.

The resulting meta-data set is also relatively large in terms of the number of columns. At the same time, some of these features may not be relevant at all, or some may be correlated and, thus, redundant. In such large data sets, feature engineering and dimensionality reduction techniques can prove interesting to find the best combination of features, that is, the smallest set of the most expressive features.[42] In this study, the recursive feature elimination with cross-validation (RFECV) was used to select important attributes that contribute the most for the performance of the prediction. As Figure 5 shows, the best results are obtained with a selection of 34 features.

To assess the influence of these transformations, four different meta-data sets were created



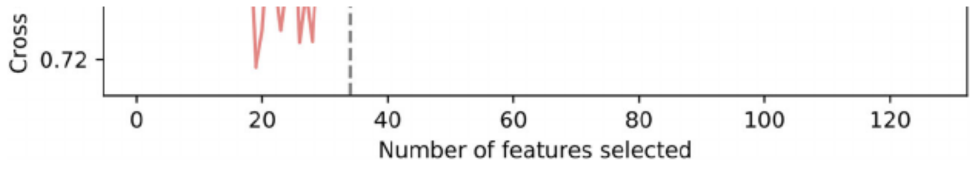


FIGURE 5 Output of the RFECV for the meta-data set with the *y_target* variable. RFECV, recursive feature elimination with cross-validation [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 5 Overview of the four meta-data sets used

Data set name	Feature Selection	SMOTE
meta-dataset_1	No	No
meta-dataset_2	No	Yes
meta-dataset_3	Yes	Yes
meta-dataset_4	Yes	No

Abbreviation: SMOTE, Synthetic Minority Oversampling Technique.

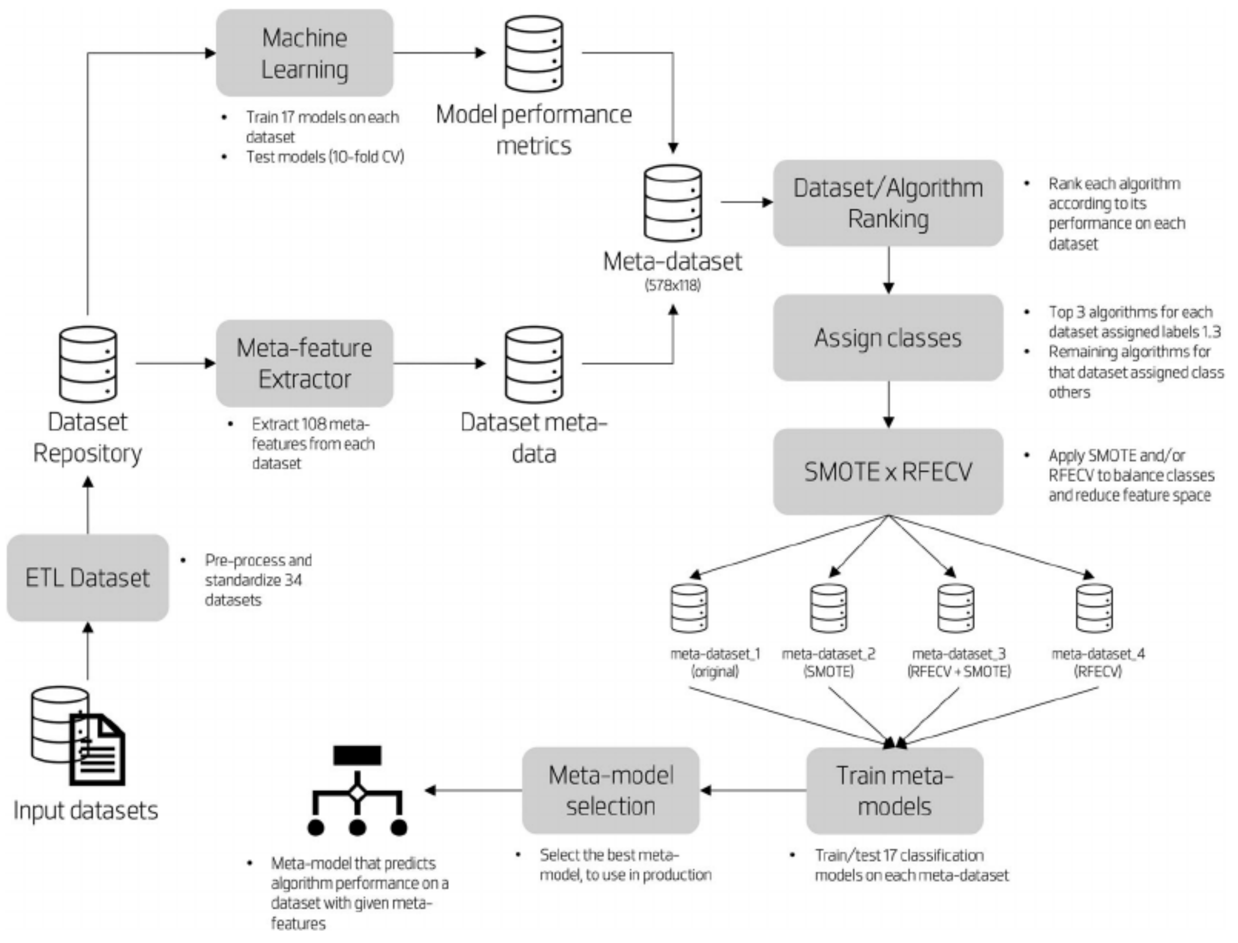

Machine Learning
Train 17 models on each dataset
Test models (10-fold CV)
Model performance metrics
Meta-dataset
(578x118)
Dataset/Algorithm Ranking
Rank each algorithm according to its performance on each dataset
Assign classes
Top 3 algorithms for each dataset assigned labels 1..3
Remaining algorithms for that dataset assigned class others
SMOTE x RFECV
Apply SMOTE and/or RFECV to balance classes and reduce feature space
Dataset Repository
Meta-feature Extractor
Extract 108 meta-features from each dataset
Dataset meta-data
ETL Dataset
Pre-process and standardize 34 datasets
Input datasets
meta-dataset_1 (original)
meta-dataset_2 (SMOTE)
meta-dataset_3 (RFECV + SMOTE)
meta-dataset_4 (RFECV)
Train meta-models
Train/test 17 classification models on each meta-dataset
Meta-model selection
Select the best meta-model, to use in production
Meta-model that predicts algorithm performance on a dataset with given meta-features

FIGURE 6 Methodology followed for implementing the Algorithm Selection mechanism. ETL, extract, transform, and load; RFECV, recursive feature elimination with cross-validation; SMOTE, Synthetic Minority

Oversampling Technique

set is selected, and the meta-model is deployed to predict the future performance of new models.

topics from the streaming platform and characterizing them as dependent or independent variables. The user must also set the frequency f at which the meta-model will be potentially updated, that is, the frequency at which a new base model will be trained to be potentially included in the ensemble. This frequency can be defined in terms of time or in terms of a number of new instances of data (e.g., update the model at each new 1000 instances).

The window of data that is used for training each new base model may, however, not be restricted to the new data that arrived in the meantime. The user can configure this through three variables:

- f—frequency at which new models are trained (in a number of instances or seconds);
- ws—the standard size of the window of data to consider (in a number of instances);
- wv—the random positive variation allowed for the window (%). A larger value will allow a higher overlap of data with previously trained models.

Figure 7 depicts this notion visually for $f = 1000$ instances, $ws = 1000$, and $wv = 100\%$. In this realistic scenario new base models are trained at each new 1000 instances of data, and considering 1000–2000 more recent instances.

The main goal of allowing this superposition between different windows of data is to reduce the sensibility of base models to noise. There is, however, a trade-off between the velocity of adaptation to new data and the robustness to noise. Smaller values of f, ws, and wv lead to a faster adaptation of concept drift but also to lower robustness to noise, and may lead the ensemble to overadapt to irrelevant, nonexisting, or transient patterns.

Whenever a new base model must be trained and a new window of data is selected, the data are obtained from the stream and split into training and test sets. A 70/30 train/test split with shuffle configuration is used, although these values can be easily changed in the configuration to suit other scenarios, according to the size of the data set. All this is done by the ML component.

The training set is used to train the new base model. Each new base model is validated using 10-fold cross-validation to obtain training performance metrics. The user can also choose to train the new base model with just a subset of the feature vector, namely, to avoid overfitting in the meta-model. The main hyperparameters that allow the user to control the complexity of each base model are thus the sample rate sr (which comes as a result from the size of the window) and the column sample rate csr.

the configuration used in the training of the model, the performance metrics (e.g., RMSE, Accuracy, and Precision), the date of training, or an index with the identifiers of the instances of data used in the training of the base model.

The test set is also added to a global test data pool. These data are used to assess the performance of the meta-model, so that it is evaluated using data that none of its base models knew during training.

During the initialization phase, the meta-model can answer prediction requests, even though the model pool is still not full. To do it, a bagging method is used in which each of the available base-models in the pool votes with the same weight towards the prediction of the meta-model. Predictions are done by the Predict component and can start as soon as the first base model is trained. To make a prediction, the Prediction module requests information from the meta-data database concerning which models are being used at the moment in the ensemble and the weight of each one. It then asks the corresponding models in the pool for their individualized prediction and combines them according to their weights, to provide the client with the final prediction.

The whole initialization process is visually depicted in Figure 8.

The initialization phase takes place between the moments in which the process is bootstrapped until the number of base models in the pool reaches the minimum size of the pool (represented by mm). Whenever this condition is reached, the system advances to the evolutionary phase, which is described in Section 5.2.2.

5.2.2 | Evolutionary phase

The Evolutionary phase is the second phase of the proposed system. It starts after the number of base models reaches mm and the system remains in this phase indefinitely. The main difference for the initialization phase is that now the number of models in the pool is constant. That is, if a new base model is added to the pool, another one must be removed. These operations are implemented by the Model Management component, which is able to load/unload models to/from memory. This only happens if this replacement is expected to result in a better meta-model. A similarity with the previous phase is that base model training continues at the same rate and following the same methodology.

In this phase the meta-model thus evolves in two different ways. On the one hand, it does so through the replacement of outdated base models, as described. On the other hand, there is also

in a more accurate meta-model. This process is implemented through a Genetic Algorithm, that runs indefinitely, and whose main goal is to find the optimum weights vector. This whole process is implemented by the Ensemble Optimization component of the architecture.

The implementation of this component follows the standard process of a Genetic Algorithm.[43] That is, an initial population of solutions, or chromosomes, is initialized. Their fitness is measured, and then a new population of solutions is generated through crossover and mutation operators that favor those solutions that have higher fitness.

The evolutionary process starts with the random creation of n solutions for the problem. In this context, a solution is a numeric vector of size mm in which each position represents the weight of a base-model in the voting scheme of the meta-model. Each solution is thus represented as a weights vector w. The initial population of the Genetic Algorithm is thus constituted by n vectors, with n representing the size of the population.

Each group of solutions is deemed a generation. The first generation is identified as g_1 and it is one of a set of G generations that will be created throughout the process. Each weight $w_{g,s,i}$, $g \in [1..G]$, $s \in [1..n]$, $i \in [1..t]$ represents the weight of the base learner i in the solution s

of generation g ($\sum_{i=1} w_{g,s,i} = 1, \forall\, s \in [1..n], \forall\, g \in [1..G]$).

For each new generation, the fitness of each solution is then calculated. To assess the fitness of each solution, all the base-models are used to predict the test data pool. Their predictions are then combined according to the solution being evaluated (i.e., a specific weight vector) and the resulting predictions (that represent the prediction of the meta-model) for the test data are compared with the actual values in the data. This step depends on the type of ML task. In regression tasks, the RMSE is calculated, while in classification tasks the accuracy is used. This process of evaluation results in a fitness vector f of size t, in which each fitness value $f_{g,s}$, $g \in [1..G]$, $s \in [1..n]$ represents the fitness of a given solution s in generation g.

Once the fitness calculation ends, the reproduction stage begins. The main goal of this stage is to generate new solutions that are potentially better by combining the characteristics of the best solutions. The reproduction process starts by selecting the b solutions with the highest fitness value. These solutions pass on to the next generation. Asides from these, a group of m new solutions created through mutation are added to the new generation. The mutation operator is implemented as a unary operator that takes as input one solution selected randomly from among the b best solutions. The selected solution is then mutated by changing two of its weights (i and j), selected at random, by an also random amount r, $r \in [0..mr]$.

The mr allows one to control the mutation rate, that is, how much we allow a solution to

This evolutionary process takes place every time a new base-model is trained, and its main consequence is to determine which base-models shall be part of the ensemble and what their weight should be.

The following list summarizes the hyperparameters that can be tuned when using this algorithm:

- sr—sample rate,
- csr—column sample rate,
- n—number of solutions in each generation,
- t—number of trees in the ensemble,
- b—number of best solutions to consider in each generation,
- mr—mutation rate, defining how much each solution should mutate,
- m—number of new solutions to generate through mutation, in each generation,
- rn—number of new random solutions to generate, in each generation,
- l—maximum number of iterations (stopping criterion),
- δ and x—define the reaching of convergence (stopping criterion).

Predictions made during the Evolutionary phase of the system follow the same logic of predictions done during initialization. However, each model may now have a different weight, according to its performance metrics. These weights are calculated by the Ensemble Optimization module and stored in the meta-data database. The Predict module thus obtains the names of the models of the ensemble and their weights from the meta-data database, asks the models for their individual predictions, and combines them according to their weights to provide a client with a prediction.

This process, through which the model pool and the weights of each model are gradually updated as new models are trained, is depicted in Figure 9.

6 | VALIDATION AND RESULTS

6 | VALIDATION AND RESULTS

This section is dedicated to evaluating the performance of both meta-learning approaches proposed in this paper. Section 6.1 is dedicated to the analysis of the results concerning algorithm selection, while Section 6.2 is dedicated to the use of meta-learning in data streaming scenarios with concept drift.

6.1 | Algorithm selection

To evaluate the performance of the approach proposed in Section 5.1, the following methodology was followed. Each of the 17 algorithms considered was used with each of the four meta-data sets described previously in Table 5, to train the respective models. All models were trained using 10-fold cross-validation and the relevant performance metrics were recorded: accuracy, precision, recall, *F*-measure, and AUC-ROC.

Accuracy is not a good metric if class labels are not uniformly distributed, as is the case. The F1 Score (the harmonic mean between precision and recall), on the other hand, combines a measure of how precise the classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances). It is generally assumed to be a better measure when dealing with imbalanced data sets. For that reason, this is the main metric that was considered to evaluate the trained models.

As Figure 10 shows, the generally best results are obtained when the SMOTE technique was applied. On the other hand, RFECV has not resulted in any benefit to the performance of the algorithms.

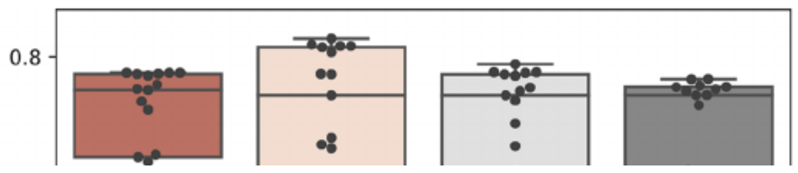
Table 6 shows the best five models (from among all the 68 models trained), and it is interesting to note that all of them are from meta-data sets in which the SMOTE technique was applied. In terms of performance metrics, ensemble methods show the best results, with the Bagging Classifier achieving 0.83 on the F1 Score.

From this study it results that the best decision is to use meta-dataset_2 and a Bagging Classifier for predicting the best algorithm for a given data set.

6.2 | Learning in streaming scenarios

To validate the proposed methodology for learning from streaming data, we tested it in two different data sets with concept drift. The first is a synthetic data set with concept drift created by the research team for the specific purpose of testing the system. The second is a well-known and publicly available weather data set with concept drift.[44]

Since the former is not publicly available, we provide a brief characterization. The data set is composed of 100,000 instances of three variables: two independent variables a and b and one target variable $f(a, b)$. Concept drift in this data occurs in different dimensions over time: (1)



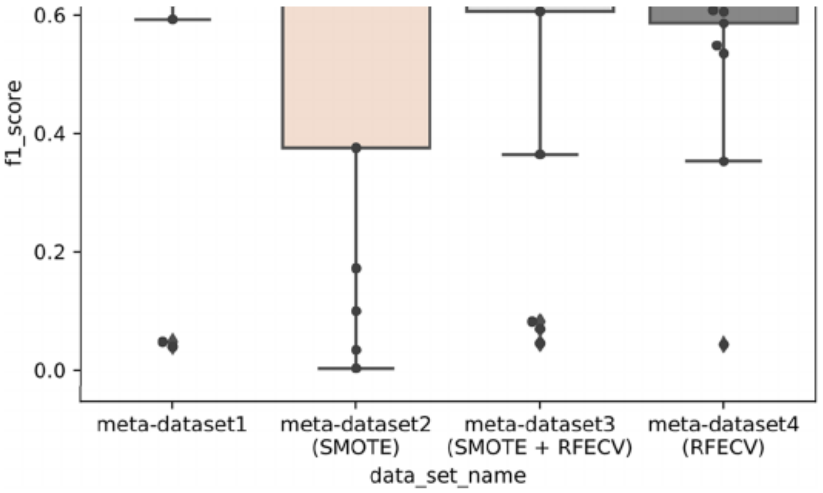


FIGURE 10 Distribution of the results for the different meta-data sets. Each boxplot combines the F1 scores of the 17 models trained with the corresponding meta-data set. RFECV, recursive feature elimination with cross-validation; SMOTE, Synthetic Minority Oversampling Technique [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 6 Top five best performing meta-data sets/algorithms concerning F1 Score

SMOTE	RFECV	Data set name	Algorithm name	Precision score	Recall score	F1 score
Yes	No	meta-dataset_2	Bagging Classifier	0.8344	0.8555	0.8309
Yes	No	meta-dataset_2	Extra Tree Classifier	0.7894	0.8555	0.8207
Yes	No	meta-dataset_2	Random Forest Classifier	0.7849	0.8555	0.8182
Yes	No	meta-dataset_2	Voting Classifier	0.7849	0.8555	0.8182
Yes	No	meta-dataset_2	K Neighbors Classifier	0.7857	0.8555	0.8161

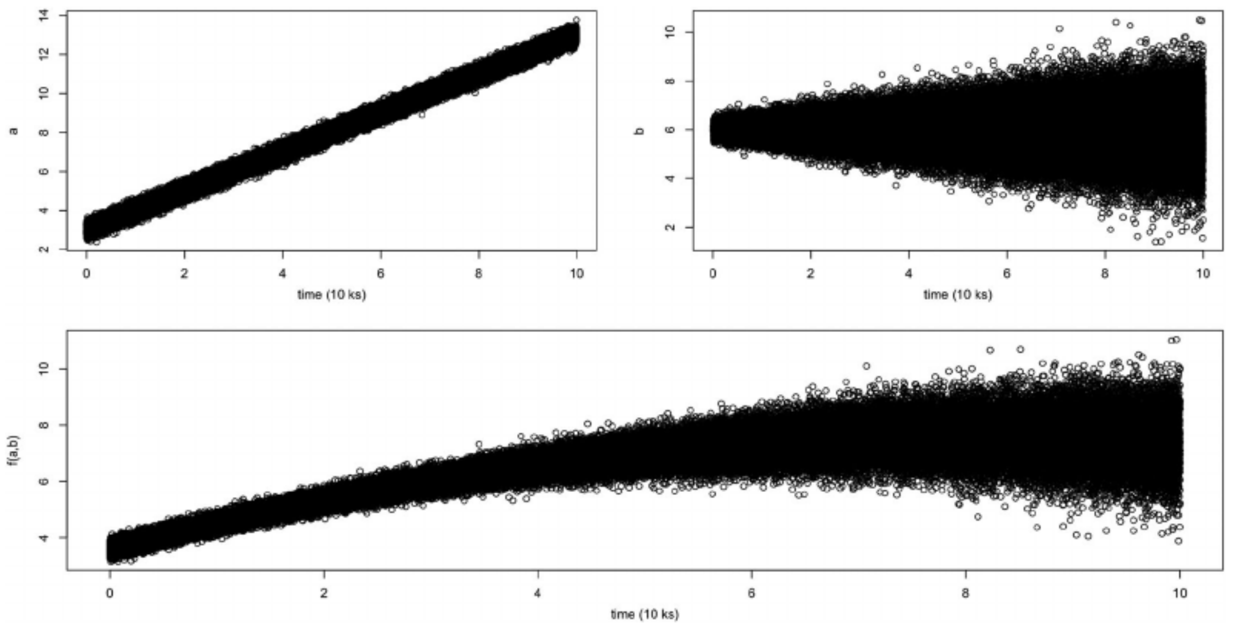
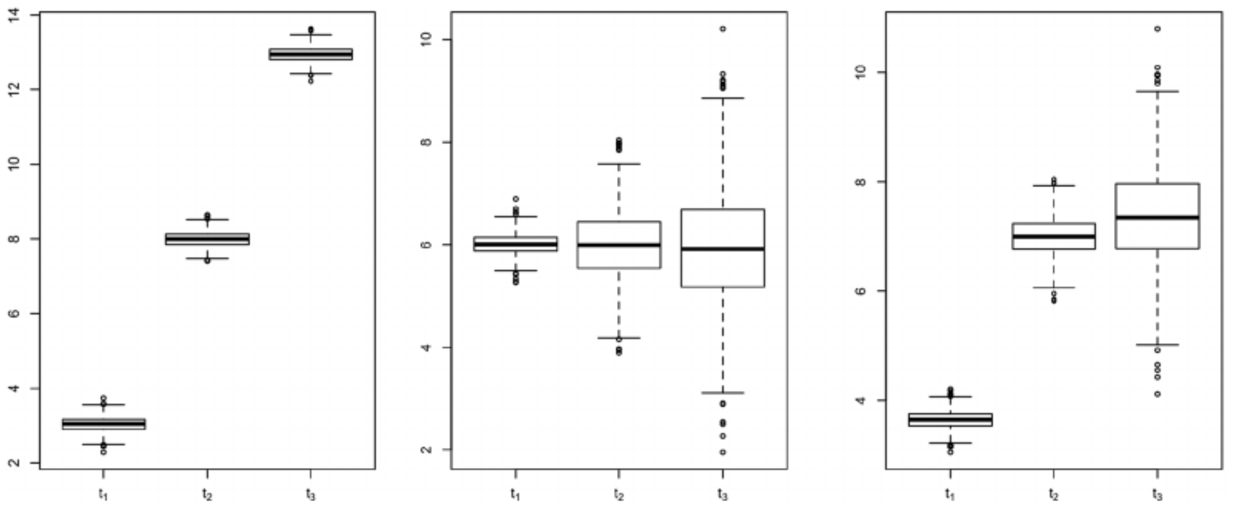


FIGURE 11 Plot of the two independent variables, a and b, and of the target variable, over time

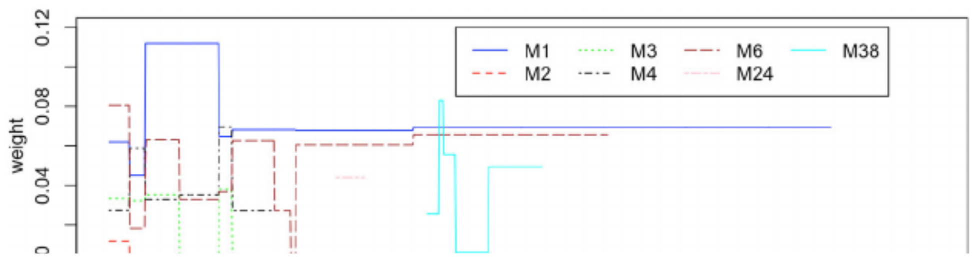


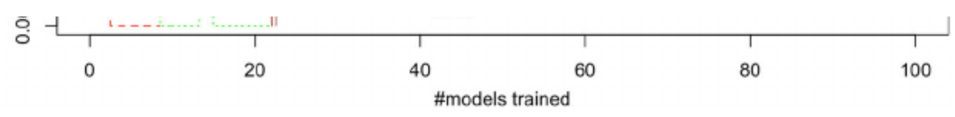
This has as consequence a change of the weights of the models in the ensemble over time, as well as some base models entering the ensemble while others leave. Figure 8 shows these dynamics over time. For simplicity only some of the 25 models of the ensemble were plotted. The figure shows how stronger base models, with higher weights, tend to remain in the ensemble longer until they are removed as they become increasingly outdated. The figure also shows when new models were trained and added to the ensemble (when their line starts) and when they are removed (when their line ends). In this figure, the x-axis represents the number of base-models trained since the beginning of the evolutionary process as no evolution takes place before the ensemble is complete and, during that initialization phase, all models weigh the same, as described previously.

Figure 13 also shows how models with smaller weights, which are related to worse performance, tend to be removed sooner (e.g., M2 and M3) while models with higher weights tend to last longer (e.g., M1 and M6).

Figure 14 shows how the measure of error (RMSE) of the ensemble evolves over time. An increase in error is visible, although it is insignificant (≈0.003). This confirms that the proposed approach can effectively maintain the predictor performance over time even when the statistical properties of the data change.

Figure 15, on the other hand, shows the evolution of a traditional ensemble over time, for the same data set. In this case, all base models have the same weight (0.04) and neither these weights nor the base models are changed once trained. The figure shows how the RMSE of the ensemble starts at a similar level to that of the proposed approach, but it quickly increases to





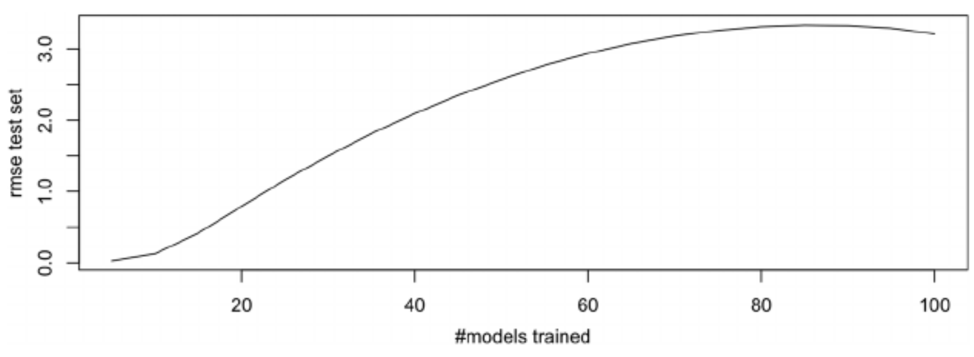


FIGURE 15 Evolution of the RMSE in a traditional ensemble, when concept drift is present. RMSE, Root-Mean-Square Error

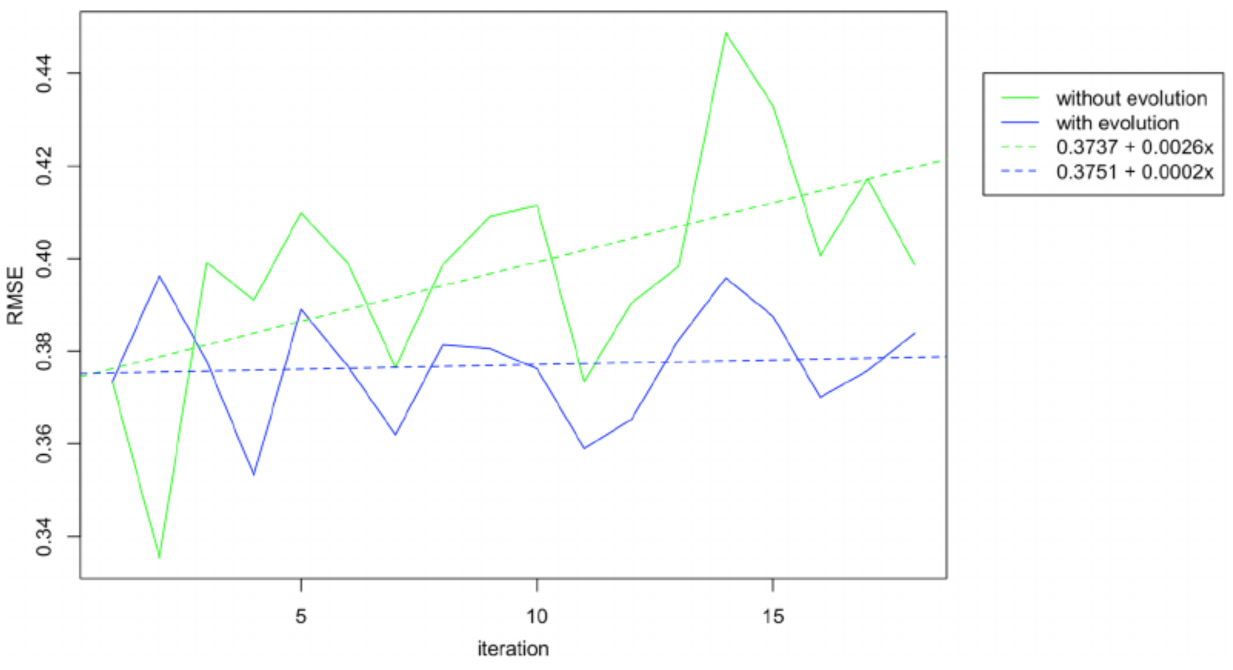


FIGURE 16 Evolution of the RMSE over time with and without the proposed method (solid lines). Dashed lines represent the trend of RMSE through a linear regression. RMSE, Root-Mean-Square Error [Color figure can be viewed at wileyonlinelibrary.com]

Results show, once again, that both ensembles start out with a similar measure of error. However, the RMSE tends to increase over time when there is no evolution of the ensemble, as the fit of a linear model to the data shows (green dashed line in Figure 11, slope = 0.0026). On the other hand, the error measure tends to remain constant over time when the proposed approach is used (blue dashed line in Figure 11, slope = 0.0002). Once again, this shows how the proposed approach is able to adapt the ensemble over time, maintaining the error measure.

Figure 17 shows how the relative importance of three of the variables changes over time in the ensemble. That is, there are variables that become more or less important over time for predicting temperature, as concepts change.

These experiments thus show how the proposed approach can effectively be used in data streaming scenarios with concept drift, allowing a model to adjust in a relatively efficient manner, by incorporating new base models while removing outdated ones.

7 | DISCUSSION AND CONCLUSIONS

In the last years, the field of ML has been facing new challenges. Most of them derive from the newly found value of data, which is nowadays collected and used in large volumes by virtually all competitive organizations and companies. However, volume is not the only challenge. Data are also more dynamic than in the past, in the sense that their statistical properties change over time, reflecting the faster changing phenomena that they represent. ML can thus no longer be a batch, one-shot operation. It is rather a continuous task, in which the search for the best model for the data at hand is constant.

The training of a model has, however, costs that are not negligible. These costs can be measured in terms of computational resources and time, and increase as the amount of data increases. Given the velocity at which decision-making happens in nowadays' world, time is actually often one of the main constraints when training a model. As a consequence, researchers are often unable to adequately explore the vast search space of algorithms and their configurations.

This paper addressed this issue by considering the potential of meta-learning. Specifically, we considered two main problems, which are also very frequent in current ML: algorithm selection and updating models in face of concept drift in the data. These two problems, which are interrelated, were addressed using a meta-learning ML framework developed by the team. While this framework current targets these two problems, it can be expanded in the future to answer more needs.

The ability of the proposed system to deal with these challenges was validated using publicly available data sets. Results point out that techniques inspired in meta-learning can be effectively used for: (1) ranking algorithms and predicting the most suitable ones for a specific ML task, speeding up the ML process by minimizing the number of models that are trained; (2) allowing trained models to evolve over time with minor modifications instead of retraining the whole model, making the adaptation to concept drift more efficient in terms of time and resources.

The work detailed in this paper has some key distinguishing characteristics, when compared with the existing literature. While the idea of using meta-learning for algorithm selection is not new (see, e.g.,Reference [45] in which a few statistical and information-theoretic features are used), we consider a much broader set of meta-features for describing data sets, of different categories. Specifically, we consider a total of 108 meta-features which, to the extent of our knowledge, is an unparalleled number. The same happens with the number and diversity of algorithms assessed: most of the existing literature focuses on a relatively small number of algorithms (e.g., References [46,47]) while this study assesses a total of 17 algorithms, of very different types. This results in an approach that is potentially more comprehensive in capturing the features of the data that really matter for predicting algorithm performance.

The approach is also more comprehensive when it comes to training/selecting the meta-model that will carry out the algorithm selection task. In the proposed approach, 17 algorithms were tested on four variants of the meta-data set, resulting in 68 candidate meta-models. Related works tend to follow a different approach: they analyze the performance of the trained models and use some heuristic to select the best algorithm. For instance, Shawkat and Smith [45] use the C4.5 (tree-based) algorithm to generate rules for algorithm selection. The proposed approach is different in the sense that it has a well-defined process that requires minimum user interaction, thanks to the incorporation of a meta-model for algorithm selection.

This is also useful in streaming scenarios. Indeed, Kerschke Pascal et al.[47] note that most of the existing work covers static offline algorithm selection. That is, scenarios in which the data sets are known beforehand and do not change with time. Nonetheless, most of the current ML

inefficient and slow to converge. In future work we will assess the performance and suitability of other optimization techniques, to select the most appropriate one.

In conclusion, this study shows that techniques inspired in meta-learning can be used to deal with some of the current challenges of ML, especially those related with the selection of the best models for a certain problem, or the continuous update of these models once trained.

ACKNOWLEDGMENTS

This study was supported by the Northern Regional Operational Program, Portugal 2020 and European Union, trough European Regional Development Fund (ERDF) in the scope of project number 39900—31/SI/2017, and by FCT—Fundação para a Ciência e Tecnologia within projects UIDB/04728/2020 and UID/CEC/00319/2019.

ORCID

José Pedro Monteiro https://orcid.org/0000-0003-3977-4786
Diogo Ramos https://orcid.org/0000-0003-2690-3489
Davide Carneiro https://orcid.org/0000-0002-6650-0388
Francisco Duarte https://orcid.org/0000-0001-9270-2226
João M. Fernandes https://orcid.org/0000-0003-1174-1966
Paulo Novais https://orcid.org/0000-0002-3549-0754

REFERENCES

1. Min C, Shiwen M, Yunhao L. Big data: a survey. *Mobile Netw Appl*. 2014;19:171-209.
2. Jie L, Anjin L, Fan D, Feng G, Joao G, Guangquan Z. Learning under concept drift: a review. *IEEE Trans Knowl Data Eng*. 2018;31:2346-2363.
3. Diego P-B, Bertha G-B. A survey of methods for distributed machine learning. *Prog Artif Intell*. 2013;2:1-11.
4. Krawczyk Bartosz M, Leandro L, João G, Jerzy S, Michal W. Ensemble learning for data stream analysis: a survey. *Inf Fusion*. 2017;37:132-156.
5. Joaquin V. Meta-learning. In: Hutter F, Kotthoff L, Vanschoren J, eds. *Automated Machine Learning*. Cham: Springer; 2019:35-61.
6. Adriano R, Garcia Luís P, Soares C, Vanschoren J, Carvalho André CPLF. Characterizing classification datasets: a study of meta-features for meta-learning. 2018. *Comp Res Repos. (CoRR)*. 2018:abs/1808.10406.

7. Bernardino R-P, Hane A, Nadia B-B, Massimiliano P. Multilinear multitask learning. In: Sanjoy D, David M, eds. *International Conference on Machine Learning*, Vol. 28. PMLR; 2013:1444-1452.
8. Pavel B, Christophe G-C, Carlos S, Ricardo V. *Metalearning—Applications to Data Mining*. Berlin

16. Murchhana T, Anita P. A study of algorithm selection in data mining using meta-learning. *Journal of Engineering Science and Technology Review*. 2017;10:51-64.
17. Wolpert David H. Stacked generalization. *Neural Net*. 1992;5:241-259.
18. Schapire Robert E, Yoav F. *Boosting: Foundations and Algorithms*. Cambridge, Massachusetts, USA: The MIT Press; 2012.
19. Amirhosein M, Sajedi HF, Bahram C, Massoud G, Dineva Adrienn A, Rafiei SE. Ensemble boosting and bagging based machine learning models for groundwater potential prediction. *Water Resour Manage*. 2021;35: 23-37.
20. Schwenker F. Ensemble methods: foundations and algorithms. In: Zhou Z-H, eds. *IEEE Computational Intelligence Magazine*. Machine Learning & Pattern Recognition Series. Vol. 8. IEEE; 2013:77-79. (Book Review). https://ieeexplore.ieee.org/document/6410720
21. Rich C. Multitask learning. *Mach Learn*. 1997;28:41-75.
22. Wolpert David H, Macready William G. No free lunch theorems for optimization. *IEEE Trans Evol Comput*. 1997;1:67-82.
23. Cullen Schaffer. A conservation law for generalization performance. In: Cohen W, Hirsh H, eds. *Machine Learning Proceedings 1994*. Elsevier; 1994:259-265. https://doi.org/10.1016/B978-1-55860-335-6.50039-8
24. Christophe G-C. Metalearning—a tutorial. In: *Tutorial at the 7th International Conference on Machine Learning and Applications (ICMLA)*. San Diego, CA, USA; 2008.
25. David H. *A Treatise of Human Nature*. Courier Corporation; 2003.
26. Pedro D. *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. New York, NY, USA: Basic Books Inc.; 2018.
27. Smith-Miles Kate A. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput Surv*. 2009;41:6:1-6:25.
28. Quan S. *Meta-learning and the Full Model Selection Problem* [Ph.D. thesis]. University of Waikato; 2014.
29. Vijayata W, Debajyoti M. Mobile agent based market basket analysis on cloud. In: *2014 International Conference on Information Technology*. IEEE; 2014:305-310.
30. João G, Indrė Ž, Albert B, Mykola P, Abdelhamid B. A survey on concept drift adaptation. *ACM Comput Surv (CSUR)*. 2014;46:1-37.
31. Sayuri IA, Paulo PJ. An overview on concept drift learning. *IEEE Access*. 2018;7:1532-1547.
32. Minku Leandro L, Xin Y. DDD: a new ensemble approach for dealing with concept drift. *IEEE Trans Knowl Data Eng*. 2011;24:619-633.
33. Rivolli A, Garcia Luís PF, Carlos S, Joaquin V, Carvalho André CPLF. Towards reproducible empirical research in meta-learning. 2018. *Comp Res Repos. (CoRR)*. 2018:abs/1808.10406.
34. Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-Learn: machine learning in Python. *J Mach Learn Res*. 2011;12:2825-2830.
35. Alexandros K. *Algorithm Selection Via Meta-learning* [Ph.D. Thesis]. University of Geneva; 2002.
36. Pavel B, João G, Bob H. Characterizing the applicability of classification algorithms using meta-level

44. Ryan E, Robi P. Incremental learning of concept drift in nonstationary environments. *IEEE Trans Neural Netw*. 2011;22:1517-1531.
45. Shawkat A, Smith KA. On learning algorithm selection for classification. *Appl Soft Comput*. 2006;6:119-138.
46. Rijn Jan N, Geoffrey H, Bernhard P, Joaquin V. Algorithm selection on data streams. In: Džeroski S, Panov P, Kocev D, Todorovski L, eds. *International Conference on Discovery Science*. Lecture Notes in Computer Science. Vol. 8777. Springer; 2014:325-336.
47. Kerschke Pascal H, Holger H, Frank N, Heike T. Automated algorithm selection: survey and perspectives.

Evol Comput. 2019;27:3-45.

How to cite this article: Monteiro JP, Ramos D, Carneiro D, Duarte F, Fernandes JM, Novais P. Meta-learning and the new challenges of machine learning. *Int J Intell Syst*. 2021;36:6240-6272. https://doi.org/10.1002/int.22549