UNIVERSITÉ NICE SOPHIA ANTIPOLIS

INTERNSHIP REPORT

Learning from experiments on Machine Learning Workflows

Author:
Miguel Fabián ROMERO RONDÓN

Supervisor:
Prof. Mireille BLAY-FORNARINO
Prof. Frédéric PRECIOSO

Report submitted in fulfillment of the requirements for the degree of Master IFI - Ubinet track

in the

I3S Laboratory, Sparks
Department of Computer Science

August 31, 2017

Université Nice Sophia Antipolis

Abstract

Faculty of Science
Department of Computer Science

Master IFI - Ubinet track

Learning from experiments on Machine Learning Workflows

by Miguel Fabián ROMERO RONDÓN

Ideally every researcher would know how all Machine Learning Algorithms perform, in order to select the appropriate Machine Learning Workflow that solves a given problem in the most efficient way. But this is far from reality, the task of selecting a Machine Learning Workflow to solve a prediction problem depends mainly on the background the researchers have, and what they consider "the best" from this knowledge. This project aims to guide the user in the selection of the most suitable workflow depending on the problem to be solved. We start from results of a previous work, where different Machine Learning Workflows were ranked according to their performance on a set of 101 training problems. The goal of this project is to predict which workflow(s) would be the best suited to solve an unseen problem.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A Machine Learning task can be expressed as a workflow mainly composed by four phases: Pre-processing, Machine Learning Algorithm, Parameters Tuning, and Evaluation. To build their Machine Learning Workflows, data scientists can use platforms like Weka [1], Orange [2], KNIME [3], RapidMiner [4] and ClowdFlows [5, 6], that allow users to create workflows by dragging and dropping components and connecting them.

Although these platforms are useful for data scientist. For them, big companies like IBM (IBM Watson), Amazon (AWS Machine Learning) and Microsoft (Azure Machine Learning) have built a set of tools that either solve the problem automatically, or advise a set of suitable algorithms according to the user needs. These platforms are limited in the sense that either they do not allow the user to know which Machine Learning Algorithm is being used in the background, or they have a very limited set of algorithms; another limitation is that they do not provide any help for the users in the generation of the whole Machine Learning Workflow suitable for the specific problem of the user.

To overcome these limitations, members of the SPARKS team (i3s laboratory), proposed the platform ROCKFlows, a platform whose goal is to help non-expert users in the creation of entire Machine Learning Workflows.

The project we present here aims to construct an important part of the ROCKFlows platform, that gives the possibility to predict the performance of different workflows over a dataset; then the system would be able to recommend to the user several workflows that would have a *good* performance on her dataset, according to what was learned from previous results.

1.1 Machine Learning Terminology

The ability to learn is one of the most fundamental attributes of intelligent behavior. The rising interest in Machine Learning is consequent with the progress in theory and computer modeling of learning processes; and factors like the growing amounts of available data, while the computational power gets cheaper with time, makes this field even more interesting [7].

Machine Learning (ML) has a wide range of applications, including social networks spam detection [8], medical diagnosis, search engines, cybersecurity [9], DNA sequence classification, speech and written language recognition, and supervised learning for Named Data Networking [10].

ML is a scientific discipline that deals with the design and development of algorithms that analyze and learn from empirical data. Learning here is in the context of capturing complex patterns of the unknown probability distribution behind the data. A more formal description comes from the book of Tom Mitchell [11]:

Definition 1.1.1. Machine Learning: An *agent* (a computer program) is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Depending on the type of feedback used to learn, a Machine Learning task is typically classified into one of the three following categories [12]:

- **Unsupervised Learning:** The *agent* learns patterns in the input, even though no explicit feedback is supplied. The most common unsupervised learning task is **clustering**: detecting potentially useful clusters of input examples.
- **Reinforcement Learning**: An *agent* takes *actions* in an *environment*, which is represented by some *reinforcements* - *rewards* or *punishments* that are fed back into the agent.
- **Supervised Learning**: The *agent* observes some example input-output pairs and learns a function that maps from input to output.

In this project we focus on supervised learning methods, but this work can be extended to the other categories, the only requirement to do so is to define the performance measures for the cases of *Unsupervised Learning* and *Reinforcement Learning*.

1.1.1 Supervised Learning

In the case of supervised learning, a dataset can be seen as examples that illustrate the relations between observed variables. The task of Supervised Learning consists in:

Definition 1.1.2. Supervised Learning: Given a training set of N example input-output pairs of the form: $(\boldsymbol{x_1}, y_1), (\boldsymbol{x_2}, y_2), ..., (\boldsymbol{x_N}, y_N)$, where $\boldsymbol{x_i}$ is the feature input vector of the i-th example and y_i the output (or target) value of the i-th example. The vector of inputs $\boldsymbol{x}$ and outputs $\boldsymbol{y}$ are related by the function $\boldsymbol{y} = f(\boldsymbol{x})$. The task of Supervised Learning consists in: Find the function h that approximates the true function f.

There are two approaches for Supervised Learning, when the target value to predict is a nominal value, the task is said to be a Statistical Classification task; and when the target value to predict is numeric, the task is said to be a Regression Analysis task [13].

Regression Analysis: is a statistical technique to analyze quantitative data and find the relationships among variables to be able to make forecasts. Regression Analysis looks for a relationship between the $\boldsymbol{x}$ (independent variable, input), and the $\boldsymbol{y}$ (dependent variable, output) [14].

Statistical Classification: is the problem of identifying to which set of categories (sub-populations) a new observation belongs, on the basis of a training dataset containing observations (or instances) whose category membership is known [15].

1.1.2 Dataset

To clarify the idea of supervised learning, we can use the dataset **Balloons**[1]. This dataset was used in an experiment done by Pazzani [17].

[1] We have included some changes from the original dataset to include the definitions of missing values and numeric attributes. You can find the original dataset in the UCI repository [16].

TABLE 1.1: Balloons Dataset

Color	**Size**	**Act**	**Age**	**Inflated (*Class*)**
Yellow	12.7	Stretch	32	T
Yellow	12.7	Stretch	13	T
Yellow	25.4	Dip	35	T
Yellow	12.7	Dip	11	T
Yellow	30.48	Stretch	40	T
Yellow	20.32	Stretch	9	F
Yellow	20.32	Dip	36	F
Yellow	30.48	Dip	7	F
Purple	12.7	?	38	T
Purple	12.7	Stretch	14	F
Purple	?	Dip	42	F
Purple	12.7	Dip	8	F
Purple	30.48	Stretch	26	T
Purple	30.48	Stretch	12	F
Purple	20.32	Dip	28	F
Purple	30.48	Dip	10	F

The subjects of the experiment first look at photographs of a person doing something with a balloon. The photographs differ in terms of the color of the balloon (yellow or purple), the size of the baloon ($\in \mathbb{R}$), the age of the person ($\in \mathbb{N}$), and the action the person is doing (stretching the balloon or dipping the balloon in water). The subjects have to predict whether the balloon will be inflated when the person blows into it.

You can refer to the Table 1.1 where we represent the dataset *Balloons* in tabular form, each sample (row in the table) consists of a set of attributes (columns in the table) and a classification value (class-column in the table). The first row of the table represents the case of an adult (32 years old) successfully inflating a small (12.7 cm long), yellow balloon that had been stretched.

We can use now this dataset, together with Definition 1.1.2, to define some terminology used in this report, it can be summarized in the following terms:

Dataset: A set of N pairs of the form $(\boldsymbol{x_i}, y_i)$, where $\boldsymbol{x_i}$ is the feature input vector and y_i the class value of the i-th instance. The example dataset *Balloons* is a set of 16 pairs of the form $(\boldsymbol{x_i} = [\{Yellow, Purple\}, \mathbb{R}, \{Stretch, Dip\}, \mathbb{N}], y_i = \{T, F\})$.

Instance: Each of the pairs $(\boldsymbol{x_i}, y_i)$ of the dataset. For example, the last row of our dataset *Balloons*: $(\boldsymbol{x_i} = [Yellow, \mathbb{R}, Stretch, \mathbb{N}], y_i = F)$, represents an instance of a child (10 years old) that couldn't inflate a small (12.7 cm long), yellow balloon that had been stretched.

Attribute: - Also known as **Feature** - is each of the dimensions of the $\boldsymbol{x_i}$ vector, it is represented as each column of the dataset in tabular form. Our example dataset has 4 attributes: $Color$, $Size$, Act, Age.

Nominal Attribute: Is an Attribute that only takes values from a finite set. In our example, $Color$ and Act are Nominal Attributes of the dataset.

Numeric Attribute: Is an Attribute that takes values from an infinite Number Set (e.g. $\mathbb{N}, \mathbb{Z}, \mathbb{R}$). In our example, $Size$ and Age are Numeric Attributes.

In addition to nominal and numeric attributes, some datasets may have further attribute types like the Date String and Relational attributes described in the ARFF format [18].

Date Attribute: Is an Attribute with date values. In the case of Weka [1], it combines the date and time format *yyyy-MM-dd-THH:mm:ss* with four digits for the year, one for the month and one for the day, then the letter T followed by the time with two digits for each of hours, minutes, and seconds.

String Attribute: Is an Attribute with textual values.

Relational Attribute: Is a type of attribute typically used in multi-instance classification problems, these are problems where a single feature vector is not enough to describe an object, then the objects are represented by sets of feature vectors [19].

Class Attribute: Is the Target Attribute that identifies the class of each instance, (y_i in our notation). In classification tasks, this attribute is nominal, while in regression tasks, this attribute is numeric. In the *Balloons* dataset, the Class Attribute is $Inflated$.

Nominal Attribute Values: Is the set of possible values a Nominal Attribute can take. For example the nominal attribute values of attribute $Color$ are $\{Purple, Yellow\}$.

Class Label: Is each value of the set of possible values the Class Attribute can take. Can be also seen as the value y_i of the i-th instance. In our example, the class label of the first instance is T.

Multi-Class vs Binary-Class problems: If the Class Attribute contains more than two Nominal Attribute Values, the classification problem is a multi-class problem. Otherwise, it is a binary class problem, like our example here, whose class labels are T or F.

Missing Values: Each instance contains one value for each attribute. If for any attribute, the value is not known, it is considered as a missing value. Missing values may exist for different reasons such as corrupted data, where the reason for missingness is completely random (Missing Completely at Random), or not recorded data, where the probability that an observation is missing depends on information that is observed (Missing at Random) or not observed (Missing Not at Random) [20]. In our example, missing values are represented by a question mark ('?').

Multi-Output problems: If there is more than one Class Attribute, the problem is a Multi-Output problem. In other words, the task is Multi-Output if the dataset is a set of N pairs of the form ($\boldsymbol{x_i}$, $\boldsymbol{y_i}$), where $\boldsymbol{x_i}$ is the feature input vector and $\boldsymbol{y_i}$ is the vector of class values of the i-th instance.

Incremental Learning: Also called online learning, is a method of Machine Learning, in which the training data is continuously used to update the learner.

Batch Learning: Opposite method of Incremental Learning, in batch learning all the instances of the training data are used to train the learner at once.

1.1.3 Supervised Classification Workflows

A supervised classification task can be expressed as a pipeline, this pipeline is what we call in this project a workflow, and is composed mainly by four phases: Preprocessing, Classification Algorithm, Parameters Tuning, and Evaluation [21] as represented in Figure 1.1.

- **Pre-processing:** The first step of a Machine Learning Workflow consists of an optional pre-processing technique. The purpose of the pre-processing technique depends on the nature of the Dataset and the Machine Learning Algorithm to be used. A pre-processing technique may be required to clean the

Dataset, to make it compatible with the Classification Algorithm; or to modify the values in the dataset in order to get a better performance of the Classification Algorithm.

- **Classifier:** After the pre-processing phase, a Classification Algorithm is used to train over the processed dataset. In the data mining literature, there are different flavors of Classification Algorithms, each one with distinctive features that result in the diversity of performances observed in the *No-Free-Lunch Theorem*, that states that "any two optimization algorithms are equivalent when their performance is averaged across all possible problems", meaning that there is not a single classifier that performs well for all possible datasets (or problems) [22].

- **Parameter Tuning:** Some algorithms depend on parameter settings that are typically tuned to get the full potential of the algorithm itself, different values on these parameters may change the performance of the Classification Algorithm, making this a critical task in the Machine Learning pipeline [23].

  - The parameters set during the *parameter tunning* are known as **hyperparameters**, they are some parameters inherent to the Machine Learning model that cannot be learned during the training process, and for this reason they need to be tunned before the training process begins. Typically the hyperparameters express high-level concepts of the model such as complexity or learning rate.

- **Evaluation:** The last phase of the workflow is the evaluation of the results. Once we have defined a pre-processing technique, chosen a classifier and set its hyperparameters, we want to test whether the workflow will lead to *good* performances or not with an evaluation method.

  - The simplest evaluation method used is to train the classifier on one subset of the dataset, called the training set, and test it on the complementary subset of the dataset, called the testing set, then the prediction made by the classifier is compared against the groundtruth values, to get a metric of the accuracy as a performance indicator of the classifier.
  - Typically, the accuracy obtained by only one test set is not a good performance metric. In order to have a more statistically valuable performance indicator, we may train and test the classifier on different subsets of train and test. The N**-fold cross validation**, is a method that splits the dataset d into N parts, called folds, $d_1, \ldots, d_N$, and then uses $N-1$ folds to train, while keeping one fold to test. This procedure is done N times, using each time a different fold to test. The performance metric for accuracy is then the average of the N accuracies.

If we assume the *evaluation* phase is the same for all the workflows, to be able to compare them, we can express a Machine Learning Workflow as a triplet (p, s, m) where p represents a set of pre-processing techniques applied on the dataset, s represents the parameter strategy used to optimize the hyperparameters of the algorithm, and m is a Machine Learning Algorithm used to learn a model over the pre-processed data, to predict results over new data.

The construction of a Machine Learning Workflow depends upon two main aspects:

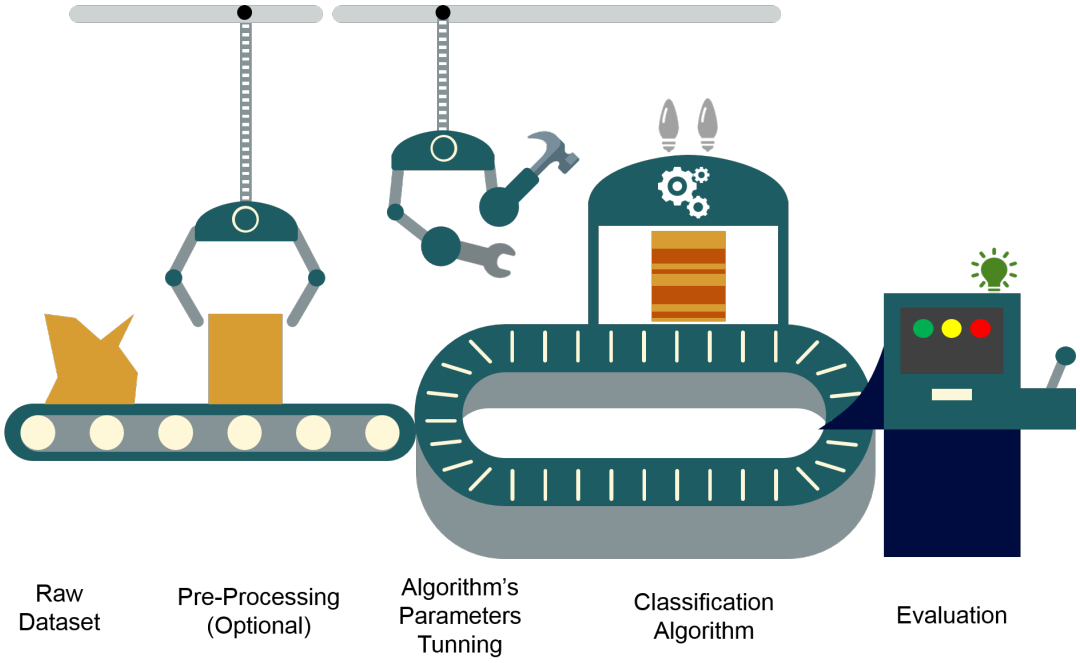


FIGURE 1.1: Typical Supervised Classification Workflow.

- The structural characteristics (size, quality and nature) of the collected data.
- How the results will be used.

1.2 Meta-Learning

The task of building a ML Workflow consists in choosing the correct pre-processing techniques together with the right algorithm as well as the correct tuning of its parameters, and this task is highly complex because we have to test over a high number of classifiers arising from different families and areas of knowledge [24].

Sometimes the researchers don't know which factors affect the performance of an algorithm over a dataset, either because the problem is too complex or because the algorithm itself is hard to analyze.

To decide which algorithm to choose, scientists often use algorithms that arise from areas in which they are experts, and can leave aside algorithms that are more exotic to them, but could perform better for the problem they are trying to solve.

The idea of Meta-Learning appears to help solving this problem. Meta-Learning consists in using Machine Learning Algorithms applied on Meta-Data about Machine Learning experiments to predict the performance of future experiments. A more formal definition comes from the survey of Lemke et al [25]:

Definition 1.2.1. Meta-Learning System: A Meta-Learning system must include a meta-learner L_{meta}, which adapts with experience, and this experience is gained by exploiting meta-knowledge T_{meta} extracted from the evaluation of previous workflows ($L_{base} = (p, s, m)$) over different datasets T_{base}.

Each dataset T_{base} in the Meta-Learning system is described by a series of features known as **Meta-Features**, they are a set of morphological characteristics of the

TABLE 1.2: Example Meta-Dataset for L_{base}=(p:Attribute Selection, s:Grid-Search, m:SVM)

Dataset Name	Number of Instances	Number of Attributes	Number of Missing Values	...	Metric - Accuracy
Arrythmia	452	279	408	...	0.71
Balloons	16	4	2	...	0.50
Breast-Cancer	286	9	9	...	0.70
...	...	...	...	...	...
Zoo	101	16	0	...	0.96

dataset that affect the performance of a classifier when it is evaluated for the specific problem T_{base}.

The meta-knowledge, also called Meta-Dataset in this report, can be seen as a dataset of datasets. Using our previous definition of dataset (1.1.2), the Meta-Dataset can be defined as *a set of N pairs of the form $(\boldsymbol{x_i}, y_i)$*, where $\boldsymbol{x_i}$ is the input vector of Meta-Features extracted from the i-th dataset (T^i_{base}), and $\boldsymbol{y_i}$ the performance measure(s) of the workflow L_{base} over the i-th dataset.

To clarify this idea we can use the *Balloons* dataset from Table 1.1 as our T_{base}, let's also assume that we have a Machine Learning Workflow L_{base}=(p:Attribute Selection, s:Grid-Search, m:SVM), and this workflow got an evaluation accuracy of 0.5 on the dataset *Balloons*. In the entry for *Balloons* Dataset in our Meta-Dataset, $\boldsymbol{x_i}$ will be the set of Meta-Features (number of instances, number of attributes, number of missing values, etc.) of the *Balloons* dataset, and y_i will be 0.5. The example Meta-Dataset for this workflow is shown in Table 1.2.

From the example Meta-Dataset in Table 1.2, we may deduce that the proportion of missing values ($\frac{number\ of\ missing\ values}{number\ of\ attributes \times number\ of\ instances}$) is correlated with the Accuracy of the workflow. The work of the Meta-Learning system is to learn these kind of rules.

1.3 ROCKFlows

Our object of study is the ROCKFlows[2] platform, it is a project aiming to help users to create their own Machine Learning Workflows by simply describing their dataset and objectives. For this purpose, ROCKFlows adopts a software product line (SPL) approach. Software Product Line engineering is concerned with systematically reusing development assets in an application domain [26, 27]. It is similar to mass customization in traditional industry, aiming to develop and evolve software systems as quality products, with reduced development effort and time-to-market.

A study done by Luca Parisi et al. [28] in 2016, conducted a series of experiments on 101 datasets from the UCI repository [16], 66 classifiers, and 12 different pre-processing techniques. After analyzing the results of these experiments, they could assess the impact of some dataset properties and pre-processing operations on the performance of the workflows.

The goal of our project is to use those results to design a Meta-Learning system that automatically generate constraints to filter out algorithms that do not match the user needs or do not perform well on the input dataset, and output the behavior (e.g. accuracy) of a workflow.

[2] http://rockflows.i3s.unice.fr

Chapter 2

Contemporary Meta-Learning Solutions

The original description of the workflow selection problem stated by Rice [29] says that given a space of instances and a space of workflows, map each instance-workflow pair to its performance. This mapping can then be used to select the best workflow for a given instance.

As almost all the contemporary approaches use Machine Learning, the Figure 2.1 sketches more accurately the systems that use Machine Learning to solve this problem. Figure 2.1, and the following description were extended from the survey done by Kotthoff [30] to include a more general description of workflow selection: "At the heart is the selection model S, which is trained using Machine Learning techniques. The data for the model comes from the workflows $W \in \boldsymbol{W}$ and the problems $x \in P$, which are characterized by features. S is created either by using training data that contains the performances of the workflows on a subset of the problems from the problem space, or feedback from executing the chosen workflow on a problem and measuring the performance. Some approaches use both data sources".

The model S makes the prediction of a specific workflow W given a problem x. This workflow is then used to solve the problem.

An effective Meta-Learning service needs to solve the fundamental problem of deciding which Machine Learning Algorithm to use on a given dataset, whether and how to pre-process its features, and how to tune the hyperparameters to get the best score for a given evaluation metric [21].

There are many different methodologies applicable to solve the problem of algorithm selection, in the following sub-sections we state the most prevalent ones.

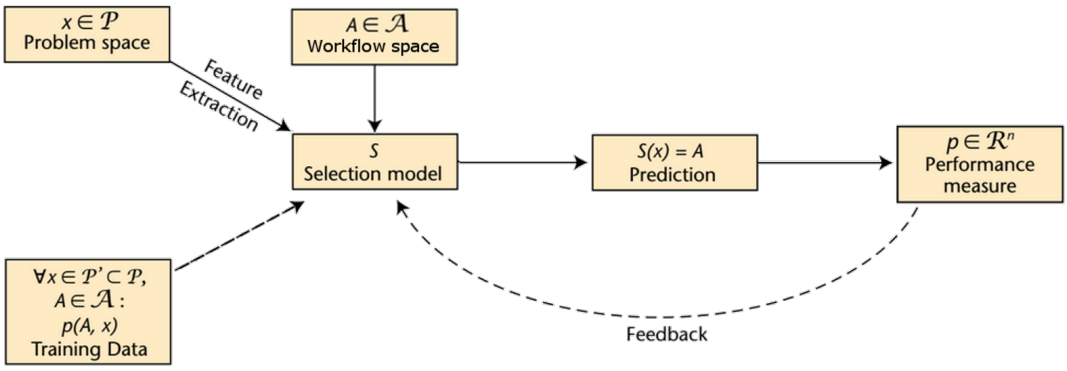


FIGURE 2.1: Contemporary workflow selection model [30].

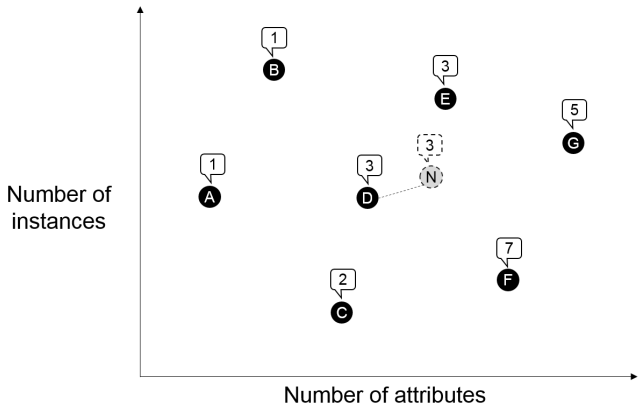


FIGURE 2.2: Case-based reasoning: Black points represent the case base (problem instances $\{A, B, C, D, E, F, G\}$ associated with selected algorithms $\{1, 2, 3, 4, 5, 6, 7\}$ from results of past performances). The gray point represents the new problem instance N (algorithm selected is 3, the same of its nearest neighbor), the nearest neighbor D is determined by computing the Euclidean distance.

2.1 Case-based reasoning

In this approach, the idea is to plot each problem as a point in a hyperplane, associating the algorithm that performs the best on each of them, and then when predicting over a new problem, select the algorithm that was associated to the nearest point of it. In Figure 2.2, we present an illustration of this approach using 2 dimensions to plot the instances. AQME system [31] is an example of a case-based reasoning system, it uses nearest-neighbor classifier to select the best algorithm.

2.2 Classification

Label each problem instance with the algorithm that should be used to solve it. Then we can use a ML Classification Algorithm to predict over a new problem instance. In Figure 2.3 we present an illustration of how this approach works.

Examples of Classification Algorithms:

- Bayesian Network Classifiers
- Decision Trees, Grafted Decision Trees, Best First Decision Trees
- Multiclass Alternating Decision Trees
- Decision Tables
- Hyper Pipes
- Propositional Rule Learners
- Support Vector Machines
- Artificial Neural Networks
- 1R classifier
- Random Trees and Random Forests

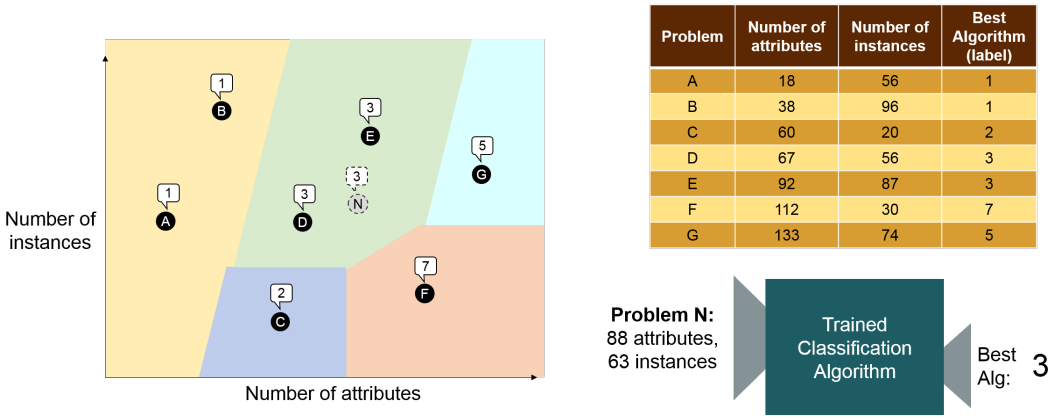


FIGURE 2.3: Classification: The Classification Algorithm is trained with the set of problem instances labeled with the algorithm used to get the best performance. Then the trained Classification Algorithm can be used to predict "the best" algorithm for a new instance.

2.3 Regression

This approach consists in predicting the performance of each algorithm on a given problem independently and then select the best one. Then we have to use the set of problems as input data of a Meta-Learning algorithm, and this will allow us to predict the performance of each algorithm on a new instance, as illustrated in Figure 2.4.

Examples of Regression Algorithms:

- Gaussian Processes
- Regression SVM
- Linear Regression
- Logistic Regression
- Multivariate Adaptive Regression Splines
- Principal Component Regression

2.4 Statistical Relational Learning

Instead of predicting labels (as in classification) or values (as in regression), this approach attempts to predict complex structures. Here we would like to predict the performance ranking of the algorithms from the set on a particular problem. An example is the Support Vector Machine SVM_{rank} instantiation of SVM_{struct} [32].

Note that all the contemporary Meta-Learning approaches solve the problem of algorithm selection, but none of them focus on the workflow selection problem. In the next chapter (See Chapter 3) we will describe our Strategy of Research to build a Meta-Learning system that is able to predict the performance of different Machine Learning Workflows, we will see if we can use the different solutions presented here, but to predict the performance of different workflows.

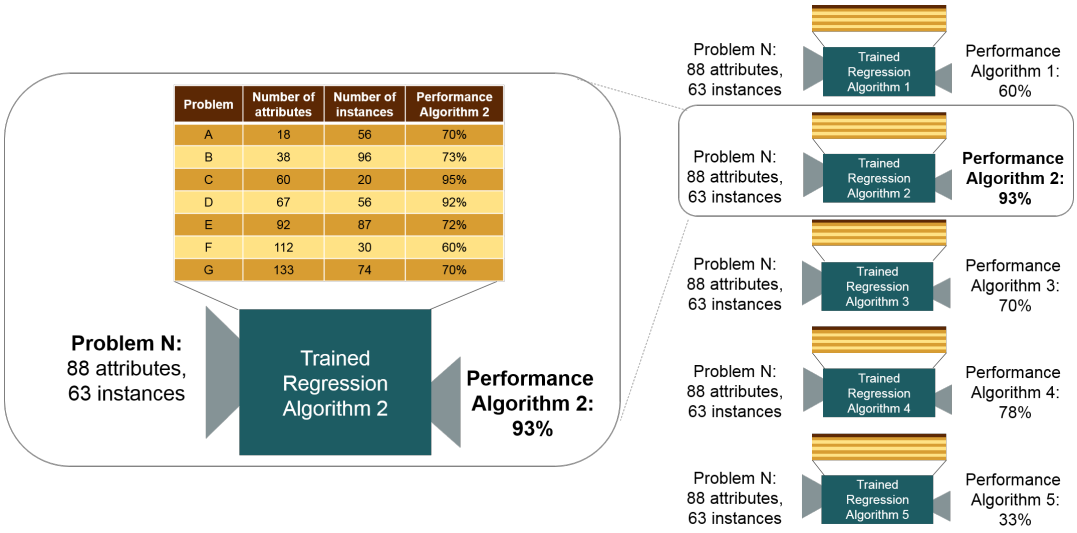


FIGURE 2.4: Regression: (**Left**) Each Machine Learning Algorithm is trained using the performance of each algorithm on all datasets. (**Right**) When you want to predict on a new instance, you have to predict the result on each algorithm and then pick the algorithm with highest performance.

Chapter 3

Strategy of Research

Our goal is to build a Meta-Learning system that takes as input a set of Meta-Features extracted from a dataset and outputs the behavior (e.g. accuracy) of a workflow.

3.1 Experiments on ML Workflows

From the work of Parisi et al. [28], we have the results of the experiments of 101 datasets, evaluated for 66 classifiers, and 12 pre-processing techniques. The list of pre-processing techniques and ML Algorithms can be found on Table 3.1 and Table 3.3 respectively. We extended the number of classifiers to 68, including the Deep Learning Algorithms: *Stack Denoising Autoencoders* (SdA, $id = 67$) and *Deep Belief Networks* (DBN, $id = 68$). The preprocessing techniques were also extended to 16, by adding the pre-processings with ids from 13 to 16 in Table 3.2. In this work we also defined a list of parameter strategies, you can find them in Table 3.4.

For each workflow, they computed the following average values (obtained either on the 4-Fold cross validation or on the 10-Fold cross validation):

- Average accuracy
- Time of:
  - Pre-processing
  - Parameter Tuning
  - Training
  - Testing
- Amount of memory occupied by the trained classifier.

Instead of using directly the averaged value, in this work they ranked the workflows to distinguish whether there are *relatively significant differences* between the performances of two workflows or not. Then they assigned to the evaluation of each workflow a rank value, which groups together the workflows who are not significantly different among each other.

The results of these experiments can be expressed in tabular form as shown in Table 3.5, where we present a subset of the results for dataset Balloons. Each row represents the results of the performance of each workflow on the dataset *Balloons*[1], a workflow is described by the first three columns (Algorithm, Pre-processing ID and Parameter Strategy), and the results are labeled according to the ranking obtained, 6 labels are used here: Best, Great, Good, Average, Bad, Worst.

[1]The full table of results can be found in https://github.com/ROCKFlows/experiments-public/blob/master/results/full-results/balloons/Final-Analysis-10Folds.xlsx

TABLE 3.1: List of pre-processing techniques used in our experiments

ID	Pre-processing Name	Description
0	$p_0 = \{\emptyset$	No pre-processing: Original dataset
1	$p_1 = \{Discretize$	Replacing numerical attributes with nominal ones by simple binning, the maximum number of bins is 10
2	$p_2 = \{RepMissVals$	Replace Missing Values: Use the attribute mean (or majority nominal value) to fill the missing values
3	$p_3 = \begin{cases} p_2 \\ p_1 \end{cases}$	Replace Missing Values and Discretize
4	$p_4 = \{NomToBin$	Nominal to Binary: Replace a nominal attribute by many binary attributes. A nominal attribute with k distinct values is transformed into k binary attributes, by using the one-attribute-per-value approach
5	$p_5 = \begin{cases} p_2 \\ p_4 \end{cases}$	First Replace Missing Values and then replace Nominal attributes to Binary ones
6	$p_6 = \{AttSel$	Attribute Selection: Reducing the number of attributes by removing the irrelevant ones. The attributes kept are those who have low correlation with the others and are highly correlated with the class attribute.
7	$p_7 = \begin{cases} p_1 \\ p_6 \end{cases}$	First Discretize and then perform Attribute Selection
8	$p_8 = \begin{cases} p_2 \\ p_1 \\ p_6 \end{cases}$	First Replace Missing Values, then Discretize, and finally perform Attribute Selection
9	$p_9 = \begin{cases} p_4 \\ p_6 \end{cases}$	Transform Nominal attributes into Binary and then perform Attribute Selection
10	$p_{10} = \begin{cases} p_2 \\ p_4 \\ p_6 \end{cases}$	Replace Missing Values, Nominal to Binary, Attribute Selection
11	$p_{11} = \begin{cases} NomToNum \\ Stndr \\ ReplMissBy0 \end{cases}$	Nominal to Numeric, Standardize and Replace Missing Values by Zero: First convert all nominal attributes into numeric by using a simple quantization: if a nominal attribute x may take discrete values $v_1, ..., v_n$ when it takes the discrete value v_i, it is converted to the numeric value $i \in 1, ..., n$. Then, standardize each attribute so that its mean is equal to zero and its standard deviation is equal to 1. Finally replace all missing values with the value 0.
12	$p_{12} = \begin{cases} p_{11} \\ p_6 \end{cases}$	Perform Attribute Selection to the dataset preprocessed with pre-processing 11.

TABLE 3.2: List of additional pre-processing techniques used in our experiments

ID	Pre-processing Name	Description
13	$p_{13} = \{Standardize$	Standardize each attribute so that its mean is equal to zero and its standard deviation is equal to 1.
14	$p_{14} = \begin{cases} DelMissVals \\ p_{13} \end{cases}$	Remove the instances with missing values then standardize.
15	$p_{15} = \begin{cases} NomToNum \\ p_{13} \end{cases}$	First convert all nominal attributes into numeric by using a simple quantization: if a nominal attribute x may take discrete values $v_1, ..., v_n$ when it takes the discrete value v_i, it is converted to the numeric value $i \in 1, ..., n$. Then, standardize each attribute so that its mean is equal to zero and its standard deviation is equal to 1.
16	$p_{16} = \begin{cases} DelMissVals \\ p_{15} \end{cases}$	First remove the instances with missing values, then convert all nominal attributes into numeric by using a simple quantization: if a nominal attribute x may take discrete values $v_1, ..., v_n$ when it takes the discrete value v_i, it is converted to the numeric value $i \in 1, ..., n$. Then, standardize each attribute so that its mean is equal to zero and its standard deviation is equal to 1.

We can use these experiments to train our Meta-Learning system to predict the performance (e.g. 'Good' or 'Bad') of a workflow for a given property (e.g. Accuracy). So let's see first how to arrange this data to fit our definition of Meta-Learning (Definition 1.2.1).

3.2 Meta-Dataset construction

Taking into account the definition of Meta-Dataset (see Section 1.2.1), the Meta-Dataset is composed by one input: the dataset (described by the Meta-Features); and one output: the performance result(s). The first thing to consider here is that instead of having only one metric, we have 3 different metrics (Accuracy, Total Time and Model Size). Then, instead of having only one output, we may have 3 different outputs. This can be solved using a **Multi-Output Classification Algorithm**.

Finally, we can arrange our Meta-Dataset as represented in Table 3.6 where we have an instance corresponding to the dataset Balloons for the Meta-Dataset of SVM (with parameter strategy: grid-search). Note here that if we want to use the least number of Meta-Datasets (hence the least number of sub-models), we need to have one Meta-Dataset for each algorithm, but in this case we may have one different output for each pre-processing technique, we assume for now that this problem can be handled by the **Multi-Output Classification Algorithm**.

As you may have noticed in Table 3.6 we have some missing values in the output. It can happen in our experiments, either because we deliberately computed the performance of the algorithm for only a few pre-processing techniques, or because

TABLE 3.3: List of Classifiers

Id	Classifier Name	Id	Classifier Name
1	Random Forest	35	Ridor
2	Bagging NBTree	36	Bagging PART
3	Hyper Pipes	37	J48
4	MultiboostAB NBTree	38	Bagging J48
5	Bagging RandomTree	39	PART
6	Logistic Regression	40	OrdinalClassClassifier J48
7	IB1	41	Bagging RepTree
8	Svm	42	Classification ViaRegression, M5P
9	Smo	43	LogitBoost Decision Stump
10	IBk	44	DTNB
11	RandomComittee RandomTree	45	Random Subspaces of RepTree
12	Decorate, J48	46	MultiboostAB, RepTree
13	Multilayer Perceptron	47	JRip
14	RotationForest RandomTree	48	Bagging JRip
15	MultiboostAB, RandomTree	49	Classification via Clustering: FarthestFirst
16	NBTree	50	Bagging Decision Table
17	Naive Bayes	51	Dagging SMO
18	AdaboostM1, J48	52	Bagging LWL
19	MultiboostAB, Decision Table	53	Rep Tree
20	MultiboostAB, Naive Bayes	54	Decision Table
21	Bagging Naive Bayes	55	LWL
22	RBF Network	56	Classification via Clustering: KMeans
23	Random Tree	57	MultiboostAB OneR
24	1-vs-1 Alternating Decision Tree	58	Bagging OneR
25	Bayesian Network	59	OneR
26	MultiboostAB J48	60	Bagging Decision Stump
27	MultiboostAB, PART	61	MultiboostAB DecisionStump
28	NNge	62	Conjunctive Rule
29	Logistic Model Tree	63	AdaboostM1, Decision Stumps
30	Bagging Hyper Pipes	64	Decision Stump
31	Simple Logistic	65	Raced Incremental Logit Boost, Decision Stumps
32	RotationForest J48	66	Alternating Decision Tree
33	MultiboostAB JRip	67	SdA
34	VFI	68	DBN

TABLE 3.4: List of parameter strategies used in our experiments

ID	Parameter Strategy Name	Description
1	default	Use default values for each hyperparameter of the Machine Learning algorithm
2	grid-search	Given the ranges of exploration for each parameter. Evaluate exhaustively each possible combination of values for the parameters
The following parameter strategies are used in Deep Learning. Here some useful definitions: **Normal:** Initialize the weights with a Normal Distribution with Mean 0 and Standard Deviation around 0.1. **UP:** Stop when the generalization loss increased in 5 successive epochs. **PQ:** Stop when the quotient between generalization loss (GL) and progress training error P_k exceeds 1, where $k = 5$ and P_k is defined as: $P_k(t) = 1000 \times \left(\frac{\sum_{t'=t-k+1}^{t} E_{tr}(t')}{k \times \min_{t'=t-k+1}^{t} E_{tr}(t')} - 1 \right)$ where $E_{tr}(t)$ is the Training error after epoch t.		
3	Normal, UP	
4	Normal, PQ	
Xavier: Initialize the weights with a Uniform Distribution $Uniform(-r, r)$, where r is defined as: $r = \sqrt{\frac{6}{fan_{in}+fan_{out}}}$ fan_{in} and fan_{out} are the number of inputs and outputs of the neuron respectively.		
5	Xavier, UP	
6	Xavier, PQ	
Patience: Stop when n iterations have passed after finding a minimum and a new optimum has not been found, where n is defined as twice the length of the test set.		
7	Xavier, Patience	
8	Normal, Patience	
GL: Stop as soon as the generalization loss (G_{Loss}) exceeds 10%, where G_{Loss} is defined as: $G_{Loss} = 100 \times \left(\frac{E_{va}(t)}{E_{opt}(t)} - 1 \right)$ $E_{va}(t)$ is the validation error after epoch t, and $E_{opt}(t)$ is the lowest validation error obtained in epochs up to t.		
9	Normal, GL	
10	Xavier, GL	

TABLE 3.5: Example of Experiment results obtained for *Balloons* dataset in the case of 10-Fold cross validation.

Algorithm	**Pre-processing ID**	**Parameter Strategy**	**Accuracy Avg**	**Status Accuracy**	**Total Time (ms)**	**Status Total Time**	**Trained Model Size (bytes)**	**Status Model Size**
Alternating Decision Tree	0	0	0.9	Best	1	Great	6620	Good
SVM	12	1	0.5	Average	6	Average	4888	Good
SVM	11	1	0.9	Best	0	Best	5656	Good
AdaBoost M1, Decision Stumps	6	0	0.7	Great	0	Best	5912	Good
Bagging LWL	4	0	0.65	Good	4	Average	32830	Bad
Conjunctive Rule	12	0	0.6	Average	3	Average	1493	Best
NNge	4	0	0.45	Bad	0	Best	20068	Bad
Hyper Pipes	4	0	0.3	Worst	0	Best	8496	Average
Decision Table	9	0	0.5	Average	34	Bad	10777	Average

TABLE 3.6: First Version of Meta-Dataset for SVM (AlgId=8, PstId=2) - Instance for dataset *Balloons*

Dataset	**Acc-P_0**	**...**	**Acc-P_{11}**	**Acc-P_{12}**	**Time-P_0**	**...**	**Time-P_{12}**	**Size-P_0**	**...**	**Size-P_{12}**
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Balloons	**?**	**...**	**Best**	**Average**	**?**	**...**	**Average**	**?**	**...**	**Good**
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

TABLE 3.7: Second Version of Meta-Dataset. Meta-Dataset $T_{meta}^{8,12,2,1}$ for SVM-$Prep_{12}$-$Pstr_2$-Accuracy - Instance for dataset *Balloons*

Dataset	**Accuracy-P_{12}-$Pstr_2$**
⋮	⋮
Balloons	**Average**
⋮	⋮

the ML Algorithm may not work for the dataset without pre-processing, in this case SVM does not work for the dataset *Balloons* without pre-processing.

The missing values in the output represent a problem for the **Multi-Output Classification Algorithm**, because if the class is missing, these algorithms typically decide not to use the instances, and we may lose data.

To solve this situation, instead of using a Meta-Dataset for each ML Algorithm, we need to use one Meta-Dataset for each workflow and each property, you can find the new Meta-Dataset format in Table 3.7. The difference of this Meta-Learning system with the one represented in Table 3.6, is that now, instead of having one learner L_{meta}^{i} for each algorithm $\in i$ (68 learners), we would need one learner $L_{meta}^{i,j,k,l}$ for each workflow (68: algorithms $\in i$, 16: pre-processings $\in j$, 10: parameter strategies $\in k$) and performance metric (3: properties $\in k$), then ≈ 3264 learners ($68 \times 16 \times 3$). Note that we exclude the parameter strategies, because in our experiments only the two Deep Learning algorithms and SVM have different parameter strategies, the other algorithms use the default parameter strategy, and then the order of magnitude in the number of sub-models is not greatly affected by it.

3.3 Meta-Features

According to our definition of Meta-Dataset (see Section 1.2.1), it is a set of N pairs of the form $(\boldsymbol{x_i}, y_i)$, where $\boldsymbol{x_i}$ is the input vector of Meta-Features extracted from the i-th dataset (T_{base}), and y_i the performance measure of the workflow $L_{base}^{j,k,l}$ (algorithm: j, pre-processing: k, parameter strategy: l), over the i-th dataset, we will now define the Meta-Features x_i that will represent each dataset.

We can start describing our datasets with simple features like the number of instances, the number of attributes, the number of nominal attributes, the number of numeric attributes, the number of classes and the number of missing values. If we describe the dataset in this way, our Meta-Dataset would look like the one shown in Table 3.8.

One should pay particular attention to the Meta-Features defined to distinguish each dataset. These Meta-Features can be conceived as a specific collection of morphological characteristics of a dataset, that jointly affect the performance of a classifier when it is applied to the particular task represented by the dataset [33].

We performed an exhaustive search on the literature to find the most important and frequently used Meta-Features, in the list below we present the Meta-Features found in papers [33, 34, 35, 36, 37, 38, 39].

For a better understanding of this list, you can refer to the following notation: The dataset has N instances and M attributes, $x_{i,j}$ denotes the value of instance i in

TABLE 3.8: Meta-Dataset for SVM-$Prep_{12}$-$Pstr_2$-Accuracy with Meta-Features – Second version

Dataset	**# inst.**	**# atts.**	**# nom. att.**	**# num. att.**	**# classes**	**# missing vals.**	**Acc-P_{12}-$Pstr_2$**
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Balloons	**16**	**4**	**2**	**2**	**2**	**2**	**Average**
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

attribute j, $x_i = \{x_{i,j} \; \forall \; j\}$. Att_j denotes the attribute j, y_i denotes the class value of instance i, and $y = \{y_1, y_2, \ldots, y_N\}$.

Other useful functions are:

$IsMissing(x)$: is a function that returns 1 if value x is missing and 0 otherwise.

$Unique(x)$: is a function that returns the set of unique values the nominal attribute x has.

$IsNominal(x)$: is a function that returns 1 if attribute x is nominal and 0 otherwise.

$IsNumeric(x)$: is a function that returns 1 if attribute x is numeric and 0 otherwise.

$IsString(x)$: is a function that returns 1 if attribute x is of type string and 0 otherwise.

$IsDate(x)$: is a function that returns 1 if attribute x is of type date and 0 otherwise.

$IsRelational(x)$: is a function that returns 1 if attribute x is relational and 0 otherwise.

$max(x)$, $min(x)$, $mean(x)$, $std(x)$, $mean_{geom}(x)$: are the common functions that return the maximum, minimum, average, standard deviation and geometric mean of array x respectively.

- **Number of Instances (N)** [33, 34, 35, 36, 37, 38, 39]: It represents the total number of samples in the dataset.
- **Natural Logarithm of Number of Instances ($\ln(N)$)** [36].
- **Number of Attributes (M)** [33, 34, 35, 36, 37, 38, 39]: Represents the total number of attributes (excluding the class) of the dataset. (i.e. the length of the input vector $\boldsymbol{x_i}$).
- **Number of Classes (C)** [33, 34, 35, 36, 37, 38, 39]: Represents the total number of output values in the dataset.

$$C = |Unique(y)|$$

- **Dimensionality (D)** [33, 35, 38, 39]: Ratio between the number of attributes and the number of instances constituting the dataset.

$$D = \frac{M}{N}$$

- **Number of Missing Values (Mv)** [35, 37, 39]:

$$Mv = \sum_{i=1}^{N} \sum_{j=1}^{M} IsMissing(x_{i,j})$$

- **Proportion of Missing Values (Pmv)** [34, 35, 37, 38, 39]: Number of missing values as a proportion of the total number of values.

$$Pmv = \frac{Mv}{M \times N}$$

- **Number of Instances with Missing Values (Imv)** [37, 38]:

$$Imv = \sum_{i=1}^{N} \begin{cases} 1, & \text{if } \exists v \in x_{i,j} \; \forall j \mid IsMissing(v) = 1 \\ 0, & \text{otherwise} \end{cases}$$

- **Proportion of Instances with Missing Values ($Pimv$)** [37]:

$$Pimv = \frac{Imv}{N}$$

- **Number of Instances with Missing Class (Imc)**:

$$Imc = \sum_{i=1}^{N} IsMissing(y_i)$$

- **Proportion of Instances with Missing Class ($Pimc$)** [37]:

$$Pimc = \frac{Imc}{N}$$

- **Number of Binary Attributes ($Mbin$)** [35, 37, 39]:

$$Mbin = \sum_{i=1}^{M} \begin{cases} 1, & \text{if } IsNominal(Att_i) \wedge |Unique(Att_i)| = 2 \\ 0, & \text{otherwise} \end{cases}$$

- **Proportion of Binary Attributes ($Pbin$)** [34, 36, 38]:

$$Pbin = \frac{Mbin}{M}$$

- **Number of Unary Attributes ($Muna$)**:

$$Muna = \sum_{i=1}^{M} \begin{cases} 1, & \text{if } IsNominal(Att_i) \wedge |Unique(Att_i)| = 1 \\ 0, & \text{otherwise} \end{cases}$$

- **Proportion of Unary Attributes ($Puna$)**:

$$Puna = \frac{Muna}{M}$$

- **Number of Date Attributes (M_{Dat})**:

$$M_{Dat} = |\{i \in \{1, \ldots, M\} \mid IsDate(Att_i) = 1\}|$$

- **Proportion of Date Attributes (P_{Dat})**:

$$P_{Dat} = \frac{M_{Dat}}{M}$$

- **Number of String Attributes (M_{Str})**:

$$M_{Str} = |\{i \in \{1, \ldots, M\} \mid IsString(Att_i) = 1\}|$$

- **Proportion of String Attributes (P_{Str})**:

$$P_{Str} = \frac{M_{Str}}{M}$$

- **Number of Relational Attributes (M_{Rel})**:

$$M_{Rel} = |\{i \in \{1, \ldots, M\} \mid IsRelational(Att_i) = 1\}|$$

- **Proportion of Relational Attributes (P_{Rel})**:

$$P_{Rel} = \frac{M_{Rel}}{M}$$

- **Number of Nominal Attributes (M_{Nom})** [35, 37, 39]: We can define the set of Nominal Attributes as:

$$NomAtts = \{i \in \{1, \ldots, M\} \mid IsNominal(Att_i) = 1\}$$

$$M_{Nom} = |NomAtts|$$

- **Proportion of Nominal Attributes (P_{Nom})** [34, 35, 36, 38, 39]:

$$P_{Nom} = \frac{M_{Nom}}{M}$$

- **Max Number of Nominal Attribute Values ($MaxNomVals$)** [35, 38, 39]:

$$MaxNomVals = \max\left(\{|Unique(Att_i)| \ \forall i \in NomAtts\}\right)$$

- **Min Number of Nominal Attribute Values ($MinNomVals$)** [35, 38, 39]:

$$MinNomVals = \min\left(\{|Unique(Att_i)| \ \forall i \in NomAtts\}\right)$$

- **Mean Number of Nominal Attribute Values ($MeanNomVals$)** [35, 38, 39]:

$$MeanNomVals = \text{mean}\ \left(\{|Unique(Att_i)| \ \forall i \in NomAtts\}\right)$$

- **Std Number of Nominal Attribute Values ($StdNomVals$)** [35, 39]:

$$StdNomVals = \text{std}\ \left(\{|Unique(Att_i)| \ \forall i \in NomAtts\}\right)$$

- **Majority Class Size ($MaxCS$)** [37, 38]:

$$MaxCS = \max\left(\left\{\sum_{i=1}^{N}\begin{cases}1, & \text{if } y_i = j\\ 0, & \text{otherwise}\end{cases} \forall j \in Unique(y)\right\}\right)$$

- **Proportion of Majority Class ($PMaxC$)** [36, 37]:

$$PMaxC = \frac{MaxCS}{N}$$

- **Minority Class Size ($MinCS$)** [38]:

$$MinCS = \min\left(\left\{\sum_{i=1}^{N}\begin{cases}1, & \text{if } y_i = j\\ 0, & \text{otherwise}\end{cases} \forall j \in Unique(y)\right\}\right)$$

- **Proportion of Minority Class ($PMinC$)**:

$$PMinC = \frac{MinCS}{N}$$

- **Number of Numeric Attributes (M_{Num})** [35, 37, 39]: We can define the set of Numeric Attributes as:

$$NumAtts = \{i \in \{1, \ldots, M\} \mid IsNumeric(Att_i) = 1\}$$

$$M_{Num} = |NumAtts|$$

- **Proportion of Numeric Attributes (P_{Num})** [35, 38, 39]:

$$P_{Num} = \frac{M_{Num}}{M}$$

- **Mean of Means of Numeric Attributes ($MMNA$)** [37, 38]

$$MMNA = \text{mean}\,(\,\{\text{mean}\,(\,\{x_{i,j} \forall j \in \{1, \ldots, N\}\})\ \forall i \in NumAtts\})$$

- **Standard Deviation Ratio (SDR)** [33, 34, 35, 37, 39]:

$$SDR = \text{mean}_{geom}\,(\,\{\text{mean}\,(\,\{x_{i,j}\ \forall j \in \{1, \ldots, N\}\})\ \forall i \in NumAtts\})$$

- **Coefficient of Variation ($VarCoef$)** [33]: We can define the Coefficient of Variation of Attribute i as:

$$VarCoef_i = \frac{\text{std}\,(\,\{x_{i,j}\ \forall j \in \{1, \ldots, N\}\})}{\text{mean}\,(\,\{x_{i,j}\ \forall j \in \{1, \ldots, N\}\})}$$

$$VarCoef = \text{mean}\,(\,\{VarCoef_i \forall i \in NumAtts\})$$

- **Covariance (Cov)** [33]: We can define the mean of the values of attribute i as:

$$att_{mean}(i) = \text{mean}\,(\,\{x_{i,j}\ \forall j \in \{1, \ldots, N\}\})$$

$$Cov_{i,j} = \sum_{k=1}^{N} \frac{x_{i,k} - att_{mean}(i)}{N-1}$$

$$Cov = \text{mean}\ (\ \{Cov_{i,j} \forall i,j \mid i,j \in NumAtts \wedge j > i\})$$

- **Multi Attribute Correlation ($MAttCorr$)** [33, 34, 35, 36, 39]: We can define the standard deviation of the values of attribute i as:

$$att_{std}(i) = \text{std}\ (\ \{x_{i,j}\ \forall j \in \{1,\ldots,N\}\})$$

$$\rho_{i,j} = \frac{Cov_{i,j}}{\sqrt{std(i) \times std(j)}}$$

$$MAttCorr = \text{mean}\ (\ \{\rho_{i,j}\ \forall i,j \mid i,j \in NumAtts \wedge j > i\})$$

- **Skewness ($Skew$)** [33, 34, 35, 36, 37, 38, 39]: We can define the skewness of attribute i as:

$$att_{skew}(i) = \frac{1}{att_{std}(i)^3} \times \frac{\sum_{k=1}^{N} (\ x_{k,i} - att_{mean}(i))\ ^3}{N}$$

$$Skew = \text{mean}\ (\ \{att_{skew}(i)\ \forall i \in NumAtts\})$$

- **Kurtosis ($Kurt$)** [33, 34, 35, 36, 37, 38, 39]: We can define the kurtosis of attribute i as:

$$att_{kurt}(i) = \frac{1}{att_{std}(i)^4} \times \frac{\sum_{k=1}^{N} (\ x_{k,i} - att_{mean}(i))\ ^4}{N}$$

$$Kurt = \text{mean}\ (\ \{att_{kurt}(i)\ \forall i \in NumAtts\})$$

- **Mean Attribute Entropy ($AttEnt$)** [33, 34, 35, 36, 37, 38, 39]: We can define the entropy of a nominal attribute as:

$$H(i) = -\sum_{j \in Unique(Att_i)} q_j \times log_2(q_j)$$

where $q_j = p(Att_i = j)$ denotes the probability that attribute i assumes the j-th value j, for $j \in Unique(Att_i)$. The normalized entropy can be computed using:

$$H_{norm}(i) = \frac{H(i)}{log_2(|Unique(Att_i)|)}$$

$$AttEnt = \text{mean}(\{H_{norm}(i), \forall i \in NomAtts\})$$

- **Class Entropy ($HClass_{norm}$)** [33, 34, 35, 36, 37, 38, 39]: Class entropy is defined similarly to the entropy of a nominal attribute:

$$HClass = -\sum_{j \in Unique(y)} q_j \times log_2(q_j))$$

where $q_j = p(y = j)$ denotes the frequency of occurrence of class value j, for $j \in Unique(y)$. The normalized class entropy is computed as:

$$HClass_{norm} = \frac{Hclass}{log_2(|Unique(y)|)}$$

- **Joint Entropy of Class and Attribute ($JEntCAtt$)** [33]: The joint entropy is defined as:

$$H(y,i) = -\sum_{j \in Unique(Att_i), k \in Unique(y)} p_{j,k} \times log_2(p_{j,k}))$$

where $p_{j,k}$ denotes the joint probability of observing the j-th value of attribute Att_i, and the k-th class value.

$$JEntCAtt = \text{mean}(\{H(y,i), \forall i \in NomAtts\})$$

- **Mutual Information of Class and Attribute ($MutInfoClassAtt$)** [33, 34, 35, 36, 37, 38, 39]: The mutual information of class and attribute i can be evaluated by $MI(y,i) = Hclass + H(i) - H(y,i)$.

$$MutInfoClassAtt = \text{mean}(\{MI(y,i), \forall i \in NomAtts\})$$

- **Max Mutual Information of Class and Attribute ($MaxMutInfoClassAtt$)** [33]:

$$MaxMutInfoClassAtt = \max(\{MI(y,i), \forall i \in NomAtts\})$$

- **Equivalent Number of Attributes (EN_{attr})** [33, 34, 35, 36, 37, 38, 39]:

$$EN_{attr} = \frac{Hclass}{MutInfoClassAtt}$$

- **Relative Equivalent Attributes ($RelEN_{attr}$)** [36]:

$$RelEN_{attr} = \frac{EN_{attr}}{M}$$

- **Noise to Signal Ratio ($NS.ratio$)** [33, 35, 36, 37, 38, 39]: We can consider the $MutInfoClassAtt$ as a measure of useful information about class, and $(AttEnt - MutInfoClassAtt)$ as a measure of non-useful information.

$$NS.ratio = \frac{AttEnt - MutInfoClassAtt}{MutInfoClassAtt}$$

For a more detailed explanation of the Statistical and Information-Theory based Meta-Features, you can refer to the work of Castiello et al. [33].

We developed, in Java, a module to extract automatically all the Meta-Features from a dataset (in arff or csv format).[2] This module is essential because it helps to create the Meta-Datasets of the Meta-Learning system and it also helps to describe a new dataset, to be able to predict its performance.

Each of our Meta-Datasets $T_{meta}^{j,k,l,m}$ (we have 1 Meta-Dataset for each workflow (j,k,l) and property m) is a *set of N pairs of the form ($\boldsymbol{x_l}$, y_i)*, where $\boldsymbol{x_i}$ is the input vector of 48 Meta-Features extracted from the i-th dataset (T_{base}^{i}), and y_i the performance measure of the workflow $L_{base}^{j,k,l}$ over the i-th dataset. Now we can train a Meta-Learner $L_{meta}^{j,k,l,m}$ to exploit this information and be able to predict future performances y_{new} for new datasets T_{base}^{new}.

[2] The jar file of this dataset-parser is available in: https://github.com/ROCKFlows/dataset-parser-public

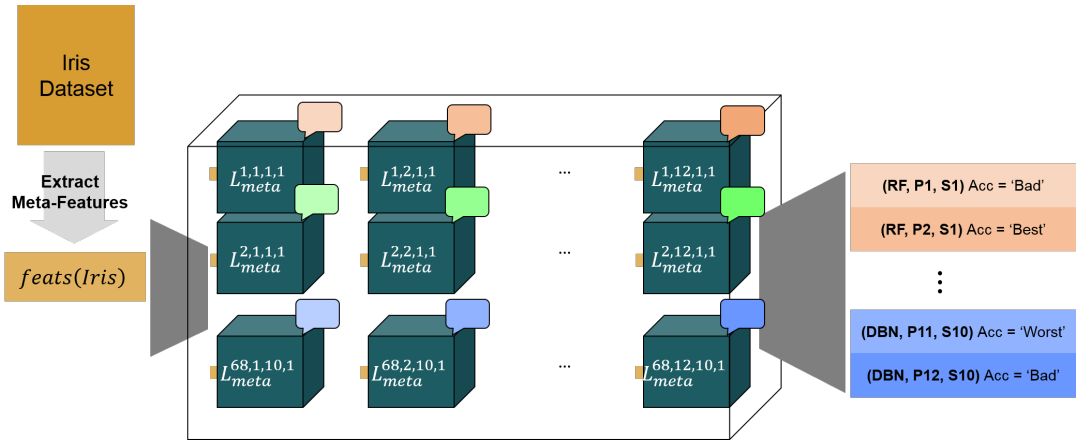


FIGURE 3.1: Prediction Process.

3.4 Operation of the Meta-Learning system

In a previous work (During the PFE of this project) we used Decisions Trees as sub-models of the Meta-Learning System, but we found that although Decision Trees are efficient to produce rule-based selection models, they overfit to the data, while Random Forest represent an efficient solution for the algorithm selection problem, because it has been proven that its rate of convergence depends only on the number of strong attributes and not on how many noisy variables there are [40, 41] then they do not overfit as more trees are added to the Forest [42].

In this subsection we will present the general operation of the Meta-Learning system and its structure. The Meta-Learning system is composed by many sub-models $L^{i,j,k,l}_{meta}$ (one sub-model for each workflow (i, j, k) and each property, l). Each of this sub-models is a Random Forest that is trained using the respective Meta-Dataset $T^{i,j,k,l}_{meta}$.

Therefore, there is, for example, a sub-model $L^{8,12,2,1}_{meta}$ to predict the accuracy ($id = 1$) of a workflow $L^{8,12,1}_{base}$ composed by algorithm SVM ($id = 8$) with parameter strategy Grid-Search ($id = 2$), and applies to the d-th dataset T^{d}_{base} the following pre-processing technique: {Nominal to Numeric, Standardize, Replace Missing Values by 0, Attribute Selection} ($id = 12$). This sub-model $L^{8,12,2,1}_{meta}$ is trained with a Meta-Dataset similar to the one shown in Table 3.8, but using the Meta-Features defined previously (see Section 3.3).

So, the Meta-Learning system is composed by sub-models identified by $L^{i,j,k,l}_{meta}$ where $i \in algorithmIds$, $j \in preprocessingIds$, $k \in paramStrategiesIds$, and $l \in propertyIds$. Each of these sub-models is trained using a Meta-Dataset $T^{i,j,k,l}_{meta}$ built with the results (for property l) of experiments (for workflow (i, j, k)) run on different base datasets T^{d}_{base} where $d \in datasetIds$, where each of these datasets is described by a vector of 48 Meta-Features.

The prediction process is illustrated on Figure 3.1, on this figure we can observe the Meta-Learning system composed by the set of sub-models $L^{i,j,k,1}_{meta}$ used to predict the metric 1: *accuracy* for the workflow (i, j, k). Once a new dataset T_{new} arrives, the first step is to extract its Meta-Features, and then use these Meta-Features as input to predict using each sub-model. Once the prediction for all the metrics and all workflows is done, the results are returned to the user.

3.4.1 Coupling the Meta-Learning system with ROCKFlows

To couple easily this model with the ROCKFlows dynamics, we decided to build a web service that takes as input the prediction request in json format, and outputs the predictions.

A request for the prediction web service has the following format:

```
{
   "dataset": {
      "META-FEATURE_1_ID" : META-FEATURE_1_VALUE,
      ...,
      "META-FEATURE_M_ID" : META-FEATURE_M_VALUE
   },
   "workflows" : {
      "WORKFLOW_1_ID" : {
         "algorithmId" : "ALGORITHM_ID_VALUE",
         "properties" : {
            "PROPERTY_1_ID" : "PROPERTY_1_VALUE",
            ...,
            "PROPERTY_N_ID" : "PROPERTY_L_VALUE",
         }
      },
      ...,
      "WORKFLOW_N_ID" : {
         "algorithmId" : "ALGORITHM_ID_VALUE",
         "properties" : {
            "PROPERTY_1_ID" : "PROPERTY_1_VALUE",
            ...,
            "PROPERTY_N_ID" : "PROPERTY_L_VALUE",
         }
      }
   }
}
```

In this format, we are requesting the prediction for a dataset described by M different Meta-Features, and N different workflows. The keyword "properties" denotes the optional properties of a workflow, for example Pre-processing or Parameter Strategy. Note that in the format there is no entry for the metric (e.g. Accuracy, or Model Size), we defined it in this way, because we always request the prediction for all the metrics we have.

An example of entry for the key-value pair of Meta-Features is:

```
"DIMENSIONALITY" : 0.1225806.
```

An example of entry for the key-value pair of "properties" of a workflow is:

```
"PRE_PROCESSOR" : "12".
```

The output of the web service has the following format:

```
{
   "WORKFLOW_1_ID" : {
      "METRIC_1_ID" : "METRIC_1_VALUE",
      ...,
      "METRIC_K_ID" : "METRIC_K_VALUE"
   },
   ...,
   "WORKFLOW_N_ID" : {
      "METRIC_1_ID" : "METRIC_1_VALUE",
      ...,
      "METRIC_K_ID" : "METRIC_K_VALUE"
   },
```

```
}
```

In this output we return the predicted values for the K different metrics of the N requested workflows.

Chapter 4

Improving for Scalability

So far we have built a working Meta-Learning system that trains using previous experiments on Machine Learning Workflows, and predicts the performance of different workflows on a new dataset using a class $\in \{Best, Great, Good, Average, Bad, Worst\}$. But now the question is, what is the procedure to follow if we want to add the result of a new experiment to our Meta-Learning system?

To illustrate this idea, we will use the example shown in Figure 4.1, where we have the dataset Balloons. This dataset was run in different workflows: (p:1, SVM:8, grid-search:1), (p:12, SVM:8, grid-search:2), (p:3, ANN:13, default-:1), and (p:12, ANN:13, default:1); and the accuracies for these workflows were $50\% \pm 25\%$, $97\% \pm 1\%$, $53\% \pm 15\%$, and $75\% \pm 10\%$, respectively. If we run our ranking algorithm, we would get the following labels: 'Worst', 'Best', 'Worst', 'Good' respectively for each workflow.

Now imagine we use the same dataset Balloons, and we run it on a new workflow (p:7, BayesNet:25, default:1), the accuracy we get is $92\% \pm 3\%$, and the label according to the ranking algorithm is 'Good'. Naturally, we would need to create or re-train the sub-model $L_{meta}^{25,7,1,1}$, to include this new instance, but note here that the ranking for the other workflows may have changed.

If we run the ranking algorithm again (including the result for the new workflow), as shown in Figure 4.1(right). The ranking class of the workflow (p:12, ANN:13, grid-search:2) have changed from 'Good' to 'Average', and then we would need to change this information by retraining the sub-model $L_{meta}^{13,12,2,1}$.

This represents a huge problem in terms of scalability, because each time we add the result of a new experiment, we would need to retrain (possibly many) other models, considering that their information becomes obsolete. This problematic situation is shown in Figure 4.2, where we have the representation of sub-models $L_{meta}^{13,12,2,1}$ and $L_{meta}^{25,7,1,1}$, and the information $T_{meta}^{13,12,2,1}$ used to train sub-model $L_{meta}^{13,12,1,0}$ (a model that has nothing to do with the new experiment) becomes obsolete and then needs to be retrained.

The solution to avoid this situation is to use a Regression Model instead of a Classification one, to predict directly the value of the metric instead of predicting the ranking of the classifiers, in the example shown in Figure 4.2, it means to predict directly the numeric value of Accuracy. Then, each time we run a new experiment and we want to add this information to our Meta-Learning system, we need to retrain only the sub-model on which we added the new instance, because the target value will not change in any other sub-model. Another advantage of this approach is that we have a more precise way to differenciate two workflows with the same ranking label.

FIGURE 4.1: (left) Example of experiments of ML Workflows on Dataset Balloons (right) Adding the experiment of ML Workflow $L_{meta}^{25,7,1,1}$.

$L_{meta}^{25,7,1,1}$ = BayesNet-P7-S1-Acc (Model with the new instance)

Dataset	Num Inst	Num Atts	Num Class	Num Miss Vals	Maj Class Size	Skewn	Cov	Class Entrop	Att Entrop	Accuracy
Balloons	16	4	2	2	2	0.5	0.8	0.87	0.79	'Good'
Car	1728	6	4	0	1210	?	?	0.60	1.00	'Bad'
...	...	...	...	...	...	...	...	...	...	...

$L_{meta}^{13,12,1,1}$ = ANN-P12-S1-Acc (Model that has nothing to do with the new instance)

Dataset	Num Inst	Num Atts	Num Class	Num Miss Vals	Maj Class Size	Skewn	Cov	Class Entrop	Att Entrop	Accuracy
Balloons	16	4	2	2	2	0.5	0.8	0.87	0.79	'Good' → 'Average'
Car	1728	6	4	0	1210	?	?	0.60	1.00	'Bad'
...	...	...	...	...	...	...	...	...	...	...

FIGURE 4.2: Scalability issues using labels: Information on other Meta-Datasets becomes obsolete.

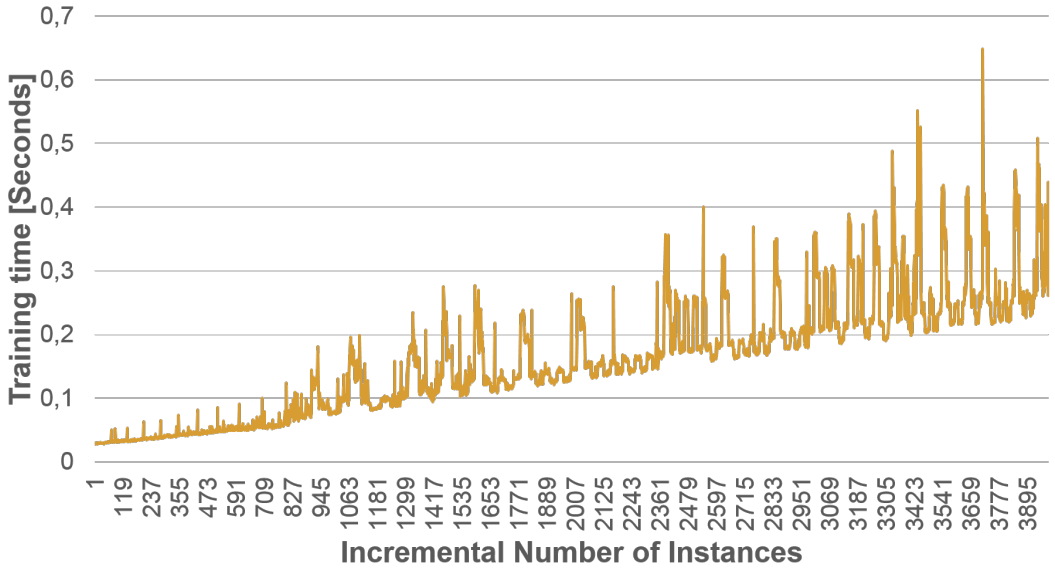


FIGURE 4.3: Random Forest Re-Training Time.

4.1 Incremental Learning

With the small adjustment of using directly the regression value in the Meta-Learning system, we reduced to only one the possible number of sub-models to train each time we add a new instance.

Another scalability problematic is the following: Assume that there are ML Experiments run constantly and therefore more training instances arrive continuously, then we would need to retrain our sub-model continuously with an increasing number of instances. In Figure 4.3, we present the re-training time of a Random Forest using a Meta-Dataset with an incremental number of instances.

Our goal now is reduce the re-training time of each model by using the incremental learning approach. For this approach we consider that we have a stream of data that arrives continuously, and we would like to use this data as soon as we have it, to train our model. We searched in the literature for different incremental learning Random Forest solutions for regression, we summarize here the studies found:

- **Mondrian Forests (MF)**: This is the work of Lakshminarayanan et al. [43], they used Mondrian Processes [44] to build ensembles of Random Decision Trees called Mondrian Forests.
- **Fast Incremental Model Trees with Drift Detection (FIMT-DD)**: This algorithm is the result of the work of Ikonomovska et al [45], it maintains a model tree whose leaves contain linear models induced online from the examples assigned to them. It also has a Drift Detection mechanism that allows to detect local drift, and updates the tree structure only locally.
- **Online Regression Trees with Options (ORTO)**: This work is the extension of FIMT-DD of Ikonomovska et al [46]. Option trees build upon regular trees by adding splitting options in the internal nodes, this way they help to improve the accuracy and stability.
- **Hoeffding Decision Trees (HDT)**: The idea of this work is to use the Hoeffding Bound [47] to define which subset of Decision Trees from the entire Forest

TABLE 4.1: Average and Median Mean Absolute Error for the solutions found for Incremental Learning for Regression

Meta-Algorithm	**Averaged Mean Absolute Error**	**Median Mean Absolute Error**
Random Forest ($\#trees = 10$)	0.1249	0.1245
Mondrian Forest ($\#trees = 10$)	0.1497	0.1462
Decision Tree	0.1462	0.1468
Mondrian Tree	0.1727	0.1686
FIMT-DD	5441878437.0000	1.0263
Bagging FIMT-DD ($\#trees = 10$)	7720287500.0000	1.3965
ORTO	5441878437.0000	1.0241

should be retrained, we based our research of Hoeffding Decision Trees for Regression on the work of Bouaziz et al [48].

Among the solutions found, we decided to exclude from our experiments the approach of *Hoeffding Decision Trees*, mainly due to the following two conditions:

1. It was not possible to find any work with an implementation for Regression, all the studies dealing with Hoeffding Decision Trees are used to solve Classification problems, and if we are willing to compare the different approaches, it would not be fair to use an application that is not adapted for regression.

2. The nature of the approach is not completely scalable, in the sense that when a percentage of Decision Trees (from the entire Forest) are selected to be retrained, we are still re-training in batch (using all the instances) the subset of trees; then the training time would be proportional to the number of instances, and this is precisely what we want to avoid with online learning.

We used 10-fold cross validation to compare the different approaches (MF, FIMT-DD, ORTO). The results for Random Forest are included to have a point of comparison. In Table 4.1 we present the mean absolute error (for the 10 folds) averaged for all the sub-models that predict accuracy (i.e. $L_{meta}^{i,j,k,1}$, $i \in \forall$ Algorithms, $j \in \forall$ Pre-processings, $k \in \forall$ Parameter strategies). In this table we also present the 50th percentile, because an eccentric error may deviate the average value.

The results shown in Table 4.1 tell us what has already been expressed in [49], the existing online RF variants are data inefficient, they require lots of training data before they can deliver good test accuracy. According to these results Mondrian Forest seems to be a relatively good approach.

In Figure 4.4 we compare the training time of Mondrian Forest against the training time of Random Forest using a Meta-Dataset with an incremental number of instances, and we can observe that the training time is reduced by one order of magnitude by using Mondrian Forests.

With Mondrian Forest we have a reduction in training time, but the error is higher than with Random Forest, and vice versa. To be able to decide which option is more valuable, we performed a more comprehensive comparison between Random Forest and Mondrian Forest, for this, we used the Scaled Error, a metric that takes into account the Mean Absolute Error of the learner (RF or MF) contrasted against the Mean Absolute Error that would be obtained by a naive-learner, that is a learner that always predicts the value 0.684 (the average output value of all the Meta-Datasets).

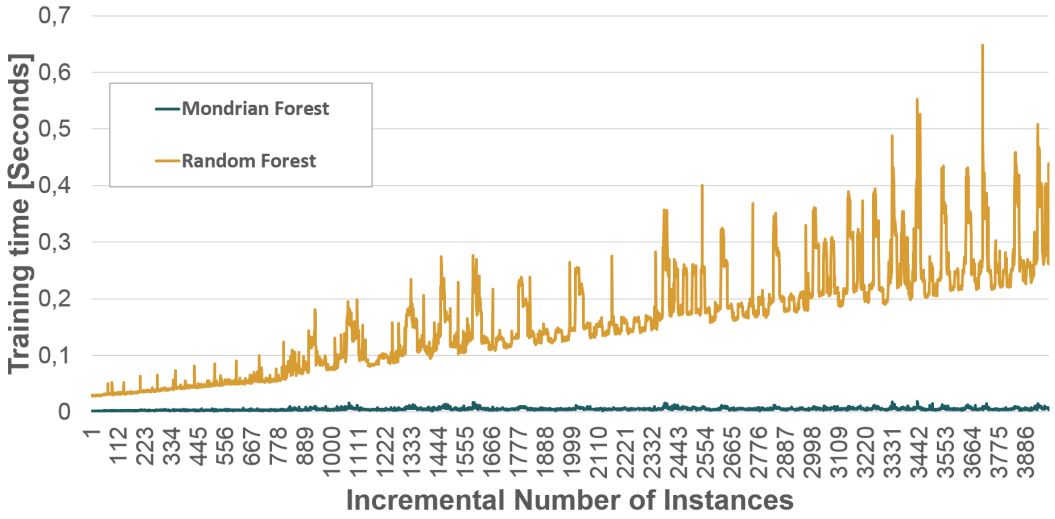


FIGURE 4.4: Re-Training Time Random Forest vs Mondrian Forest.

Define $N_{j,k,l}$ as the number of instances of the Meta-Dataset of workflow with algorithm id j, pre-processing id k, and parameter strategy l; $pred(i,j,k,l,m)$ is the prediction result of $L_{meta}^{j,k,l,m}$ for instance i; and $y_{i,j,k,l,m}$ is the groundtruth value of metric m in workflow (j:algorithm, k:pre-processing, l:parameter strategy), for instance i.

The total number of predictions is:

$$TotalNumOfPreds = \sum_{k \in Preps} \sum_{j \in Algs} \sum_{l \in ParSt(j,k)} N_{j,k,l} \tag{4.1}$$

The prediction of the Naive Learner is given by the equation:

$$Naive_{pred} = \frac{\sum_{k \in Preps} \sum_{j \in Algs} \sum_{l \in ParSt(j,k)} \sum_{i=1}^{N_{j,k,l}} y_i}{TotalNumOfPreds} \tag{4.2}$$

The formula for the mean Scaled Error is given by the following equation:

$$MeanScaledErr = \frac{\sum_{k \in Preps} \sum_{j \in Algs} \sum_{l \in ParSt(j,k)} \sum_{i=1}^{N_{j,k,l}} \frac{|pred(i,j,k,l,1) - y_{i,j,k,l,1}|}{|Naive_{pred} - y_{i,j,k,l,1}|}}{TotalNumOfPreds} \tag{4.3}$$

Where $Algs$ is the set of ids of Machine Learning Algorithms, $Preps$ is the set of ids of Pre-processing techniques, and $ParSt(j,k)$ is the set of appropriate Parameter strategies for algorithm j and pre-processing k. If $MeanScaledErr > 1$ means that the Naive learner is (on average) better than the Meta-Learner used. In Table 4.2 we summarize these results, we also present the Median and the smaller percentile for which $MeanScaledErr > 1$.

Analyzing the results presented on Table 4.2, in the case of the average, it is not possible to conclude anything, as there may be an out-of-range value that affects the value of the mean, but then for the median we observe that the Random Forest learner is better than the Mondrian Forest learner, we can also observe that for $\approx 63\%$ of the predictions the Random Forest model is better than the Naive Learner, and for $\approx 56\%$ of the predictions the Mondrian Forest model is better than the Naive

TABLE 4.2: Error comparison: Mondrian Forest vs Random Forest

Algorithm	**Average Absolute Error**	**Median Absolute Error**	**Smaller percentile for which Mean-ScaledErr > 1**
Random Forest ($\#trees = 10$)	2.9138	0.7724	Percentile: 0.627
Mondrian Forest ($\#trees = 10$)	2.3988	0.9070	Percentile: 0.558

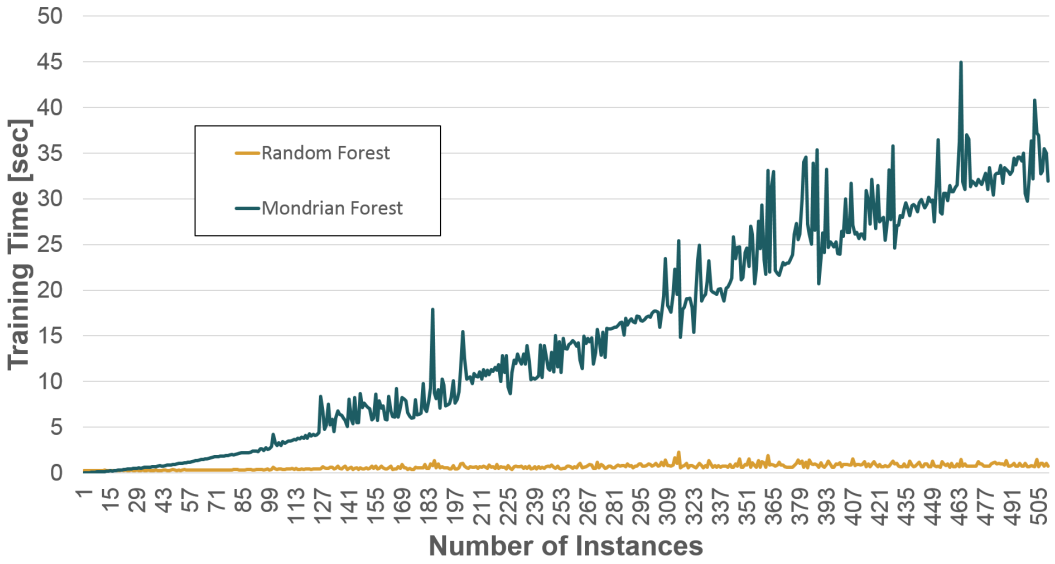


FIGURE 4.5: Batch Training Time using an incremental number of instances for MF and RF (The instances are from $T^{1,1}_{meta}$).

Learner. We have decided to use a batch Random Forest learner and not an online learning model, because the accuracy of the sub-models is greatly reduced when using Mondrian Forest.

Another disadvantage of Mondrian Forest is its pure-online nature. Mondrian trees can be updated online in an efficient manner, and the distribution of trees sampled from the online algorithm is identical to the corresponding batch counterpart [43], but when it comes to the time of a batch update or a total batch training of the Mondrian Forest, this time is much longer than the total batch training time of a Random Forest. In other words, compared to Random Forests, in terms of training time, Mondrian Forests are efficient for online (incremental) learning but inefficient for batch learning. To depict this deduction, in Figure 4.5 we present the training time in batch of an incremental number of instances for Mondrian Forest (MF) and Random Forest (RF).

Chapter 5

Evaluation and Results

5.1 Evaluation Methodology

Recall that the Meta-Learning system is nothing more than the collection of different sub-models $L_{meta}^{i,j,k,l}$, where i represents the algorithm id, j the pre-processing id, k the parameter strategy id, and l the property or metric id. Each of the sub-models is trained using a Meta-Dataset $T_{meta}^{i,j,k,l}$, where each instance of the Meta-Dataset represents an experiment consisting in running the m-th dataset T_{base}^{m} on the ML Workflow (i, j, k) (pre-processing j, algorithm i and parameter strategy k) and measuring the metric l.

It is also possible that a dataset T_{base}^{d} can be used in workflow $T_{meta}^{p,q,r,s}$ while it cannot be used in workflow $T_{meta}^{l,m,n,o}$ (for $p \neq l$, $q \neq m$ or $r \neq n$). According to this, each Meta-Dataset $T_{meta}^{i,j,k,l}$ may have different number of instances.

In order to test the Meta-Learning system, and observe the impact of the number of instances used in each sub-model, we set the following experiment:

1. Arrange the sub-models $L_{meta}^{i,j,k,l}$ according to the number of instances in its Meta-Dataset $T_{meta}^{i,j,k,l}$.

2. Do cross validation. For this experiment we used 10-folds, it means that we divided the Meta-Dataset in 10 subsets, then 9 subsets were used to train and the remaining subset was used to test, this procedure was done 10 times (with each of the subsets being the testing set once) as shown in Figure 5.1.

3. The predictions of each fold of the testing set are compared with the groundtruth values. For this comparison between predicted value and groundtruth value, we computed the coefficient of determination (or R^2 score). The R^2 score is a regression score function, and it can be computed using the following equation:

$$R^2 score = 1 - \frac{\sum_{i=1}^{\#instances}(y_i - f_i)^2}{\sum_{i=1}^{\#instances}(y_i - \bar{y})^2} \tag{5.1}$$

   Where y_i is the groundtruth value of the i-th instance, f_i is the predicted value of the i-th instance and $\bar{y} = \frac{\sum_{i=1}^{\#instances} y_i}{\#instances}$. The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

4. Average the R^2 score for the 10 folds for each model.

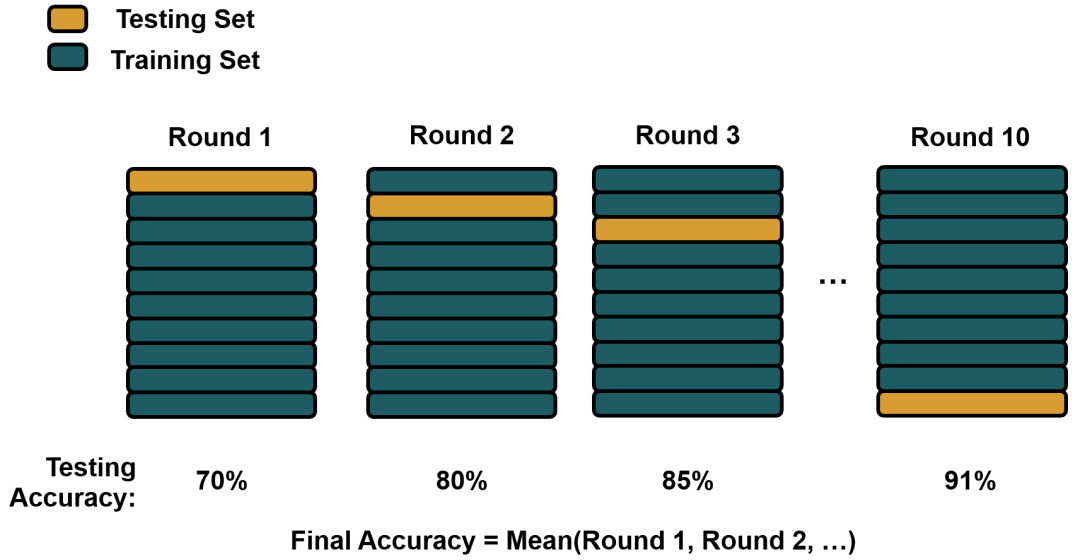


FIGURE 5.1: Example of Meta-Dataset splitting in training and test set by cross validation.

The Figure 5.2 shows us the computation of the average R^2 score for each model, around 75% of the models have a negative R^2 score, we interpret this as the models being obsolete. Then, we can conclude that around 75% of the models are obsolete, and it is mainly due to the reduced number of instances used to train them.

5.2 New Meta-Learning system Proposed

From the previous section we have concluded that the number of instances affects drastically the performance of the Machine Learning models, considering it from a statistical point of view, this is not an unexpected result, and this problem is widely known in Machine Learning [50]. From the work of Parisi et al. [28] we have the results of the experiments of 101 datasets, and the task of finding more training samples (datasets) is complex, from the UCI repository[1], for example, only 108 datasets are suitable to be included in our experiments (small-scale datasets in the common UCI format, for classification tasks) [24].

In our Meta-Learning system, we can find some sub-models that contain less than 10 instances (datasets) in their Meta-Dataset, then our goal was to cut down the number of models with a reduced number of instances. One solution is to avoid using the sub-models whose Meta-Dataset contains less instances than a defined threshold, this way we bypass the problem of giving wrong predictions by simply not giving any prediction. Another solution is to find a way to increase the number of instances on each model, there are many options here, either we create artificial datasets (see Section 6.1.1), we extend the data from the information we already have (see Section 6.1.2), or we can join the Meta-Datasets from different sub-models that share similar characteristics. The latter is the approach we take in this section.

The new Meta-Learning system proposed has the following structure. Instead of having one sub-model $L_{meta}^{i,j,k,l}$ for each algorithm i, pre-processing technique j,

[1] see https://archive.ics.uci.edu/ml/datasets.html

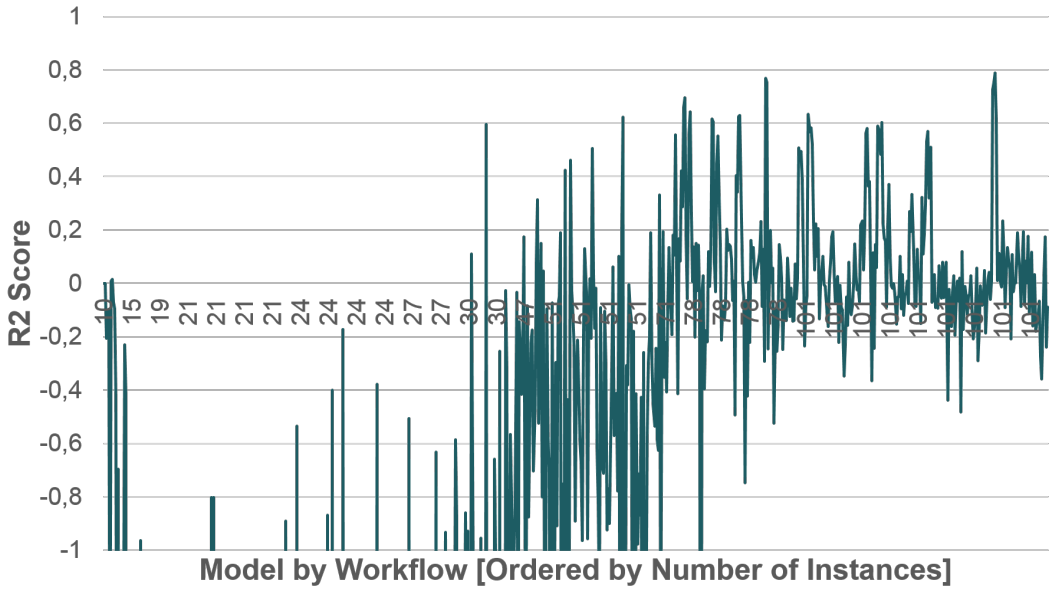


FIGURE 5.2: R^2 score of each sub-model ($L^{i,j,k,1}_{meta}$, $i \in$ {all algorithms}, $j \in$ {all pre-processings}), $k \in$ {applicable parameter strategy for algorithm i and pre-processing j}) ordered by number of instances in their Meta-Datasets.

parameter strategy k, and metric l, we will have one sub-model $L^{i,l}_{meta}$ for each algorithm i and metric l; then, instead of using the pre-processing technique j and parameter strategy k as subdivision for the sub-models, we will include them as new Meta-Features in the Meta-Datasets. Therefore, the sub-model $L^{i,l}_{meta}$ of the new Meta-Learning system will contain all the information of the Meta-Datasets of the sub-models $L^{i,j,k,l}_{meta}$ for $j \in \{\forall$ pre-processing ids$\}, k \in \{\forall$ parameter strategy ids$\}$, of the previous Meta-Learning system, as represented in the illustration of Figure 5.3.

The benefits of this new Meta-Learning system structure are listed below:

1. We have less sub-models to manage: Before we had *# of algorithms* $\times$ *# of pre-processing techniques* $\times$ *# of properties* sub-models, and now this quantity is reduced to *# of algorithms* $\times$ *# of properties* sub-models, in our case this changed from having $(68 \times 12 \times 3 =)$ 2448 sub-models to have $(68 \times 3 =)$ 204 sub-models. Note that we did not take into account the number of parameter strategies, because in our experiments (except for the algorithms SVM, SdA, and DBN) we used only 1 (default) parameter strategy.

2. Our hypothesis is that this new structure benefits the prediction performance of each sub-model, because in the previous Meta-Learning system, the sub-model $L^{i,j,k,l}_{meta}$ had no information about sub-model $L^{i,m,n,l}_{meta}$ ($j \neq m \vee k \neq n$ two sub-models with same algorithm that predict the same metric) it is possible that the new sub-models can benefit from the information about different experiments using the same algorithm but different pre-processing techniques (or parameter strategies).

3. If more pre-processing techniques or parameter strategies are used, the number of sub-models is not increased

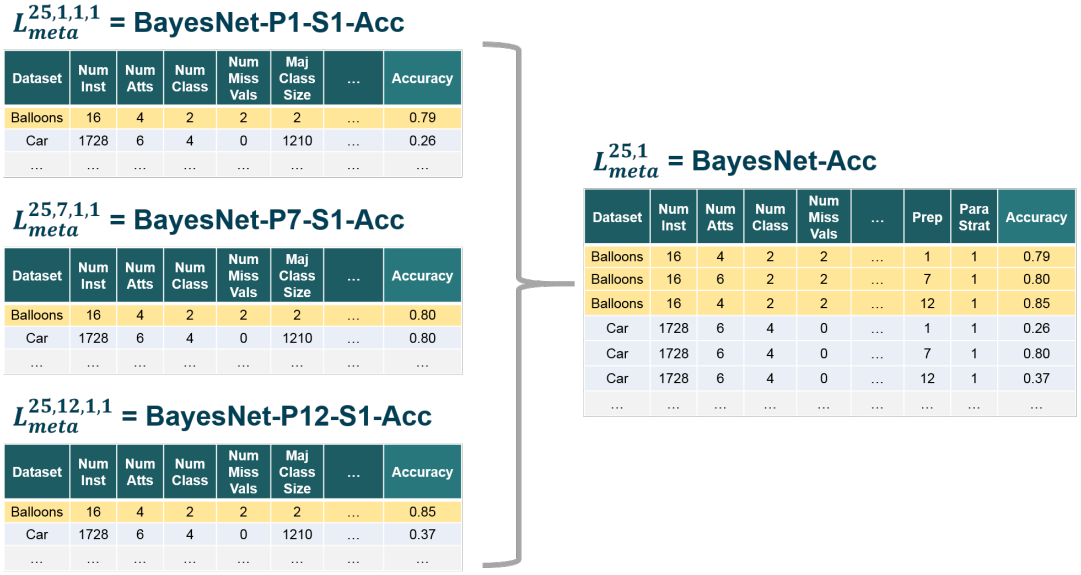


FIGURE 5.3: Second Meta-Learning system Structure.

4. This structure makes also possible to include numeric features about the workflow, for example the number of trees used in a Random Forest, or the exact values of the hyperparameters of a SVM (e.g. *gamma* and *C* of RBF). This was impossible to achieve with the previous structure because we would need to discretize the range of values to avoid having an infinite amount models (as the index of the model would be a numeric value).

5. The size of each sub-model is in the order of Megabytes, the requirements, of primary memory (RAM) for the computing unit, to be able to load and manage each of the sub-models independently is low.

Note here that we could have reduced the number of models even more, for example using one sub-model L_{meta}^{k} for each property; combining all the Meta-Datasets of the sub-models $L_{meta}^{i,k}$ for $i \in \{\text{all algorithm ids}\}$. We did not do it, because the size of the sub-models L_{meta}^{k} is in the order of Gigabytes (L_{meta}^{1}[acc]: 3GB, L_{meta}^{10}[time]: 0.5GB, and L_{meta}^{12}[size]: 0.7GB), and then, the primary memory (RAM) requirements for the computing unit, to load each of the sub-models is much higher, also taking into account that the size of the model is supposed to grow each time we add more instances to it.

5.3 Evaluation Methodology - New Meta-Learning System

The evaluation method to test the performance of the new Meta-Learning system, is similar to the one used for the previous model (see Section 5.1), although there is one subtle difference that changes the way of spliting each Meta-Dataset into folds.

In this new Meta-Learning system is posible to find two or more datasets with different pre-processing techniques or parameter strategy in the same Meta-Dataset. It means that there may be two instances I_a and I_b with the same Meta-Features except the Meta-Feature of the pre-processing technique (or parameter strategy). When we perform 10-fold cross validation, the Meta-Dataset is divided in 10 subsets, then 9 subsets are used to train and the remaining subset is used to test. If one of the

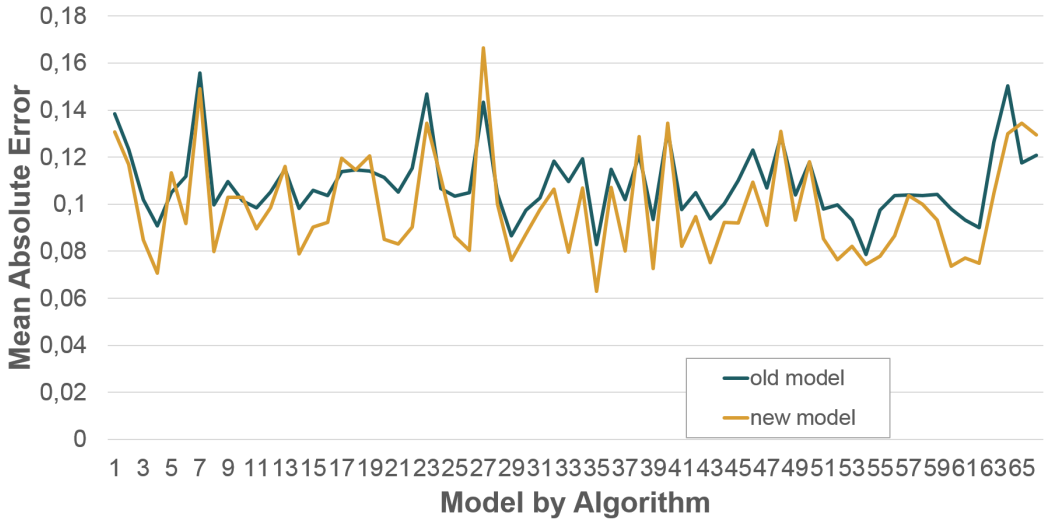


FIGURE 5.4: Mean absolute error of new Meta-Learning System: sub-models $L^{i,1}_{meta}$ (for $i \in$ {all algorithms}) vs Mean absolute error of previous Meta-Learning System: sub-models $L^{i,j,1}_{meta}$ (for $j \in$ {all pre-processings}).

instances (I_a or I_b) is in the training set and the other is in the testing set, the learner may have some advantage by knowing this information.

The solution to avoid this situation is to do the folding of the Meta-Dataset by spliting by datasets, basically all the instances that contain the same dataset (with different pre-processing techniques) must be in the same fold.

Then, we compute the average absolute error of all the predictions of all folds of the testing set against the groundtruth values. In Figure 5.4 we can find these results for each sub-model compared with the results of the sub-models in the previous Meta-Learning System.

The average R^2 score of the previous structure was 0.34, while the average R^2 score of the new Meta-Learning system is 0.40, it means that the new model structure is better, so our hypothesis, that this new structure benefits from the information about different experiments using the same algorithm but different pre-processing techniques, is true.

5.4 Hyperparameter optimization

In Machine Learning, the Hyperparameters are parameters of the Learning Model whose values are set prior to the training process. In the case of Random Forest, that is the Learner used in each sub-model of the Meta-Learning system, the main hyperparameter is the number of decision trees in the forest.

We present in Figure 5.5 the evolution of the Mean Absolute Error of the Meta-Learning system (to predict accuracy) when we increase the number of trees. In Figure 5.6a we present the evolution of Average Model Size in Memory when we increase the number of trees, we also analyze the average Training Time, the average Time it takes to load the Model and the average Prediction Time against the number of trees in Figure 5.6b, Figure 5.6c and Figure 5.6d respectively.

In this Figures we can observe that after using 250 trees, if we increase the number of trees, the error does not decrease much more, while the memory size, training

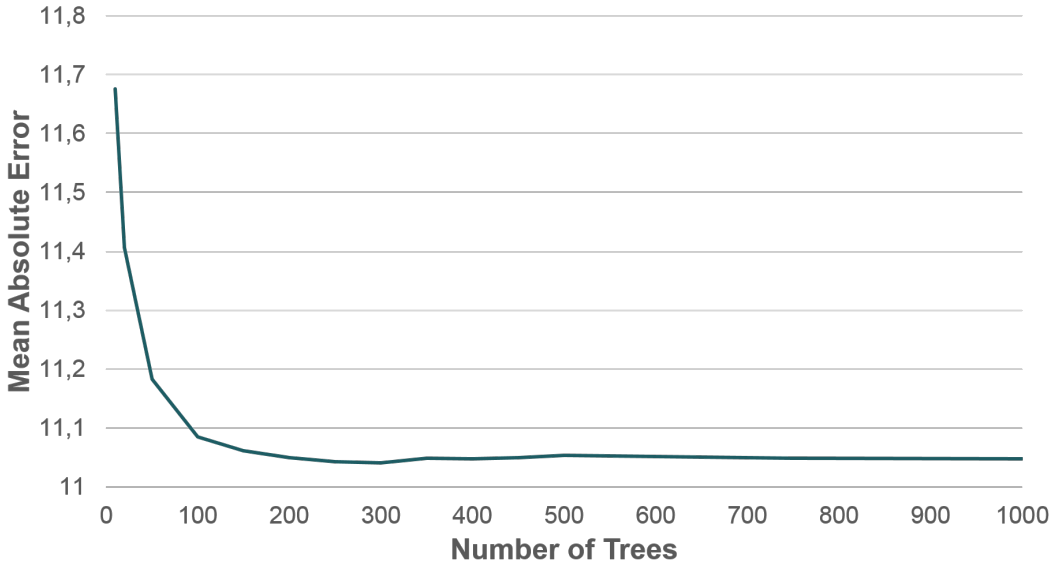
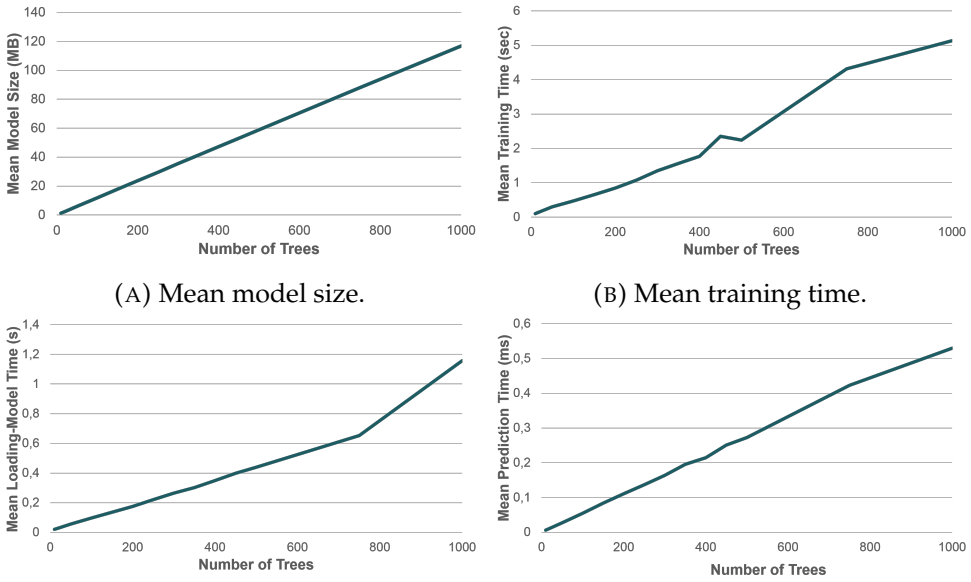


FIGURE 5.5: Mean absolute error of sub-models $L^{i,1}_{meta}$ (for $i \in$ {all algorithms}) increasing the number of trees in the Random Forests.



(A) Mean model size.

(B) Mean training time.

(C) Mean time to load.

(D) Mean prediction time.

FIGURE 5.6: Model Size, Training Time, Loading Time and Prediction Time of sub-models $L^{i,1}_{meta}$ (for $i \in$ {all algorithms}) increasing the number of trees in the Random Forests.

time, loading model time and prediction time are directly proportionally to the number of trees used in the Random Forest, for this reason we decided to use 250 trees in all our sub-models.

Chapter 6

Future Work

In this section we present the possible future work for this project, and some advances already done.

6.1 Extending the data in our Meta-Datasets

6.1.1 Dataset Generation from Meta-Features

One possible direction in the continuation of this project is to extend the data in each Meta-Dataset. As real-world datasets are hard to obtain, one possible way to extend the number of datasets in our experiments is to use artifially created datasets, often called *datasetoids*.

There have been some approaches in this regard [51, 52], in [53], Reif et al. propose a data-generator able to create datasets with specific Meta-Features, this data-generator uses a genetic approach and allows the user to incorporate arbitrary Meta-Features. We took this work as reference to devise the following algorithm:

Data: *tol*: Error tolerance, *population*: Initial random population
Result: *population*: a population of datasets that minimize the objective function
while *meanerr* $>$ *tol* **do**
  offspring $\leftarrow$ *varAnd*(*population*, cxpb=0.6, mutpb=0.1);
  fitVals $\leftarrow$ *evaluate*(*offspring*);
  err $\leftarrow$ 0 ;
  for *fit* $\in$ *fitVals*, *individual* $\in$ *offspring* **do**
    individual.fitness.values $\leftarrow$ *fit*;
  end
  population = *select*(*offspring*, k=len(*population*));
  err $\leftarrow$ 0 ;
  for *individual* $\in$ *population* **do**
    err $\leftarrow$ err + *individual.fitness.values* ;
  end
  meanerr $\leftarrow$ err / len(*population*);
end

Algorithm 1: Genetic Algorithm for Dataset-Generation

In this genetic algorithm each *individual* is a dataset, the *population* is a set of datasets (*individuals*), and the *offspring* is the result of the mutation and combination of the datasets in the *population*.

The function *evaluate* computes the objective function f for each individual i in the population. The objective function is built using the weighted sum over the

Meta-Feature vector $\boldsymbol{x}_i$ of size n, the vector of desired values y, and the vector of corresponding weights w:

$$f(i) = \sum_{j=1}^{n} w_j \times \left| \boldsymbol{x}_{i_j} - y_j \right| \tag{6.1}$$

The function *select* picks the k best individuals from the *offspring* according to their fitness values *individual.fitness.values*.

The function *varAnd* mutates the datasets by shifting data points and recombines two datasets by swapping fractions of them, it returns a collection of individuals, following this algorithm:

Data: *population*: Set of individuals (datasets), *cxpb*: The probability of mating two individuals, *mutpb*: The probability of mutating an individual
Result: *offspring*: The new generation of datasets
Crossover: Here we perform a two copy point among each pair of individuals in the population ;
for $i \in \{1, ...len(population)\}$ **do**
  for $j \in \{i+1, ...len(population)\}$ **do**
    if *True with probability cxpb* **then**
      $col_{init} \leftarrow$ random value $\in \{0, \ldots,$ population[i]$_{cols}\}$;
      $row_{init} \leftarrow$ random value $\in \{0, \ldots,$ population[i]$_{rows}\}$;
      $width \leftarrow$ random value $\in \{0, \ldots,$ population[i]$_{cols} \mid col_{init} + width <$population[i]$_{cols}\}$;
      $height \leftarrow$ random value $\in \{0, \ldots,$ population[i]$_{rows} \mid row_{init} + height <$population[i]$_{rows}\}$;
      $rect \leftarrow$ rectangle$(col_{init}, row_{init}, width, height)$;
      offspring[i] = *population*[i] ;
      offspring[j] = *population*[j] ;
      offspring[i][$rect$] = *population*[j][$rect$] ;
      offspring[j][$rect$] = *population*[i][$rect$] ;
    end
  end
end
Mutation: Here we mutate the offspring *indpb* $\leftarrow$ Probability of mutating a value in the individual ;
for *individual* $\in$ *offspring* **do**
  if *True with probability mutpb* **then**
    for *datapoint* $\in$ *individual* **do**
      Change *datapoint* on dataset *individual* with probability *indpb*;
    end
  end
end

Algorithm 2: Crossover and Mutation of the offspring

We implemented this algorithm in Python using the framework for genetic algorithms DEAP [54]. To initialize the population, we create different datasets keeping fixed the number of columns (features) and number of rows (instances), and assigning random values from a uniform distribution between -100 and 100. Since the best possible value of the objective function is zero, we can iterate until the genetic

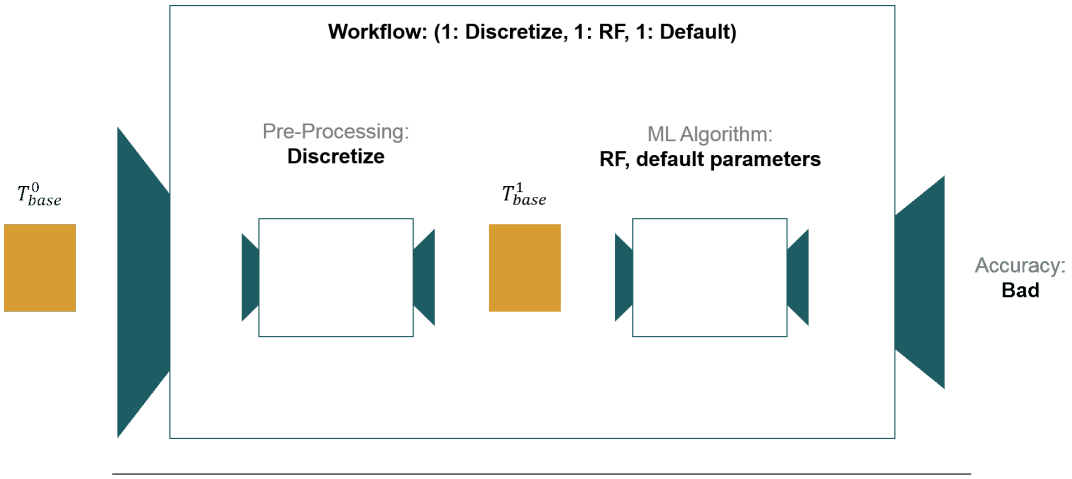


FIGURE 6.1: Experiment of dataset T^0_{base} for workflow $(1, 1, 1)$.

algorithm achieves a certain error.

This approach seems promising, but we still need to add the computation of all the Meta-Features (for the moment we only have the Meta-Features: Mean of Means of Numeric Attributes, Standard Deviation Ratio, Skewness, Kurtosis and Covariance), and also study the conflicts between Meta-Features, because it is possible that when the error of the objective function for a Meta-Feature is reduced, it may increase the error of another Meta-Feature. A simple example of this is the case of requesting to get a dataset with kurtosis $kurt$ and skewness $skew$, where $kurt < skew^2 + 1$ [55].

6.1.2 Exploiting the already existing information

Another way to increase the number of instances in the Meta-Datasets, is to exploit the information we already have from the experiments on ML Workflows, for a better illustration of this idea, we will use the Figure 6.1, where we have an example of using the dataset T^0_{base} to train the workflow (1:(pre-processing=Discretize), 1:(algorithm=RF), 2:(parameter-strategy=default)), the performance evaluation is $Accuracy = 0.83$. Then, the sub-model $L^{RF:1,Acc:1,Pst:1}_{meta}$ will contain an entry with the Meta-Features of T^0_{base}, the pre-processing 1, the parameter strategy 1 and the target value (class) 0.83.

Observe in Figure 6.1 that inside the workflow we obtain the pre-processed dataset T^1_{base}, the result of Discretizing the numeric attributes of dataset T^0_{base}. In Figure 6.2, we present what we consider can be seen as a different experiment, we have the dataset T^1_{base} as input of the workflow (0:(pre-processing=$\emptyset$), 1:(algorithm=RF), 1:(parameter-strategy=default)), the performance evaluation is $Accuracy = 0.83$ (the same of the experiment in Figure 6.1). Then we can add the entry with the Meta-Features of dataset T^1_{base}, the pre-processing 0, the parameter strategy 1, and the target value (class) 0.83.

Our idea is to use the Meta-Features of the pre-processed datasets to add more information to the Meta-Learning System. We propose the following algorithm to do this process:

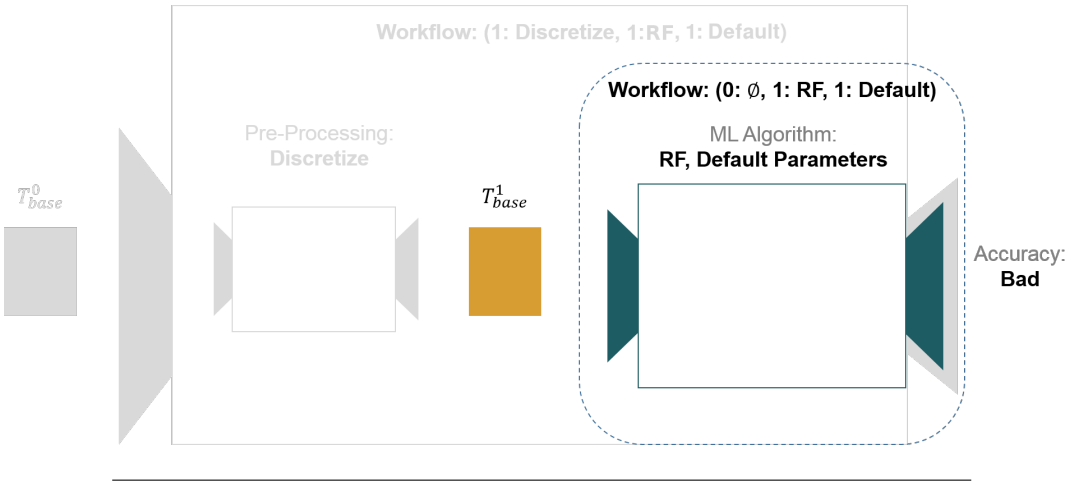


FIGURE 6.2: Experiment of dataset T^1_{base} for workflow $(0, 1, 1)$.

Data: *Datasets*: list of datasets, *prepId*: Pre-processing Id, *algId*: Algorithm Id, *pstId*: Parameter Strategy, *propId*: Property Id.
Result: *MLSystem*: a Meta-Learning System with Meta-Datasets including additional information
for $T_{base} \in Datasets$ **do**
  result = *getExpeResult*(T_{base}, prepId, algId, propId, pstId);
  for $p \in$ *getRelPreps*$(prepId)$ **do**
    features = FeatureExtraction(T^p_{base});
    addEntryToMLSystem($L^{algId,propId}_{meta}$, features, excludeFromPrep(p, prepId), pstId, result);
  end
end

Algorithm 3: Genetic Algorithm for Dataset-Generation

The procedure *getExpeResult*(*dset*, *prepId*, *algId*, *propId*, *pstId*) returns the result of the metric performance *propId* of workflow $(prepId, algId, pstId)$ for Dataset *dset*.

The procedure *FeatureExtraction*(*dset*) returns the list features of dataset *dset*.

The procedure *addEntryToMLSystem*(*mod*, *feats*, *prepId*, *pstId*, *res*) adds to Meta-Dataset of sub-model *mod* the entry where y_i is the value *res*, and $\boldsymbol{x_i}$ is the concatenation of:

- The list of Meta-Features *feats*,
- the pre-processing id *prepId*,
- and the parameter strategy *pstId*.

The procedure *getRelPreps*(*prepId*) returns the list of pre-processings that are contained in the given pre-processing *predId*. In the example, getRelPreps(1) would return the list $\{0\}$.

Finally the procedure *excludeFromPrep*(*prepIdSub*, *prepId*) returns the pre-processing Id of the subtraction of pre-processing *prepIdSub* from pre-processing *prepId*. In the example, excludeFromPrep(0, 1) would return 0.

Please refer to Table 3.1 to clarify the composition of pre-processings, in Table 6.1 we present the results of the last two methods for the pre-processings used in this work.

TABLE 6.1: Composition of pre-processings: Values returned by functions *getRelPreps* and *excludeFromPrep*

Pre-processing	**getRelPreps**	**excludeFromPrep**
0	$\{\emptyset\}$	$\emptyset$
1	$\{0\}$	$(0,1) = 0$
2	$\{0\}$	$(0,2) = 0$
3	$\{0,2\}$	$(0,3) = 0$
		$(2,3) = 1$
4	$\{0\}$	$(0,4) = 0$
5	$\{0,2\}$	$(0,5) = 0$
		$(2,5) = 4$
6	$\{0\}$	$(0,6) = 0$
7	$\{0,1\}$	$(0,7) = 0$
		$(1,7) = 6$
8	$\{0,2,3\}$	$(0,8) = 0$
		$(2,8) = 7$
		$(3,8) = 6$
9	$\{0,4\}$	$(0,9) = 0$
		$(4,9) = 6$
10	$\{0,2,5\}$	$(0,10) = 0$
		$(2,10) = 9$
		$(5,10) = 6$
11	$\{0\}$	$(0,11) = 0$
12	$\{0,11\}$	$(0,12) = 0$
		$(11,12) = 6$
13	$\{0\}$	$(0,13) = 0$
14	$\{0\}$	$(0,14) = 0$
15	$\{0\}$	$(0,15) = 0$
16	$\{0\}$	$(0,16) = 0$

6.2 Feature Selection on the Meta-Learning System

The Meta-Features build the feature space of the Meta-Learning System, if the space is high-dimensional it represents a problem for the Meta-Learning System, and in general, for any pattern recognition method, this problem is known as the *curse of dimensionality*.

One approach to reduce the dimensionality of the Meta-Datasets is to use feature selection, we can select the most relevant Meta-Features and build the models (Random Forests) using a reduced subset of Meta-Features.

Feature Selection is useful for the following reasons:

- The models are easier to interpret as they are simplified by having less variables.
- The training times are shorter: The algorithm learns faster because it has less data to train.
- It avoids the *curse of dimensionality* [56].
- The generalization is enhanced by reducing the overfitting [57], because redundant data (noise) is reduced.

Our first idea is to select the most relevant features given by the evaluation of Feature Importance of Random Forest. The challenge here is that the implementation of Random Forest in Weka (the library used to build the Meta-Learning System) does not have this functionality for regression. Then, another option is to evaluate the worth of a Meta-Feature by measuring it's correlation with the class value, we can use the Class CorrelationAttributeEval of the Weka Library[1] as reference to build a similar class for our Attribute Selection method, taking into account that the cross-validation evaluation must avoid to have two instances with the same dataset in different folds (See Section 5.3).

[1] See http://weka.sourceforge.net/doc.dev/weka/attributeSelection/CorrelationAttributeEval.html

Chapter 7

Conclusions

In this project we worked on the construction of a Meta-Learning system, the core module of the platform ROCKFlows, a system that aims to help non-expert users on the selection of the Machine Learning Workflows to solve their Machine Learning problems. The theoretic model we have designed is focused on supervised classification problems, although it can be extended to unsupervised, or supervised regression problems.

We managed to use the results of different Machine Learning experiments, specifically the results of 1086 workflows (composed by 16 pre-processing techniques, and 68 classifiers, and 10 parameter strategies) tested over 101 datasets from the UCI repository. This input data was processed to produce the Meta-Datasets with which we trained different sub-models to predict three properties defined: Accuracy, Total Time and Model Size.

We developed a module to extract the 48 most representative Meta-Features of a given dataset, this module is an important part of the prediction flow. When a user comes with her dataset, the Meta-Features of the dataset are extracted, and then, thanks to this description, the prediction is done by exploiting the similarities between the vector of Meta-Features of the given dataset and the Meta-Features of the datasets already in the Meta-Datasets of the Meta-Learning system.

We designed and implemented a Meta-Learning Model that is scalable in the sense that it does not need to make other computations (e.g. ranking the results) before adding a new instance to the knowledge base. We enhanced the scalability of the model by using Regression Algorithms instead of Classification ones.

We studied the online learning approach, to reduce the incremental training time of the model, but among the different incremental learning solutions for regression in the literature, we did not find an algorithm comparably good to its batch learning counterpart, for this reason each sub-model of the Meta-Learning system uses a Random Forest regressor.

The hyperparameter optimization results show us that after using 250 trees in the Random Forests, a law of diminishing return is observed, in other words, the accuracy is not greatly reduced by using more than 250 trees in the sub-models of the Meta-Learning system.

The structure of the Meta-Learning system proposed was largely studied throughout the project, searching for a solution that provides *good* accuracy results, manageable model size, and training, loading and prediction times, that satisfy the requirements of the web service implemented for the ROCKFlows platform. The final structure for this system consists in using one sub-model $L^{i,j}_{meta}$ for each classifier i and property j, and including in the Meta-Features the pre-processing technique and the other workflow steps (i.e. Parameter Strategy).

The research of this work is not exhaustive, and is limited by the number of datasets used and the number of workflows defined. We propose to extend the

research in this direction, looking for ways to increase the number of datasets in the knowledge base of the Meta-Learning system, to understand better the effect of the nature of the dataset (given by the Meta-Features) on the performance of the workflows. Finally, a similar work should be conducted to extend this work in the prediction of other Machine Learning problems such as clustering, regression and reinforcement learning tasks.

Appendix A

Datasets Tested

In Table A.1, we present the list of 101 tested datasets coming from the UCI repository, the most widely used database in the classification literature. These 101 datasets represent the whole UCI database for classification tasks, excluding the large-scale problems (with very high # attributes and/or # instances).

TABLE A.1: List of 101 tested datasets coming from the UCI repository

Dataset Name	**# Atts.**	**# Inst.**
ac-inflam	6	120
ac-nephritis	6	120
adult	14	32561
annealing	38	798
arrhythmia	279	472
audiology-std	69	171
balance-scale	4	625
balloons	4	16
bank	16	4521
blood	4	748
breast-cancer	9	286
bc-wisc	8	699
bc-wisc-diag	29	569
bc-wisc-prog	32	198
breast-tissue	9	106
car	6	1728
ctg-3classes	45	2126
chess-krvkp	36	3196
congress-voting	16	435
conn-bench-sonar	60	208
conn-bench-vowel	12	528
contrac	9	1473
credit-approval	15	690
cylinder-bands	37	512
dermatology	34	366
echocardiogram	12	131
ecoli	7	336
fertility	9	100
flags	28	194
glass	9	214

Continued on next page

Table A.1 – *Continued from previous page*

Dataset Name	**# Atts.**	**# Inst.**
haberman-survival	3	306
hayes-roth	4	132
heart-cleveland	13	303
heart-hungarian	13	294
heart-switzerland	13	123
heart-va	13	200
hepatitis	19	155
hill-valley	100	606
horse-colic	22	300
ilpd-indian-liver	10	583
image-segmentation	19	210
ionosphere	34	351
iris	4	150
led-display	7	1000
lenses	4	24
libras	90	360
low-res-spect	101	531
lung-cancer	56	32
lymphography	18	148
magic	10	19020
mammographic	5	961
molec-biol-promoter	57	106
monks-1	6	124
monks-2	6	169
monks-3	6	122
mushroom	22	8124
musk-1	166	476
musk-2	166	6598
nursery	8	12960
ozone	72	2536
page-blocks	10	5473
parkinsons	22	195
pendigits	16	7494
pima	8	768
pb-MATERIAL	11	107
pb-REL-L	11	107
pb-SPAN	11	107
pb-T-OR-D	11	107
pb-TYPE	11	107
planning	12	182
post-operative	8	90
primary-tumor	17	330
ringnorm	20	7400
seeds	7	210
spambase	57	4601
spect	22	80
spectf	44	80

Continued on next page

Table A.1 – *Continued from previous page*

Dataset Name	**# Atts.**	**# Inst.**
st-aust-cred	14	690
st-germ-cred	20	1000
st-heart	13	270
st-landsat	36	4435
st-shuttle	9	43500
st-vehicle	18	846
steel-plates	27	1941
synth-ctrl	60	600
teaching	5	151
thyroid	21	3772
tic-tac-toe	9	958
titanic	3	2201
trains	32	10
twonorm	20	7400
vc-2classes	6	310
vc-3classes	6	310
wall-following	24	5456
waveform	21	5000
waveform-noise	40	5000
wine	13	179
wine-qual-red	11	1599
wine-qual-white	11	4898
yeast	8	1484
zoo	16	101

Bibliography

[1] Mark Hall et al. "The WEKA data mining software: an update". In: *ACM SIGKDD explorations newsletter* 11.1 (2009), pp. 10–18.

[2] Janez Demšar et al. "Orange: From experimental machine learning to interactive data mining". In: *Knowledge discovery in databases: PKDD 2004* (2004), pp. 537–539.

[3] MR Berthold et al. "KNIME: The Konstanz Information Miner In: Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007)". In: *Springer* 11 (2007), pp. 319–326.

[4] Ingo Mierswa et al. "Yale: Rapid prototyping for complex data mining tasks". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2006, pp. 935–940.

[5] Janez Kranjc, Vid Podpečan, and Nada Lavrač. "Clowdflows: a cloud based scientific workflow platform". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2012, pp. 816–819.

[6] Janez Kranjc, Vid Podpecan, and Nada Lavrac. "Real-time data analysis in ClowdFlows". In: *Big Data, 2013 IEEE International Conference on*. IEEE. 2013, pp. 15–22.

[7] Ryszard S Michalski, Jaime G Carbonell, and Tom M Mitchell. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013, p. v.

[8] Alex Hai Wang. "Detecting Spam Bots in Online Social Networking Sites: A Machine Learning Approach." In: *DBSec* 10 (2010), pp. 335–342.

[9] Sumeet Dua and Xian Du. *Data mining and machine learning in cybersecurity*. CRC press, 2016.

[10] Leonce Mekinda and Luca Muscariello. "Supervised Machine Learning-Based Routing for Named Data Networking". In: *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE. 2016, pp. 1–6.

[11] Thomas M. Mitchell. *Machine Learning*. 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997, p. 2. ISBN: 0070428077, 9780070428072.

[12] Stuart. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall, 2009, pp. 694–695. ISBN: 9780136042594, 0136042597.

[13] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2014, pp. 21–35.

[14] John B Guerard Jr and Eli Schwartz. "Regression analysis and forecasting models". In: *Quantitative Corporate Finance*. Springer, 2007, pp. 277–301.

[15] Jiliang Tang, Salem Alelyani, and Huan Liu. "Feature selection for classification: A review". In: *Data Classification: Algorithms and Applications* (2014), p. 37.

[16] Catherine L Blake. "UCI repository of machine learning databases". In: *http://www. ics. uci. edu/~ mlearn/MLRepository. html* (1998).

[17] Michael J Pazzani. "Influence of prior knowledge on concept acquisition: Experimental and computational results." In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 17.3 (1991), p. 416.

[18] Ian H Witten et al. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[19] Veronika Cheplygina and David MJ Tax. "Characterizing multiple instance datasets". In: *International Workshop on Similarity-Based Pattern Recognition*. Springer. 2015, pp. 15–27.

[20] A Rogier T Donders et al. "A gentle introduction to imputation of missing values". In: *Journal of clinical epidemiology* 59.10 (2006), pp. 1087–1091.

[21] Matthias Feurer et al. "Efficient and robust automated machine learning". In: *Advances in Neural Information Processing Systems*. 2015, pp. 2962–2970.

[22] Yu-Chi Ho and David L Pepyne. "Simple explanation of the no-free-lunch theorem and its implications". In: *Journal of optimization theory and applications* 115.3 (2002), pp. 549–570.

[23] James Bergstra, Daniel Yamins, and David Cox. "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures". In: *International Conference on Machine Learning*. 2013, pp. 115–123.

[24] Manuel Fernández-Delgado et al. "Do we need hundreds of classifiers to solve real world classification problems". In: *J. Mach. Learn. Res* 15.1 (2014), pp. 3133–3181.

[25] Christiane Lemke, Marcin Budka, and Bogdan Gabrys. "Metalearning: a survey of trends and technologies". In: *Artificial intelligence review* 44.1 (2015), pp. 117–130.

[26] Paul Clements and Linda Northrop. *Software product lines*. Addison-Wesley, 2002.

[27] Klaus Pohl, Günter Böckle, and Frank J van Der Linden. *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.

[28] Luca Parisi et al. "Comparison of Workflows: a Step Further". In: (), p. 20.

[29] John R Rice. "The algorithm selection problem". In: *Advances in computers* 15 (1976), pp. 65–118.

[30] Lars Kotthoff. "Algorithm selection for combinatorial search problems: A survey". In: *Ai Magazine* 35.3 (2014), pp. 48–60.

[31] Luca Pulina and Armando Tacchella. "A self-adaptive multi-engine solver for quantified Boolean formulas". In: *Constraints* 14.1 (2009), pp. 80–116.

[32] Thorsten Joachims. "Training linear SVMs in linear time". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2006, pp. 217–226.

[33] Ciro Castiello, Giovanna Castellano, Anna Maria Fanelli, et al. "Meta-data: Characterization of input features for meta-learning". In: *MDAI*. Springer. 2005, pp. 457–468.

[34] Pavel Brazdil, João Gama, and Bob Henery. "Characterizing the applicability of classification algorithms using meta-level learning". In: *European conference on machine learning*. Springer. 1994, pp. 83–102.

[35] Alexandros Kalousis and Melanie Hilario. "Feature selection for meta-learning". In: *Advances in knowledge discovery and data mining* (2001), pp. 222–233.

[36] Alexander K Seewald. "Meta-Learning for stacked classification". In: *audiology* 24.226 (2002), p. 69.

[37] Martijn J Post, Peter van der Putten, and Jan N van Rijn. "Does Feature Selection Improve Classification? A Large Scale Experiment in OpenML". In: *International Symposium on Intelligent Data Analysis*. Springer. 2016, pp. 158–170.

[38] Besim Bilalli et al. "Automated data pre-processing via meta-learning". In: *International Conference on Model and Data Engineering*. Springer. 2016, pp. 194–208.

[39] Alexandros Kalousis and Theoharis Theoharis. "Noemon: Design, implementation and performance results of an intelligent assistant for classifier selection". In: *Intelligent Data Analysis* 3.5 (1999), pp. 319–337.

[40] GÃŠrard Biau. "Analysis of a random forests model". In: *Journal of Machine Learning Research* 13.Apr (2012), pp. 1063–1095.

[41] Miles E Lopes. "Measuring the Algorithmic Convergence of Random Forests via Bootstrap Extrapolation". In: *Davis CA: Department of Statistics, University of California* (2015).

[42] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[43] Balaji Lakshminarayanan, Daniel M Roy, and Yee Whye Teh. "Mondrian forests for large-scale regression when uncertainty matters". In: *Artificial Intelligence and Statistics*. 2016, pp. 1478–1487.

[44] Daniel M Roy and Yee W Teh. "The mondrian process". In: *Advances in neural information processing systems*. 2009, pp. 1377–1384.

[45] Elena Ikonomovska, João Gama, and Sašo Džeroski. "Learning model trees from evolving data streams". In: *Data mining and knowledge discovery* 23.1 (2011), pp. 128–168.

[46] Elena Ikonomovska et al. "Speeding-up hoeffding-based regression trees with options". In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, pp. 537–544.

[47] Wassily Hoeffding. "Probability inequalities for sums of bounded random variables". In: *Journal of the American statistical association* 58.301 (1963), pp. 13–30.

[48] Ameni Bouaziz et al. "Interactive generic learning method (IGLM): A new approach to interactive short text classification". In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. ACM. 2016, pp. 847–852.

[49] Balaji Lakshminarayanan, Daniel M Roy, and Yee Whye Teh. "Mondrian forests: Efficient online random forests". In: *Advances in neural information processing systems*. 2014, pp. 3140–3148.

[50] Claudia Beleites et al. "Sample size planning for classification models". In: *Analytica chimica acta* 760 (2013), pp. 25–33.

[51] Gert Loterman and Christophe Mues. "Learning algorithm selection for comprehensible regression analysis using datasetoids". In: *Intelligent Data Analysis* 19.5 (2015), pp. 1019–1034.

[52] Ricardo BC Prudêncio, Carlos Soares, and Teresa B Ludermir. "Combining meta-learning and active selection of datasetoids for algorithm selection". In: *International Conference on Hybrid Artificial Intelligence Systems*. Springer. 2011, pp. 164–171.

[53] Matthias Reif, Faisal Shafait, and Andreas Dengel. "Dataset generation for meta-learning". In: *KI-2012: Poster and Demo Track* (2012), pp. 69–73.

[54] De Rainville et al. "Deap: A python framework for evolutionary algorithms". In: *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*. ACM. 2012, pp. 85–92.

[55] J Shohat. "Inequalities for moments of frequency functions and for various statistical constants". In: *Biometrika* (1929), pp. 361–375.

[56] Eamonn Keogh and Abdullah Mueen. "Curse of dimensionality". In: *Encyclopedia of Machine Learning*. Springer, 2011, pp. 257–258.

[57] Gareth James et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013.