Université de Nice - Sophia Antipolis

Master Ubinet
Internship Report

A Knowledge Flow as a Software Product Line

Presented by:
Luca Parisi

Supervisor:
Prof. Frédéric Precioso (I3S)
Prof. Mireille Blay Fornarino (I3S)

Abstract

Constructing a data mining workflow depends at least on the collected dataset and the objectives of users. This task is complex because of the increasing number of available algorithms and the difficulty in choosing the suitable and parametrized algorithms. Moreover, in order to decide which algorithm has the best performances, data scientists often need to use analysis tools to compare performances of different algorithms. The purpose of this project is to lay the foundations of a software system that leads the construction of such workflows into the right direction toward the best ones.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1: Introduction

1.1 Supervised Classification

Mobile sensors and social networks are only a few examples of applications that generate a huge amount of data each day. Such a collected big data can be analysed and new valuable knowledge can be extracted from it. The overall goal of data mining is to extract valuable information from datasets in order to retrieve new knowledge for further use. In the data mining field there exist several problems, one of which is the supervised classification problem. In order to explain better what supervised classification means, Table 1.1 reports an extract of the Iris dataset. Iris is a genus of one flowering plant, and its genus is defined according to the sepal length, sepal width, petal length and petal width of each flower. In the Iris dataset, each flower is one row of the dataset. In this example, there are three genuses: Iris-setosa, Iris-versicolor and Iris-virginica. In the first six rows of the dataset the Iris genus is known, but in the last one it is unknown. The goal of classification is to predict the right Iris genus for each flower where we do not know its genus, in this case the last row of the dataset. Predictions are made by a classification algorithm. Supervised classification means that before predicting the Iris genus, the classification algorithm needs to learn some rule from the flowers with a known Iris genus. That is, the algorithm needs to be trained in order to build a model that knows how to predict the Iris genus. Let K be the subset of the flowers with a known Iris Genus, in this example the first six rows of the dataset. Once we choose a classification algorithm and we build a model that is able to predict the Iris genus, how can we know if the predicted genuses are the correct ones? We need to evaluate whether the predictions done by the model are good or not. We can check the performances of the model by making it predict the Iris genuses of the flowers that we already know. To this purpose, we divide K in two datasets, K_1 and K_2, such that $K_1 \cap K_2 = \emptyset$ and $K_1 \cup K_2 = K$. K_1 is the training set, and it is used to train the classification algorithm and to build the model, while K_2 is the test set, and it is used check if the predictions done by the training model correspond with the acutal Iris genus of the flower. In order to check the performances of the model, we can calculate the accuracy, which is the percentage of correct predicted instances of the test set. After the evaluation, if the accuracy obtained by the model is good, the same model can be used to predict the Iris genus of the flowers with an unknown genus. Otherwise, we should build a different model that leads to a better accuracy during the evaluation of the model.

Sepal Length (cm)	**Sepal Width (cm)**	**Petal Length (cm)**	**Petal Width (cm)**	**Iris Genus**
...	...	...	...	...
5.1	3.5	1.4	0.2	Iris-Setosa
4.9	**?**	1.4	0.2	Iris-Setosa
6.4	3.2	4.5	1.5	Iris-Versicolor
6.9	3.1	4.9	1.5	Iris-Versicolor
6.5	3.0	5.2	2.0	Iris-Virginica
6.2	3.4	5.4	2.3	Iris-Virginica
6.3	3.1	4.2	1.8	**???**
...	...	...	...	...

Table 1.1: Subset of Iris Dataset

1.1.1 Vocabulary

This section aims to define well the terminology used in this report to refer to classification problems. For non-data scientist readers, it may be not so easy to understand fully the concepts illustrated here if they do not know the proper terminology. Furthermore, different terminologies are used in other books or articles, so sometimes it is hard to understand well the content of an article if the terminology is not well specified. The Iris dataset shown in Table 1.1 is used as an example dataset. The terminology used in this report can be summarized in the following terms:

- Instance: each row of the dataset is one instance.
- Attribute: each column of the dataset is one attribute. Examples of attributes are: "sepal length", "sepal width", "petal length", "petal width" and "Iris genus".
- Missing Value: each instance contains one value for each attribute. If, for any attribute, the value is not known, it is a missing value. An example of a missing value is the value "sepal width" of the second instance.
- Numeric Attribute: one attribute is numeric if it contains only numeric values. Examples of numeric attributes are "sepal length", "sepal width", "petal length" and "petal width" are numeric attributes.

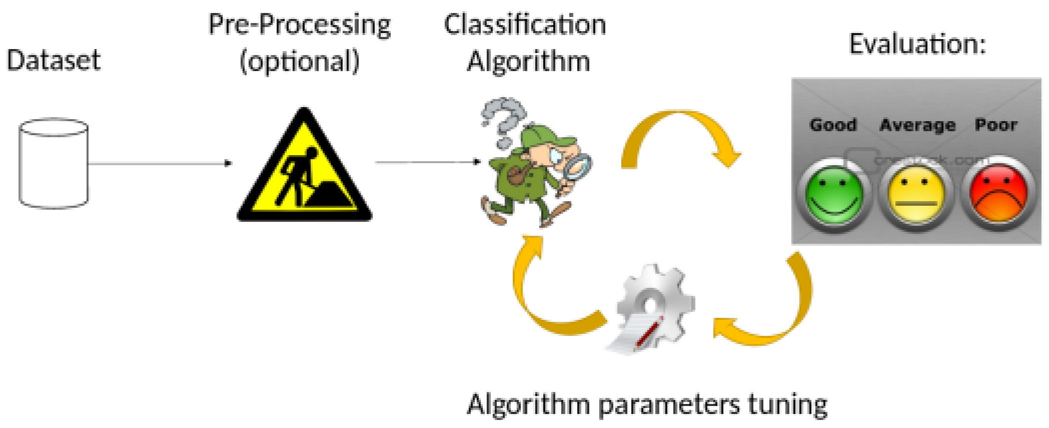


Figure 1.1: General Supervised Classification Workflow

- Nominal Attribute: one attribute is nominal if it contains only categories (text values). "Iris genus" is an example of nominal attribute.
- Class Attribute: it is one attribute of the dataset that identifies the class of each instance. In the classification problems, the class attribute must be nominal, it cannot be numeric. In the Iris dataset, the class attribute is "Iris genus".
- Class Label: each instance contains one class label, which is the value of the class attribute. An example of class label is "Iris-Setosa".
- Multi-Class vs Binary-Class Problem: If the class attribute contains more than two distinct categories, the classification problem is a multi-class problem. Otherwise, it is a binary class problem (called also two-class problem).
- Classifier: synonym of "classification algorithm".

1.1.2 Supervised Classification Workflows

A supervised classification process can be defined as a workflow of phases, as shown in Figure 1.1. An optional pre-processing to apply to the dataset is the first phase. The purpose of pre-processing is either to make the dataset compatible with the classification algorithm that we want to apply on it, or to modify the data in order to expect better accuracies when predicting classes. Typical pre-processing techniques are:

- Data Selection: only the relevant attributes are selected from the dataset.
- Data Cleaning: for example, if the dataset contains some missing values and the chosen classifier cannot deal with missing values, we need to replace them with another value.
- Data Transformation: the original data is transformed into a different kind of data. For example, numeric attributes may be discretized into nominal attributes, or transformed into a different scale, etc...

After the pre-processing phase, we need to choose a classification algorithm to apply on the pre-processed dataset. In the data mining literature there exist a lot of classification algorithms, each one with different performances. Some algorithms have some parameters to be set, and different values assigned to these parameters may change the performances of the classification algorithm. The last phase of the workflow is the evaluation of results. Once we have defined a pre-processing technique, chosen a classifier and set its parameters, we want to test whether the workflow will lead to good performances or not with an evaluation method. The simplest evaluation method is to train the classifier on one training set, to test it on one test set, and to calculate the accuracy obtained as performance indicator. Anyway, the accuracy obtained on only one test set may not be a good performance indicator, because it may not represent well the future instances to be predicted. In order to have an evaluation more statistically sound, we may train and test the classifier on several training sets and test sets. An evaluation method that does this is the N-fold cross validation, which evaluates for N times a classifier on different training sets and test sets. Let K be the dataset used in the evaluation phase, where in each instance the class label is known. Then, K is divided into N folds, $K_1, K_2, ..., K_N$, and for N times, (N-1) folds are used as training set and 1 fold is used as test set. The final accuracy is the average of the N accuracies found on each of the N test sets. If the accuracy obtained during the evaluation is good enough, the evaluated model can be used to predict future instances with unknown class labels, otherwise it should be changed. There are three ways to change the workflow:

1. change the pre-processing technique applied to the original dataset
2. choose a different classifier
3. set different values to the parameter

Each one of these changes leads to a different workflow that can obtain better or worse accuracies in the evaluation phase.

1.2 Motivations

When there is a classification problem to be solved, data scientists, according to their knowledge and to their experience, know the best practices that solve it the best. They know how to manipulate the problem and which algorithms will probably lead to the best results. On the other hand, non-expert users neither know the best practices nor have any experience in the data mining field. Both for data scientists and non-expert users, there exist a lot of platforms that help them in building workflows. Section 1.2.1 reports the state of the art of such existing systems. Currently, we are implementing a system that leads users into the right direction in finding their most suitable workflow, depending on the problem to be solved. Section 1.2.2 reports the differences among the existing platforms and the system that we are implementing, called ROCKFlows [1].

1.2.1 State Of The Art

Data scientists can use platforms like Weka [6], Orange [10], KNIME [11], RapidMiner [12] and ClowdFlows [14] [15], in order to build workflows more easily. On these platforms, users can create and execute workflows by selecting a set of components and by connecting them. Figure 1.2 shows an example of a workflow built on Weka's knowledge flow, which allows to build workflows by dragging and dropping components on the screen and to connect them easily by using arrows.

Although such platforms are useful for data scientists, they may be a little bit complex for non-expert users. For them, it may be more helpful a system that either solves automatically the problem, or advise a set of suitable workflows to use, or suggests the components to select. Big companies have proposed their own cloud platforms, which aim to be such a system. The IBM Watson platform [18] offers an interface to analyze unstructured data (such as text files) and take as input questions in plain English. Then, the system presents answers to the questions submitted by analyzing automatically the dataset given by the user. Amazon's product [17] is a black box similar to the IBM's platform, but it is focused on supervised classification and regression problems. The workflow is built and executed automatically by the platform, and minimal responsibility for the choice of the workflow is left to the users. On the other hand, the solution proposed by Microsoft Azure [16] is different both from IBM's and from Amazon's platform. Instead of finding the solution automatically for users, it gives them advices on which components to select, based

[1]ROCKFlows stands for Request your Own Convenient Knowledge Flows.

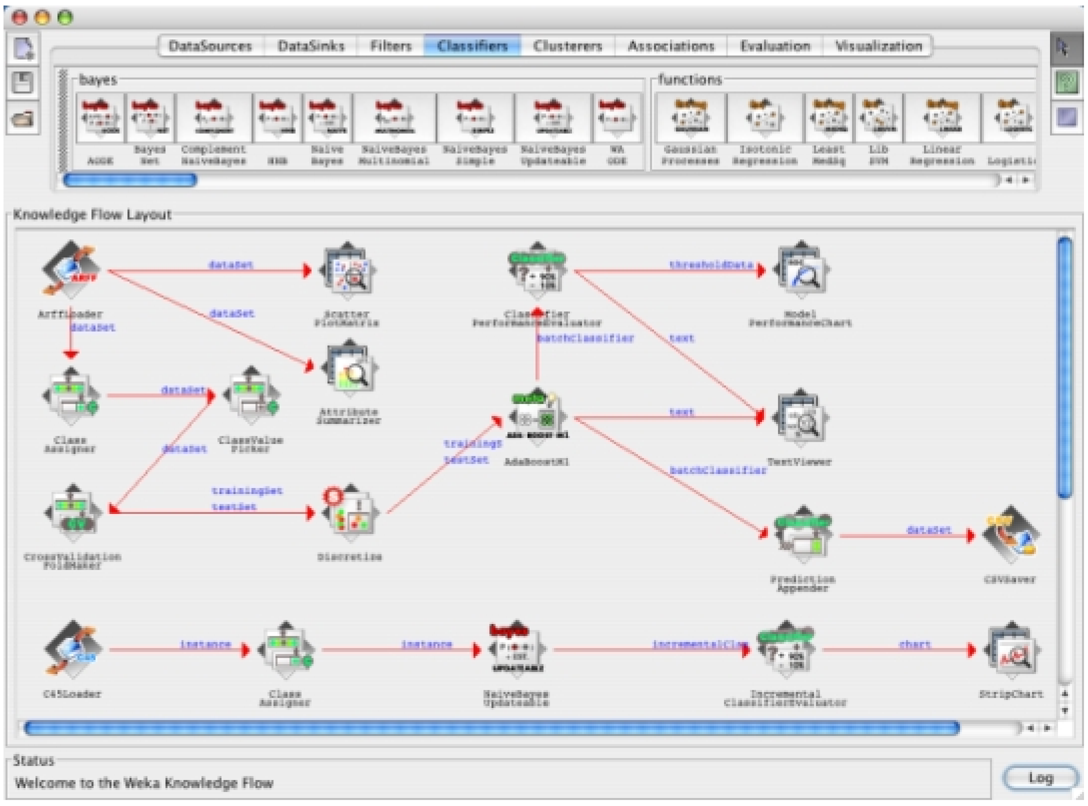


Figure 1.2: Example of a Workflow built on Weka's Knowledge Flow.

on the known machine-learning best practices. Figure 1.3 shows the Microsoft cheat sheet that orients users in choosing the proper classification algorithm, in case of two-class classification problems.

Finally, there exists a system born in the academic world, named MLBase [13]. It aims to free users from the algorithm choice by building a sophisticated cost-based model. It provides early answers to the users, then it keeps working in background while users use the system, in order to optimize and to improve the workflow.

1.2.2 ROCKFlows Features

ROCKFlows aims to lay the foundations of a system intended both to data scientists and to non-expert users. Data scientists are free to build their desired workflows, while non-expert users can ask the system to select automatically components of the workflow. For example, if data scientists want to use a specific algorithm, they are free to choose it. Similarly, if non-expert users do not know which algo-

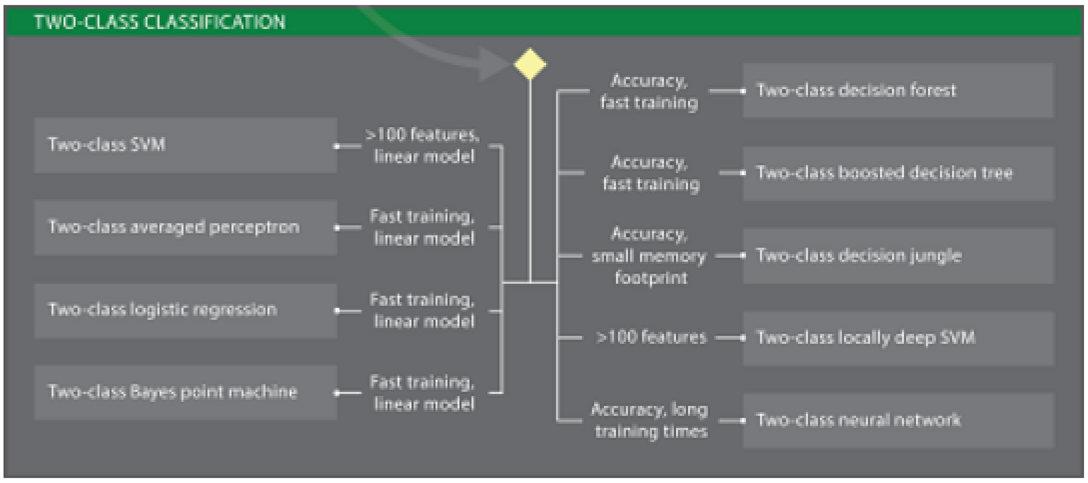
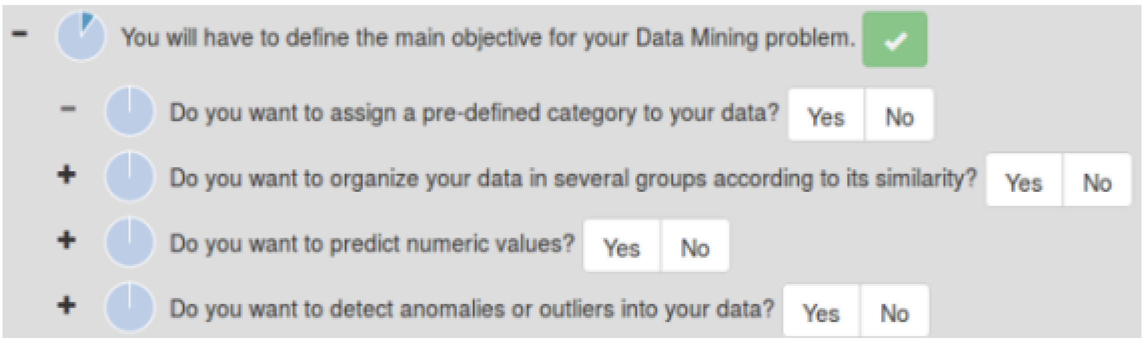

TWO-CLASS CLASSIFICATION
Two-class SVM
>100 features, linear model
Two-class averaged perceptron
Fast training, linear model
Two-class logistic regression
Fast training, linear model
Two-class Bayes point machine
Fast training, linear model
Accuracy, fast training
Two-class decision forest
Accuracy, fast training
Two-class boosted decision tree
Accuracy, small memory footprint
Two-class decision jungle
>100 features
Two-class locally deep SVM
Accuracy, long training times
Two-class neural network

Figure 1.3: Microsoft Azure cheat sheet that orients users in choosing the proper classification algorithm, in case of two-class classification problems


You will have to define the main objective for your Data Mining problem. ✓

Do you want to assign a pre-defined category to your data? Yes No

Do you want to organize your data in several groups according to its similarity? Yes No

Do you want to predict numeric values? Yes No

Do you want to detect anomalies or outliers into your data? Yes No

Figure 1.4: GUI of ROCKFlows: questions are asked in plain english, and users answer by selecting pre-built answers

rithm is better to use, the system will select one of the most suitable algorithm depending on their dataset. With respect to the Weka knoledge flow (Figure 1.2), users do not need to select and to connect the components with arrows. Once the components are selected, ROCKFlows knows how to connect them and how to build the workflow. Similar to the IBM Watson platform, ROCKFlows asks users questions in plain english, and they answer by choosing one on the available pre-built answers. Figure 1.4 reports an example of questions / answers asked by the GUI of ROCKFlows.

With respect to the solution proposed by Microsoft Azure (Figure 1.3), ROCK-

Flows aims to consider workflows which does not correspond necessarily to knwon best practices. Our purpose is to explore more the space of the solutions of machine learning workflows. For example, in the two-class classification problem, Microsoft Azure advise only nine algorithms, depending on the dataset properties, on the speed of training and on the level of accuracy that users want. Finally, with respect to MLBase, the objective of ROCKFlows is not to keep working in background in order to improve the workflows advised to the users. ROCKFlows aims to predict right away the most suitable workflows, without comparing all the possible workflows in background in order to find the best one.

1.3 Goals Of The Internship

The work done for this internship is the theoretical research that there is behind ROCKFlows. For starters, we face only the supervised classification problem. The main goal of this internship is to define a strategy that let ROCKFlows predict right away the most suitable workflows to the users, without comparing the performances of different workflows as the Amazon or MLBase solutions do. To reach this goal, ROCKFlows needs to know in advance the performances that workflows will have on the user's dataset, that is, it should be able to skip the evaluation phase. From users' point of view, the most suitable workflow can be either the one who reaches the best accuracy on their datasets, or the one who reaches a good accuracy by respecting, in the same time, some users' constraints. Users' constraints may concern the performances of the execution of workflows, for example, the total time of execution of workflows or the amount of RAM occupied by the process executing the workflow. To this purpose, the second goal of this internship is to predict both the time of execution of a workflow and the memory used by the process executing it, depending on the user's dataset.

1.4 Challenges

As explained in Section 1.1.2, when we build a classification workflow we need to choose at least three components:

- An optional pre-processing technique to apply to the dataset
- A classification algorithm to choose
- The values for the parameters requested by the classifier

The first challenge of this internship is the high variability coming from the construction of workflows. Each modification to one of these three components leads to a different workflow. So, even if we would like to compare all the possible workflows among them in order to find the most suitable for the user, it would be impossible because of the huge time that we should wait. This is also one of the hot topic research in the data mining field. In terms of accuracy, predictions of workflows would be easier if, among all the classification algorithms that exist in the literature, only one of them had always better accuracies than the others. That is, the construction of workflows would be easier if we had to select only one classifier because it is always the best regardless the users' datasets. Unfortunately, the No-Free-Lunch Theorem [2] states that the best classifier will not be the same for each dataset. The same holds for the pre-processing choice. The application of a pre-processing technique on the user's dataset changes the performances of the algorithms of classification, and in the data mining literature there does not exist one pre-processing technique that guarantees always the best performances than the others. The second challenge of this project is to know in advance the performances that workflows will have during their execution, without doing the evaluation phase. That is:

- *How can we predict the accuracy that a workflow will have on users datasets, without doing the evaluation phase?*
- *How can we predict the total time of execution that a workflow will have on users datasets, without doing the evaluation phase?*
- *How can we predict the memory usage that the process executing the workflow will have on users datasets, without doing the evaluation phase?*

Chapter 2: Related Work

The most related work to the research done for this internship is the paper written by M.F.Delgado et al. (2014): *Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?* Journal of Machine Learning Research 15 3133-3181. This paper does not consider all the phases of a general workflow as we do in this project, it is focused on the choice of the best classification algorithm for solving supervised classification problems. In order to know which classifiers work best in general, the authors have tested 179 classifiers on 121 datasets, of which 117 coming from the UCI repository [5]. About the 179 classifiers, they have tested the implementation of the same algorithms done on different frameworks, such as R [7], Weka [6], C and Matlab (by using the Neural Network Toolbox) [8].

Regarding the pre-processing, they have applied the same pre-processing to each dataset. At first they have converted all the nominal attributes into numeric attributes 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, they have standardized each attribute, so in the end each attribute is numeric, where its mean is equal to zero and its standard deviation is equal to one. Finally, they have replaced all the missing values with the value 0. Apart from these operations, they have not done any further pre-processing. For each classifier, the authors have reported the values assigned to its parameters. Anyway, some parameters need to be tuned on the specific dataset before being assigned. In this case, the authors have tuned them on a validation dataset, which is extracted by each of the tested dataset before evaluating the algorithm. The validation dataset is formed by one training set and by one test set. The training set contains the 50% of the total number of instances of the dataset, while the test set contains the remaining 50%. About the datasets of the UCI repository [5], some of them are already separated in one training set and one test set, while others are defined in a unique file. For the evaluation phase, the authors have decided to use two different evaluation methods. If the dataset is defined in a unique file, the classifier (with the tuned parameter values) is evaluated with a 4-fold cross validation, otherwise, it is trained on the given training set and tested on the given test set.

As results of the paper, the authors have reported a final ranking of the 179 classifiers, where for each classifier they have calculated the Friedman rank [3], the average accuracy obtained on the 121 datasets and the Cohen k [4] as performance indicators. From their experiments, the authors have found that the algorithm that obtained the best results is Random Forest implemented in R and accessed via caret, which achieves 94.1% of the maximum accuracy, while the second best one is the SVM with Gaussian kernel implemented in C using LibSVM, which achieves 92.3% of the maximum accuracy. As conclusion, they have compared the families of classifiers. They say that random forest is clearly the best family of classifiers (3 out of 5 bests classifiers are RF), followed by SVM (4 classifiers in the top-10), neural networks (5 members in the top-20) and boosting ensembles (3 members in the top-20). In order to make the experiments reproducible, the authors have made their 121 pre-processed datasets available for download [1]. Furthermore, for each dataset, they have also published two files: `conxuntos.dat` defines the validation dataset used for the parameter tuning of the algorithms, while `conxuntos_kfold.dat` defines the folds used by the 4-fold cross validation.

2.1 Critics

The results reported into the paper are very interesting and show that, generally, there are some families of classifiers that work better than the others. Since in general we obtain good performances with the Random Forest or Svm families of classifiers, we could base the choice of the classification algorithm only by considering these two families of classifiers. Anyway, their experiments present five points that may change the results reported by the authors:

1. In the evaluation phase, they use two different evaluation methods in order to calculate the same final ranking of classifiers. If the choice of the evaluation method affects the results of the evaluation, it will affect also the final ranking of the classifiers reported by the authors. For example, the authors could have obtained a different ranking of classifiers if they had used the 4-Fold cross validation method also on the datasets already separated into one training set and one test set. So, the first critic can be expressed by the following questions: *What is the impact of the evaluation method used for the evaluation phase? Can we find significant differences in the final ranking of classifiers if we consider two different evaluation methods?*

2. They perform only one pre-processing technique, because they say: *the impact of these transforms can be expected to be similar for all the classifiers; however, our objective is not to achieve the best possible performance for each data set (which eventually might require further pre-processing), but to compare classifiers on each set.* The second critic can be expressed by the following questions:

[1] `http://persoal.citius.usc.es/manuel.fernandez.delgado/papers/jmlr/data.tar.gz`

What if the impact of the pre-processing is not similar for all the classifiers? Can we find significant differences in the final ranking if we consider also other pre-processing techniques?

3. Some Weka classifiers cannot work directly on the pre-processed datasets made by the authors, because they can work only on nominal attributes. In this case, Weka converts automatically the numeric attributes into nominal attributes in order to make the dataset suitable for these classifiers. The Weka classifiers that can work only on nominal attributes are the following:

   - weka.classifiers.bayes.BayesNet
   - weka.classifiers.rules.OneR
   - weka.classifiers.meta.Bagging OneR
   - weka.classifiers.meta.MultiBoostAB OneR

   The consequence of this is that the results obtained by these classifiers refer to datasets different from the ones used by the authors, so maybe the final ranking of classifiers has been affected by this. As we wondered in the previous critic, if the impact of pre-processing is significant, we might find a different final ranking if we test the classifiers that can work also on numeric attributes on the dataset pre-processed by Weka that contain only nominal attributes.

4. In the pre-processing technique used by the authors, they have replaced the missing values with the value 0. Anyway, some algorithms can manage missing values, so for them it is not necessary to replace them. So, the third critic can be expressed by the following questions: *What is the impact of missing values on these algorithms? Can we find significant differences in the ranking for the classifiers that can manage missing values, if we do not replace them?*

5. They have tested some meta-classifiers implemented in Weka with the default base classifier, the zeroR classifier [?]. This classifier always predicts the mode of the nominal attributes, that is, in case of the class labels, it predicts always the most frequent class of the dataset. This is the reason why in the final ranking reported into the paper, the following meta-classifiers have bad results:

   - weka.classifiers.meta.Grading
   - weka.classifiers.meta.MultiScheme

- weka.classifiers.meta.Vote
- weka.classifiers.meta.Stacking: the authors use the stacking ensemble method only with one zeroR base classifier, while the method is intended to test several base classifiers, not only one.
- weka.classifiers.meta.StackingC: the same reason as Stacking.

Chapter 3: Strategy of Research

In order to skip the evaluation phase and to predict the best workflows on untested datasets, ROCKFlows needs to know at least what their performances are on some datasets. Basing on the information retrieved, we can find some rules that allow the system to predict the performances of the workflows on untested datasets, without doing the evaluation phase. To this purpose, we have defined a strategy that can be summarized in four phases:

1. *Experiments* (Section 3.1): each workflow is tested on some datasets and its performances are retrieved and stored in Excel files.

2. *Analysis* (Section 3.2): we analyse how much the accuracy obtained by the workflows on each dataset is good, without considering only the average accuracy as value of comparison.

3. *Significant Impacts on Classifier Performances* (Chapter 4): by using the *Analysis* phase, we study which factors have a significant impact on the performances among different workflows. By replicating partially the experiments done by Delgado et al. (Section 2), we want to find an answer to the questions presented in critics #1, #2 and #4 of Section 2.1.

4. *Predictions* (Chapter 5): by exploiting the results retrieved in the *Analysis* phase (Section 5.1), we want to find some rules that allow to predict the performances of workflows on untested datasets, without doing the evaluation phase. Hence, we can predict the workflows that, allegedly, will reach the best accuracy (Section 5.1) on untested datasets. Similarly, we want to predict the total time of execution of each workflow and the memory usage of the process executing it (Sections 5.2 and 5.3).

3.1 Experiments

The research done for this internship is based on the experiments reported in Section 2. We have replicated in part the experiments of Delgado et al., but we have limited our experiments on the Weka's platform (version 3.6.14), by testing 65/179 classifiers implemented with the Weka's APIs on 101/121 datasets of the UCI repository [5]. Anyway, since this project is focused on workflows and not only in classifiers, we have extended the experiments done by the authors by considering 12 pre-processing techniques along with the original dataset (without pre-processing). For each algorithm we have set the same parameters reported by the authors, hence, except for the parameters of the classifier LibSvm, we do not tune the parameters before testing the classifier on each dataset. Hence, we define a workflow as a 2-tuple (p_i,c_j) composed by:

- p: the choice of a sequence of pre-processing techniques applied to the dataset
- c: the choice of the algorithm of classification

Any variation to p_i or to c_j defines a different workflow that the system may take into account when predicting the best workflow to the users. For research sake reproducibility, we have made available the code used to do the experiments. [1]

3.1.1 Datasets (D)

In our experiments, we have tested 101/121 datasets coming from the UCI repository [5]. In the rest of this report, the set of 101 datasets is referred as D.

Since we based our experiments on Weka's platform, we have defined each dataset $d_k \in D$ in .arff files, which is the format used by Weka to read them. In the UCI repository, for each dataset is given its raw data and the description of the problem, which explains what each attribute refers to. Generally, each nominal attribute is represented with a range of numbers, where each number refers to a specific category. For example, an attribute *Size* that have categories *small, medium, large* can be expressed as three numbers: $0, 1, 2$ }. Since datasets in the UCI repository are not in the .arff format, we had to define manually the categories for each nominal attribute. Likewise, we have set a numeric attribute only when its values are real numbers, for example, the measure coming from a sensor. In our experiments, the difference between numeric and nominal attributes is very important because the nature of the attributes of the dataset defines which pre-processing techniques are applicable on them, as explained in Section 3.1.2. Moreover, from the description of datasets, we have removed all the instance identifiers, that is, all the attributes that present a unique value for each instance. We have done this manual pre-processing because the identifiers may affect the results obtained by the experiments.

Dataset Properties

Furthermore, in our experiments it is important to know the structure of the datasets, which we identify with the following properties:

[1]`https://github.com/ROCKFlows/experiments-public/tree/master/sourceCode`

- Number of Attributes
- Number of Instances
- Number of Classes: 2 (binary-class problem) or more (multi-class problems)
- Type of Attributes: nominal, numeric or both of them.
- Missing Values: Does the dataset contain any missing value?

Dataset List

Table 3.1 shows the set of datasets D used in our experiments, along with the properties of each dataset. For reproducible research sake, we have made the 101 datasets available for public download. [2]

3.1.2 Pre-Processing (P)

From each dataset $d_k \in D$ to test, several pre-processed datasets with different properties are created by applying a set of pre-processing techniques P to d_k. In our experiments, we have decided to test the original dataset along with 12 pre-processing techniques, that is, from 1 to 12 pre-processed datasets may be created from the original one, depending on if the pre-processing technique $p_i \in P$ is applicable to the original dataset (Section 3.1.2). Each pre-processing technique $p_i \in P$ may contain one or more operations, in this case they are applied sequentially to the original dataset. The set of pre-processing techniques P used in our experiments is reported in the following:

p_0 No pre-processing (original dataset)

p_1 {Discretize}

p_2 {Replace Missing Values}

p_3 {Replace Missing Values, Discretize}

p_4 {Nominal To Binary}

p_5 {Replace Missing Values, Nominal To Binary}

[2]`https://github.com/ROCKFlows/experiments-public/tree/master/Resources/datasets/work.tar.gz`

Dataset	#Att.	#Inst.	#Cls.	Att. Types	M.V.
ac-inflam	6	120	2	Both	no
ac-nephritis	6	120	2	Both	no
adult	14	32561	2	Both	yes
annealing	38	798	6	Both	yes
arrhythmia	279	472	16	Both	yes
audiology-std	69	171	18	Nominal	yes
balance-scale	4	625	3	Numeric	no
balloons	4	16	2	Nominal	no
bank	16	4521	2	Both	no
blood	4	748	2	Numeric	no
breast-cancer	9	286	2	Nominal	yes
bc-wisc	8	699	2	Nominal	yes
bc-wisc-diag	29	569	2	Numeric	no
bc-wisc-prog	32	198	2	Numeric	yes
breast-tissue	9	106	6	Numeric	no
car	6	1728	4	Nominal	no
ctg-3classes	45	2126	3	Numeric	yes
chess-krvkp	36	3196	2	Nominal	no
congress-voting	16	435	2	Nominal	yes
conn-bench-sonar	60	208	2	Numeric	no
conn-bench-vowel	12	528	11	Both	no
contrac	9	1473	3	Both	no
credit-approval	15	690	2	Both	yes
cylinder-bands	37	512	2	Both	yes
dermatology	34	366	6	Both	yes
echocardiogram	12	131	2	Both	yes
ecoli	7	336	8	Numeric	no
fertility	9	100	2	Numeric	no
flags	28	194	8	Both	no
glass	9	214	7	Numeric	no
haberman-survival	3	306	2	Both	no
hayes-roth	4	132	3	Numeric	no
heart-cleveland	13	303	5	Both	yes
heart-hungarian	13	294	2	Numeric	yes
heart-switzerland	13	123	5	Numeric	yes

Table 3.1: List of 101 tested datasets coming from the UCI repository. M.V. stands for "Missing Values"

Dataset	**#Att.**	**#Inst.**	**#Cls.**	**Att. Types**	**M.V.**
heart-va	13	200	5	Numeric	yes
hepatitis	19	155	2	Both	yes
hill-valley	100	606	2	Numeric	no
horse-colic	22	300	2	Both	yes
ilpd-indian-liver	10	583	2	Both	no
image-segmentation	19	210	7	Numeric	no
ionosphere	34	351	2	Numeric	no
iris	4	150	3	Numeric	no
led-display	7	1000	10	Nominal	no
lenses	4	24	3	Nominal	no
libras	90	360	15	Numeric	no
low-res-spect	101	531	9	Numeric	no
lung-cancer	56	32	3	Nominal	yes
lymphography	18	148	4	Both	no
magic	10	19020	2	Numeric	no
mammographic	5	961	2	Numeric	no
molec-biol-promoter	57	106	2	Nominal	no
monks-1	6	124	2	Nominal	no
monks-2	6	169	2	Nominal	no
monks-3	6	122	2	Nominal	no
mushroom	22	8124	2	Nominal	yes
musk-1	166	476	2	Numeric	no
musk-2	166	6598	2	Numeric	no
nursery	8	12960	5	Nominal	no
ozone	72	2536	2	Numeric	yes
page-blocks	10	5473	5	Numeric	no
parkinsons	22	195	2	Numeric	no
pendigits	16	7494	10	Numeric	no
pima	8	768	2	Numeric	no
pb-MATERIAL	11	107	3	Both	yes
pb-REL-L	11	107	3	Both	yes
pb-SPAN	11	107	3	Both	yes
pb-T-OR-D	11	107	2	Both	yes
pb-TYPE	11	107	6	Both	yes
planning	12	182	2	Numeric	no

Continuation of Table 3.1

Dataset	**#Att.**	**#Inst.**	**#Cls.**	**Att. Types**	**M.V.**
post-operative	8	90	3	Nominal	yes
primary-tumor	17	330	15	Nominal	yes
ringnorm	20	7400	2	Numeric	no
seeds	7	210	3	Numeric	no
spambase	57	4601	2	Numeric	no
spect	22	80	2	Nominal	no
spectf	44	80	2	Numeric	no
st-aust-cred	14	690	2	Both	no
st-germ-cred	20	1000	2	Both	no
st-heart	13	270	2	Numeric	no
st-landsat	36	4435	6	Numeric	no
st-shuttle	9	43500	7	Numeric	no
st-vehicle	18	846	4	Numeric	no
steel-plates	27	1941	7	Numeric	no
synth-ctrl	60	600	6	Numeric	no
teaching	5	151	3	Both	no
thyroid	21	3772	3	Both	no
tic-tac-toe	9	958	2	Nominal	no
titanic	3	2201	2	Nominal	no
trains	32	10	2	Nominal	yes
twonorm	20	7400	2	Numeric	no
vc-2classes	6	310	2	Numeric	no
vc-3classes	6	310	3	Numeric	no
wall-following	24	5456	4	Numeric	no
waveform	21	5000	3	Numeric	no
waveform-noise	40	5000	3	Numeric	no
wine	13	179	3	Numeric	no
wine-qual-red	11	1599	11	Numeric	no
wine-qual-white	11	4898	11	Numeric	no
yeast	8	1484	10	Numeric	no
zoo	16	101	7	Nominal	no

Continuation of Table 3.1

p_6 {Attribute Selection}

p_7 {Discretize, Attribute Selection}

p_8 {Replace Missing Values, Discretize, Attribute Selection}

p_9 {Nominal To Binary, Attribute Selection}

p_{10} {Replace Missing Values, Nominal To Binary, Attribute Selection}

p_{11} The same pre-processing described in Section 2 (we have taken directly the dataset released by the authors)

p_{12} We apply pre-processing p_6 to the pre-processed dataset p_{11}

In the rest of this report, each pre-processed dataset is identified with the same identifier of the above pre-processing techniques. For example, we can refer to "the pre-processed dataset p_4" in order to specify the pre-processed dataset obtained after applying the pre-processing technique p_4 to the original dataset.

Pre-processing $\{p_i \in P : 1 \leq i \leq 10\}$ are based on four Weka filters:

- weka.filters.unsupervised.attribute.ReplaceMissingValues: replaces missing values with the mean (if the attribute is numeric) or the mode (if the attribute is nominal)
- weka.filters.unsupervised.attribute.Discretize: discretizes, by simple binning, a numeric attribute into a nominal attribute. The maximum number of bins is 10.
- weka.filters.unsupervised.attribute.NominalToBinary: converts a nominal attribute into a set of binary numeric attributes. A nominal attribute with k distinct values is transformed into k binary attributes 0-1, by using the one-attribute-per-value approach.
- Attribute Selection: we have created a custom Weka filter that selects a subset of attributes that are highly correlated with the class and have in the same time a low intercorrelation among them. The selection is done by the class weka.attributeSelection.CfsSubsetEval

Domain of Applicability

We apply pre-processing $\{p_i \in P : 1 \leq i \leq 10\}$ to the original dataset (p_0) if and only if the application of p_i to p_0 creates a pre-processed dataset with different dataset properties with respect to p_0. For example, if p_0 does not contain any missing value, we do not apply to it the pre-processing p_2. The same holds for the attribute types of the original dataset. For example, if p_0 contains only nominal attributes, we cannot apply to it the pre-processing techniques $\{p_1, p_3, p_7, p_8\}$, because the Weka filter "Discretize" can be applied only on numeric attributes. To this purpose, in order to know which pre-processing is applicable to p_0, we need to check whether it contains or not any missing value and what attribute types it has. Table 3.2 reports the set of applicable pre-processing with respect to these to properties. Regarding pre-processing p_{12}, it can be always applied to the pre-processed dataset p_{11}, because it is simply a selection of its attributes.

Type of Attributes	**Missing Values**	**Applicable pre-processing**
Nominal and Numeric	Yes	$\{\forall p_i \in P \mid 1 \leq i \leq 10\}$
Nominal and Numeric	No	$\{p_1, p_4, p_6, p_7, p_9\}$
Numeric	Yes	$\{p_1, p_2, p_3, p_6, p_7, p_8\}$
Numeric	No	$\{p_1, p_6, p_7\}$
Nominal	Yes	$\{p_2, p_4, p_5, p_6, p_9, p_{10}\}$
Nominal	No	$\{p_4, p_6, p_9\}$

Table 3.2: Domain of applicability of pre-processing $\{p_i \in P : 1 \leq i \leq 10\}$ with respect to the dataset properties

3.1.3 Classifiers (C)

In our experiments we have tested 65/179 classifiers of the ones tested by Delgado et al. in their paper [1]. In the rest of this report, the set of 65 classifiers is referred as C. As we already said in Critic #3 of Section 2.1, if a classifier cannot work directly on the dataset, Weka applies a hidden pre-processing on it in order to make the dataset compatible for the classifier. Since the experiments done for this project take into account the pre-processing phase as part of the workflow, this behavior should be avoided because it changes the workflow that we want to test. In such a case, we simply say that the classifier is not compatible with the dataset, so we cannot test the workflow. For example, we cannot apply the Svm classifier on a dataset $d_k \in D$ that has both numeric and nominal attributes, because it can work only on numeric attributes. That is, we cannot test the workflow (p_0, Svm) on d_k. However, if we apply pre-processing p_4 to d_k, Svm becomes compatible with the pre-processed dataset, because it contains only numeric attributes, but this is a different workflow (p_4, Svm). Since the Weka documentation does not say anything about the hidden pre-processing that it applies on the datasets, in order to know the kind of datasets that a classifier can work on, we had to look directly its java source code implementation. Then, we have defined three boolean properties that specify the kind of datasets on which a classifier can work on:

- Require Numeric Attributes: true if the classifier can work only on numeric attributes. False if it can work on nominal attributes, too.
- Require Nominal Attributes: true if the classifier can work only on nominal attributes. False if it can work on numeric attributes, too.
- Manage Missing Values: true if the classifier can manage missing values. False otherwise.

If a classifier cannot work on multi class problems, we have used the strategy 1-vs-1, which trains $\frac{K \cdot (K-1)}{2}$ binary classifiers for a K-class problem. Each binary classifier learns to distinguish two class labels, then, at prediction time, a voting scheme is applied. All the $\frac{K \cdot (K-1)}{2}$ classifiers predict the class label of an unknown instance, and the class that receives more votes is selected as class label for the instance.

3.1.4 Classifier List

This section reports the set of classifiers C used in our experiments, organized according to the java Weka's packages. For each classifier $c_j \in C$, we report only the values of the parameters that are different from the default values assigned by Weka (version 3.6.14). The same parameters have been used by Delgado et al. in their experiments [1].

weka.classifiers.functions:

1. Logistic

2. LibSVM : Svm with gaussian kernel. It requires the tuning of the parameters C and γ before being applied on datasets. For this reason, for each dataset $d_k \in D$, we have used the validation dataset defined into the file `conxuntos.dat`. That is, for each dataset we have created a distinct validation dataset, on which the parameters of Svm have been tuned with the following values:
   - Parameter C: [0.1, 1, 10, 100, 1000]
   - Parameter γ: [0.00001, 0.0001, 0.001, 0.01, 0.1, 1]
3. MultilayerPerceptron
4. SimpleLogistic
5. SMO
6. RBFNetwork. Number of clusters = half of training instances, ridge = 10^{-8} for the linear regression. This classifier can be applied only on datasets with standardized numeric attributes (pre-processing p_{11} and p_{12}).

weka.classifiers.trees:

1. J48
2. NBTree
3. RandomForest. The number of trees is set to 500.
4. LMT
5. ADTree
6. RandomTree. At least 2 instances per leaf, unclassified patterns are allowed.
7. REPTree
8. DecisionStump

weka.classifiers.bayes:

1. NaiveBayes
2. BayesNet

weka.classifiers.lazy:

1. IBk
2. IB1
3. LWL

weka.classifiers.rules:

1. JRip
2. PART. Confidence threshold for pruning = 0.5.
3. DTNB
4. Ridor
5. DecisionTable
6. ConjunctiveRule
7. OneR
8. NNge. Number of folder for computing the mutual information = 1, Number of attempts of generalisation = 5.

weka.classifiers.misc:

1. HyperPipes
2. VFI

weka.classifiers.meta:

In this section we report the list of meta classifiers that we have tested. Each of the java classes representing them accepts only one base classifier as input. So, if we want to test two different base classifiers, we need to test two different meta classifiers.

1. Bagging. The base classifiers that we have tested with Bagging are:
    - RepTree

- J48
- NBTree
- DecisionStump
- RandomTree
- OneR
- JRip
- DecisionTable
- PART
- NaiveBayes
- HyperPipes
- LWL

2. MultiBoostAB. The base classifiers that we have tested with MultiBoostAB are the same reported for Bagging, except HyperPipes and LWL.
3. RandomCommittee. The base classifier that we have tested is RandomTree.
4. LogitBoost. Threshold on likelihood improvement = 1.79. The base classifier that we have tested is LogitBoost DecisionStump.
5. RacedIncrementalLogitBoost. The base classifiers that we have tested is Raced-IncrementalLogitBoost DecisionStump.
6. AdaBoostM1. The base classifiers that we have tested are:
    - DecisionStump
    - J48
7. RandomSubSpace. The base classifier that we have tested is RepTree.
8. OrdinalClassClassifier. The base classifier that we have tested is J48.
9. Dagging. Number of folds = 4. The base classifier that we have tested is SMO.
10. Decorate. 15 J48 (Section 3.1.4 #1) are used as base classifiers. Weka generates an exception if the dataset contains a nominal attribute with only one distinct category.

11. ClassificationViaRegression. The base classifier that we have tested is `weka.classifiers.trees.M5P`.

12. ClassificationViaClustering. The base classifiers that we have tested are:

    - weka.clusterers.SimpleKMeans
    - weka.clusterers.FarthestFirst

3.1.5 Evaluation

When we want to test the workflow (p_i, c_j) on a dataset $d_k \in D$, we need to use an evaluation method in order to compute its average accuracy. As we wondered in critic 1 of Section 2.1, we have decided to test if two different evaluation methods leads to a significant difference of the average accuracy obtained by the workflows. In our experiments, we have used:

- 4-fold cross validation: we have built the exact 4-folds defined into the file `conxuntos_kfold.dat` released by M.F.Delgado et al.

- 10-fold cross validation: we have used the Weka's APIs `Instances.trainCV` and `Instances.testCV`, which build the folds deterministically. In this way, the 10 folds can be reproduced by other researchers who want to replicate our experiments.

This way of building the folds guarantees that they are built equally for each workflow, so the way on which data is folded does not bias the results obtained among different workflows.

For a tested dataset (the original one or a pre-processed one), the the performances obtained by each tested workflow are stored into Excel files, along with the pre-processing time required by the pre-processing phase. As example, Table 3.3 reports the results obtained for five classifiers on the original dataset *Adult* (p_0), in case of 4-Fold cross validation. Logistic Regression and Svm are not compatible with the dataset p_0 *Adult*, because these two classifiers can work only on numeric attributes and they do not manage missing values, while the dataset p_0 *Adult* has both numeric and nominal attributes and missing values. In this example, the pre-processing time is 0 because we do not apply to any pre-processing to the original dataset.

For each workflow, we calculate the following average values, obtained either on the 4-Folds cross-validation or on the 10-Fold cross-validation.

			pre-proc. time = 0 s		
Algorithm	**Comp.**	**Accuracy (%)**	**Training Time (s)**	**Test Time (s)**	**Memory (Mb)**
Logistic Regression	no	-	-	-	-
Svm	no	-	-	-	-
J48	yes	86,23	20	0,05	0,2
NBTree	yes	86,06	201	0,78	7
Random Forest	yes	84,91	975,6	87	900

Table 3.3: Example of results collected for 5 classifiers on the original dataset (p_0) *Adult*, in case of 4-Fold cross validation. Comp. = Compatible.

- average accuracy
- pre-processing time
- training time
- test time
- amount of memory occupied by the trained classifier, that is, the amount of memory occupied by the java object representing the Weka implementation of the classifier. The memory is computed by the java library Classmexer (ref.), which is an external agent that can read the memory used by any Java object.

Regarding the memory required by the execution of workflows, we have not been able to detect dynamically the RAM occupied by the java process that executes the workflow, so we have limited this study on the amount of memory occupied by the trained classifier (java object).

3.2 Analysis

Once all the results from the experiments have been gathered, for each dataset $d_k \in D$ we need to define a strategy that tells us how much the workflow (p_i, c_j) is good with respect to the accuracy, without taking into account only the average accuracy value. We want to distinguish whether there are significant differences between two average accuracies or not, because the average accuracy is related only to the evaluation phase, where the class labels of each instance are known. That

p_i	**Algorithm**	**Rank**	**Accuracy (%)**	**Total Time (s)**	**Memory (Mb)**
0	Decorate	1	86,83	1569	17
4	RotationForest J48	1	86,71	1173	11
11	Bagging PART	1	86,70	979,2	17
5	RotationForest J48	1	86,65	644,1	10
2	Decorate	2	86,64	1797	7
...	...	...	...	...	...
0	J48	6	86,23	20,5	0,2
...	...	...	...	...	...

Table 3.4: Example of ranking of workflows for the dataset *Adult*, in case of 4-Fold cross validation.

is, we may obtain different accuracies when we use the same workflow on other unknown instances, which have not been used during the evaluation phase.

Hence, we have assigned to the results obtained by each workflow during the evaluation phase a rank value, which groups together the workflows who are not significantly different among each other. If there is a significant difference between two workflows, they will have a different rank, otherwise they will have the same rank. For example, Table 3.4 reports the top 5 ranking of the workflows obtained on the dataset *Adult*, in case of 4-Fold cross-validation. From the ranking, we can see that there is not a significant difference between the average accuracies of the workflows (p_0, Decorate) and (p_4, RotationForest J48), because both of them have rank=1. On the other hand, there is a significant different between the average accuracies of the workflows (p_5, RotationForest J48) and (p_2, Decorate), because the former has rank=1 and the latter has rank=2.

The lower the rank, the better is the workflow. All the workflows that have rank=1 are considered as best workflows for the dataset. In the *Adult* dataset example (Table 3.4), the workflows (p_0, Decorate), (p_4, RotationForest J48), (p_{11},Bagging PART) and (p_5, RotationForest J48) are the best workflows in terms of accuracy, since they have rank=1. For each workflow, the total time of execution is calculated by adding the pre-processing time, training time and test time stored into the Excel file of the related pre-processed dataset. For example, in Table 3.4 the total time of execution of the workflow (p_0, J48) is calculated by summing the pre-processing time, training time and test time found on Table 3.3.

Algorithm 1 Rank Calculation Algorithm

```
procedure setRank(List<Workflow> W)
    sort W from the best avg to the worst one
    for each w $\in$ W
        w.rank $\leftarrow$ 1
    maxRank $\leftarrow$ 1
    top:
    for i = 1 to maxRank
        toAnalyze $\leftarrow$ W.getWorkflowsByRank(i)
        size $\leftarrow$ toAnalyze.getSize()
        first $\leftarrow$ toAnalyze.getWorkflow(0)
        for k = 2 to size
            current $\leftarrow$ toAnalyze.getWorkflow(k)
            if first.avg $\neq$ current.avg
                if significantDifference(first.array, current.array)
                    for each w $\in$ W | w.avg $\leq$ current.avg
                        w.rank $\leftarrow$ w.rank + 1
                    maxRank $\leftarrow$ maxRank + 1
                    goto top.
```

The algorithm that calculates the rank is reported in Algorithm 1. It needs to know, for each workflow, both the average accuracy value and the array of accuracies that led to that specific average. In the pseudo code, a workflow is identified as an object Workflow, in which the average value is stored in the field Workflow.avg, the array of values in the field Workflow.array, and the output rank in the field Workflow.rank. The significant difference among the average accuracies is checked with the statistical hypothesis tests, as explained in Section 3.2.1.

3.2.1 Statistical Hypothesis Tests

The significant difference between the average accuracies of two workflows is checked with statistical hypothesis tests, which compare two arrays of values. This is the reason why in Algorithm 1 we need the array of values that form the average accuracy. In case of 4-Fold cross validation, the array contains the four accuracies found on each fold, while in case of 10-Fold cross validation the array contains ten accuracies. As null hypothesis, we set that the difference of the means of the arrays is zero. If we can reject with a degree of confidence of 95% the null hypothesis, we can say that the difference of the means of the two arrays is not zero, so the average accuracies of the two workflows are significantly different. Otherwise, we accept the null hypothesis because we do not have enough confidence to reject it, that is, we say that the two workflows are not significantly different. The methodology used to do the tests is the following: first of all, the two arrays are sorted from the lowest value to the highest one. Then, we check with the Shapiro-Wilk method if each of the two arrays follows a normal distribution. If this is the case, we perform the statistical tests with the Student's paired TTest, which assumes that data is normally distributed. Otherwise, we use the Wilcoxon Signed Rank test, which does not require the normal distribution of the data.

The Shapiro-Wilk method is implemented into the java library `jdistlib.disttest.NormalityTest`, while the Student's paired TTest and the Wilcoxon Signed Rank test are implemented into the java classes `org.apache.commons.math3.stat.inference.TTest` and `org.apache.commons.math3.stat.inference.WilcoxonSignedRankTest`

Chapter 4: Significant Impacts on Rankings of Classifiers

In this chapter, we study which factors have a significant impact when we want to compare the performances among different classifiers. By replicating partially the experiments done by Delgado et al. (Section 2), we want to find an answer to the questions presented in critics #1, #2 and #4 of Section 2.1. If the choice of one evaluation method, the choice of a pre-processing technique and the management of missing values have a significant impact on the accuracies obtained by classifiers, we have to take into account also these factors before saying that a classifier c_a is better than a classifier c_b. Depending on the question to answer, we calculate two rankings of classifiers based on the rank value, as explained in Section 3.2. Depending on the factor studied, if for the same classifier c_j we find that it has two different rank values between the two rankings, we can conclude that the factor has a significant impact on the accuracies obtained by classifiers. Since the rankings presented in this chapter are based on classifiers, the average accuracy is calculated among the datasets $d_k \in D$, and one rank value is assigned to each classifier.

4.1 Evaluation Impact

In order to check if the choice of the evaluation method has a significant impact on the accuracies of classifiers (critic #1, Section 2.1), we check if there are significant differences between the ranks obtained with the two evaluation methods used in our experiments: the 4-Fold cross validation and the 10-Fold cross validation. For each dataset $d_k \in D$, the results come from its pre-processed dataset p_{11}. The left ranking of Table 4.1 reports the top 10 classifiers in case of the 4-Fold cross validation, while the right ranking reports the top 10 classifiers in case of the 10-Fold cross validation. The full comparison is not reported in this report because it would be too much large, anyway, it is anyway available for public download [1]. As Table 4.1 shows, the two rankings are significantly different. For example, in case of 4-Fold cross validation Logistic Model Tree has rank=3 and RotationForest RandomTree rank=2, while in case of 10-Fold cross validation Logistic Model Tree has rank=2 and RotationForest RandomTree rank=3. In this example, RotationForest RandomTree is better than Logistic Model Tree with the 4-Fold cross-validation, while Logistic Model Tree is better than RotationForest RandomTree with the 10-Fold

[1] `https://github.com/ROCKFlows/experiments-public/tree/master/Resources/comparisons/crossValidationImpact/Cross-Validation-Comparison.xlsx`

Pre-Processing p_{11}					
4-Fold Cross Validation			**10-Fold Cross Validation**		
Algorithm	**Rank**	**% Avg. Accuracy**	**Algorithm**	**Rank**	**% Avg. Accuracy**
Rot.For. J48	1	82,72	Svm	1	76,48
Svm	1	82,68	Rot.For. J48	1	76,45
Random Forest	1	82,47	Random Forest	2	75,99
Bagging PART	2	81,60	L.M.T	2	75,92
MAB, PART	2	81,58	Rot.For. R.Tree	3	75,77
Rot.For. R.Tree	2	81,50	Bagging PART	4	75,40
Bag. NBTree	3	81,43	Bagging J48	5	75,09
Bagging J48	3	81,32	MAB J48	5	75,07
L.M.T.	3	81,28	MAB NBTree	5	75,07
MAB J48	3	81,22	MAB, PART	6	74,97

Table 4.1: Top 8 of classifier rankings among the pre-processed datasets p_{11}.
Left Ranking: 4-Fold cross validation.
Right Ranking: 10-Fold cross validation.
Rot.For.=RotationForest, MAB=MultiBoostAB, Bag.=Bagging, L.M.T.=Logistic Model Tree, R.Tree=RandomTree

cross-validation.

So we can conclude that the choice of the evaluation method has a significant impact on the ranking of classifiers, and this factor has to be taken into account when comparing the performances among classifiers or workflows.

4.2 Pre-Processing Impact

In order to check if the choice of a pre-processing technique has a significant impact on the accuracies of classifiers (critic #2, Section 2.1), we check if there are significant differences between the ranks obtained from the pre-processed datasets p_{11} and from the best pre-processed datasets. That is, the first ranking is calculated with the accuracies from the pre-processed datasets p_{11}, while the second ranking of classifiers is calculated with the best accuracies coming from one of the pre-processed datasets $p_i \in P$. Since we have proved in Section 4.1 that the choice of the evaluation method has a significant impact on the rankings, we have analysed

4-Fold Cross Validation					
Pre-Processing p_{11}			**Best Pre-Processing $p_i \in P$**		
Algorithm	**Rank**	**%Avg. Accuracy**	**Algorithm**	**Rank**	**%Avg. Accuracy**
Rot.For. J48	1	82,72	Rot.For. J48	1	85,99
Svm	1	82,68	Random Forest	1	85,91
Random Forest	1	82,47	Rot.For R.Tree	1	85,83
Bagging PART	2	81,60	Bag. NBTree	2	85,32
MAB, PART	2	81,58	Svm	2	84,96
Rot.For. R.Tree	2	81,50	MAB NBTree	3	84,83
Bag. NBTree	3	81,43	R.C. R.Tree	3	84,75
Bagging J48	3	81,32	L.M.T.	3	84,75

Table 4.2: Top 10 of classifier rankings in case of 4-Fold cross validation.
Left Ranking: accuracies come from p_{11} datasets.
Right Ranking: accuracies come from the best $p_i \in P$ datasets.

10-Fold Cross Validation					
Pre-Processing p_{11}			**Best Pre-Processing $p_i \in P$**		
Algorithm	**Rank**	**%Avg. Accuracy**	**Algorithm**	**Rank**	**%Avg. Accuracy**
Svm	1	76,48	Svm	1	80,39
Rot.For. J48	1	76,45	Rot.For. J48	2	80,11
Random Forest	2	75,99	Rot.For. R.Tree	3	79,89
L.M.T.	2	75,92	L.M.T.	4	79,80
Rot.For. R.Tree	3	75,77	Random Forest	5	79,57
Bagging PART	4	75,40	MAB NBTree	6	79,19
Bagging J48	5	75,09	MLP	6	79,17
MAB J48	5	75,07	Bag. NBTree	6	79,06

Table 4.3: Top 8 of classifier rankings in case of 10-Fold cross validation.
Left Ranking: accuracies come from p_{11} datasets.
Right Ranking: accuracies come from the best $p_i \in P$ datasets.
Rot.For.=RotationForest, MAB=MultiBoostAB, Bag.=Bagging, L.M.T.=Logistic Model Tree, R.Tree=RandomTree, R.C.=RandomCommittee

the pre-processing impact both with the 4-Fold cross validation and with the 10-Fold cross validation. Table 4.2 reports the analysis based on the 4-Fold cross validation: the left ranking reports the top 8 classifiers by considering only pre-processing p_{11}, while the right ranking reports the top 8 classifiers by considering the best pre-processing technique where a classifier gets its highest accuracy. Similarly, Table 4.3 reports the analysis based on the 10-Fold cross validation. The full comparisons are not reported in this report because they are too much large, anyway, they are available for public download [2]. As both Table 4.2 and Table 4.3 show, the two rankings are significantly different. For example, Table 4.2 shows that with the p_{11} datasets, RotationForest RandomTree has rank=2 and Svm rank=1, while with the best $p_i \in P$ datasets, RotationForest RandomTree has rank=1 and Svm rank=2. In this example, Svm is better than RotationForest RandomTree if we consider only pre-processing p_{11}, but RotationForest RandomTree is better than Svm if we consider the best pre-processing $p_i \in P$. Similarly, Table 4.3 shows that with the p_{11} datasets, Random Forest has rank=2 and RotationForest RandomTree rank=3, while with the best $p_i \in P$ datasets, Random Forest has rank=5 and RotationForest RandomTree rank=3. In this example, Random Forest is better than RotationForest RandomTree if we consider only pre-processing p_{11}, but RotationForest RandomTree is better than Random Forest if we consider the best pre-processing $p_i \in P$.

So we can conclude that the choice of the pre-processing has a significant impact on the final ranking of classifiers, both in case of 4-Fold cross validation and in case of 10-Fold cross-validation. Hence, pre-processing has to be taken into account when comparing the performances among classifiers or workflows.

4.3 Missing Values Impact

In order to check if the treatment of missing values has a significant impact on the accuracies of classifiers (critic #4, Section 2.1), we check if there are significant differences between the ranks obtained from the datasets that contain missing values and the pre-processed datasets without missing values. For this analysis, only the 30/101 datasets that contain missing values and only the classifiers $c_j \in C$ that can work on missing values are considered for building the two rankings of classifiers.

[2] 4-Fold cross validation `https://github.com/ROCKFlows/experiments-public/tree/master/Resources/comparisons/preProcessingImpact/p11-vs-bestPi-4Fold-Comparison.xlsx`
10-Fold cross validation: `https://github.com/ROCKFlows/experiments-public/tree/master/Resources/comparisons/preProcessingImpact/p11-vs-bestPi-10Fold-Comparison.xlsx`

The first ranking is calculated with the accuracies of the original datasets (p_0), while the second ranking is calculated with the accuracies of the pre-processed datasets p_2, where the missing values are replaced with the mean/mode of the attributes. Since we have proved in Section 4.1 that the choice of the evaluation method has a significant impact on the rankings, we have analysed the impact of missing values both with the 4-Fold cross validation and with the 10-Fold cross validation. Table 4.4 reports the analysis based on the 4-Fold cross validation: the left ranking reports the top 6 classifiers on the datasets with missing values, while the right ranking reports the top 6 classifiers on the pre-processed datasets p_2. Similarly, Table 4.5 reports the analysis based on the 10-Fold cross validation: the left ranking reports the top 9 classifiers on the datasets with missing values, while the right ranking reports the top 9 classifiers on the pre-processed datasets p_2. The full comparisons are not reported in this report because they are too much large, anyway, they are available for public download [3]. As both Table 4.4 and Table 4.5 show, the two rankings are significantly different. For example, Table 4.4 shows that, in case of missing values, Logistic Model Tree has rank=1 and Bagging J48 rank=2, while when the missing values are replaced, both Logistic Model Tree and Bagging J48 have rank=2. In this example, Logistic Model Tree is better than Bagging J48 only on datasets with missing values, while on datasets without missing values they can be considered equal because they have the same rank. Similarly, Table 4.5 shows that, in case of missing values, Logistic Model Tree has rank=3 and Bagging PART rank=1, while when the missing values are replaced, Logistic Model Tree has rank=2 and Bagging PART rank=3. In this example, Bagging PART is better than Logistic Model Tree only on datasets with missing values, while on datasets without missing values Logistic Model Tree is better than Bagging PART.

So we can conclude that the treatment of missing values has a significant impact on the ranking of classifiers, and this is another factor to take into account when comparing the performances among the classifiers.

[3] 4-Fold cross-validation: `https://github.com/ROCKFlows/experiments-public/tree/master/Resources/comparisons/missingValuesImpact/mv-4Folds.xlsx`
10-Fold cross-validation: `https://github.com/ROCKFlows/experiments-public/tree/master/Resources/comparisons/missingValuesImpact/mv-10Folds.xlsx`

4-Fold Cross Validation					
Pre-Processing p_0			**Pre-Processing p_2**		
Algorithm	**Rank**	**% Avg. Accuracy**	**Algorithm**	**Rank**	**% Avg. Accuracy**
Bag. NBTree	1	79,64	Rot.For. J48	1	78,89
Rot.For. J48	1	79,28	Bag. NBTree	1	78,72
L.M.T.	1	79,08	Rot.For. R.Tree	2	78,42
Bagging JRip	2	78,77	L.M.T.	2	78,26
MAB NBTree	2	78,76	Bagging J48	2	78,13
Bagging J48	2	78,63	Bagging JRip	3	78,08

Table 4.4: Top 6 of classifier rankings in case of 4-Fold cross-validation. *Left Ranking*: accuracies come from the original datasets with missing values. *Right Ranking*: accuracies come from the pre-processed datasets p_2 without missing values.

10-Fold Cross Validation					
Pre-Processing p_0			**Pre-Processing p_2**		
Algorithm	**Rank**	**% Avg. Accuracy**	**Algorithm**	**Rank**	**% Avg. Accuracy**
Rot.For. J48	1	73,88	Rot.For. J48	1	74,14
MAB, PART	1	73,83	Bag. NBTree	1	74,12
Bag. NBTree	1	73,76	Bagging J48	1	73,80
Bagging PART	1	73,69	MAB, PART	2	73,59
Bagging J48	1	73,50	L.M.T.	2	73,32
MAB NBTree	2	73,22	Bagging PART	3	73,17
A.D.T.	2	73,09	MAB J48	4	73,04
Rot.For. R.Tree	3	73,08	A.D.T.	4	73,03
L.M.T.	3	72,93	Rot.For. R.Tree	4	72,85

Table 4.5: Top 9 of classifier rankings in case of 10-Fold cross-validation. *Left Ranking*: accuracies come from the original datasets with missing values. *Right Ranking*: accuracies come from the pre-processed datasets p_2 without missing values.

Rot.For.=RotationForest, MAB=MultiBoostAB, Bag.=Bagging, L.M.T.=Logistic Model Tree, R.Tree=RandomTree, A.D.T=Alternating Decision Tree

Chapter 5: Predictions of Workflows

The last step of this research is to predict the performances that a workflow (p_i, c_j) will have on untested datasets (the user's datasets), without doing the evaluation phase and without comparing all the possible workflows among them. Regarding the accuracy, we want to predict the workflows that will reach the highest accuracies on untested datasets. Similarly, we want to predict approximate values both for the total time of execution of the workflow and the amount of RAM used during its execution. As we explained in Section 3.1.5, we have not been able to detect dynamically the amount of RAM occupied by the java process that executes the workflow, so we have limited this study on the memory occupied by the trained classifier, which in our experiments is a java object. The total time of execution of the workflow is the sum of the pre-processing time, training time and test time.

Section 5.1 reports how to predict workflows that, allegedly, will have the best accuracy on untested datasets, while Section 5.2 reports how to predict the total time of execution and the memory usage of the trained classifier (Java object).

5.1 Accuracy Predictions

In order to predict the accuracies that workflows will have on untested datasets, we want to find some rule among the results of our experiments (Section 3.1). Our approach is to work on data patterns: all the datasets that match the same data pattern δ are grouped together, then we check if a workflow (p_i, c_j) has rank=1 on each dataset that match δ. If we were able to find at least one workflow for a data pattern δ that satisfies these conditions, we may suppose that it will have the best accuracy on untested datasets that match δ. With this method, we can predict workflows without doing the evaluation phase and without comparing all the possible workflows each other. But now the question is: *How do we define a data pattern?* We base the definition of data pattern according to the dataset properties reported in Section 3.1.1. By setting the number of attributes, the number of instances, the type of the attributes, etc... of the dataset, we can specify a data pattern.

Once a data pattern δ has been defined, we need to check if there is at least one workflow that has the best accuracies on each dataset that matches δ. As we said in Section 3.2, on a dataset $d_k \in D$ we consider a workflow as best if it has rank=1. That is, we consider a workflow as best for the data pattern δ if it has an average rank=1 among the datasets that match δ. Since in Section 4.1 we proved that the impact of the evaluation method is a factor that has to be taken into account when comparing the workflows, we need to check that its average rank is 1 both on the 4-Fold and on the 10-Fold cross validation.

Among the results of our experiments, we have found only one data pattern δ_1 in which a workflow has an average rank=1. The data pattern is defined as:

- the number of attributes is in the range [16, 21]
- the number of instances is greater than 211
- multi-class problems
- numeric attributes
- missing values are not present

3/101 datasets match this data pattern: {pendigits, statlog-vehicle, waveform}. Table 5.1 and Table 5.2 report the top 5 workflows ordered by the average rank obtained on each of the three datasets, in case of 4-Fold and 10-Fold, respectively. The full rankings are not reported here because they would be too much large, anyway, they are available for bublic download [1]. On each ranking, the workflow (p_{11}, Svm) has rank=1. Hence, we may suppose that the workflow (p_{11}, Svm) will have the best accuracies on the untested datasets that match δ_1. Since we have only 3 datasets that match the data pattern δ_1, we could not investigate further the performances of the workflow (p_{11}, Svm) on other datasets. So, this is not a strong evidence on which to state that the workflow (p_{11}, Svm) has always best performances on the data pattern δ_1, anyway it is a good candidate to predict. With this strategy, we can find other good candidate workflows, which do not have necessarily the average rank=1. For example, another good candidate workflow is (p_0, Svm), which second in the ranking both in case of 4-Fold cross validation (Table 5.1) and in case of 10-Fold cross-validation (Table 5.1).

Other Data Patterns

During this internship, we have spent a lot of time in searching also other data patterns, but we have found only data patterns where workflows either had rank=1

[1] 4-Fold cross-validation: `https://github.com/ROCKFlows/experiments-public/tree/master/Resources/comparisons/dataPattern/analysis-4Folds.xlsx`
10-Fold cross-validation: `https://github.com/ROCKFlows/experiments-public/tree/master/Resources/comparisons/dataPattern/analysis-10Folds.xlsx`

p_i	**Algorithm**	**Avg. Rank**	**%Avg. Accuracy**	**# Rank=1**
11	Svm	1	90,27	3/3
0	Svm	1,33	89,18	2/3
0	Logistic Model Tree	3,33	89,35	2/3
11	Logistic Model Tree	3,33	89,35	2/3
11	RotationForest J48	3,66	87,86	0/3

Table 5.1: Top 5 ranking of workflows on the data pattern δ_1 in case of 4-Fold cross-validation, ordered by the average rank.

p_i	**Algorithm**	**Avg. Rank**	**%Avg. Accuracy**	**# Rank=1**
11	Svm	1	90,72	3/3
0	Svm	2	89,32	1/3
12	Svm	4	86,79	1/3
11	Logistic Model Tree	4,67	89,65	1/3
0	Logistic Model Tree	4,67	89,65	1/3

Table 5.2: Top 5 ranking of workflows on the data pattern δ_1 in case of 10-Fold cross-validation, ordered by the average rank.

in case of the 4-Fold cross-validation or in case of the 10-Fold cross-validation, not both of them. Since we have proved in Section 4.1 that the choice of the evaluation method has a significant impact on the results obtained, we cannot state that such workflows are good candidates to be the best ones on untested datasets with the same data pattern.

5.2 Impacts on Time and Memory

In order to predict the total time of execution of workflows and the memory usage of the trained classifier (java object), we study how these values change with respect to the structure of the datasets. The structure of dataset is what determine its size, for example:

- the number of instances of the dataset

- the number of attributes of the dataset
- the number of classes of the dataset

At first, we want to check if the structure of datasets affect time and memory values. If this is the case, we want to find some dependencies from the structure of datasets and the time and the memory requested by workflows on untested datasets. If we knew how these values change with respect to the structure of datasets, it would suffice only to check the structure of the untested datasets in order to predict time and memory values.

For this study, we have considered only the results coming from the 10-Fold cross validation and from the pre-processed dataset p_{11}. We have decided to use only the results of the 10-Fold cross-validation because its training sets are bigger than the training sets of the 4-Fold cross-validation ($\frac{9}{10} > \frac{3}{4}$), and generally, the training phase lasts more time than the test phase. We have decided to use only the pre-processing p_{11} because each classifier $c_j \in C$ is compatible with it, except for BayesNet, OneR, Bagging OneR and MultiBoostAB OneR. Anyway, during this internship we have had time to check the impact of the structure of datasets only on 10 classifiers: Random Forest, Logistic Model Tree, Bagging NBTree, MultiBoostAB NBTree, Decorate, Logistic Regression, Svm, J48, NBTree and Multillayer Perceptron. We have chosen both base-classifiers and meta-classifiers, in order to check if there are differences among their performances.

5.2.1 Number of classes Impact

In order to check if the number of classes impacts the time of execution of a workflow and the memory requested by the trained classifier, we fix the number of attributes and the number of instances of the datasets, then we check how these values vary with respect to the number of classes. From the tested datasets D, we have compared the performances between the *mushrooms* and the *pendigits* dataset. These datasets have almost the same number of attributes (22 and 16, respectively) and almost the same number of instances (8124 and 7494, respectively), the substantial difference is the number of classes (2 and 10, respectively). Table 5.3 reports the different time and memory results obtained by the 10 classifiers on the *mushrooms* and *pendigits* datasets. From Table 5.3 we can see that, on the *pendigits* dataset, both time and memory values increase when the number of classes increases. The only exception is the time of execution of SVM, which goes from 30 seconds to 16 seconds. The only responsible for these increments is the higher number of classes

	mushrooms		***pendigits***	
Algorithm	**Time (s)**	**Mem. (Mb)**	**Time (s)**	**Mem. (Mb)**
Random Forest	164	3	393	37
Logistic Model Tree	779	0,1	2234	0,6
Bagging NBTree	775	17	1790	97
MultiBoostAB NBTree	109	3	1646	79
Decorate	157	0,8	437	22
Logistic Regression	4	0,03	185	0,03
Svm	30	0,01	16	0,01
J48	1	0,02	3	0,1
NBTree	115	3	358	10
Multillayer Perceptron	255	0,04	310	0,06

Table 5.3: Time and memory values obtained by 10 classifiers on the *mushrooms* and the *pendigits* datasets

of the *pendigits* dataset, because it has less instances and less attributes than the *mushrooms* dataset. From these results, we can conclude that the number of classes of datasets affects both the time of execution of the workflow and the memory occupied by the trained classifer. Why is the time of execution of Svm lower on the *pendigits* dataset? Maybe it is because it is more sensible to the number of attributes or to the number of instances, which are higher on the *mushrooms* dataset.

5.2.2 Number of instances Impact

In order to check if the number of instances impacts the time of execution of a workflow and the memory requested by the trained classifier, we fix the number of attributes and the number of classes of the datasets, then we check how these values vary with respect to the number of instances. To this purpose, we have taken the *musk-2* dataset, which contains 6598 instances, and we have divided it into two other datasets: the first one contains 2362 instances and the second one contains 3956 instances. Table 5.4 reports the different time and memory results obtained by the 10 classifiers on the three datasets. For each classifier, we can see that both time and memory values increase when the number of instances inreases. So we can conclude that the number of instances of the dataset affects both the time of execution of workflows and the memory occupied by the trained classifer.

	2362 instances		3956 instances		6598 instances	
Algorithm	**Time (s)**	**Mem. (Mb)**	**Time (s)**	**Mem. (Mb)**	**Time (s)**	**Mem. (Mb)**
Random Forest	9	9	18	18	410	22
Logistic Model Tree	154	1	311	2	3074	2
Bagging NBTree	2993	156	5659	262	8419	284
MultiBoostAB NBTree	1799	95	4882	238	5876	249
Decorate	64	2	145	4	550	3
Logistic Regression	1	0,2	2	0,2	38	0,2
Svm	3	0,08	7	0,08	70	0,08
J48	1	0,1	1	0,1	24	0,1
NBTree	255	19	396	28	3134	29
Multillayer Perceptron	291	0,9	476	0,9	3392	0,9

Table 5.4: Time and memory values obtained by 10 classifiers on three *musk-2* datasets, with different number of instances: 2362, 3956 and 6598

5.2.3 Number of attributes Impact

In order to check if the number of attributes impacts the time of execution of a workflow and the memory requested by the trained classifier, we fix the number of instances and the number of classes of the datasets, then we check how these values vary with respect to the number of attributes. To this purpose, we have compared these values between two pair of datasets:

- *waveform* and *waveform-noise*: the unique difference between these dataset is the number of attributes: 21 the former and 40 the latter
- *adult* p_{10} and *adult* p_5: the pre-processing technique p_5 leads to a higher number of attributes than the original *adult* dataset because it transforms each nominal attributes into two or more binary numeric attributes. Contrarily, the pre-processing technique p_10 leads to a lower number of attributes because it performs an attribute selection on the original dataset. The unique difference between the two pre-processed datasets is the number of attributes: 12 (p_5) and 106 (p_{10}).

Table 5.5 reports the different time and memory results obtained by the 10 classifiers between the *waveform* and *waveform-noise* datasets, while Table 5.6 reports

	waveform		***waveform-noise***	
Algorithm	**Time (s)**	**Mem. (Mb)**	**Time (s)**	**Mem. (Mb)**
Random Forest	640	48	748	50
Logistic Model Tree	425	0,02	716	0,04
Bagging NBTree	2265	51	3356	98
MultiBoostAB NBTree	2317	49	3467	98
Decorate	444	11	312	11
Logistic Regression	26	0,03	55	0,05
Svm	30	0,01	85	0,02
J48	22	0,2	24	0,2
NBTree	376	3	526	5
Multillayer Perceptron	237	0,05	593	0,1

Table 5.5: Time and memory values obtained by 10 classifiers on the *waveform* and the *waveform-noise* datasets

	p_{10} ***Adult***		p_5 ***Adult***	
Algorithm	**Time (s)**	**Mem. (Mb)**	**Time (s)**	**Mem. (Mb)**
Random Forest	960	53	1675	520
Logistic Model Tree	842	0,7	1403	4
Bagging NBTree	1111	23	12308	198
MultiBoostAB NBTree	908	20	7951	127
Decorate	514	2	3022	21
Logistic Regression	14	0,03	156	0,2
Svm	191	0,02	836	0,1
J48	13	0,04	175	0,4
NBTree	185	5	3338	27
Multillayer Perceptron	365	0,03	5543	0,4

Table 5.6: Time and memory values obtained by 10 classifiers on the pre-processed *Adult* p_{10} and *Adult* p_5 datasets.

the differences between the *adult* dataset p_{10} and p_5. From these results, we can see that when the number of attributes increases, time and memory values increase as well. There are only two exception, that we can find in Table 5.5: the time of execution of the classifier Decorate decreases when the number of attributes increases, and similarly, the memory required by the trained Random Forest decreases. So, we can conclude that the number of attributes of the dataset affects both the time of execution of workflows and the memory occupied by the trained classifer. Why are the time of execution of Decorate and the memory usage of Random Forest lower when the number of attributes increase? Maybe because they are more sensible to the number of instances or to other factors.

5.3 Time and Memory Predictions

After having showed that at least the number of classes, the number of instances and the number of classes have an impact on the time of execution of workflows and on the memory required by the trained classifier (java object), from the results of our experiments we want to find some dependencies among these factors and time and memory values. So, on untested datasets, we will be able to predict time and memory values only by checking the structure of the dataset. We have found four mathematical functions that approximate these values with respect to the number of instances of 10 datasets: $\hat{D} = \{$acute-inflammation, blood, statlog-german-credit, titanic, thyroid, bank, twonorm, mushrooms, magic, adult$\}$. These datasets are all binary-class problems and their number of attributes is in the range [3, 21]. We could not fix the number of attributes because otherwise we do not have enough datasets with a different number of instances. The number of instances change among these values: $\{120, 748, 1000, 2201, 3772, 4521, 7400, 8124, 19020, 32561\}$. The mathematical functions are shown in the following figures:

- Figure 5.1: the time of execution of the workflow (11, Svm) can be approximated with the function: $y = \frac{1}{130} \cdot x$, where x is the number of instances of datasets and y is the time expressed in seconds. Its time of execution depends at most on the training time of the Svm classifier.
- Figure 5.2: the time of execution of the workflow (11, Bagging LWL) can be approximated with the function: $y = \frac{1}{2} \cdot x$, where x is the number of instances of datasets and y is the time expressed in seconds. Its time of execution depends at most on the test time of the Bagging LWL classifier.
- Figure 5.3: the memory usage of the workflow (11, NBTree) can be approximated with the function: $y = \log \frac{x}{250}$, where x is the number of instances of

datasets and y is the memory expressed in Mb.

- Figure 5.4: the memory usage of the workflow (11, Bagging LWL) can be approximated with the function: $y = \frac{1}{2000} \cdot x$, where x is the number of instances of datasets and y is the memory expressed in Mb.

So, we can use these mathematical functions in order to predict time and memory performances with respect to the number of instances of each untested dataset that is a binary-class problem and that has a number of attributes contained in the range [3, 21].

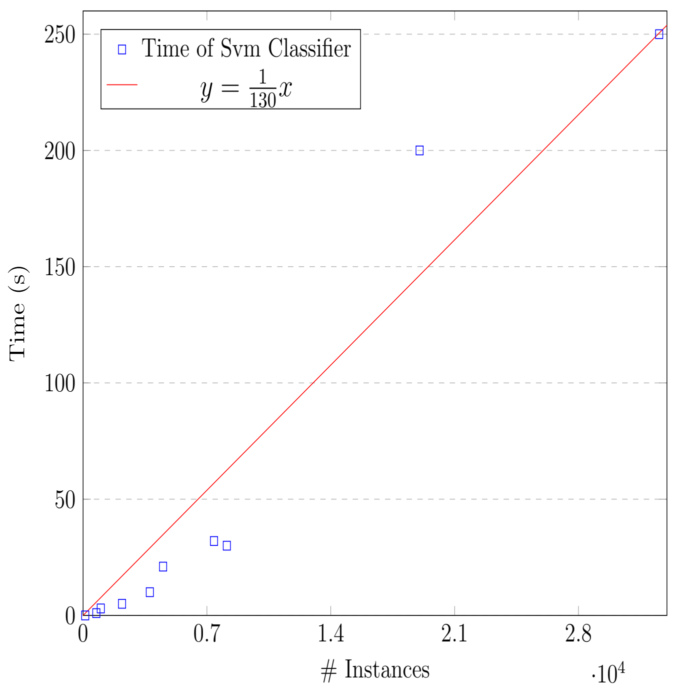


Figure 5.1: Time of execution of the workflow (p_{11}, Svm) on the datasets $\hat{D}$

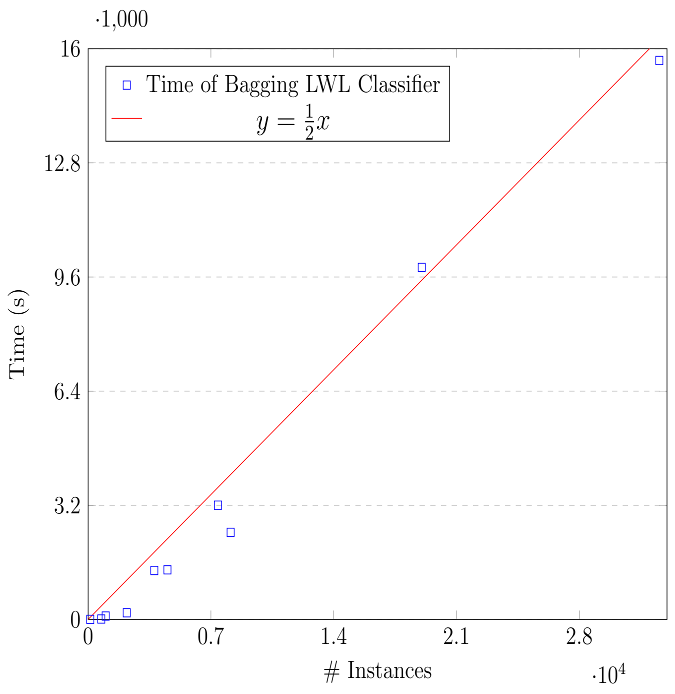


Figure 5.2: Time of execution of the workflow (p_{11}, Baging LWL) on the datasets $\hat{D}$

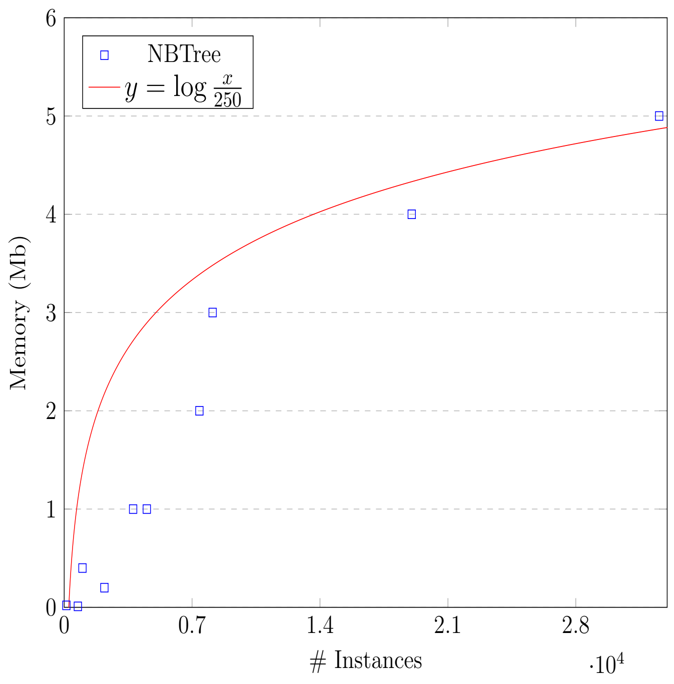


Figure 5.3: Memory usage of the workflow (p_{11}, NBTree) on the datasets $\hat{D}$

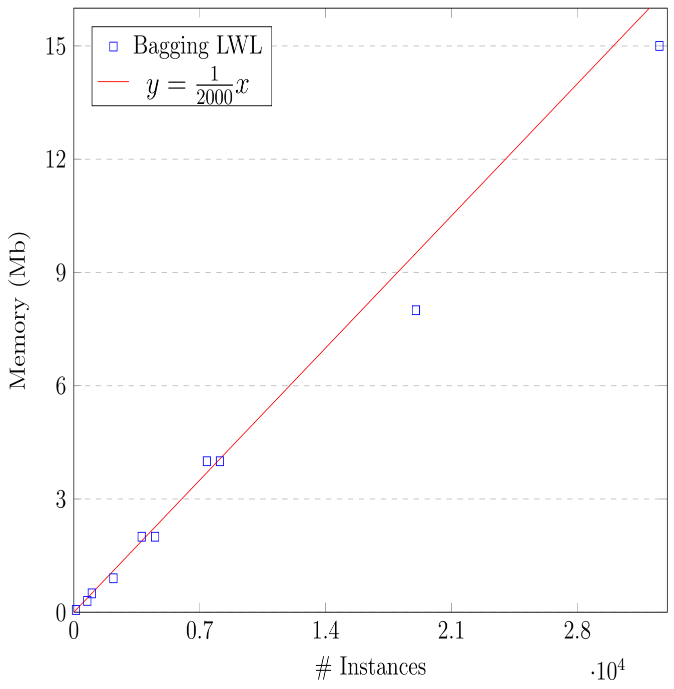


Figure 5.4: Memory usage of the workflow (p_{11}, Baging LWL) on the datasets $\hat{D}$

Conclusions

The research done for this project is the theoretic work that there is behind ROCKFlows, the system that we are implementing that aims to advice non-expert users the best workflows that can solve their machine learning problems. In order to start the project, we focused on the supervised classification problems, by considering 12 pre-processing techniques and 65 classifiers implemented on the Weka platform. At first, we have defined a strategy based on statistical hypothesis tests that groups together workflows that are not significantly different. Then, we have used this strategy in order to answer the three questions we wondered in Section 2.1. We have proved that if we want to compare several classifiers among them, we have to take into account three factors: the choice of the evaluation method, the choice of the pre-processing and the treatment of missing values. Each of these factors have a significant impact on the results obtained by the comparison of the classifier. Moreover, we have proposed a strategy based on data patterns in order to predict the workflows that, allegedly, will have the best accuracy on untested datasets, without doing the evaluation phase and without comparing all the possible workflows. For example, from our experiments we have found one data pattern where the workflow (11, Svm) has an average rank=1, both in case of 4-Fold and 10-Fold cross-validation. This result may suggest that we can expect the workflow (11, Svm) to reach the best accuracy on untested datasets that match the same data pattern, or at least it would be a good candidate workflow to predict. The future related research may concern to study more in depth the data patterns of datasets, in order to find workflows that are supposed to get the best results in terms of accuracy. Finally, we have studied how the time of execution of workflows and the memory required by the trained classifier vary with respect to the structure of datasets. We have proved that at least three factors affect time and memory values: the number of classes, the number of attributes and the number of instances of datasets. Then, we have searched if there exists some dependency among these factors and time and memory performances. For example, from 10 datasets of our experiments, we have found four mathematical functions that approximate time and memory performances of four workflows with respect to the number of instances of datasets. So, we can predict time and memory performances of four workflows on untested datasets only by looking at its number of instances. The future related research may concern to study more in depth the dependecies between the structure of datasets and time and memory performances.

The research presented in this report is not exhaustive, it is limited by the number of datasets, by the number of pre-processing techniques and by the number of classifiers tested. Moreover, it is limited to the Weka platform, which can affect the implementation of classifiers. This work may and should be extended into this direction, in order to fully understand the relations among the nature of datasets and accuracy, time and memory peformances of the execution of workflows. To conclude, a similar work might and should be done on other machine learning problems, such as clustering, regression and anomaly detection.

Appendices

Appendix A: RockFLOWS

The purpose of the ROCKFlows project (Request your Own Convenient Knowledge Flows) is to lay the foundations of a software platform that helps the construction of machine learning workflows. Beyond the predictions of the most suitable workflows showed into this report, the system also let data scientists build their desired workflows. We have based our project on the Software Product Line (SPL) engineering method, which separates two processes: domain engineering for defining the variability of the product line and application engineering for deriving product line applications [20]. In the case of ROCKFlows, the products are the machine learning workflows, which can be generated as a product line.

A.1 Software Product Line

SPL development starts with a first phase of domain engineering, where the variability of the domain is captured into a feature model, which is a tree of features. The semantics of a feature model is the set of feature configurations that it permits. It can be expressed with mathematical logic, by means of propositional formulas, where each feature is represented as a boolean variable. Feature models can express relationships among parent and child features and among cross-tree features. Table A.1 reports the basic primitives of the parent-child relationships, while Table A.2 reports the basic primitives of the cross-tree constraints.

A.2 Separation of Concerns

The feature model of ROCKFlows is based on the principle of the separation of concerns, that is, each element responsible for the creation of workflows is defined in a separate feature. The main features of ROCKFlows are the following:

- Input Dataset: specifies the dataset properties of the input dataset
- Pre-processing: specifies the pre-processing techniques to apply to the input dataset
- Processing: specifies the machine learning algorithms usable into the system
- Functional Objectives: specifies what users want to do on their input dataset. For example, if they have to solve a supervised classification problem or a clustering problem.

Parent-Child Primitive	**Semantic**
f_1 optional sub-feature of f	$f_1 \Rightarrow f$
f_1 mandatory sub-feature of f	$f_1 \Leftrightarrow f$
$f_1, ..., f_n$ alternative sub-features of f (XOR)	$(f_1 \vee \cdots \vee f_n \Leftrightarrow f) \wedge \bigwedge_{i<j} \neg(f_i \wedge f_j)$
$f_1, ..., f_n$ or sub-features of f (OR)	$f_1 \vee \cdots \vee f_n \Leftrightarrow f$

Table A.1: Parent-Child primitives of Feature Model

Cross-Tree Primitive	**Semantic**
f_1 excludes f_2	$\neg(f_1 \wedge f_2)$
f_1 requires f_2	$f_1 \Rightarrow f_2$

Table A.2: Cross-Tree primitives of Feature Model

- Performances: specifies the expected performances by users. For example, users may have a constraint regarding the time of execution of workflows

These features have dependencies among them, managed by cross-tree constraints. For example, the functional objectives affect the choice of the machine learning algorithm. If users have a clustering problem to solve, they can choose only a clustering algorithm, they cannot choose an algorithm of classification.

Cross-tree constraints are foundamental to define the set of the possible workflows that is possible to build from the feature model. Hence, we need to be careful to add only the constraints that exclude the construction of workflows that we are sure that they are not valid. The hard part of the formalization of features and cross-feature constraints is to deal with the not knowledge. Table A.3 reports an example where we know the accuracy performances of three algorithm on a specific data pattern. If users have a dataset that matches $Pattern1$ and they want only algorithms with high accuracies, we can discard the algorithm B from the choice of the algorithms, because we know for sure that it will have low accuracies on $Pattern1$. So the feature model will contain a constraint like: $Pattern1 \wedge HighAccuracy \Rightarrow \neg(B)$. But what can we say about the algorithm C? We do not have enough information about its accuracy performances on the data pattern $Pattern1$, so we cannot set any constraint that removes C from the choice of the algorithms.

Data Pattern	Algorithm	Accuracy
Pattern 1	A	High
Pattern 1	B	Low
Pattern 1	C	???

Table A.3: Example of not knowledge

A.3 ROCKFlows: Simplified Overview

In its current version, ROCKFlows has roughly 300 features and more than 5000 constraints. Because of this, it is not possible to describe here all the work done. This section shows a simplified overview on how the system ROCKFlows works.

A.3.1 Feature Model

The feature model of ROCKFlows has been defined on the online platform SPLOT research [19], which allow to define custom feature models and to test product configurations. Cross-tree constraints are defined as CNF clauses:

- f_1 requires f_2: $\neg f_1 \vee f_2$
- f_1 excludes f_2: $\neg f_1 \vee \neg f_2$

Figure A.1 shows the simplified feature diagram of ROCKFlows and two cross-feature constraints. The feature diagram is simply a visual notation of the feature model. In the diagram, the root feature $Workflow$ contains three features: *Dataset* (mandatory), $Pre - Processing$ (optional) and $Classifier$ (mandatory). Since $Pre - Processing$ is an optional feature, during the construction of the workflow (product configuration) we can decide whether to select it or not, depending on whether we want to apply some pre-processing technique to the input dataset or not. Each of these three features is a tree which contains more sub-features. For example, the feature $Classifier$ contains an XOR sub-feature, which in turn contains the *Svm* and $J48$ sub-features. For the XOR feature, SPLOT will allow us to select only one of the two classifiers during the construction of the workflow. Contrarily, the feature *Attribute Type* contains an OR sub-feature, which in turn contains the $Nominal$ and $Numeric$ sub-features. For the OR feature, SPLOT will allow us to select both of the featues during the construction of the workflow, for example if the dataset contains both numeric and nominal attributes. Once the feature model has been formalized, SPLOT let it export into the SXFM format, which is a pseudo xml file that defines the features and the cross-tree constraints.

From the feature model we can create several products, which in the case of ROCKFlows are machine learning workflows. The product configuration is simply a selection of a subset of features that belongs to the feature model. By selecting one feature, automatically the model may select other features or disable them, depending on the cross-tree constraints. An example is shown in Figure A.2a. If we select the feature *Nominal* (that is, the input dataset has nominal attributes), then the model automatically disable the feature *Svm*, because the Svm algorithm cannot work on nominal attributes. Another example is shown in Figure A.2b. If we select the algorithm *Svm* and the feature *Missing Value* (that is, the input dataset has missing values), then the model automatically select the pre-processing *Replace Missing Values*, because the Svm algorithm cannot work on missing values. This example is also a product (workflow), because all the features have been selected or disabled. The workflow can be summarized as follow:

- The input dataset contains missing values and only numeric attributes
- Apply pre-processing *Replace Missing Values* to the input dataset
- Apply the Svm classifier to the pre-processed dataset

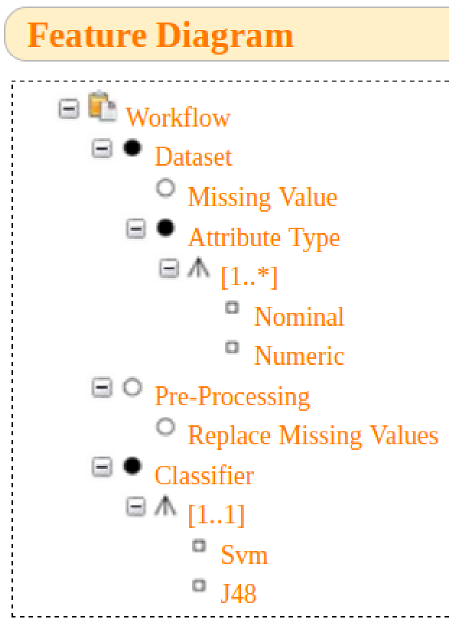
The product configuration does not allow to create workflows that does not satisfy the cross-tree constraints. For example, within the feature model showed into Figure A.1, it is not possible to select the following workflow:

- The input dataset does not contain any missing value, it has only nominal attributes
- Do not apply any pre-processing to the input dataset
- Apply the Svm classifier to the input dataset

This workflow is not valid because of the constraint: $\neg Nominal \wedge \neg Svm$, which says: the feature *Svm* excludes the feature $Nominal$, and viceversa.

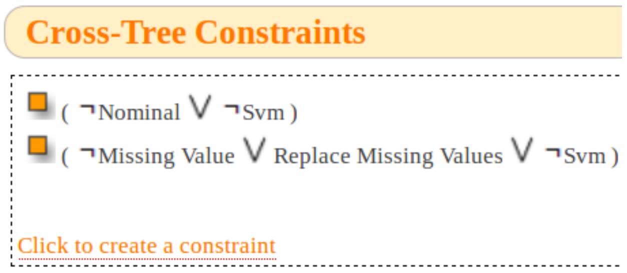
A.3.2 Code Generation Area

Once users have defined their workflow, SPLOT generates an xml file that contains the information about the product configuration, that is, which features have



Feature Diagram

- Workflow
  - ● Dataset
    - ○ Missing Value
    - ● Attribute Type
      - [1..*]
        - Nominal
        - Numeric
  - ○ Pre-Processing
    - ○ Replace Missing Values
  - ● Classifier
    - [1..1]
      - Svm
      - J48



Cross-Tree Constraints

(¬Nominal ∨ ¬Svm)

(¬Missing Value ∨ Replace Missing Values ∨ ¬Svm)

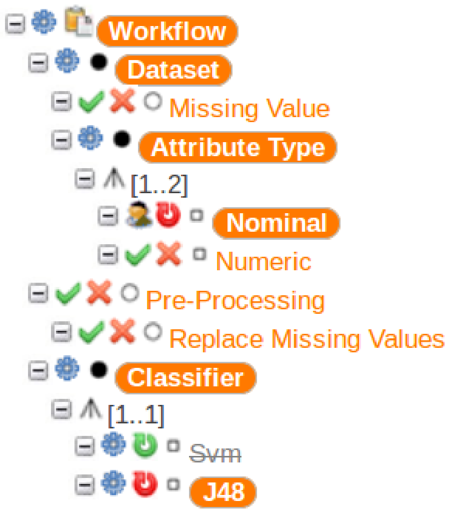
Click to create a constraint

(a) (b)

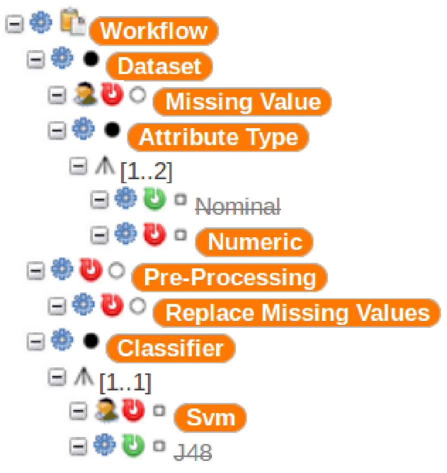
Figure A.1:
(a): Simplified Feature Diagram of ROCKFlows
(b): 2 Cross-Feature Constraints
black dot = mandatory, white dot = optional, [1..*] = or, [1..1] = xor

been selected and which are not. For example, the xml configuration of the workflow shown in Figure A.2b is reported in the following frame:

```
<configuration model="Workflow">
        <feature id="_r">
                <name>Workflow</name>
                <type>templateModel</type>
                <value>1</value>
                <decisionType>propagated</decisionType>
                <decisionStep>1</decisionStep>
        </feature>
        <feature id="_r_1">
                <name>Dataset</name>
                <type>mandatory</type>
                <value>1</value>
                <decisionType>propagated</decisionType>
                <decisionStep>1</decisionStep>
        </feature>
```



(a)



(b)

Figure A.2:
(a): The selection of Missing Values Disables Svm
(b): Missing Values and Svm require Replace Missing Values

```
<feature id="_r_1_2">
        <name>Missing Value</name>
        <type>optional</type>
        <value>1</value>
        <decisionType>manual</decisionType>
        <decisionStep>2</decisionStep>
</feature>
<feature id="_r_1_5">
        <name>Attribute Type</name>
        <type>mandatory</type>
        <value>1</value>
        <decisionType>propagated</decisionType>
        <decisionStep>1</decisionStep>
</feature>
<feature id="_r_1_5_6_7">
        <name>Nominal</name>
        <type>grouped</type>
        <value>0</value>
        <decisionType>propagated</decisionType>
        <decisionStep>4</decisionStep>
</feature>
<feature id="_r_1_5_6_8">
```

```
                <name>Numeric</name>
                <type>grouped</type>
                <value>1</value>
                <decisionType>manual</decisionType>
                <decisionStep>3</decisionStep>
        </feature>
        <feature id="_r_9">
                <name>Pre-Processing</name>
                <type>optional</type>
                <value>1</value>
                <decisionType>propagated</decisionType>
                <decisionStep>5</decisionStep>
        </feature>
        <feature id="_r_9_10">
                <name>Replace Missing Values</name>
                <type>optional</type>
                <value>1</value>
                <decisionType>manual</decisionType>
                <decisionStep>5</decisionStep>
        </feature>
        <feature id="_r_11">
                <name>Classifier</name>
                <type>mandatory</type>
                <value>1</value>
                <decisionType>propagated</decisionType>
                <decisionStep>1</decisionStep>
        </feature>
        <feature id="_r_11_12_13">
                <name>Svm</name>
                <type>grouped</type>
                <value>1</value>
                <decisionType>manual</decisionType>
                <decisionStep>4</decisionStep>
        </feature>
        <feature id="_r_11_12_14">
                <name>J48</name>
                <type>grouped</type>
                <value>0</value>
                <decisionType>propagated</decisionType>
                <decisionStep>4</decisionStep>
        </feature>
</configuration>
```

Each feature is identified by the $< name >$ field. The $< value >$ field contains two values: 0 if the feature is selected, 1 otherwise. In order to generate automatically the java source code, the code generator of ROCKFlows at first reads the xml file of the configuration, and each feature that has $< value >$=1 is retrieved. However, the xml structure used by SPLOT does not allow to recognize easily which the important features are, because the parent-child relationship are lost into the xml file, that is, each feature is at the same xml node level. For example, to the purpose of building the source code of a workflow, it is not useful to know that the feature *Dataset* is selected, it will be selected always in any workflow, because it is a mandatory feature, so it will have always $< value >$=1. To this purpose, the code generator contains a filter that selects only the relevant features with $< value >$=1. In this example, the relevant features are: *Missing Value*, *Numeric*, *Replace Missing Values*, *Svm*. Then, the code generator contains a mapper, which retrieves for each relevant feature three things:

- The dependencies to import into the main java class
- The piece of code necessary to write into the java class in order to execute the feature
- The rank value, which is used to sort the features.

So, in the end, the java class is built by adding at first all the dependencies required by the relevant features, then by adding the pieces of code of the sorted features. In this workflow example, at first it is written the code necessary to load the input dataset. The input dataset is passed as parameter to the program, along with the index of the class attribute. The dataset is an external file in .arff or csv format. Then, the pre-processing *Replace Missing Values* is applied to it. In this ROCKFlow's overview, the input dataset is split in one training set and in one test set. Then, the Svm classifier is trained and tested on the respected datasets, in order to calculate its accuracy. The java main class created from the example workflow showed above is the following frame:

```
package ROCKFlows.Overview;

import weka.classifiers.Classifier;
import weka.core.Instances;
import weka.core.converters.ConverterUtils;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.ReplaceMissingValues;
import weka.classifiers.functions.LibSVM;
import weka.core.Instance;
```

```
public class MainExperiment {

    public static void main(String[] args) throws Exception {

        //read parameters
        String path = "";
        int index = -1;
        int numParameters = args.length;
        for (int iii = 0; iii < numParameters; iii++) {
            switch (args[iii]) {
                case "-path": {
                    path = args[++iii];
                    break;
                }
                case "-index": {
                    index = Integer.parseInt(args[++iii]);
                    break;
                }
            }
        }
        //read input dataset
        Instances dataSet = ConverterUtils.DataSource.read(path);
        dataSet.setClassIndex(index);
        //apply pre-processing
        Filter current = new ReplaceMissingValues();
        current.setInputFormat(dataSet);
        Instances preProcessed = Filter.useFilter(dataSet, current);
        //create training and test set
        Instances trainingSet = preProcessed.trainCV(2, 0);
        Instances testSet = preProcessed.testCV(2, 0);
        //Svm classifier
        Classifier proc = new LibSVM();
        //build classifier
        proc.buildClassifier(trainingSet);
        //test classifier
        int correct = 0;
        for(int i = 0; i < testSet.numInstances(); i++){
            Instance currentToClassify = testSet.instance(i);
            double actualValue = currentToClassify.value(
                currentToClassify.classAttribute());
            double predictedValue =
                proc.classifyInstance(currentToClassify);
            if (predictedValue == actualValue) {
                correct++;
            }
        }
        //compute accuracy
        double accuracy = correct / testSet.numInstances();
        //output
        System.out.println("Accuracy = " + accuracy);
    }
```

A.3.3 Graphic User Interface

The current version of ROCKFlows is available online at the url: `http://rockflows.i3s.unice.fr/#/`. It presents a user interface that asks users questions in plain english, and users can answer by selecting the answers proposed by the system. The user interface hides users the feature model on which ROCKFlows is based. The user interface is a web-service that uses the SPLAR libraries [22], which manages the constraints of the feature model. When users select some features, the web-service changes automatically the available options also on the user interface, depending on the constraints of the feature model.

Bibliography

[1] M.F.Delgado: *Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?* 15(Oct):3133−3181, 2014

[2] Wolpert, David (1996), *The Lack of A Priori Distinctions between Learning Algorithms*, Neural Computation, pp. 1341-1390.

[3] David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures.* CRC Press, 2006.

[4] Jean Carletta. *Assessing agreement on classification tasks: The kappa statistic.* Computational Linguistics, 22(2):249–254, 1996.

[5] `http://mlr.cs.umass.edu/ml/datasets.html`

[6] `http://www.cs.waikato.ac.nz/ml/weka/`

[7] `http://www.r-project.org`

[8] `http://www.mathworks.es/products/neural-network`

[9] `http://persoal.citius.usc.es/manuel.fernandez.delgado/papers/jmlr/data.tar.gz`

[10] J. Demˇsar, B. Zupan, G. Leban, and T. Cur: *Orange: From experimental machine learning to interactive data mining* in PKDD, ser. Lecture Notes in Computer Science, J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, Eds., vol. 3202. Springer, 2004, pp. 537–539.

[11] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kotter, T. Meinl, P. Ohl, C. Sieb, K.Thiel, and B. Wiswedel: *KNIME: The Konstanz Information Miner* in GfKl, ser. Studies in Classification, Data Analysis, and Knowledge Organization, C. Preisach, H. Burkhardt, L. Schmidt-Thieme, and R. Decker, Eds. Springer, 2007, pp. 319–326.

[12] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler: *Yale: Rapid prototyping for complex data mining tasks* in KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, L. Ungar, M. Craven, D. Gunopulos, and T. EliassiRad, Eds. New York, NY, USA: ACM, August 2006, pp. 935–940.

[13] Tim Kraska, Ameet Talwalkar, John Duchi, Rean Griffith, Michael J. Franklin, Michael Jordan: *MLbase: A Distributed Machine-learning System*

[14] Janez Kranjc, Vid Podpe and Nada Lavra: *ClowdFlows: A Cloud Based Scientific Workflow Platform*

[15] Janez Kranjc, Vid Podpe, Nada Lavra: *Real-TimeDataAnalysisinClowdFlows*, 2013 IEEE International Conference on Big Data

[16] `https://azure.microsoft.com/fr-fr/services/machine-learning/?WT.srch=1&WT.mc_ID=SEM_fJvxYgAf`

[17] `http://www.ibm.com/smarterplanet/us/en/ibmwatson/what-is-watson.html`

[18] `http://aws.amazon.com/fr/machine-learning/`

[19] `www.splot-research.org`

[20] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer-Verlag, 2005.

[21] `https://maven.apache.org/`

[22] M. M. Mendonça, M. Branco, D. Cowan, and M. Mendonca. *S.P.L.O.T.: software product lines online tools.* In OOPSLA, pages 761–762. ACM Press, 2009.