Intelligent Assistance for Data Pre-processing

Besim Bilalli[a,*], Alberto Abelló[a], Tomàs Aluja-Banet[a], Robert Wrembel[b]

[a] *Universitat Politécnica de Catalunya, BarcelonaTech, Barcelona, Spain*
[b] *Poznan University of Technology, Poznan, Poland*

Abstract

A data mining algorithm may perform differently on datasets with different characteristics, e.g., it might perform better on a dataset with continuous attributes rather than with categorical attributes, or the other way around. Typically, a dataset needs to be pre-processed before being mined. Taking into account all the possible pre-processing operators, there exists a staggeringly large number of alternatives. As a consequence, non-experienced users become overwhelmed with pre-processing alternatives. In this paper, we show that the problem can be addressed by automating the pre-processing with the support of meta-learning. To this end, we analyzed a wide range of data pre-processing techniques and a set of classification algorithms. For each classification algorithm that we consider and a given dataset, we are able to automatically suggest the transformations that improve the quality of the results of the algorithm on the dataset. Our approach will help non-expert users to more effectively identify the transformations appropriate to their applications, and hence to achieve improved results.

Keywords: Data pre-processing, Data mining, Meta-learning

1. Introduction

Recently, more and more non-experts are using data mining tools to perform data analysis. These users require off the shelf solutions that will assist them throughout the process. The process itself, a.k.a. knowledge discovery, consists of several steps, such as *data selection*, *data pre-processing*, *data mining*, and *evaluation* or *interpretation* [1], see Figure 1. One of the most important steps of this process is the data pre-processing step. Data pre-processing is so important that usually 50-80% of analysis time is spent on it [2]. The reason for this, is that, a properly prepared/pre-processed dataset yields better results. One can apply the best learning algorithm, but if the data is not well-prepared, the algorithm may perform poorly (e.g., bad predictive accuracy) [3].

Since data pre-processing is so important and typically it is performed by a non expert-user, there is a need to support the user by means of automating the process as much as possible.

In this paper, we propose a solution to this problem. We aim at assisting the user by recommending transformations i.e., pre-processing operators, that will ultimately improve the result of the analysis, that usually happens to be a classification task. In order to do that, we make use of the concept of *meta-learning*, which consists of two phases, such as *learning* and *predicting*. For a given dataset and a selected classification algorithm we are able to suggest transformations that once applied yield an improved classification performance (e.g., predictive accuracy).

Contributions. The main contributions of this paper can be summarized as follows:

- We leverage ideas from meta-learning to present a technique for ranking pre-processing operators depending on their impact on the final result of data analysis.
- We show the benefits of our approach by implementing a tool that is capable of automatically recommending pre-processing operators to the user.
- We show experiments that demonstrate the effectiveness and quality of our approach.

The rest of the paper is organized as follows: the related work is discussed in Section 2. An overview of

*Corresponding author
Email addresses: bbilalli@essi.upc.edu (Besim Bilalli), aabello@essi.upc.edu (Alberto Abelló), tomas.aluja@upc.edu (Tomàs Aluja-Banet), robert.wrembel@cs.put.poznan.pl (Robert Wrembel)

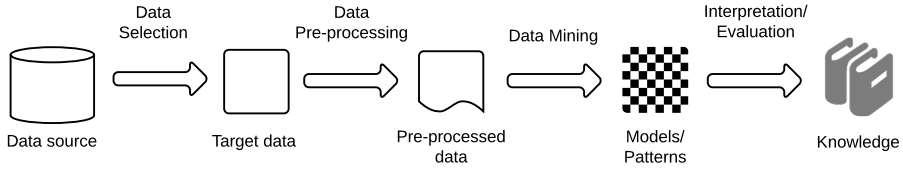

Figure 1: Data Analysis/Knowledge Discovery Process, adapted from [1]

data pre-processing, together with its benefits is given in Section 3. Our proposed solution is formally defined in Section 4. A brief look at the materialization of our proposed approach in terms of a prototype solution is given in Section 5. The results of the experimental evaluations are reported in Section 6. Finally, Section 7 summarizes our work and outlines some future work.

2. Related Work

A lot of research has been conducted in terms of providing user support for different steps of data analysis. The focus however, has usually been on the data mining step, and data pre-processing has generally been overlooked.

Weka [4], an open source tool for data mining, allows users to apply pre-processing algorithms but it does not provide assistance in terms of which one to apply. However, since different data mining algorithms have different requirements regarding the dataset, some pre-processing is applied by default inside some of the algorithms. This pre-processing is usually a simple transformation that does not aim at improving the performance of an algorithm but it aims at transforming the dataset so that it can fit to the data mining algorithm. Furthermore, note that only few algorithm implementations in Weka contain these kind of on the fly transformations.

In AutoWeka [5], user assistance is provided, however, only with regard to the data mining step. That is, the system suggests the best learning algorithm to use with it's proper parametrisation without considering the pre-processing step. Hence, the user needs to deal with the pre-processing on his own.

In AmazonML[1], the system recommends an initial recipe for pre-processing, which is prepared taking into consideration the attributes of the dataset, including the response (i.e., the attribute to be predicted). The recipes provided, however, are pre-formatted instructions for common transformations and do not guarantee improvements of the final result. Hence, they are recommended only because they are applicable to the particular dataset, whereas we are interested in performing pre-processing with the only goal of improving the final result of the analysis.

eIDA [6], which is a product of the eLico[2] project, aims at autonomously constructing workflows that are combinations of pre-processing and data mining algorithms. In order to do that, the problem of workflow construction is viewed as a planning problem, in which a plan must be built consisting of operators that transform the initial data into models or predictions. In order to find the plans, an exhaustive combination of all applicable transformations with all applicable algorithms is performed. Taking into consideration the number of algorithms (e.g., hundreds in RapidMiner[3] – the project is built on top of RapidMiner), the search space of the problem is unfeasible to compute, hence, the optimal solution may not be found. Moreover, in this approach, independent support, exclusively for pre-processing is not provided. As a matter of fact, a *take it all, or leave it* solution is given. In contrast, we focus only on pre-processing, which not only reduces the search space but at the same time allows independent support, hence, the data mining algorithm can be chosen at will.

There exist some other systems [7, 8, 9], however, they also focus on providing support for the data mining step only.

3. Overview on Data Pre-processing

In this section we give a general overview of the pre-processing step in data analysis by first explaining the different existing pre-processing operators/algorithms. Next, we examine and discuss the impact of data pre-processing operators on the final result of data analysis.

[1] https://aws.amazon.com/machine-learning

[2] http://www.e-lico.eu

[3] http://rapidminer.com

Transformation	Technique	Attributes	Input Type	Output Type
Discretization	Supervised	Local	Continuous	Categorical
Discretization	Unsupervised	Local	Continuous	Categorical
Nominal to Binary	Supervised	Global	Categorical	Continuous
Nominal to Binary	Unsupervised	Local	Categorical	Continuous
Normalization	Unsupervised	Global	Continuous	Continuous
Standardization	Unsupervised	Global	Continuous	Continuous
Replace Miss. Val.	Unsupervised	Global	Continuous	Continuous
Replace Miss. Val.	Unsupervised	Global	Categorical	Categorical
Principal Components	Unsupervised	Global	Continuous	Continuous

Table 1: List of Transformations (Data Pre-processing Operators)

3.1. Data Pre-processing Operators

Traditionally, data mining has been performed on transactional data consisting of continuous attributes. The continuous scale of these attributes has enabled the use of conventional statistical methods, such as logistic regression. However, the advances in computational and storage capacity have enabled the accumulation of ordinal, nominal, and binary data, giving rise to datasets of heterogeneous scales. This has induced: 1) advances in the application of data driven methods (e.g., decision trees, bayesian algorithms, nearest neighbours, support vector machines, etc.) capable of mining large datasets, 2) challenges in transforming attributes of different scales into mathematically feasible and computationally suitable formats [3]. Indeed, each attribute may require special treatment, such as discretization of numerical attributes, rescaling of ordinal attributes and encoding of categorical ones. Hence, different transformations may be required.

For the sake of this paper, we consider the transformations shown in Table 1. They are available in the form of open source packages in different data mining tools (e.g., Weka, RapidMiner). We aimed at selecting some of the most important transformations that cover a wide range of data pre-processing tasks, which are distinguished as *data reduction* and *data projection*. The purpose of *data reduction* is to decrease the size of the dataset (e.g., instances selection or feature selection). The purpose of *data projection* is to alter the representation of the dataset (e.g., mapping continuous values to categories or encoding nominal attributes) [10].

In Table 1, a transformation is described in terms of: 1) the *Technique* it uses, which can be `Supervised` — the algorithm knows the class of each instance and `Unsupervised` — the algorithm is not aware of the class, 2) the *Attributes* it uses, which can be `Global` — applied to all compatible attributes and `Local` — applied to specific compatible attributes, 3) the *Input Type*, which denotes the compatible attribute type for a given transformation, which can be `Continuous` — it represents measurements on some continuous scale, or `Categorical` — it represents information about some categorical or discrete characteristics, 4) the *Output Type*, which denotes the type of the attribute after the transformation and it can similarly be `Continuous` or `Categorical`.

3.2. Impact of Pre-processing

In the following we devise a brief example that reveals the importance of data pre-processing for a prediction (e.g., classification) problem. For more in depth analysis of the impact of pre-processing we refer the reader to [3, 11].

Let us suppose that a user wants to apply the `Logistic` algorithm to the `Automobile`[4] dataset. The summary of `Automobile` is given in Table 2. This dataset specifies *autos* in terms of their various characteristics like *fuel type*, *aspiration*, *num-of-doors*, *engine-size*, etc. The response attribute (i.e., class)

[4] https://archive.ics.uci.edu/ml/support/Automobile

Metadata	Value
Instances	205
Attributes	26
Classes	2
Categorical Atts.	11
Continuous Atts.	15
Miss. Values	59

Table 2: Summary of `Automobile`

Transformation	Attribute	PA
Unsup. Discretiz.	1,9,10,11,12,13	0.81
Unsup. Discretiz.	1,9,10	0.80
Unsup. Discretiz.	All Cont. Atts.	0.75
Sup. Nom. To Bin.	All Cat. Atts.	0.73
Unsup. Normaliz.	All Cont. Atts.	0.71

Table 3: The Impact of Transformations on the `Automobile` Dataset

is *symboling*. *Symboling* is a categorical attribute that indicates the insurance risk rate, and its range is: `-3,-2,-1,0,1,2,3`. Value `3` indicates that the auto is risky, `-3` that it is pretty safe. The problem is to build a model that will predict the insurance risk rate for a new auto.

Now, if `Logistic` is applied to the original non-transformed dataset, a predictive accuracy of `0.71` is obtained with a 10 fold cross-validation. Note that for this run the Weka implementation of `Logistic` with a default parametrization is used. On the other hand, if some pre-processing is first performed on `Automobile` and then the data mining algorithm is applied, the results shown in Table 3 are obtained. In Table 3, the first column denotes the transformation applied, the second denotes the index values of the attributes to which the transformation is applied and the third is the predictive accuracy (PA) obtained after the `Logistic` algorithm is applied on the transformed dataset. Note that for instance, if the transformation `Unsupervised Discretization` (with default parametrization) is applied to attributes {`1,9,10,11,12,13`}, an improvement of 14% is obtained in terms of the predictive accuracy. A non-experienced user would not be aware of that. Hence, a proper recommendation of transformations would ease user's task and at the same time it would improve the final result.

Indeed, to alleviate this problem, in the next section we propose an approach that leverages meta-learning to recommend transformations that ultimately improve the result of the data analysis.

4. Meta-learning for Data Pre-processing

Meta-learning is a general process used for predicting the performance (e.g., predictive accuracy) of an algorithm on a given dataset. It is a method that aims at finding relationships between dataset characteristics and data mining algorithms [12].

However, taking into consideration the above mentioned scenario where a user needs to be provided with some transformations to be applied, we propose to use meta-learning in order to find relationships between transformations and data mining algorithms.

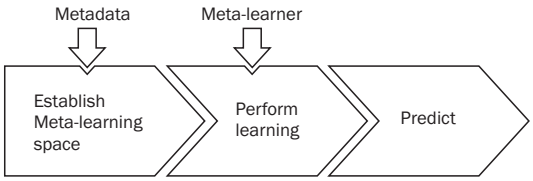

Figure 2: Phases of the Ranking Process

This can be done, since transformations, through the changes they cause in the dataset characteristics, they impact the results of the data mining algorithms. Using meta-learning, we can *learn* this impact and we can **rank** transformations according to their capability of improving the final result of the data mining algorithm.

The process of ranking consists of three phases, see Figure 2. First, a *meta-learning space* is established using metadata. The metadata consist of dataset characteristics along with some performance measures for data mining algorithms on those particular datasets. Then, the *meta-learning phase* generates a model (i.e., predictive meta-model) which defines the area of competence of the data mining algorithm [7]. Finally, when a transformed dataset (i.e., a transformation was applied on the dataset) arrives, the dataset characteristics are extracted and fed to the predictive meta-model, which predicts the performance of the algorithm on the transformed version of the dataset. At this point, we are able to obtain predictions for different transformed datasets (e.g., different transformations applied to the same dataset). By comparing the obtained predictions for the different transformations, we are able to rank the transformations depending on their predicted impact on the given dataset. This concludes the *prediction* phase.

For the sake of concreteness, let us assume that, the user wants to apply Logisitc Regression to a dataset, to deal with a classification problem at hand. Our system, firstly, takes the dataset and applies several transformations to it (i.e., one at a time to avoid a combinatorial problem). As a result, several transformed versions of the dataset are obtained. Next, the system extracts the necessary meta-features (see Section 4.1.1) from all the transformed versions of the dataset and uses them as input to the predictive meta-model which is specifically built for the Logistic Regression algorithm. The meta-model is built by training a meta-learner (e.g., Random Forest or any other regression algorithm) on existing/historical metadata consisting of dataset characteristics and a performance measure (e.g., predictive ac-

curacy) of Logistic Regression on the datasets. This meta-model is used to produce a prediction for each transformed dataset. Informally, this is what the system thinks will be the result (i.e., predictive accuracy) of applying Logistic Regression on each transformed dataset. Thus, these values are used to rank the transformations. That is, if the prediction is higher the higher will stand the transformation – that caused this prediction, in the ranking. The top ranked transformations will be recommended to the user.

Two necessary ingredients for performing the aforementioned process are the **metadata** and the **meta-learner**. In the following we give details on each one of them.

4.1. Metadata.

In our previous work [13], we studied and classified all types of metadata that can be used by systems that intelligently support the user during the process of data analysis. These systems may vary in terms of the methodology they follow (e.g., case based reasoning, planning systems, etc.) [14] and may use different metadata. When it comes to meta-learning however, metadata consist of: 1) dataset characteristics – **meta-features**, and 2) a performance measure for the algorithms considered – **meta-response**. In statistics, the former are called *predictors* and the latter is called *response*.

4.1.1. Meta-features

Meta-features characterize a dataset, and two main classes have been proposed :

- *General measures:* include general information related to the dataset at hand. To a certain extent they are conceived to measure the complexity of the underlying problem. Some of them are: the number of instances, number of attributes, dataset dimensionality, ratio of missing values, etc.

- *Statistical and information-theoretic measures:* describe attribute statistics and class distributions of a dataset sample. They include different summary statistics per attribute like mean, standard deviation, class entropy, etc.

In the literature, other meta-features have also been proposed, such as *Landmarking and model-based* [15, 16] measures. These measures are not classical dataset characteristics, but involve performing simple data mining algorithms on datasets and then use these as values of the features. We do not consider them as dataset characteristics and, since in big data settings, they may introduce significant computational overhead, they do not participate as meta-features in our experiments. Yet, various systems may use various meta-features for the construction of the meta-space. The meta-features we specifically consider are shown in Table 4. These are the set of meta-features extracted in OpenML [17], which is an open science platform developed with the aim of allowing researchers to share their datasets, implementations and experiments (machine learning and data mining) in such a way that they can easily be found and reused by others. OpenML is the biggest source of data and metadata for advancing meta-learning studies.

Column *Type* in Table 4, specifies the type of the meta-feature, and it can be `Continuous` – the meta-feature can be extracted only from datasets that contain attributes of continuous type, `Categorical` – the meta-feature can be extracted only from datasets that contain attributes of categorical type, `Generic` – the meta-feature can be extracted from any dataset, regardless of the types of it's attributes. Furthermore, in Table 4, column *Modifiable* indicates whether the meta-features are modifiable through the transformations we use, shown in Table 1. If meta-features are not modifiable/transformable, we do not consider them, because they remain constant and they do not reflect the impact of transformations.

Yet, note that, in the set of meta-features considered (excluding the non-modifiable ones), not all the meta-features are independent or non-correlated. In order to remedy this, we perform *feature extraction* and then *feature selection* on the original/initial set of meta-features. The method is depicted in Figure 3 and consists of two steps, which are explained next.

Feature Extraction. As previously mentioned, some of the meta-features considered may be very correlated or even redundant (e.g., we calculated the correlation between *Noise to Signal Ratio* and *Equivalent Number of Attributes* on a sample with 570 datasets, and they appeared to be correlated with a Pearson coefficient of 0.85)[5]. As a matter of fact, performing meta-learning on top of correlated meta-features will not lead to a good performance of the meta-learning system. Therefore, in order to remove the dependency and extract the most important information from the meta-features, we first perform a Principal Component Analysis (PCA) [18] to the original set of meta-features. PCA is the predominant linear dimensionality reduction technique, and it

[5] http://www.openml.org/search?type=measure

No	Name	Type	Modifiable
1..2	[Number|Percentage] of Continuous Attributes	Continuous	Yes
3..6	Min[Means|Std|Kurtosis|Skewness] of Continuous Attributes	Continuous	Yes
7..10	Mean[Means|Std|Kurtosis|Skewness] of Continuous Attributes	Continuous	Yes
11..14	Max[Means|Std|Kurtosis|Skewness] of Continuous Attributes	Continuous	Yes
15..17	Quartile [1|2|3] of Means of Continuous Attributes	Continuous	Yes
18..20	Quartile [1|2|3] of Std of Continuous Attributes	Continuous	Yes
21..23	Quartile [1|2|3] of Kurtosis of Continuous Attributes	Continuous	Yes
24..26	Quartile [1|2|3] of Skewness of Continuous Attributes	Continuous	Yes
27	Number of Categorical Attributes	Categorical	Yes
28	Number of Binary Attributes	Categorical	Yes
29	Percentage of Categorical Attributes	Categorical	Yes
30	Percentage of Binary Attributes	Categorical	Yes
31..33	[Min|Mean|Max] Attribute Entropy	Categorical	Yes
34..36	Quartile [1|2|3] Attribute Entropy	Categorical	Yes
37..39	[Min|Mean|Max] Mutual Information	Categorical	Yes
40..42	Quartile [1|2|3] Mutual Information	Categorical	Yes
43	Equivalent Number of Attributes	Categorical	Yes
44	Noise to Signal Ratio	Categorical	Yes
45..48	[Min|Mean|Max|Std] Attribute Distinct Values	Categorical	Yes
49	Number of Instances	Generic	Yes
50	Number of Attributes	Generic	Yes
51	Dimensionality	Generic	Yes
52,53	[Number|Percentage] of Missing Values	Generic	Yes
54,55	[Number|Percentage] of Instances with Missing Values	Generic	Yes
56	Number of Classes	Generic	No
57	Class Entropy	Generic	No
58,59	[Minority|Majority] Class Size	Generic	No
60,61	[Minority|Majority] Class Percentage	Generic	No

Table 4: Meta-features (Dataset Characteristics)

has been widely applied on datasets in all scientific domains, from the social sciences and economics, to biology and chemistry. In short, PCA seeks to reduce the dimension of a large number of directly observable features into a smaller set of indirectly observable features – *latent features*. More precisely, the goals [19] of PCA are, to:

- extract the most important information from the dataset,
- compress the size of the dataset by keeping only this important information,
- explain and simplify the description of the dataset, and
- analyze the structure of observations (instances) and variables.

In order to achieve these goals, PCA computes new features, which are called *principal components*. These

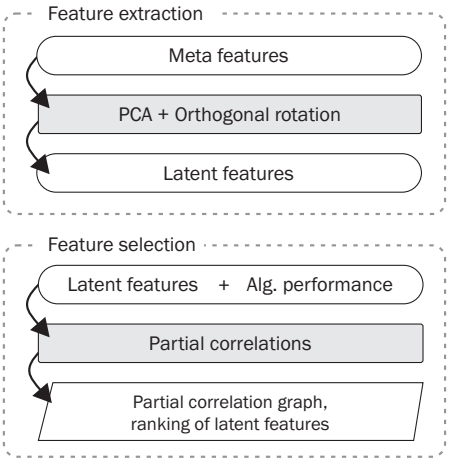

Figure 3: Feature Extraction and Feature Selection

features are obtained as linear combinations of the original features. The first principle component is required to have the largest possible variance to "explain" the largest part of the variance of the dataset (i.e., meta-dataset). Then, the rest of the components are computed under the following constraints: 1) each component needs to be orthogonal to the previous one, and 2) each component needs to have the largest possible variance. The values of these new features are called *factor scores* and are geometrically interpreted as the *projections* of the instances onto the principal components. PCA finds a subspace of size p, where the features are clustered depending on their projections into the factor space. The feature clusters actually form *latent-features*. A set of p components $p \leq n$ is then selected. Each component represents a certain part of the total variance of the dataset. We retain all the components (latent features) that cumulatively represent at least 90% of the total variance.

Next, to facilitate interpretation, after having determined the number of components, we perform a rotation of the components retained. Two types of rotations are mainly used: *orthogonal* – the new axes are required to be orthogonal to each other and *oblique* – the new axes are not required to be orthogonal. Note that, the part of variance explained by the total subspace after rotation is the same as it was before the rotation. In this paper, orthogonal rotation or more precisely VARIMAX [20] method is chosen to perform a *transformation* of the data. VARIMAX method assumes that a simple solution means that each component has a small number of large loadings, and a large number of zero loadings. After the rotation, the set of components – *latent features*, are independent, more interpretable, and they, of course, are defined by their respective meta-features – the ones that are most correlated (a latent feature is calculated as a mean of its corresponding meta-features).

As a final remark, PCA followed by VARIMAX rotation removes the dependency/correlation between features, however, it does not guarantee that all the latent-features retained are equally relevant for predicting the performance of a data mining algorithm. Thus, in order to retain only the most relevant latent features (i.e., the ones that have higher predictive power), in the second step (see Figure 3), we perform latent-feature selection, which is explained next.

Feature selection. The first step in Figure 3, produces a set of candidate latent-features for meta-learning. However, it does not provide a measure on the relevance of the latent-features. The question is: "*How relevant is a latent-feature for predicting the response?*". Indeed, we are interested on the subset of latent-features that are the most relevant for predicting the response. In order to find and retain only the most relevant latent-features, in this step, first, an additional feature (i.e., response) is attached to the set of latent-features. The additional feature can be any of the performance measures (e.g., predictive accuracy, see Section 4.1.2) of the algorithms evaluated over the datasets (the instances of the meta-dataset).

Next, we calculate the partial correlation [21] between the features. This allows us to generate partial correlation graphs that represent the relationships and the strengths of the relationships between features. Our focus is only on the relationships between the latent-features (extracted in the first step) and the response. That is, we measure how relevant are the latent-features for predicting the response. The graph allows us to visualize only the latent-features that have a direct link with the response. Furthermore, these links have a strength which is measured through the significance value (i.e., *p-value*). We consider as significant only the links with a value less than or equal to 0.05 ($p\text{-value} \leq 0.05$). Hence, at the end, only a subset of latent-features is retained. Given the fact that, latent-features are defined through the original meta-features, as explained in the previous section, ultimately this step allows us to retain a subset of the original meta-features. Hence, in the whole process then, we use only the meta-features identified in this step. A schematic representation of a partial correlation graph is shown in Figure 4.

4.1.2. Performance measures (meta-response)

Performance measures are different outputs that can be obtained after the evaluation of data mining algorithms. Since we are dealing with classification prob-

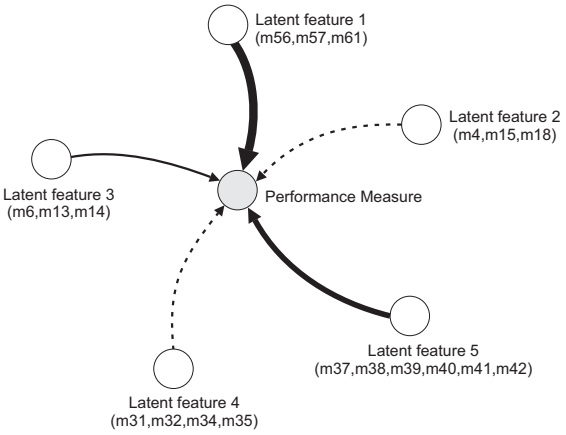

Figure 4: Schematic representation of a partial correlation graph for *Decision Tree* algorithm and *Predictive Accuracy* as a performance measure. The thickness of the edges denotes the significance of the relationship between two nodes. The dashed edges denote negative correlation. m[n] denotes a meta-feature and corresponds to the meta-features in Table 4

Measure	**Formula**
Accuracy	$\frac{TP+TN}{TP+FP+FN+TN}$
Precision	$TP/(TP+FP)$
Recall	$TP/(TP+FN)$
AUC	$\frac{1}{2}\left(\frac{TP}{TP+FN}+\frac{TN}{TN+FP}\right)$

TN - True Negatives, TP - True Positives, FN - False Negatives, FP - False Positives

Table 5: Classification algorithm performance measures

lems, and hence the algorithms we consider are of classification type, the performance is usually measured in terms of *predictive accuracy*, *precision*, *recall* or *area under the roc curve (AUC)*. Moreover, classification algorithms are usually evaluated using 10-fold cross-validation [22].

In Table 5, formulas for calculating these measures are given. Briefly, *Accuracy* is a measure of the overall effectiveness of a classifier. *Precision* is the class agreement of the instance labels with the positive labels given by the classifer. *Recall* measures the effectiveness of a classifier to identify positive labels. Finally, one can think of *AUC* as the classifier's ability to avoid false classification. For more details regarding these measures and how they extend to multi-class classification problems we refer the reader to [23].

4.2. Meta-learner

Having stored an algorithm performance characteristic (see Table 5) and a set of dataset characteristics (see Table 4), the goal is to predict the performance of an algorithm in a transformed dataset. Formally, the problem can be defined as follows. Given algorithm A and a limited number of training data $D = (\mathbf{x}_1, y_1)...(\mathbf{x}_n, y_n)$, the goal is to find a meta learner with optimal/good generalization performance. Generalization performance is estimated by splitting D into disjoint training and validation sets $D^{(i)}_{train}$ and $D^{(i)}_{valid}$. We use leave-one-out validation [22], which splits the training data into n partitions $D^{(1)}_{valid}, ..., D^{(n)}_{valid}$, and sets $D^{(i)}_{train} = D \backslash D^{(i)}_{valid}$ for $i = 1, ..., n$. Note that $\mathbf{x} \in x_1, x_2...x_n$ are the dataset characteristics and y_1 is a chosen measure of the performance of algorithm A run on that particular dataset. Hence, $\mathbf{x}$ and y altogether are the extracted metadata. Since y consists of 4 different performance measures for algorithm runs, we build meta-spaces for each specific measure separately. Then for each meta-space (meta-dataset), we generate meta-models – using a meta-learner.

A few basic criteria were followed for selecting the meta-learner to use. First, the problem in the meta-learning space is of regression type – a number needs to be predicted (i.e., a value in the range of [0, 1]) rather than a class.

The second criterion is that the meta-learner needs to be more *sensitive*. By this we mean that the meta-learner needs to be able to capture even the slight changes that transformations might apply on datasets. This is because we need to predict the impact of the transformations on the data mining results and we need to be able to compare the impacts of different transformations. This comparison needs to be done at a finer granularity. Otherwise, in the worst case, all the transformations may end up having the same impact. For instance, as a first approach we considered simple regression trees [24] as meta-learners, and they suffer from this problem. Their limitation is that they contain a discrete number of leaves, and hence a discrete number of possible predictions.

The third criterion is that the meta-learner should handle missing values. Recall that some dataset characteristics can be calculated on datasets that necessarily contain either continuous or categorical attributes (see Table 4). As a matter of fact, our second trial of using *Logistic Regression* as meta-learner did not give good results. Because *Logistic Regression* cannot han-

dle missing values.

Hence, finally the meta-learner we decided to use is *Random Forest*. *Random Forest* complies with all the above mentioned criteria. It can be used for regression problems. It suffers far less from the discreteness of the leaves, because internally, a lot of trees (i.e., 500 trees) are built at random and at the end averages are taken to be used as predictions. Finally, it performs well when missing values are present.

Thus, we use *Random Forest* to build models for each data mining algorithm or more precisely for each classification algorithm that we consider.

In particular, the classification algorithms that we consider are representative algorithms for all, except two classes of algorithms in Weka. In Weka, the classification algorithms are classified into: *bayes, functions, lazy, rules, trees, meta-methods, and miscellaneous*. We aimed at considering one algorithm for each one of the first five classes, and they are: *Naive Bayes, Logistic, IBk, PART, and J48* respectively. The last two classes were omitted due to the fact that they are more complex and are not commonly used by non experienced users.

5. Solution Prototype

In this section, we discuss the materialization of the approach proposed in Section 4, into a prototype solution. The general architecture of the developed prototype solution is depicted in Figure 5. The solution's main processes, **Learning** and **Recommending**, are implemented independently of each other. Below we give detailed explanations for each one of them.

5.1. Learning phase

In the previous sections we mentioned that in order to build a model (i.e., predictive meta-model), we must firstly establish the meta-space — denoted as *Learning phase* in Figure 5. In our context, the meta-space needs to be constructed out of metadata that can be extracted from datasets and from the executions of classification algorithms on those datasets. As a matter of fact, we needed to fetch hundreds of datasets, extract their characteristics, run different algorithms on them and get different evaluation measures with 10 fold cross validation. Finally, use all of these to feed the *Meta-database*.

In order to do the aforementioned, we first used OpenML to fetch several hundred datasets (i.e., 570). Next, from each dataset we extracted the 55 dataset characteristics – highlighted as modifiable in Table 4, and on each dataset we applied 5 classification algorithms in order to extract the performance measures – shown in Table 5. Then, we performed feature extraction – using PCA followed by VARIMAX on the set of dataset characteristics (meta-features), and feature selection – using the partial correlation graphs, for every classification algorithm and every performance measure considered. Finally, for each classification algorithm and for each performance measure, we obtained a meta-dataset that was fed to the *Meta-database*. In Figure 5, this whole process is represented via the *Metadata Generator* module and was developed in `Java`.

After obtaining the metadata, hence constructing the meta-space, we continued on building the *Models* (or predictive meta-models) using the *Meta-learner* (i.e., Random Forest) we considered. We used the `R` language to construct a model for each one of the algorithms and for each one of the performance measures considered. After that, the models were exported to PMML [25] files, and were next fed to the *Predictor* in the recommending phase.

Note that this process is not specifically tailored for datasets from the OpenML repository, but it can work on any collection of datasets. The models obtained are expected to slightly change from one collection to another.

5.2. Recommending phase

When a user wants to analyze a dataset, he/she selects an algorithm to be used for the analysis and then the system automatically recommends transformations to be applied, such that the final result is improved. In order to do that, the system first, applies different transformations to the dataset through the *Transformation executor* module. Then, the meta-features of the transformed dataset are extracted through the *Meta-feature Extractor* module and they are fed to the *Predictor*, which using the meta-model (i.e., PMML file) corresponding to the classification algorithm selected by the user, predicts the impact of the transformation/s. The gain here is that, the classification algorithms are not applied for real to the transformed datasets – which is a costly proces. Instead, meta-models are used to predict the outputs of the classification algorithms on the transformed datasets. Hence, finally, transformations are ranked according to their impact on the final result – according to whether they improve the final result. The modules of the *Recommending phase* are entirely developed in `Java`.

6. Evaluation

We perform an experimental study of the performance that can be achieved by our approach on vari-

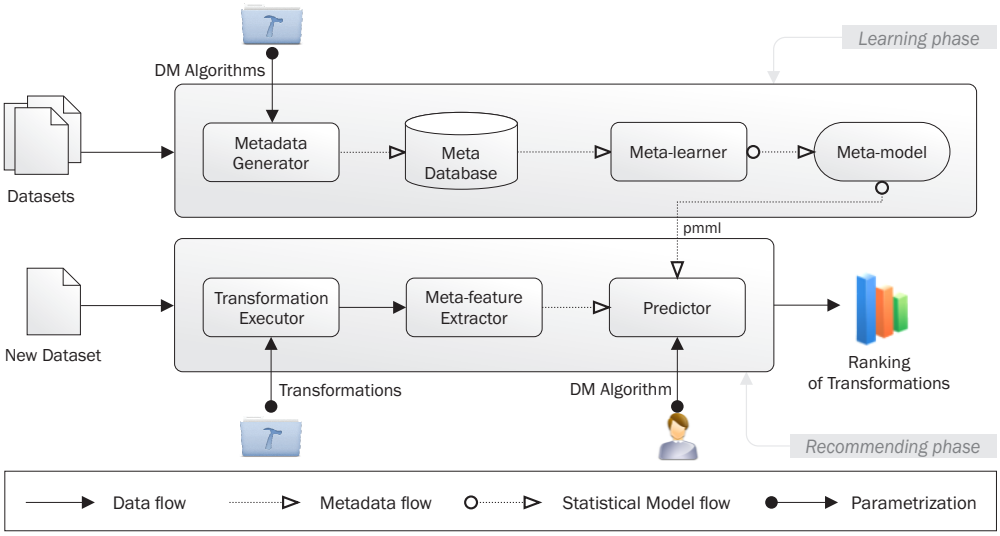

Figure 5: Solution Architecture

ous algorithms and various datasets. After specifying our experimental environment, we evaluate our system's ability to predict the transformations that will improve the final result of the analysis.

6.1. Experimental setup

Recall that when building the meta-learners, we use leave-one-out validation for evaluating them. Likewise, in order to enable a larger number of datasets for performing the experiments, each time we performed the leave-one-out validation, we created a meta-model using the subset of datasets (i.e., withholding the dataset that was left-out). Hence, for each data mining algorithm, we created as many meta-models as datasets considered for the respective algorithm. As a matter of fact, in order to perform experiments for an algorithm, we can use the entire set of datasets for testing, only bearing in mind that for each dataset, in the *Predictor*, we use the meta-model that was built without using that particular dataset.

In this context, an experiment – depicted in Figure 6, is performed in the following way. First, a dataset and a classification algorithm to be used for performing analysis (i.e., classification) on the dataset, is selected. Next, the system finds the **impact** of a **set of transformations** on the final result of the classification.

The **set of transformations**, consists of iteratively applying the transformations shown in Table 1, however each time changing the set of attributes to which the transformation is applied. Note that the transformations which are denoted as `Global` in the table, are applied only once to the set of all compatible attributes (altogether), whereas the transformations, which are denoted as `Local` are applied to: 1) every compatible attribute separately (one by one), and 2) all the set of compatible attributes (altogether). Indeed, transformations are not applied to combinations of attributes and hence there is no 'combinatorial explosion'. However, the user may apply the method several times in iteration and, as such, arrive to a combination that may induce better results. Yet, this depends on the user and his availability to use the method iteratively.

The **impact**, is the effect of transformations to the final result (i.e., predictive accuracy) of the selected algorithm, and it can be, *predicted impact* or *computed impact*.

The *predicted impact* is calculated by applying the set of transformations, as defined above, and subsequently extracting the characteristics of the transformed datasets, to use them as inputs for predicting the performance of the respective algorithm on the transformed datasets.

The *computed impact* is calculated by similarly applying the set of transformations, but then, subsequently applying the respective classification algorithm for real to the transformed datasets, and hence obtaining the real performance (e.g., predictive accuracy) of the classification algorithm on the transformed datasets. In terms of computational complexity, the latter is a costly process, and it is performed only for the sake of evaluating the system.

The experiments were performed on an Intel Core i5

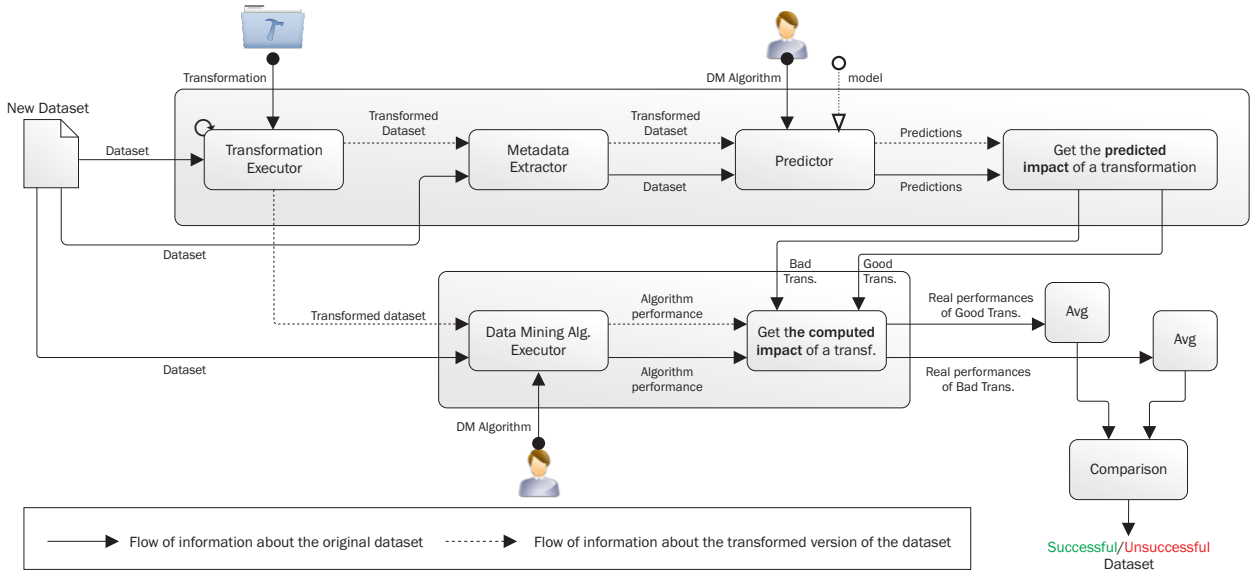

Figure 6: Experimentation Scheme

machine, running at 1.70 GHz with 8 GB of main memory. An experiment for a single algorithm, on average took approximately 4 CPU hours.

On each run, the system internally categorizes a transformation, into one of the following three categories:

- *Good* – an improvement of the final result for the respective algorithm is predicted if the transformation were to be applied, compared to the prediction obtained on the non-transformed version of the dataset,
- *Bad* – a worsening of the final result for the respective algorithm is predicted if the transformation were to be applied, compared to the prediction obtained on the non-transformed version of the dataset,
- *Neutral* – neither improvement nor worsening is predicted if the transformation were to be applied.

The aim of the experiments is roughly to verify whether the categorizations made by the system are true for real (i.e., whether a transformation categorized as *Good*, is Good for real and improves the performance of the algorithm). This, as previously mentioned – though costly, is done by executing the data mining algorithms on the transformed datasets and computing the real impact of the transformations (see Figure 6).

In this context, we mark as *Successful*, the cases (i.e., datasets) on which the *real/computed average improvement* we get from all the transformations categorized as *Good* for a dataset, is greater than the *real/computed average improvement* we get from the transformations that were categorized as *Bad* for the same dataset. That is, the transformations predicted as *Good*, ”beat” on average the transformations predicted as *Bad*. In contrast, we mark as *Unsuccessful*, the cases on which the transformations predicted as *Good* cannot ”beat” on average the transformations predicted as *Bad*.

6.2. Results for Random Forests

In Figure 7, we show the results obtained when *Random Forests* are used as meta-learners. In the figure, we show the comparison between the number of *Successful* cases — the green bar, and the number of *Unsuccessful* cases — the red bar. In addition, the gray bar, denotes the total number of cases (datasets) for which we performed the experiments on each respective algorithm. Furthermore, the bars highlighted with *dashes*, *dots*, and *back slashes* refer to the results for *predictive accuracy*, *precision* and *AUC*, respectively. Notice that the results for *recall* are omitted due to the fact that identical values with *predictive accuracy* were obtained (see weighted recall in Weka[6]).

Observe that, the sum of *Successful* (green) and *Unsuccessful* (red) cases does not coincide with the total number of datasets (gray). This is because, for some

[6]http://weka.sourceforge.net/doc.stable/weka/classifiers/Evaluation

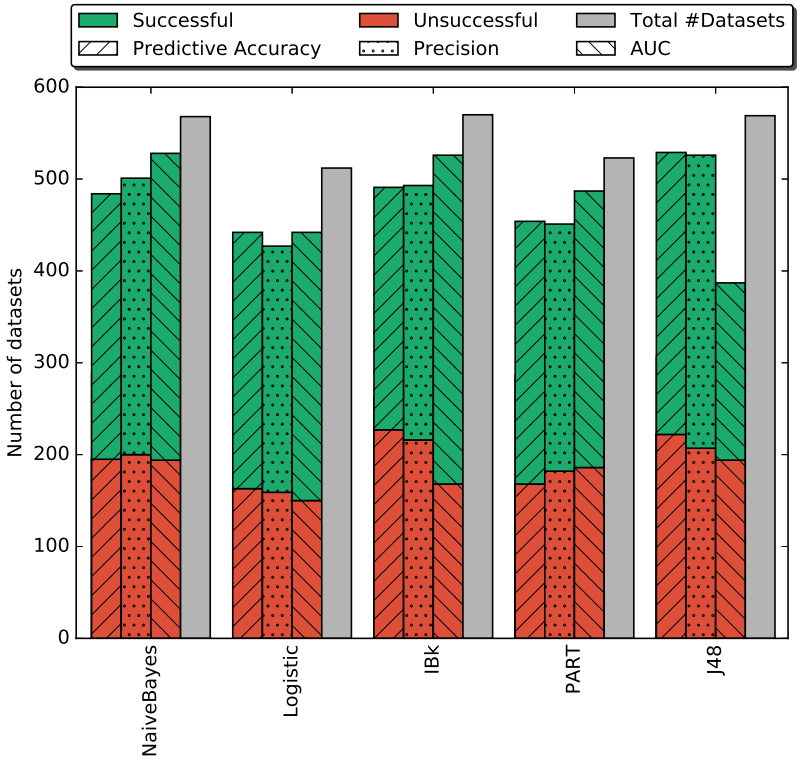

Figure 7: Random Forest Results

datasets we either do not find *Good* transformations (14.3%), or we do not find *Bad* transformations (7.9%). This happens because the datasets already belong to the best or the worst leaves of all the internal trees of the *Random Forests*, hence there can be no transformations that can move them to a better or worse leaf respectively. As a matter of fact, in those particular cases we cannot compare the *Good* versus *Bad*, hence, they do not appear neither as *Successful* nor as *Unsuccessful*.

In order to understand whether the numbers shown in the figure are significant, we performed a binomial distribution test, comparing the number of *Successful* cases to the number of *Successful* + *Unsuccessful* cases with respect to the theoretical probability which is equal to `0.5`. The results obtained are shown in Table 6. The column *p-value* denotes how significant is the difference between the values of *Successful* and the population of *Successful* + *Unsuccessful*. We assume the difference to be significant if the *p-value* is less than or equal to `0.05`. Observe that our method gives significant values for all the algorithms and all the performance measures with the only exception of algorithm *J48* with performance measure *AUC*. Yet, recall that when we performed feature selection using the partial correlation graphs between features and the response, we retained only the features that had a significant relationship within the limits of 0.05, which happens to be too restrictive for *J48* with *AUC* (i.e., some relevant feature is left out). If we increase the threshold to 0.1 (less restrictive) we obtain significant results for this case too. The *p-value* we obtain is 0.0011.

7. Conclusions and Future Work

In this work, we have shown that the daunting problem of data pre-processing can be alleviated by a practical, automated tool. This is made possible through meta-learning which enables predicting the impact of transformations on the final performance of algorithms on the corresponding datasets, and in turn, allows ranking the transformations according to their impact on the final result.

We built a tool that draws on a range of classification algorithms in Weka and makes it easy for non-experts to perform data pre-processing. An extensive evaluation on hundreds of datasets showed that for the set of algorithms considered, even blindly (e.g., users without

	Predictive Accuracy			Precision			AUC		
Algorithm	Suc.	Suc.+Uns.	p-value	Suc.	Suc.+Uns.	p-value	Suc.	Suc.+Uns.	p-value
Naive Bayes	289	484	7.40E-06	301	501	2.41E-06	334	528	3.36E-10
Logistic	279	442	1.09E-08	268	427	4.34E-08	292	442	3.39E-12
IBk	264	491	4.31E-02	277	493	2.59E-03	358	526	0
PART	286	454	9.73E-09	269	451	1.61E-05	301	487	6.42E-08
J48	307	529	8.95E-05	319	526	3.79E-07	193	387	5.00E-01

Table 6: Binomial Significance Test for Random Forests

any prior knowledge in data mining) applying the recommended transformations improves the final result of the algorithms on average. We believe that this can be a handy tool for experienced users as well, because they can discriminate within the recommended transformations and pick the ones that are potentially more suitable for their problem at hand.

As future work, we see potential value in customizing the transformations depending on the class of algorithms (e.g., trees) or even specific algorithms. We also aim at extending the range of the classification algorithms that we have considered so far.

Acknowledgments. This research has been funded by the European Commission through the Erasmus Mundus Joint Doctorate "Information Technologies for Business Intelligence - Doctoral College" (IT4BI-DC). The work of R. Wrembel is supported from the National Science Center grant No. 2015/19/B/ST6/02637.

References

[1] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, From data mining to knowledge discovery in databases, AI Magazine 17 (3) (1996) 1–34.

[2] M. A. Munson, A study on the importance of and time spent on different modeling steps, SIGKDD Explor. Newsl. 13 (2) (2012) 65–71.

[3] S. F. Crone, S. Lessmann, R. Stahlbock, The impact of preprocessing on data mining: An evaluation of classifier sensitivity in direct marketing, European Journal of Operational Research 173 (3) (2006) 781 – 800.

[4] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, et al., The weka data mining software: An update, ACM SIGKDD Explorations Newsletter 11 (1) (2009) 10–18.

[5] C. Thornton, F. Hutter, H. H. Hoos, et al., Auto-weka: Combined selection and hyperparameter optimization of classification algorithms, in: KDD, 2013, pp. 847–855.

[6] J. Kietz, F. Serban, S. Fischer, A. Bernstein, Semantics Inside! But Let's Not Tell the Data Miners: Intelligent Support for Data Mining, in: ESWC, 2014, pp. 706–720.

[7] K. Alexandros, H. Melanie, Model selection via meta-learning: A comparative study, International Journal on Artificial Intelligence Tools 10 (04) (2001) 525–554.

[8] D. Michie, D. J. Spiegelhalter, C. C. Taylor, J. Campbell (Eds.), Machine Learning, Neural and Statistical Classification, Ellis Horwood, 1994.

[9] M. Charest, et al., Bridging the gap between data mining and decision support: A case-based reasoning and ontology approach, Intelligent Data Analysis 12 (2) (2008) 211–236.

[10] D. Pyle, Data Preparation for Data Mining, Morgan Kaufmann, 1999.

[11] T. Dasu, T. Johnson, Exploratory data mining and data cleaning, Vol. 479, John Wiley & Sons, 2003.

[12] P. Brazdil, C. Giraud-Carrier, C. Soares, R. Vilalta, Metalearning: Applications to Data Mining, 1st Edition, Springer Publishing Company, Incorporated, 2008.

[13] B. Bilalli, A. Abelló, T. Aluja-Banet, R. Wrembel, Towards intelligent data analysis: The metadata challenge, in: Proceedings of the International Conference on Internet of Things and Big Data, 2016, pp. 331–338.

[14] F. Serban, J. Vanschoren, J.-U. Kietz, A. Bernstein, A survey of intelligent assistants for data analysis, ACM Computing Surveys 45 (3) (2013) 31:1–31:35.

[15] B. Pfahringer, H. Bensusan, C. G. Giraud-Carrier, Meta-learning by landmarking various learning algorithms, in: Proceedings of the Seventeenth International Conference on Machine Learning, 2000, pp. 743–750.

[16] Y. Peng, P. A. Flach, C. Soares, P. Brazdil, Improved Dataset Characterisation for Meta-learning, 2002, pp. 141–152.

[17] J. Vanschoren, J. N. van Rijn, B. Bischl, L. Torgo, Openml: Networked science in machine learning, SIGKDD Explorations Newsletter 15 (2) (2014) 49–60.

[18] H. Hotelling, Analysis of a complex of statistical variables into principal components, Journal of Educational Psychology 24 (6) (1933) 417–441.

[19] M. Morchid, R. Dufour, P. Bousquet, G. Linarès, J. Torres-Moreno, Feature selection using principal component analysis for massive retweet detection, Pattern Recognition Letters 49 (2014) 33–39.

[20] H. F. Kaiser, The varimax criterion for analytic rotation in factor analysis, Psychometrika 23 (3) (1958) 187–200.

[21] K. Baba, R. Shibata, M. Sibuya, Partial correlation and conditional correlation as measures of conditional independence, Australian and New Zealand Journal of Statistics 46 (4) (2004) 657–664.

[22] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: Proceedings of the International Joint Conference on Artificial Intelligence, 1995, pp. 1137–1143.

[23] M. Sokolova, G. Lapalme, A systematic analysis of performance measures for classification tasks, Information Processing and Management 45 (4) (2009) 427–437.

[24] B. Bilalli, A. Abelló, T. Aluja-Banet, R. Wrembel, Automated data pre-processing via meta-learning, in: Proceedings of the International Conference on Model and Data Engineering, 2016, pp. 194–208.
[25] A. Guazzelli, M. Zeller, W.-C. Lin, G. Williams, Pmml: An open standard for sharing models, The R Journal 1 (2009) 60–65.