Media Engineering and Technology
German University in Cairo
and
Multimedia Analysis and Data Mining Competence Center
German Research Center for Artificial Intelligence (DFKI GmbH)
Kaiserslautern, Germany

Classifiers' Accuracy Prediction based on Data Characterization

Bachelor Thesis

Author:	Sarah Daniel Abdelmessih
Reviewer:	Prof. Dr. Andreas Dengel
Supervisors:	Dr. Faisal Shafait
	Christian Kofler
Submission Date:	27 August, 2010

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowlegement has been made in the text to all other material used

---

Sarah Daniel Abdelmessih
27 August, 2010

Acknowledgement

"The God of heaven, He will prosper us; therefore we His servants will arise and build."
(Nehemia 2:20).

I eagerly proclaim that this bible verse was my driving force that motivated me to pursue all the work done and achieve best results.

First of all I would like to thank my parents for their emotional and financial support and encouragement. Furthermore, I would like to show my gratitude towards **Prof. Slim Abdennadher** and **Prof. Andreas Dengel**, who gave me the opportunity to do my bachelor thesis at the German Research Center for Artificial Intelligence (DFKI).

Special thanks to:

Dr. Faisal Shafait for his supervision and technical support.

Christian Koefler for his supervision, encouragement and technical support.

Matthias Reif and Markus Goldstein for their technical support.

Moheb Elmasry and Nazly Sabbour for reviewing this report, encouragement, and support.

I am also very thankful to Dr. Thomas Kieninger, Brigitte Selzer, and Leivy Michelly Kaul for taking care of all the paper work and administrative issues. In addition, I would like to thank my sister, Suzette, for her support and encouragement.

Abstract

In machine learning, picking the best classifier for a given problem is a challenging task. A recent research field called meta-learning automates this procedure by using a meta-classifier in order to predict the best classifier for a given dataset. Using regression techniques, even a ranking of preferred learning algorithms can be determined. However, all methods are based on a prior extraction of meta-features from datasets. Landmarking is a recent method of computing meta-features, which uses the accuracies of some simple classifiers as characteristics of a dataset. Considered as the first meta-learning step in RapidMiner, a new operator called landmarking *has been developed. Evaluations based on 90 datasets, mainly from the UCI repository, show that the landmarking features from the proposed operator are useful for predicting classifiers' accuracies based on regression. In this work a collaborative user interface, which is integrated with a tool called Classifier Recommender, is introduced. It mainly facilitates the use of machine-learning methods or algorithms over the web and provides users with the functionality of sharing data and experiments.*

Contents

Chapter 1

Introduction

Data Mining is involved with automating the process of searching for patterns and trying to classify data into categories, based on the regularities recognized in the data. *Pattern Recognition* and *Machine Learning* are growing fields in this research area. They have great potential for other research fields and applications, such as face recognition, image analysis, speech recognition, handwritten character recognition, sequence analysis in the field of bioinformatics or cancer diagnosis. However, such systems are data-driven, which means that they have to be adapted according to the environment and the problem. Therefore, most of the pattern recognition and machine learning systems, methods or tools are mostly used by experts. Another problem is the integration of these methods in real-world software systems; it is still a trying experience for software developers and users.
Herein, the Pattern Recognition Engineering (PaREn)[1] project aims at facilitating pattern recognition and machine learning methods for non-expert users. The project target is to develop new tools for methods and algorithms that support automating parameter optimization, model selection, machine learning system construction, rapid testing, validation, and on-line adaptivity. In addition, a powerful tool called *Classifier Recommender*, was developed. This tool suggests learning algorithms for a problem by predicting their accuracies and then evaluating algorithms selected by the user. This tool was also integrated in a web application, called Collaborative User Interface (CUI), as part of this thesis. RapidMiner (Mierswa et al., 2006), which is an open-source system for *data mining*, was chosen as a software platform, to deliver PaREn technologies and tools as open-source.

1.1 Motivation

Predicting the performance of classifiers, ranking learning algorithms, and a dynamic selection of a suitable learning algorithm for a given problem are recent research topics in machine learning (Brazdil et al., 2003; Soares and Brazdil, 2000; Vilalta and Drissi, 2002). As derived from the *No Free Lunch Theorem*, no learning algorithm can be specified as outperforming on the set of all real-world problems (Wolpert and Macready, 1995).

[1] This project is initialized by the *German Research Center for Artificial Intelligence (DFKI)* and funded by the *Federal Ministry of Education and Research*. For more details see http://madm.dfki.de/paren/.

This implies that a learning algorithm has reasonable performance on a set of problems, defined as its *area of expertise* (Vilalta and Drissi, 2002). As illustrated in Figure 1.1, each algorithm has its own set of problems on which it performs well, however, some algorithms share a subset of their area of expertise.

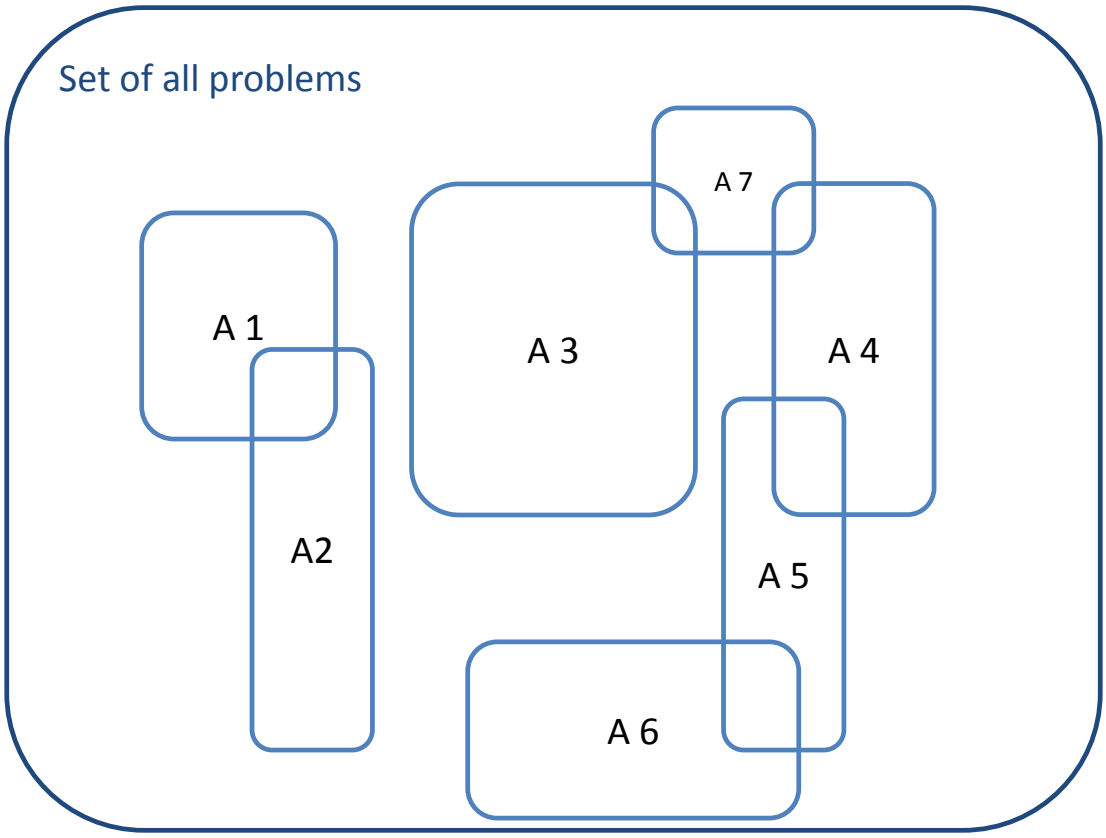


Figure 1.1: A dummy example of the area of expertise of algorithm A 1 to A 7 defined in the set of all problems. Each algorithm has a different area of expertise. However, some algorithms share some datasets, such as A 1 and A2.

In this context, *meta-learning* was developed: it relates algorithms to their area of expertise using specific problem characteristics. The idea of *meta-learning* is to learn about classifiers or learning algorithms, in terms of the kind of data, for which they actually perform well (Smith-Miles, 2008; Ali and Smith, 2006). Using dataset characteristics, which are called *meta-features* (Castiello et al., 2005; Ali and Smith, 2006), one predicts the performance results of individual learning algorithms. These features are divided into several categories (Castiello et al., 2005):

- simple/general features (number of attributes, number of categorical attributes, number of samples, . . .).

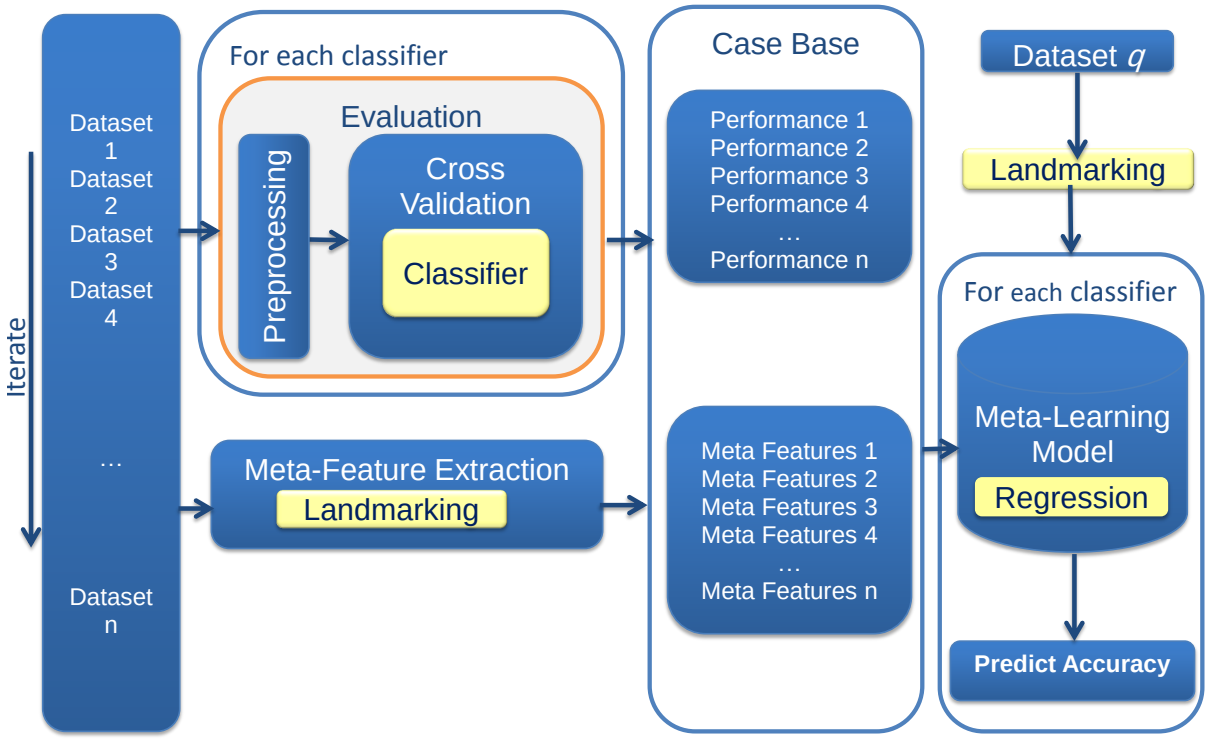


Figure 1.2: Schematic view of meta-learning: on the left, the case base is created, on the right an unknown dataset is processed and a suitable classifier is recommended.

- statistical features (canonical discriminant correlations, skew, kurtosis, ...) (King et al., 1995).
- information theoretic features (class entropy, signal-to-noise ratio, ...).

In addition, a new feature category called *landmarking* was proposed (Pfahringer et al., 2000). *Landmarking features* are classification accuracies of some simple but fast computable learning algorithms, mostly related to more complex classifiers.
Figure 1.2 gives a general schematic overview of a meta-learning system: it shows, that for a number of datasets, multiple classifiers are evaluated. From these datasets *landmarking features* are extracted, as explained in Section 2.2.4. All results are stored in a central *case base* as later explained in Section 4 and a meta-learning model is trained. Later, if an unknown dataset q needs to be processed, *landmarking features* are extracted and using the meta-learning model, a classifiers' accuracies are predicted.

As a contribution of this thesis, a *landmarking operator* was developed on top of PaREn as the first *meta-learning* step in RapidMiner. Details about this operator are provided in Section 2.3.

1.2 Thesis Outline

The report is composed of the following major parts:

Chapter 2 is divided into four sections. The first section provides some important definitions in the field of Pattern Recognition and elaborates different approaches for meta-learning. The subsequent section gives definitions about some meta-features. Then, an overview of the developed landmarking operator is described. In the last section, the basic idea of predicting classifier's accuracy based on regression is explained.

Chapter 3 shows the idea of the CUI and gives an insight into its implementation and the integration of the Classifier Recommender into it.

Chapter 4 provides experiment setup and important results of the evaluation of the landmarking features compared to other meta-features, based on UCI and StatLib-datasets.

Chapter 5 concludes the developed tools and the evaluation. Furthermore, it states interesting discussion points.

Chapter 2

Meta-Learning

This chapter presents background knowledge about classification, regression and different approaches for meta-learning. Furthermore, it explains the different meta-features and the prediction of algorithms accuracy on a given problem using regression. In addition, an overview of the developed landmarking operator is provided.

2.1 State of the art

In this section, an overview of basic definitions and meta-learning approaches are provided.

2.1.1 Classification

Classification is defined as associating input data to discrete categories, where the number of categories is finite. For example, if a handwritten digit is to be recognized, a classification algorithm will assign it to one of the categories from zero to nine. This is achieved by executing a learning algorithm that classifies data, which is called *classifier*. Any classification process has two phases:

1. *Training or learning phase*: in this phase the learning algorithm is applied on a subset of the dataset, called training data. This results in a trained model.

2. *Test phase*: in this phase another subset of the data, called test data, is evaluated using the model created in the training phase. The ability of the model to categorize the test data is called *generalization*.

Due to the variability of problem types, complexities, and different attribute types (e.g., numerical or nominal attributes), many learning algorithms were developed, where each learning algorithm has an *area of expertise* (Vilalta and Drissi, 2002). To reduce the complexity of a problem and speed up the classification process, *preprocessing* or *feature extraction* is applied on the dataset. Preprocessing aims to make the problem easier to solve, by reducing the variability of class categories.
Some learning algorithms create their model using input data and some adjustable parameters. These parameters affect the model's accuracy and should be set according to the problems features and complexity.

In this context, the concept of parameter optimization was developed. Consequently, a recent research area called automatic parameter optimization was initiated. A commonly used method for parameter optimization is *grid search*. This method takes at least two inputs:

1. A problem.
2. n number of parameters of an algorithm. Note, that several algorithms' parameters can be given as input to the grid search algorithm.

The grid search algorithm further tries to find the optimal combination for the n parameter values, such that the best performance of the algorithm on the given dataset can be achieved. The grid search algorithm has some parameters that should be adjusted for each of the n input parameters. These are:

- Range: is the range of the values in which the algorithm searches for the optimal values for the parameters that should be optimized.
- Steps: is the number of values that should be chosen from the specified range.
- Scale: is the methodology by which the values should be chosen from the specified range. The values can be chosen in a linear, quadratic or logarithmic manner.

The total number of combinations of parameter values tried by the grid search is equal to the product of the steps of all the parameters. For example, if two parameters should be optimized, each having ten steps, then the number of combinations equals 100.

Cross-Validation

To enhance the performance of a learning model and have a better generalization, *validation* should be applied on a given problem. The idea is to have a subset of the dataset, called validation set, that is not included in the training set, and is considered as a test set. One of the most powerful validation techniques is the *cross-validation*. In this technique the performance of an algorithm on an input dataset is averaged over several rounds of evaluation. In each round the analysis is done on the training set, and then the validation is performed using the validation set, where the subsets differ from one round to the other. An important issue is how many rounds should be performed. The best performance is achieved if the validation set consists of one example and the number of iterations is equal to the size of the dataset. That is called *leave-one-out cross-validation*, however, this is very expensive for large and complex problems, due to the time and resource consumption the computations need. Therefore, the *k-fold cross-validation* is commonly used, where k specifies the number of rounds.

Learning Algorithms

In this section learning algorithms that were used throughout the work are described:

Naive Bayes is a probabilistic classifier, based on *Bayes' Theorem*:

$$p(X|Y) = \frac{p(Y|X) \cdot p(X)}{p(Y)} \tag{2.1}$$

where $p(X)$ is the prior probability and $p(X|Y)$ is the posterior probability. It is called *naive*, because it assumes independence of all attributes to each other.

K-Nearest Neighbor (K-NN) is an instance-based learning algorithm. This type of algorithm does not perform generalization. Instead, it compares the instances of the test set with instances in the training set. This algorithm simply checks the k-nearest neighbors to the current test instance, and assigns it to the dominant class by voting, where the dominant class is the most common class in the k-neighbors. These neighbors are specified by measuring the distance between the attributes.

LibSVM is a library for Support Vector Machine (SVM). The SVM model represents the dataset examples as points in space by trying to cluster separate categories by the widest gap possible. It constructs an N-dimensional hyperplane that separates the different categories. This algorithm has some parameters, such as C and γ, that are provided by the LibSVM library.

MultiLayer Perceptron (MLP) is a feedforward neural network, which means that the connections between the layers do not form directed cycles. It is an extension of the linear perceptron algorithm. The linear perceptron algorithm consists of two layers: input and output. The sum of the weighted input values is passed to an activation function and then the output is predicted. In the MLP the same procedure is applied, but an extra layer called hidden layer is added between the input and output layer. This algorithm is more powerful than the linear perceptron, as it can distinguish data that is not linearly separable.

Decision Tree is a tree in which the nodes represent attributes of a dataset. The attributes are chosen according to some criterion, such as information gain. The information gain indicates how informative an attribute is with respect to the classification task using its entropy. The higher the variability of the attribute values, the higher its information gain. The attributes with the better criterion value (e.g., higher information gain) are chosen to create the next node. The leaves of the tree represent the classes.

Random forest is based on the Decision Tree algorithm. It randomly creates different decision trees and outputs the class that is the mode of the output classes of the individual trees. Mode is defined as the value that occurs most frequently.

One-attribute-Rule (OneR) is an algorithm for finding a rule, such that the minimum error attribute is chosen for prediction. A rule is created for each attribute and the one with the lowest error is chosen. A single node decision tree consisting of the chosen attribute as root is then built as a classification model.

2.1.2 Regression Analysis

In contrast to classification, regression is a task for which the output is one or more continuous variables. In the regression analysis a relation between dependent and independent variables is established. Dependent variables are the output values of a dataset, and the dataset attributes are considered as independent variables. Throughout the analysis, it

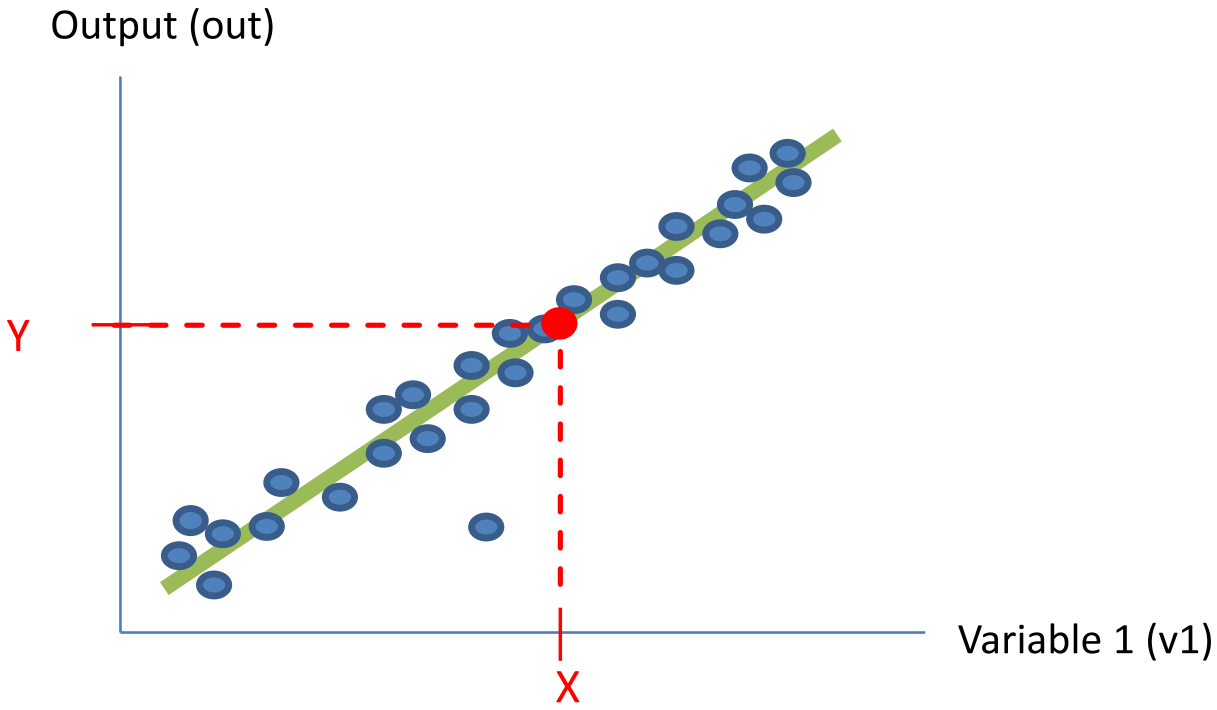


Figure 2.1: Example for linear regression on dummy data. The line drawn in green is the regression model created, based on the values of variable (v1) and the output (out). The dashed red line shows how a new variable value x can be mapped to an output value y using the regression model.

is aimed to find out how the values of the dependent variables change, whenever one or more independent variables vary. Based on this analysis a model or function is created to be able to map a set of independent variables to an output value. A regression model can be a line, a plane, or a hyperplane, depending on the number of variables or parameters combined.
One of the simplest forms of regression is *linear regression*, where a linear combination of parameters is constructed to build the model. Figure 2.1 illustrates an example of a linear regression model on one variable ($v1$) only. In this figure, it is shown how a new value x for the $v1$ can be mapped to an output value y using the regression model, which is a line in this case.

2.1.3 Different Approaches for Meta-Learning

Meta-learning is learning about learning algorithms, so that one can predict classifiers' accuracies, select the best suitable algorithm for a specific problem, or optimize algorithm parameters to be optimal for a problem instance. Many approaches were made trying to map a problem instance to a learning algorithm (Smith-Miles, 2008):

- Rice (1976) constructed a model for algorithm selection, which is illustrated in Figure 2.2. As depicted in the model, for a given problem x, and its extracted features $f(x)$, an algorithm α is selected by the mapping $S(f(x))$, such that α maximizes the performance mapping $y(\alpha(x))$. The aim was to find the mapping

function $S(f(x))$ that finds the best performant learning algorithm for a given problem x. However, Rice (1976) could not find the mapping function for algorithm selection problem and the model was considered as an abstract model.

- The next approach for algorithm selection was presented by Rendell and Cho (1990). They tried to experiment the effect of certain features on learning algorithms. These features are related to the size and concentration of the classes in a problem. Then this idea was developed to use simple features, described in Subsection 2.2.1, in order to create rules to decide whether an algorithm should be used for the given feature set or not. This technique was then extended to develop rules that state whether an algorithm performs well on an instance problem, in order to recommend a better classifier at run-time.

- The European project titled *StatLog* (King et al., 1995) aimed to evaluate the various classification approaches and to relate the performance of learning algorithms to problem characteristics. In this project, more feature types were taken into consideration, such as statistical and information theoretic features (Michie et al., 1994; King et al., 1995). To learn a rule for an algorithm, the decision tree learner was used. The nodes of the tree contained the features and their values, for which the algorithm outperformed other learning algorithms.

- The idea of the StatLog project was extended by Gama and Brazdil (1995). Instead of using decision trees to construct rules for each algorithm, regression was used. Based on this idea the evaluations, described in Chapter 4, were performed. In addition, the Classifier Recommender tool of the PaREn project was developed using the regression approach.

- In contrast to developing rules, Case-based reasoning (CBR), which is the process of solving a new problem based on the solutions of similar past problems, was used for meta-learning (Lindner and Studer, 1999). CBR is used to calculate the similarity between the problems based on their meta-features. Based on this similarity a set of one or more learning algorithms is recommended to the user.

- METAL Petrak (2002) is another project that was created based on the success of the StatLog project. The goal of this project was to provide model or algorithm selection approaches, supported in an on-line environment. The performance results were based on a ten-fold cross-validation. In addition, a tool called METAL Data Characterization Tool (DCT) was developed and was used in the evaluation as described in Subsection 4.1.1.

- Another approach in meta-learning is the *ranking* technique (Brazdil et al., 2003; Soares and Brazdil, 2000). Instead of selecting a single algorithm for a problem, a set of learning algorithms are ranked according to their performance on the problem. Time and accuracy were the major performance criteria on which the approach was based. This methodology used the K-NN algorithm to identify the similarity between datasets, based on their meta-features. Then, the algorithms of the k most similar datasets to the current problem instance are chosen to be ranked. These algorithms are then evaluated on the problem and their performance accuracy is provided to the user.

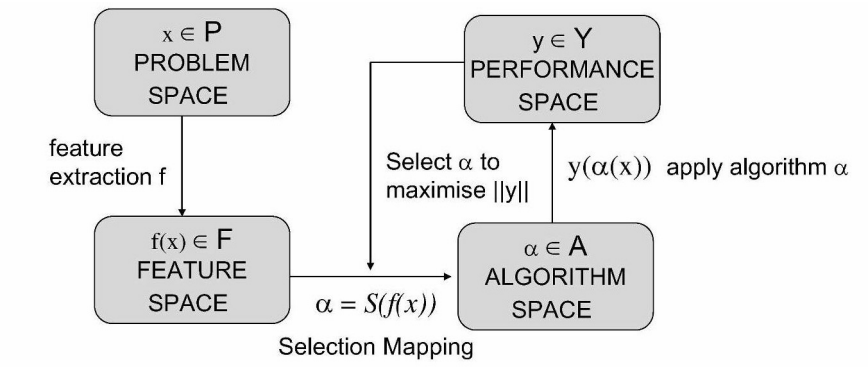


Figure 2.2: Schematic diagram of Rice's algorithm selection problem model. The objective is to determine the selection mapping S that returns the best algorithm α. This figure is taken from Smith-Miles (2008).

In the Statlog project it was stated that the effort for calculating features is sometimes greater than that for running some simple algorithms. In this context, the idea of relating simple algorithms to more complex algorithms was proposed and was developed in the METAL project. This approach is called landmarking. It is the major contribution of this thesis and will be discussed in the next two sections.

2.2 Meta-Features Extraction

In this section an overview of the different categories of meta-features is provided, wherein the landmarking features are discussed in detail (King et al., 1995; Castiello et al., 2005).

2.2.1 Simple Features

Simple features are general information that can be extracted from a dataset, which measure the problem's size or complexity. The following features are the most important simple features:

Number of samples represents the total number of samples observed by the dataset.

Number of attributes is the total number of attributes contained in the dataset, including numerical and nominal attributes. The larger the number of attributes of a problem, the more complex its classification is.

Number of output values represents the number of output values or classes in the dataset.

Dataset dimensionality is the ratio between the two features: number of attributes and number of samples of the dataset.

Number of categorical attributes represents the number of nominal attributes in the dataset. Nominal attributes may be complex for some algorithms in terms of calculations; therefore, they may increase the execution time and complexity of the

problem. Some other algorithms, such as the linear discriminant analysis can not handle these attributes. Therefore, preprocessing needs to be applied on the dataset (e.g., convert the attributes to numerical ones) to be able to evaluate it.

2.2.2 Statistical Features

The statistical features are statistical measures that indicate the distribution of attributes and their correlation. Before explaining these features some basic principles have to be defined:

Standard deviation: measures the variability or dispersion in the distribution.

Covariance: is a measure for how two *variables*, in that case *attributes*, change with respect to each other.

Principle Component Analysis (PCA): is the procedure of reducing the complexity of high dimensional data by reducing the number of variables. A principal component is a linear combination of variables, which accounts for as much of the variability in the data as possible. Each succeeding component accounts for as much of the remaining variability as possible and is orthogonal to all the previous principle components.

Canonical correlation analysis: assesses the relationship between two sets of variables. In this technique, a linear combination of variables from each set is derived, such that the correlation between the two combinations is maximized.

A description of statistical measures that were mainly used in the evaluation represented in Chapter 4, is provided below:

Homogeinity of covariance: is a measure of the affinity of the covariance of the attributes. As defined in King et al. (1995), it is "the geometric mean ratio of standard deviations of the populations of individual classes to the standard deviations of the sample".

Canonical discriminant correlation: is based on PCA, canonical correlation, and projection of the attribute's space. It tries to successively find linear combinations of attributes that discriminate between the examples and are orthogonal to each other. As stated in King et al. (1995) "Canonical discriminants systematically project the n attribute space to $n-1$, maximizing the ratio of between-mean distances to within cluster (population centers of examples) distances; successive discriminants are orthogonal to earlier discriminants".

Variance by the first q canonical discriminants: is related to the canonical discriminant analysis. It is explained by King et al. (1995) as "The sum of the first q eigenvalues of the canonical discriminant matrix divided by the sum of all the eignvalues represents the "proportion of total variation" explained by the first q canonical discriminants".

Skew is a measure of asymmetry of the distribution of an attribute. If it has a value of zero, then it is a uniform variable that is normally distributed.

Kurtosis is a measure of the "peakedness" of the probability distribution of an attribute. The higher its value, the sharper is the peak of the distribution.

Mean absolute correlation coefficient: is based on the correlation between attributes, which is calculated for each class separately. It is considered as a measure of the interdependence between attributes. As stated in King et al. (1995) the nearer the correlation between the attributes to unity, the more redundant data is in the attributes.

2.2.3 Information Theoretic Features

The information theoretic features are indicators for characteristics of datasets containing categorical attributes. These features measure the randomness of an instance to indicate whether a problem has a structure or not, implying how hard a problem is. The following information theoretic measures can be distinguished:

Class entropy: in general, entropy is a measure for the randomness or distribution of a variable. High entropy indicates the uniformity of the distribution of the variable, while low entropy implies high variations in the distribution of the variable. In this context, class entropy measures how much information is needed to specify a given class.

Noise to signal ratio: is a measure of the noise, which is the amount of irrelevant information contained in the dataset.

2.2.4 Landmarking

Every problem or dataset has certain characteristics that relates it to an area of expertise for which a specific learning algorithm exists. In this context, *landmarking features* are defined as dataset characteristics representing the performance of some simple learners on this dataset. These simple learners are called *landmarkers*.
The basic hypothesis behind *landmarking* is that the simple *landmarkers* are somehow related to more advanced and complex learners. This means that *landmarkers* or combinations thereof are able to estimate the performance of more sophisticated algorithms for a given problem. This raises the question of how to choose the learners that are ideal as landmarkers. They have to satisfy the following conditions (Pfahringer et al., 2000):

- The algorithm has to be simple, which requires its execution time to be short, implying minimal computational complexity of the learner (Bensusan and Giraud-Carrier, 2000).
- The landmarkers have to differ in their bias, mechanism, property measurements, or area of expertise (Vilalta and Drissi, 2002).
- Every landmarker has to be simpler than the targeted advanced learner. Otherwise the landmarker will be useless, since the targeted learning algorithms could be evaluated directly, avoiding a potentially error prone prediction step and saving time (Bensusan and Giraud-Carrier, 2000).

The landmarkers used in the implemented RapidMiner landmarking operator are (Pfahringer et al., 2000; Bensusan and Kalousis, 2001; Giraud-Carrier, 2008):

Naive Bayes Learner described in Subsection 2.1.1.

Linear Discriminant Learner is a type of discriminant analysis, which is understood as the grouping and separation of categories according to specific features. Linear discriminant is basically finding a linear combination of features that separates the classes best. The resulting separation model is a line, a plane, or a hyperplane, depending on the number of features combined.

One Nearest Neighbor Learner is a K-NN classifier with k equal to one. In this algorithm a test point is assigned to the class of the nearest point within the training set.

Decision Node Learner is a classifier based on the information gain of attributes. This learner selects the attribute with the highest information gain. Then, it creates a single node decision tree consisting of the chosen attribute as a split node.

Randomly Chosen Node Learner is a classifier that results also in a single decision node, based on a randomly chosen attribute.

Worst Node Learner is a classifier that calculates the highest information gain for a split for all the attributes. Then it chooses the attribute with the lowest information gain among all attributes to model a single node decision tree.

Average Node Learner calculates the average performance of single node decision trees, where each node corresponds to one attribute.

The accuracies of the above defined algorithms are used as landmarking features for the task of meta-learning.

2.3 Landmarking Operator in RapidMiner

The landmarking operator, developed as part of this thesis, is the starting point for meta-learning using RapidMiner. It can be easily extended and used to build meta-learning processes, such as classifier recommender or automatic algorithm selection. In this section, an overview of the design of the landmarking operator is provided.

2.3.1 RapidMiner Terminology

Before going into details of the landmarking operator, some RapidMiner terminology has to be defined:

- **Operator** is the super-class of any operator implemented in RapidMiner. To be precise, an operator is an object that accepts an array of input objects, does some operations or processes based on its defined role, and then returns an array of output objects. These inputs and outputs of an operator are delivered or received through *ports* that can be connected to other operators' ports. Operators can have

parameters as settings for its operations or processes. The processes or operations of an operator are performed in a method called *doWork*, which has to be implemented by every operator. This method is invoked when the operator is executed in order to start its processes. Moreover, an operator can be encapsulated or embedded in another operator as a subprocess.

- **ExampleSet** is an interface that represents datasets. It can be passed to operators as input or delivered by an operator as output. This ensures that example sets can be preprocessed or processed by an operator.

2.3.2 Landmarking Operator Functionality

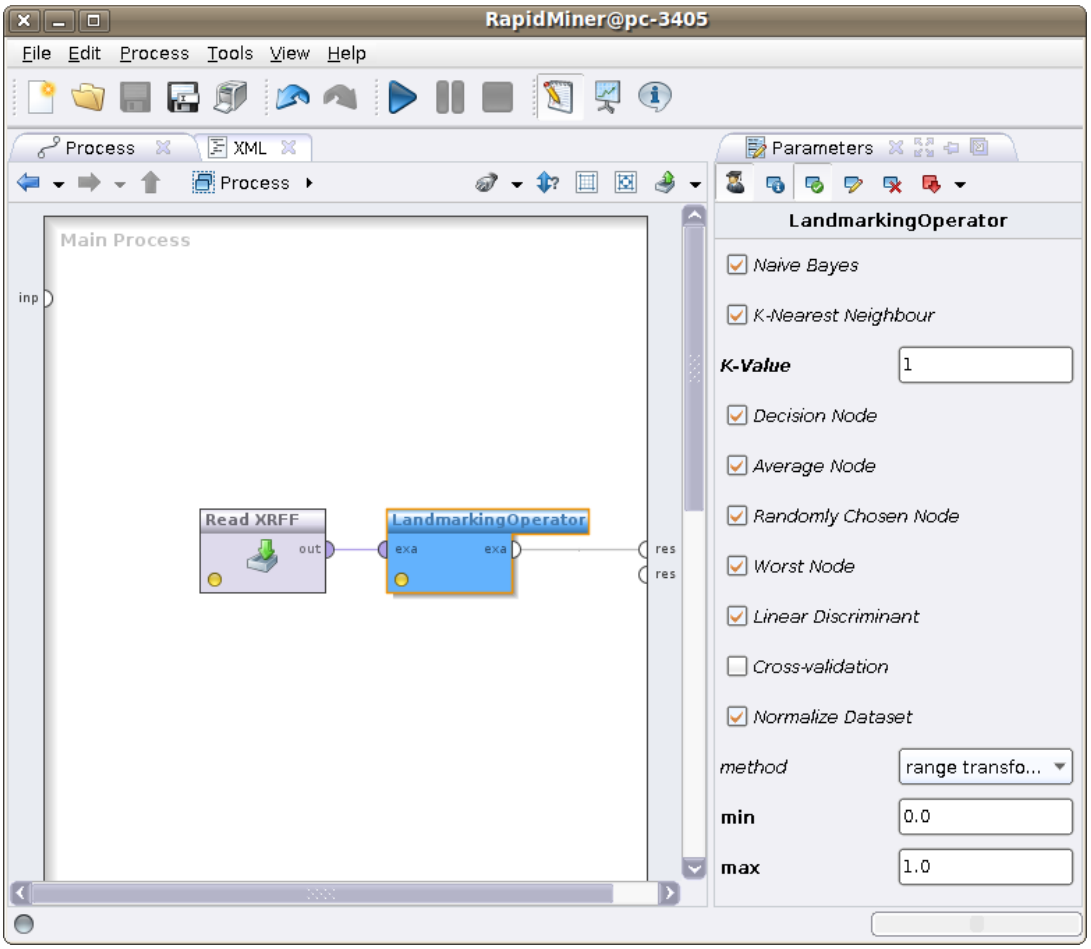


Figure 2.3: The landmarking operator in RapidMiner. In the middle of the figure, the process tab is depicted. In this tab the landmarking operator is connected to a ReadXRFF operator to get an input *ExampleSet*. The parameters of the operator are represented in the Parameters tab on the right. These are the default settings of the parameters of the landmarking operator.

As shown in Figure 2.3, the landmarking operator has two ports: an input port that should receive an *ExampleSet*, and an output port. The delivered output contains the landmarking results in the form of an *ExampleSet*, having the landmarkers as attributes,

and their accuracies as attribute values. Some parameter settings are provided to make the operator adequate to problem requirements. The user has control over:

- the landmarkers to be evaluated.
- whether the landmarkers should be encapsulated into a *cross-validation* process or not. If this parameter is not enabled, the input dataset is considered as training and test set at the same time.
- the *preprocessing* of the input *ExampleSet* by *normalization*.

By default all the landmarkers described in Section 2.2.4 are enabled and the normalization property is set to true with a range [0, 1]. However, since several classifiers have special properties or requirements, some landmarkers have to preprocess the input dataset or have to use the classifier in a certain way. For example, it is illogical to evaluate the One Nearest Neighbor landmarker without using cross-validation, as the prediction will be always correct by picking the point itself as the nearest neighbor, resulting in a 100% accuracy. Another specially handled landmarker is the Linear Discriminant Learner: It is set to be never validated using cross-validation, because of the inability of the RapidMiner operator to run always correctly in this case. Furthermore, some preprocessing is applied on its input *ExampleSet*. The preprocessing steps employed are:

1. Removing mappings of label values that actually do not occur in the examples of the dataset, as RapidMiner's linear discriminant operator can not handle this case.
2. Converting all *polynominal* attributes to *binominal*, then to *numerical* attributes, as the linear discriminant analysis operator does not support nominal attributes.

If cross-validation is applied on an *ExampleSet* with a total number of examples smaller than the chosen number of folds, a *leave-one-out* cross-validation is performed instead automatically.

2.3.3 Landmarking Operator Architecture

The developed landmarking operator consists of three important components, namely *AbstractLandmarker*, *LandmarkingOperator*, and *LandmarkingResults*. **AbstractLandmarker** is the super-class of all landmarkers, as illustrated in Figure 2.4. The main method of this class is *learnExampleSet*, which evaluates the performance of the model of the passed *Operator* on the input *ExampleSet*, according to the parameter settings described in Section 2.3.2. Any landmarker has to be a sub-class of AbstractLandmarker, having its own operator and specific preprocessing or parameter settings. Figure 2.4 shows the relation between the landmarkers and RapidMiner operators. All decision node landmarkers use RapidMiner's *DecisionStumpLearner* to create their decision node. This is achieved by reducing the attribute set of their input dataset according to their definition and using the information gain of the attributes. The other landmarkers are simply mapped to their corresponding operators in RapidMiner, as depicted in Figure 2.4. To integrate the landmarkers in a single operator in RapidMiner, the *LandmarkingOperator* has been developed as a sub-class of *Operator*. The interaction point between the operator and the landmarkers is the *doWork* method of the landmarking operator. As illustrated in Figure 2.5, the following sequence of operations is performed in this method:

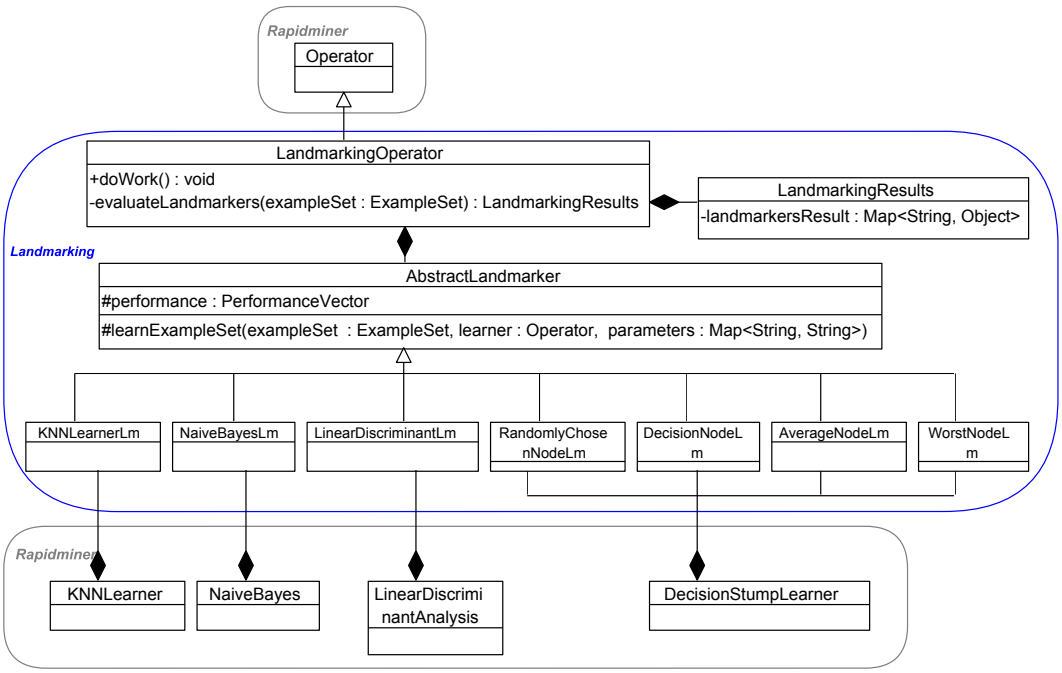


Figure 2.4: Class diagram for RapidMiner's landmarking operator

- By calling the major method *evaluateLandmarkers*:
  - A map containing the parameters described in Section 2.3.2 as key is created. For any additional parameters that should be passed to a landmarker, a modification of the map can be easily done in the LandmarkingOperator class.
  - The constructed parameters Map is passed to the landmarkers and each landmarker process is managed separately. The result of each landmarker is added to a *LandmarkingResults* object x.
  - The *LandmarkingResults* object x is returned.
- The result of the evaluated landmarkers is controlled by *LandmarkingResults* class. This result object x contains a map, *landmarkersResult*, that is filled after the evaluation of a landmarker with its name and accuracy. Then, the LandmarkingOperator parses the LandmarkingResults object to an *ExampleSet*.
- The *ExampleSet* is delivered to the output port of the operator.

Figure 2.6 shows a sample output of the landmarking operator. If one is interested in results other than the accuracy (e.g., absolute error, RMSE, ...), a key and a value can be easily added to the result map.

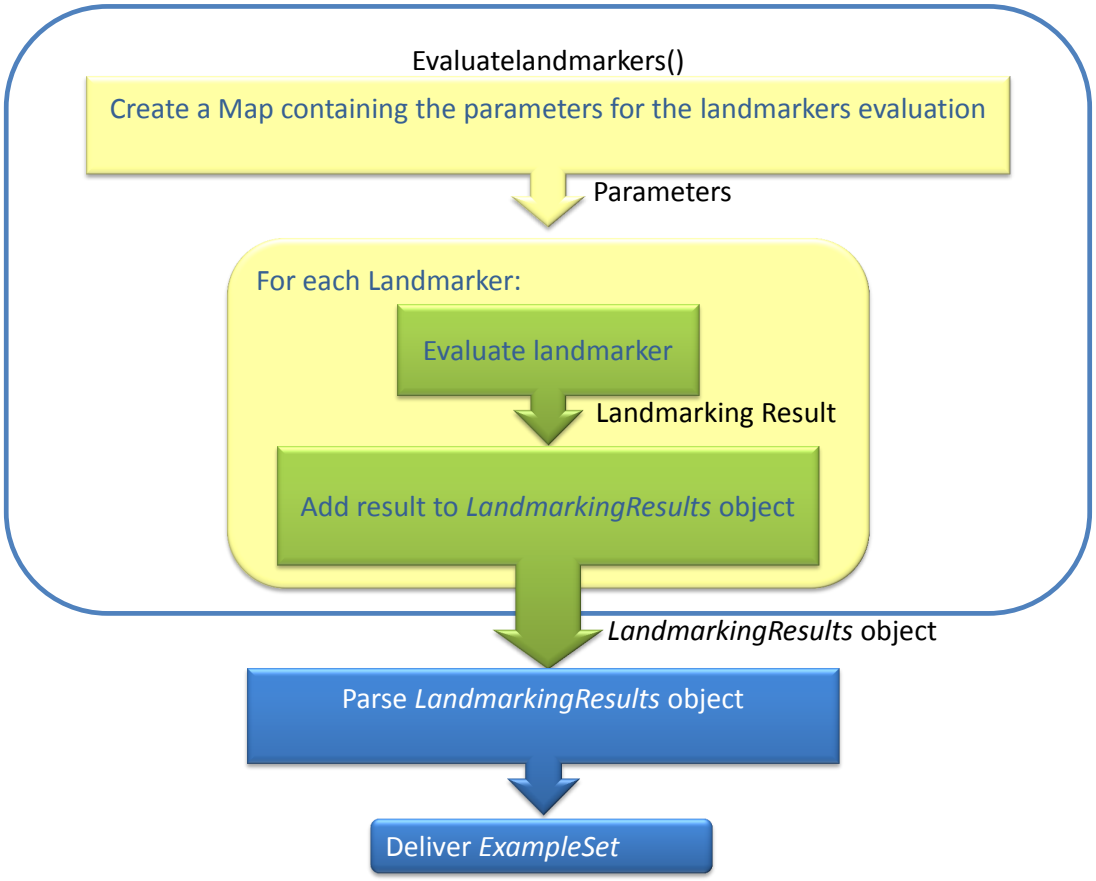


Figure 2.5: Sequence of the operations performed in the doWork() method of RapidMiner's landmarking operator

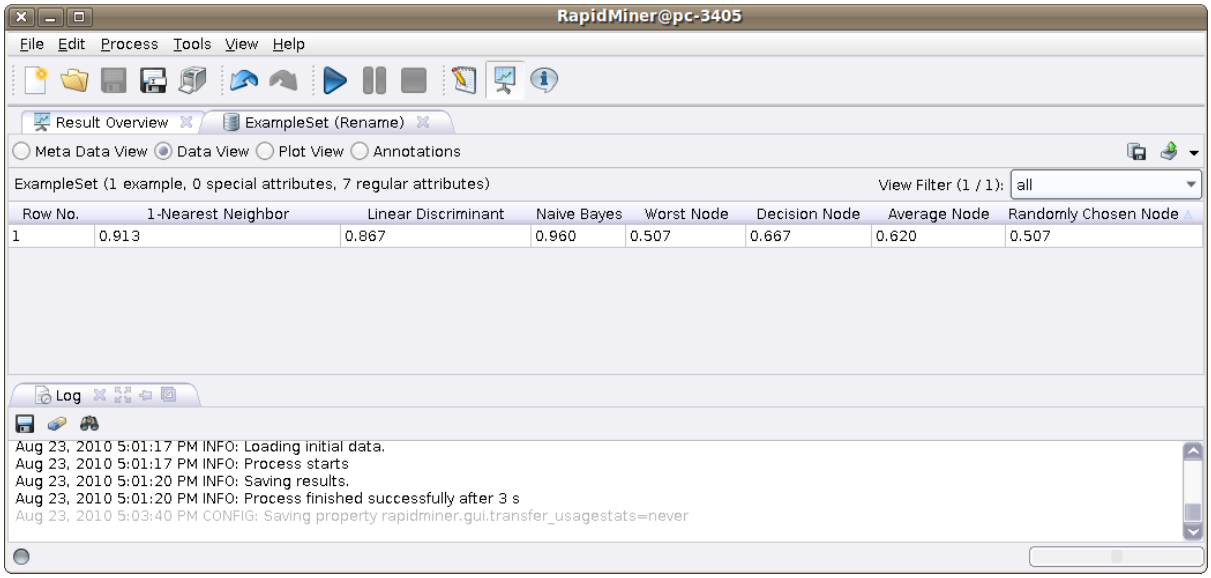


Figure 2.6: Sample output of the landmarking operator. In the header of the table, the names of the different landmarkers that were evaluated are represented. The first row, shows the calculated accuracy of each landmarker.

2.4 Classifier's Accuracy Prediction

As clarified in the introduction and in Section 2.1, there are several approaches for meta-learning. Classifier accuracy prediction is part of meta-learning and there are several techniques proposed for such a system. In this chapter, the basic idea of how regression can be used to predict classifier's accuracy based on meta-features is provided[1].
Every problem has some meta-features that can be extracted from it. These meta-features are very useful properties that can be used as input variables for a regression model. In this approach, a regression model is used as a meta-learning model for the meta-learning system, as illustrated in Figure 1.2. To predict the classifier's accuracy using meta-features and regression the following steps should be employed:

1. Choose one or more meta-features (described in Section 2.2), on which the prediction should be based.

2. Extract these meta-features from the training datasets.

3. Evaluate the classifier on all the training datasets, computing its accuracy.

4. Train the regression model on these datasets by using the meta-features along with the computed accuracies as input variables for the regression analysis.

5. Giving the meta-features of any new problem to the constructed regression model, the model will return the predicted accuracy of the classifier.

This approach for predicting classifier's accuracy based on dataset characteristics was used for the evaluation of landmarking features (provided in Chapter 4).

[1] A paper about regression for meta-learning will be published soon

Chapter 3

Collaborative User Interface

In this chapter, an overview of the Collaborative User Interface (CUI), which provides pattern recognition and meta-learning functionalities over the web, is presented.

3.1 Overview

The CUI is part of the PaREn project. It is a web application integrated with the rich PaREn tools and functionalities. The aim of the development of this application is to facilitate the usability of the PaREn tools for non-expert users or beginners. Furthermore, it provides the users with a repository for sharing data, experiments, and experience.

This thesis contributed in transforming the previously developed prototype into a more stable release. In this section, an overview of some functionalities related to meta-learning and classifier recommendation is provided. Some of these functionalities are:

- uploading datasets to the users repository on the server. The developed application accepts only XRFF format datasets.
- sharing datasets with other users
- recommending a classifier for a dataset by performing two steps:
    1. predicting the accuracies of some classifiers
    2. evaluating the classifiers selected by the users on the dataset to get their accuracies.
    3. Save the optimized pipeline, in order to provide it to the user.
- downloading the optimized pipeline of the evaluated classifiers as RapidMiner processes in eXtensible Markup Language (XML) format.

3.2 Application Description

In this section, an overview of the architecture and design of the CUI is provided to understand how it can be modified or extended.

3.2.1 Tools and Libraries

The application is built on top of the J2EE framework. It is developed using Java Server Pages (JSP), Java Servlets, and the following tools and libraries:

Glassfish Server[1] as a server for the web application.

JavaScript Object Notation (JSON) is a lightweight data-interchange format used to send data from the client side to the server side and vice versa. Data can be represented as objects or arrays of values. An object has simply the structure of name\value pairs placed between curly braces (e.g., {name1:value1, name2:value2,...}). The data stored in an array is represented as values. For instance, an array of objects can have this structure: array=[{name1:value1},{name2:value2},{name3:value3},...]. *FLEXJSON*[2] was used as a library for JSON in Java. The main advantage of this library is its ability to serialize Java objects into JSON and to deserialize JSON strings into Java objects.

Dojo a JavaScript tool kit that facilitates some operations, such as creating widgets, uploading files, and manipulating JSON data.

RapidMiner was integrated in the application as a Java ARchive (JAR) library, in order to use its objects and operators.

3.2.2 Architecture

To have an agile architecture, the application was developed based on the *Model-View-Controller (MVC)* architecture. As shown in Figure 3.1, the system is composed of several packages. The *Data Access Object (DAO)* and *Beans* packages represent the model part of the architecture, where the Servlets package acts as a controller for incoming client requests. As part of the architecture, the *Web Content* depicted on the right side of Figure 3.1, represents the view that manages the graphical user interface. Each package has a main responsibility that is achieved by its classes:

- *Beans:* is responsible for classes that represent stored data as Java bean objects.
- *DAO:* The DAO classes access, update, and retrieve stored data from the local disk. These classes use the bean objects to wrap stored data. In addition, they manipulate retrieved data as beans that are returned to the Servlets.
- *Servlets:* The Servlet classes in this package respond to HTTP requests fired by the client and control the data and actions that are sent to the client as a response. In order to send data from the server side to the client side, the servlets communicate with DAOs to retrieve the needed information as objects or beans.
- *RapidMiner Handler:* classes in this package handle RapidMiner processes, such as the classifier recommender steps described later.
- *Utils:* is responsible for instance for file services or for FLEXJSON operations, such as serialization and deserialization of objects.

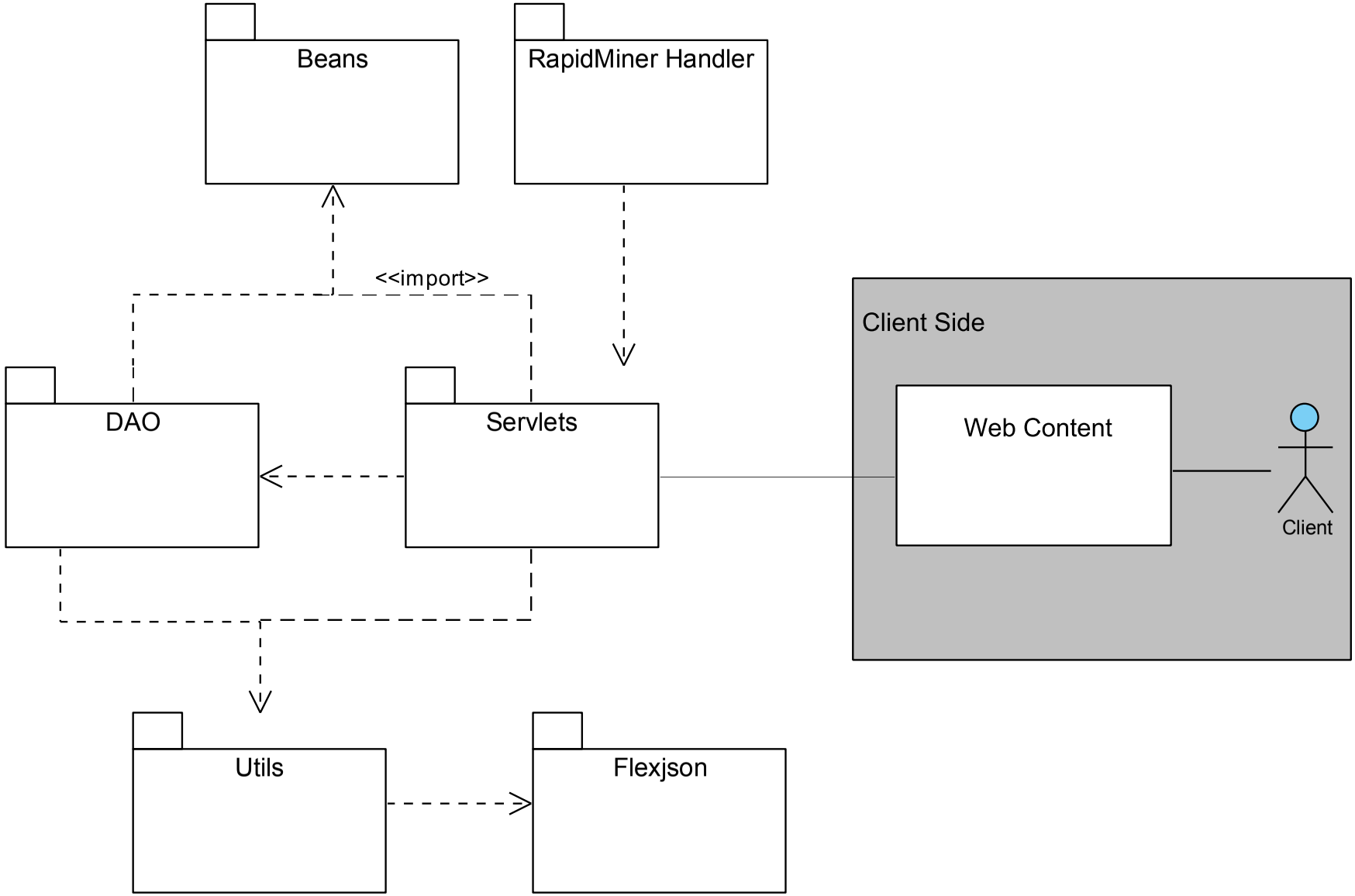


Figure 3.1: Use Case Package Diagram for the Collaborative User Interface

The data is stored in JSON format. A reason for why it was chosen to store data in files is to avoid costly operations of accessing a database. Furthermore, JSON format was used, as it makes the communication between the server and client easier and faster. Another reason is that there is no need neither to convert the stored data to any other format to send it to the client nor to convert the received data by the server into another format to store it. Thus, using DOJO and FLEXJSON library the data was directly and simply manipulated using serialization and deserialization.
Analogous to databases, an entry of the stored data has an id. Most bean objects have some important attributes, as depicted in Figure 3.2, that are used to control and manipulate stored data easily:

- *DatasetBean and ExperimentBean:* These beans have an id that identifies the datasets and the experiments uniquely. In addition, they have an owner id that represents the user id, who uploaded the dataset or executed the experiment. A list of users' ids, with whom the user shares his data or experiments, is part of these objects. In addition, any experiment bean has a dataset id that corresponds to the dataset on which the experiment should be evaluated.

- *Repository beans:* These beans have several attributes that manage the other beans:

  - An *idCounter* variable that represents the available id number that will be assigned to a new experiment, dataset, or user, respectively, that should be added to the storage.
  - A *list including other bean objects* that will be referred to as an entry. As shown in Figure 3.2, the DatasetsRepositoryBean and the ExperimentsRepositoryBean have a list of DatasetBean and ExperimentBean objects. Each of these entries include information about the datasets or experiments, such as name, owner id, and shared users' ids.

The lists of data entries of the repository beans are stored in JSON as an array and the idCounter is stored as an object. Each object in the array has its own name\value pairs, as illustrated in Figure 3.3. These data entries are updated using deserialization of the stored JSON data. For example, if a new dataset should be added to the user's repository, the JSON data is retrieved by the DatasetsDAO class upon a request from the DatasetsServlet class. Then the JSON string is deserialized to a DatasetsRepository object and a new DatasetBean is added to the list of beans of this object. After the update of this repository object, it is serialized to JSON format and stored on the disk again.

Classifier Recommender

An important part of the application is the *Classifier Recommender*. It basically predicts some classifiers' accuracies based on meta-features and regression, as explained in Section 2.4. When a user uploads a dataset, the classifier recommender experiment is automatically carried out on it. In this experiment, seven target classifiers are taken into consideration, namely Naive Bayes, K-NN, LibSVM, MLP, Decision Tree, Randomforest Weka, and OneR. The classifier recommender experiment is divided in two steps:

[2] http://flexjson.sourceforge.net/

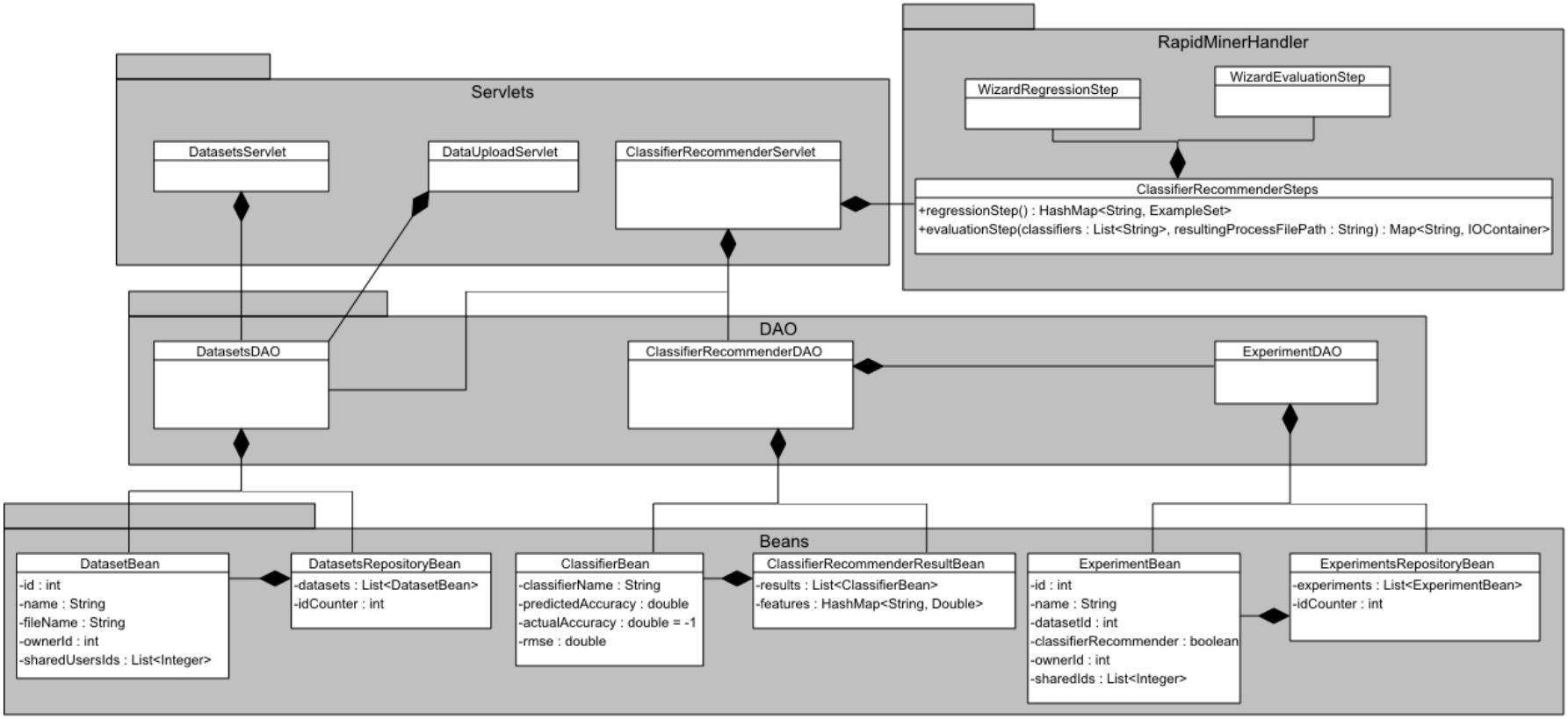


Figure 3.2: Class Diagram for the Collaborative User Interface

```
{"datasets":[

{"date":null,"fileName":"cars.xrff","id":0,"name":"cars","ownerId":
0,"sharedUsersIds":[],"size":105086},

{"date":null,"fileName":"backache.xrff","id":1,"name":"backache","
ownerId":0,"sharedUsersIds":[],"size":158086},

{"date":null,"fileName":"anneal.xrff","id":2,"name":"anneal","owne
rId":1,"sharedUsersIds":[],"size":751934}

],

"idCounter":3}
```


Figure 3.3: An Example of stored datasets' information in JSON format.

1. *Regression Step*: the class WizardRegressionStep executes this part of the experiment by using an operator called *Regressioner* that was developed by a co-worker as part of the PaREn project. First it extracts the meta-features from the input dataset. Specifically, it extracts landmarking features, described in Subsection 2.2.4. Then it applies a regression model on the extracted features, in order to predict the accuracy of the target classifiers. These regression models are trained on datasets and stored as a *case base* in order to speed up the prediction task. At the end of the process the landmarking features and the predicted accuracies are returned as *ExampleSets* in a Map, so that they can be viewed by the user.

2. *Evaluation Step*: after displaying the predictions of the recommended classifiers, the user can choose one or more classifiers to be evaluated, as depicted in Figure 3.6. The *WizardEvaluationStep* class takes the responsibility of evaluating the selected classifiers. First, an operator called *CaseBaseOperator* creates an optimized process for each of the classifiers. Then this process is run and the accuracies of the classifiers are computed and returned. Afterwards the optimized pipelines of the evaluated classifiers are stored in the users experiment results directory, so that the user can download the required pipelines.

These classifier recommender steps are managed by the class *ClassifierRecommenderSteps* whenever the *ClassifierRecommenderServlet* receives a request from the client, as depicted in the class diagram in Figure 3.2. To respond to such a request the *ClassifierRecommenderServlet* reads the experiment result from the disk. An example for such results is shown in Figure 3.4. The retrieved results are deserialized into an object of type *ClassifierRecommenderResultBean* that has a boolean variable called *evaluated*, which indicates whether some classifiers are already evaluated or not. If one or more classifiers are evaluated, then the predicted accuracies were already calculated for all the classifiers. Otherwise, it is checked whether the results list is empty or not. In case it is empty the predictions of all the classifiers will be calculated. If the results of the predicted accuracies are already stored, then they are just returned to the client side.

{"evaluated":true,"features":{},"results":[

{"actualAccuracy":0.7390846130683529,"classifierName":"NearestNeighbors","p
redictedAccuracy":0.8095663348287998,"rmse":0.08331182214072527},

{"actualAccuracy":0.8423968684131286,"classifierName":"DecisionTree","predi
ctedAccuracy":0.7446337460054715,"rmse":0.11062798605761938},

{"actualAccuracy":0.6798554652213189,"classifierName":"NaiveBayes","predict
edAccuracy":0.7165527860333101,"rmse":0.07326741389493535},

{"actualAccuracy":0.7932249322493224,"classifierName":"LibSVM","predictedA
ccuracy":0.8227500577839257,"rmse":0.1026168517032764},

{"actualAccuracy":0.7978319783197831,"classifierName":"NeuralNetImproved",
"predictedAccuracy":0.8281605665087648,"rmse":0.0950599225022344},

{"actualAccuracy":0.6502258355916892,"classifierName":"OneR","predictedAcc
uracy":0.6982805394952205,"rmse":0.07602753803746337},

{"actualAccuracy":-1.0,"classifierName":"RandomForest-
Weka","predictedAccuracy":0.8399062739829013,"rmse":0.0824752416262678
2}]}

Figure 3.4: An Example of classifier recommender results stored in JSON format.

As depicted in the class diagram in Figure 3.2, a *ClassifierBean* contains classifier variables, such as name, predicted accuracy, RMSE, and computed accuracy. By default the accuracy of any classifier is -1, as the accuracy has to be predicted for the classifier first. The value of the accuracy is then updated as soon as an evaluation request for the classifier is successfully completed. The predicted accuracies of the classifiers and their RMSE values are provided to the user as shown in Figure 3.5. After the user selects the classifiers to be evaluated and the evaluation completes successfully, two other columns with the computed accuracy and the downloadable pipeline are presented, as illustrated in Figure 3.7. The evaluation of the predicted and computed accuracies is performed only once.

Data Storage Structure

There are two categories of files stored on the server. First, some files are created to store general information about users, datasets, and experiments in JSON format, which are then used in the operations of setting, getting, and storing data, as explained above. These files include an array of JSON objects along with an object representing the next available id index. Hitherto, three files of this category exist, namely users, datasets, and experiments, as illustrated in Figure 3.8. The second category of files is either files uploaded by the users (e.g., datasets) or experiments' results, such as classifiers' accuracies or RapidMiner pipelines. The folder structure of the stored data is shown in Figure 3.8.

As illustrated, each user has his own folder, named after his id, that is encapsulated inside the "Users" folder. Each user's folder contains two sub-folders namely, "Datasets" and "Experiments". The "Datasets" folder contains all the datasets uploaded by the user, while the "Experiments" folder includes a sub-folder for each user's experiment. The experiments folders are named after the experiments' ids and each folder contains the results of its experiment. For example, for a classifier recommender experiment the results are stored in a file named "cars.xrff.results", where "cars.xrff" is the name of the dataset which was evaluated in the experiment. In addition to the result file, there are RapidMiner pipelines, which are stored in XML files saved under a name composed of the dataset's name and the classifier's name (e.g., cars.xrff.NaiveBayes).

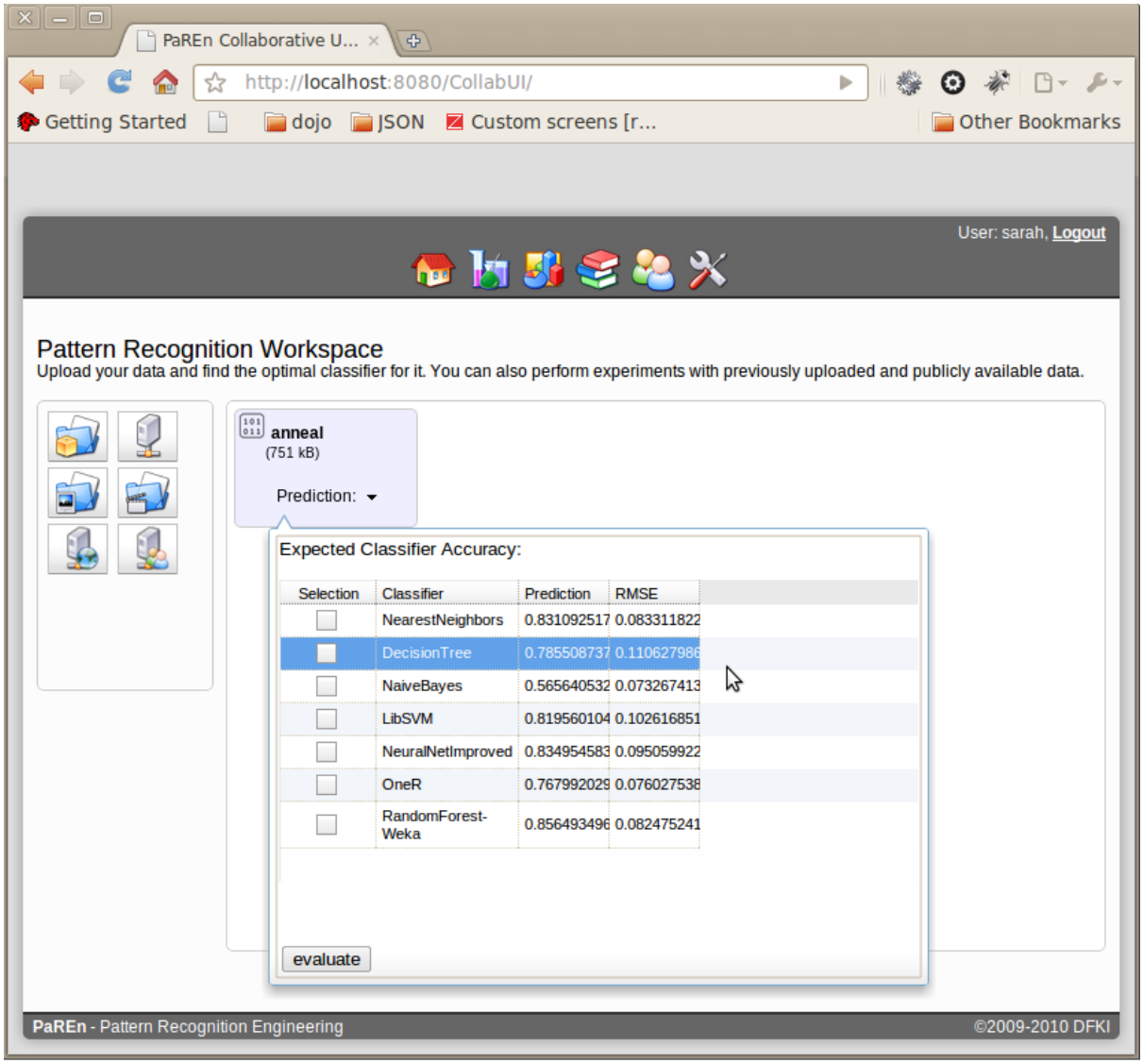


Figure 3.5: The pattern recognition workspace of the user is shown. In the middle there is a widget for the dataset anneal that is being evaluated. The computed predicted accuracies and the RMSE values of the classifiers are listed, so that the user can choose the appropriate classifier to be evaluated.

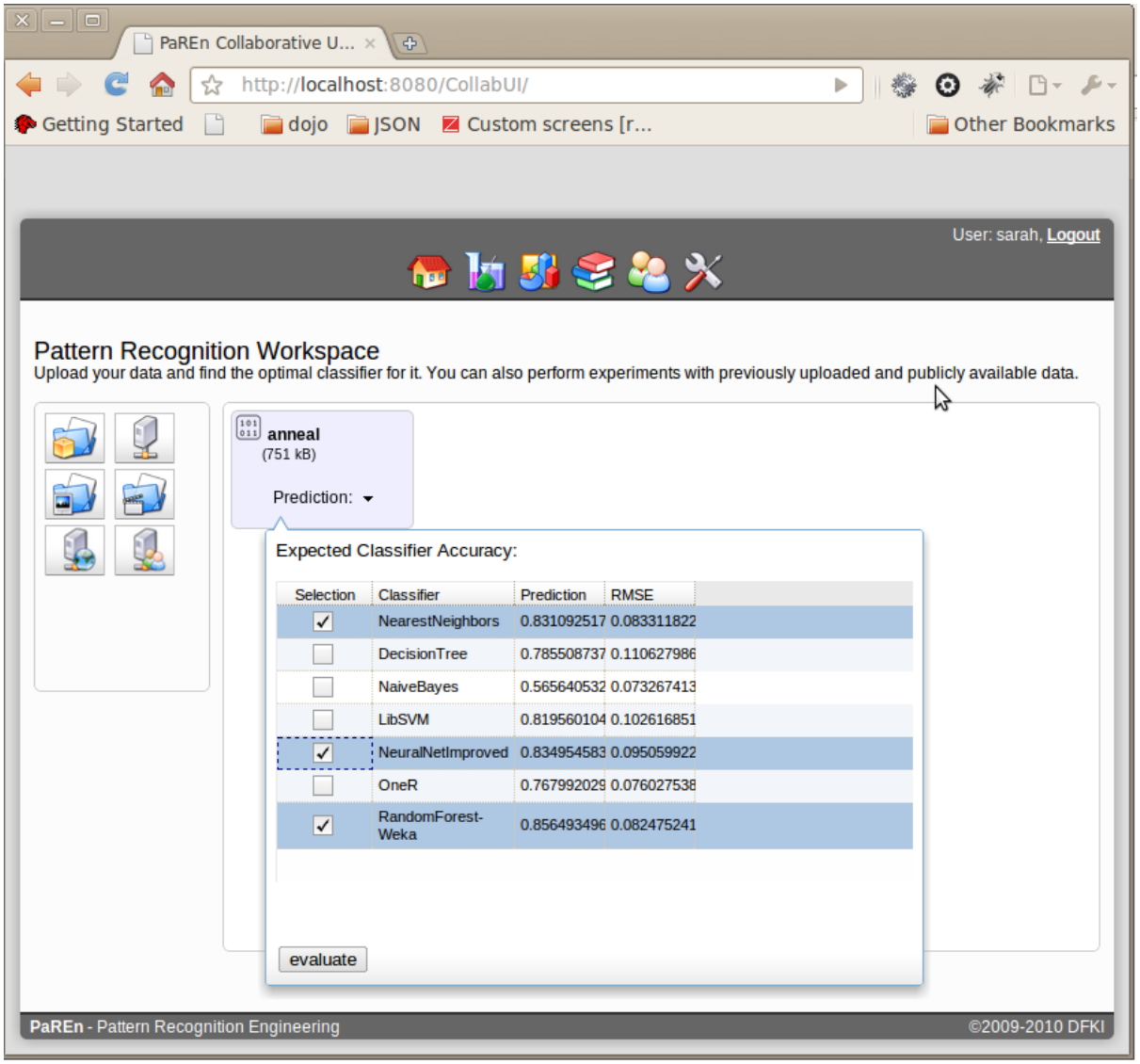


Figure 3.6: The pattern recognition workspace of the user is shown. In the middle there is a widget for the dataset "anneal" that is being evaluated. The predicted accuracies and the RMSE of the classifiers is already computed and the user selected two classifiers to be evaluated.

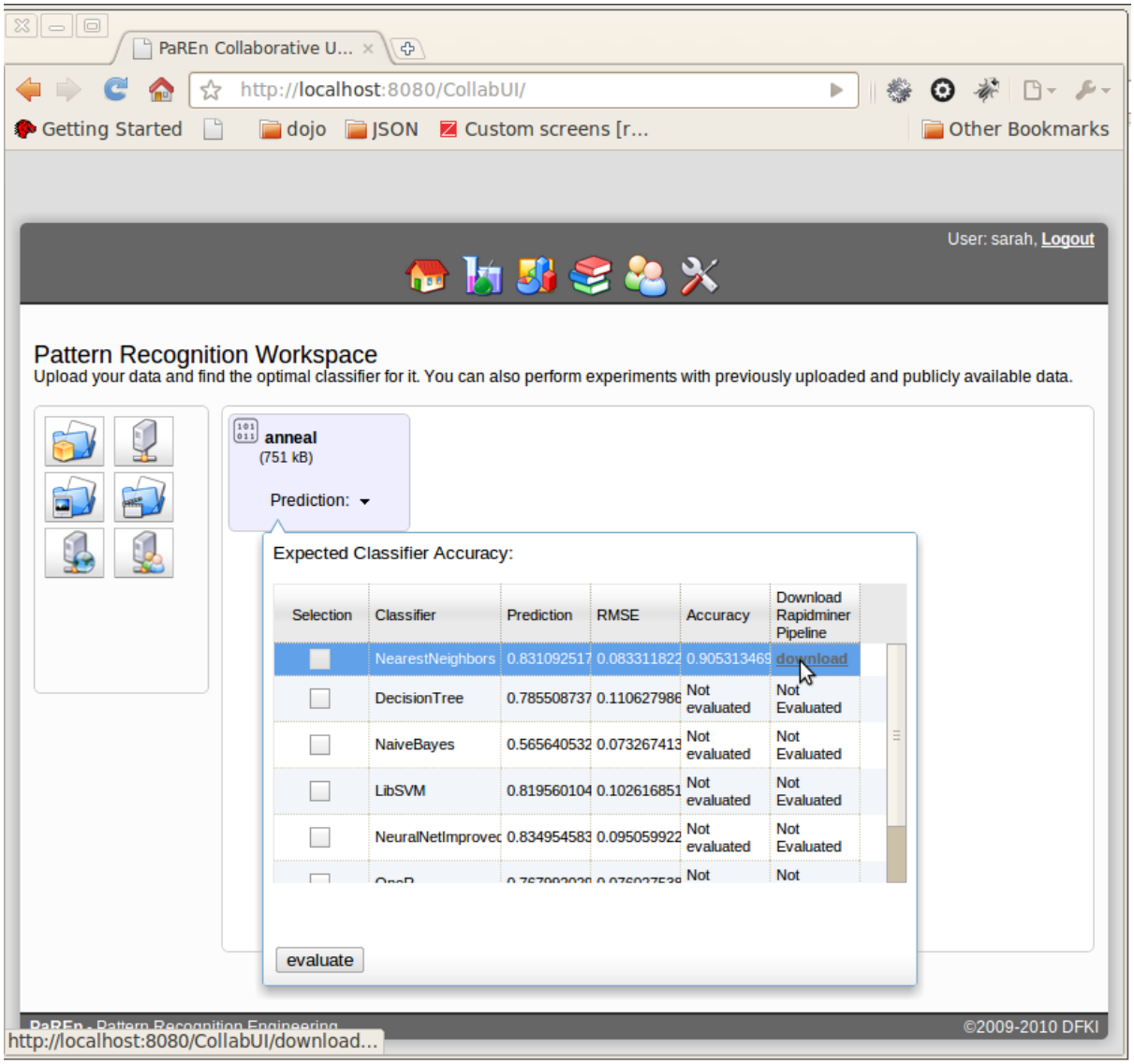


Figure 3.7: The pattern recognition workspace of the user is shown. In the middle there is a widget for the dataset "anneal" that is being evaluated. The computed predicted accuracies and the RMSE of all classifiers evaluated classifiers are listed. In addition, the computed accuracies of the evaluated classifiers along with a link to the RapidMiner pipeline are viewed to the user. The user can download any of the evaluated classifiers pipelines in XML format.

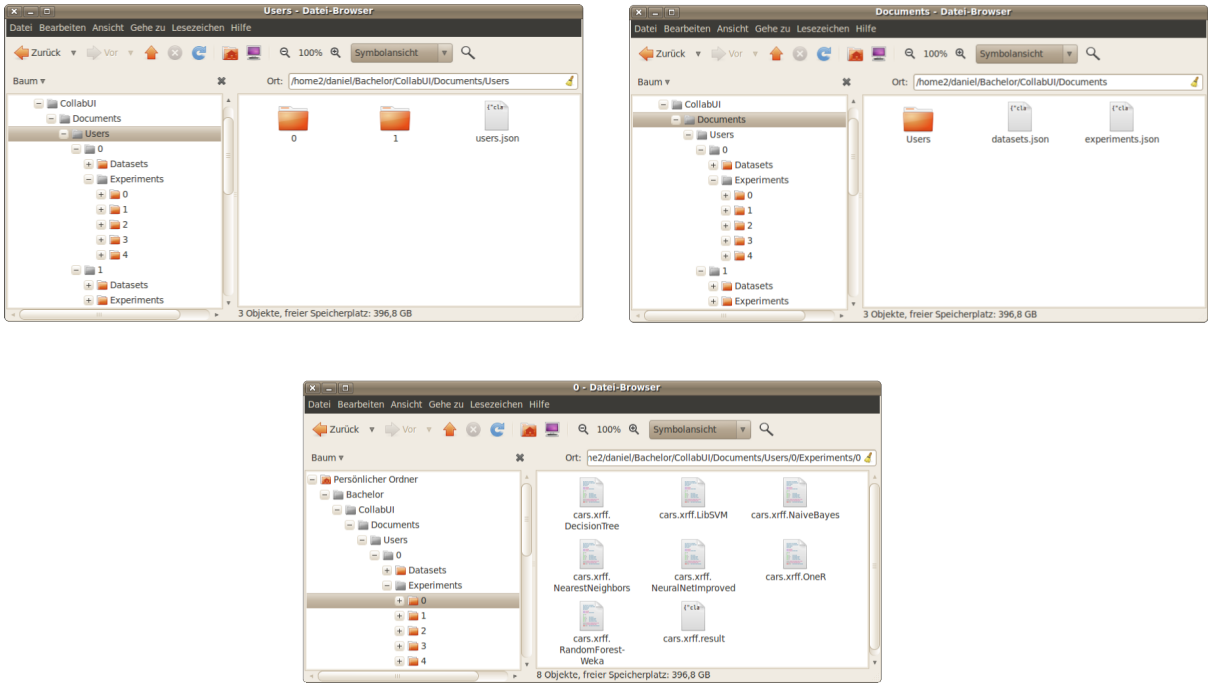


Figure 3.8: Stored data files and folders structure.

Chapter 4

Evaluation

The main goal of the evaluation is to determine whether landmarking features are useful for meta-learning or not compared to other meta-features. Accordingly, meta-features were evaluated by predicting the accuracies of some classifiers and observing the confidence of prediction calculated for the different classifiers. Further experiments were carried out to observe which landmarkers combinations and settings perform better, trying to correlate landmarkers and classifiers.

4.1 Experiment Setup

The experiments were applied on 90 datasets in total, where 67 datasets were from the UCI repository (A. Asuncion, 2007) and 23 datasets were from StatLib, as shown in Table A.1. In order to have reliable accuracy estimates, all chosen datasets have more than 100 samples. RapidMiner processes were used to manage the experiments. The target algorithms, for which the accuracy was predicted, are:

- Naive Bayes
- K-NN
- Multilayer Perceptron
- OneR
- Random Forest Weka[1]
- Decision Tree
- LibSVM

The chosen classifiers origin from different algorithm categories and were chosen because of their prominence. However, not all chosen classifiers are complex learning algorithms. The classifers that can be considered as complex are Decision Tree, LibSVM, MLP,and Random Forest. A grid search parameter optimization was applied on important parameters of the target classifiers, as illustrated in Table 4.1. For each of the classifiers, the

[1]Operator in the Weka plug-in

accuracy was computed for all the datasets. These accuracies, along with the dataset names were stored in a dataset considered as a *case base* as illustrated in Figure 4.1. In order to estimate the predicted accuracies of the chosen classifiers and the confidence of their prediction, *regression* was applied (King et al., 1995). The results were then analyzed using different measurements, as described in Subsection 4.1.4. The evaluation steps are illustrated in Figure 4.1 and are explained in detail in the following subsections.

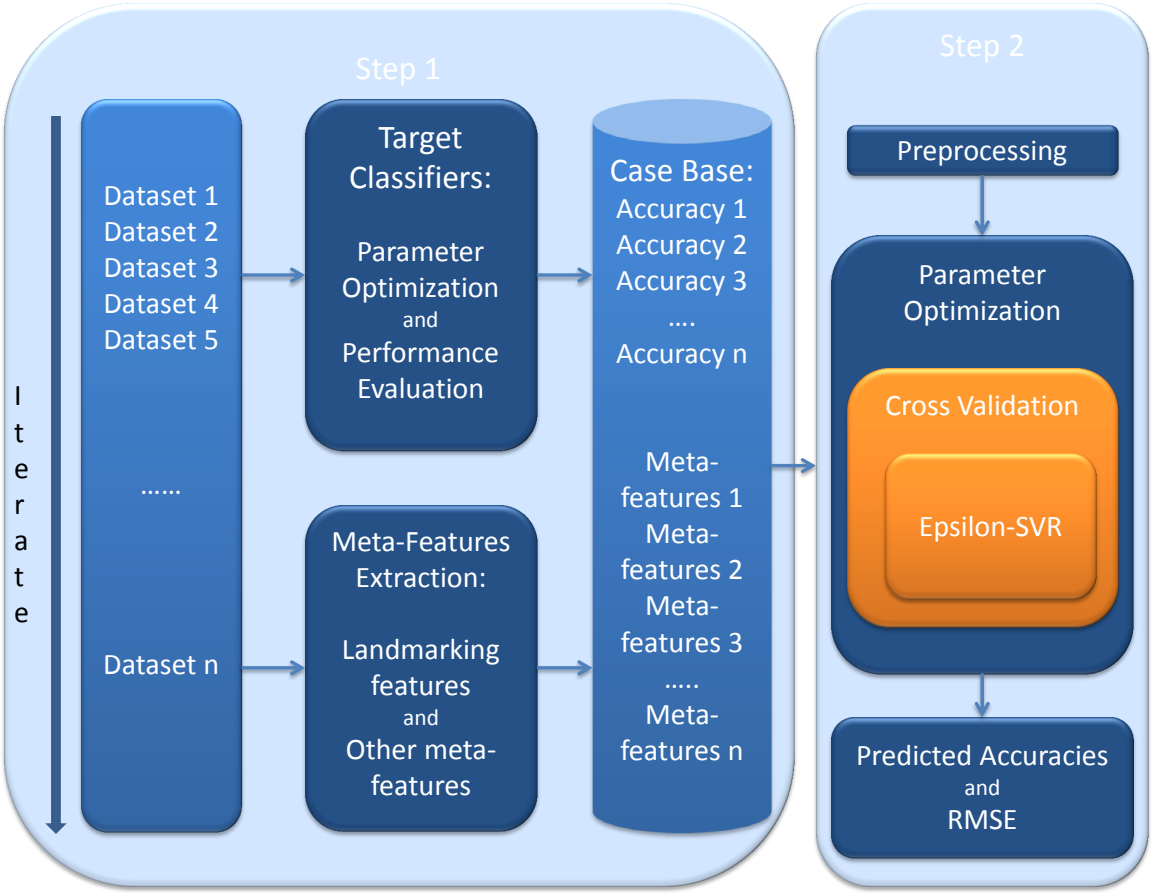


Figure 4.1: Flow Chart for the Evaluation. In this chart the evaluation steps are illustrated. First the meta-features and the accuracies are computed and stored in the *Case Base*. Then a regression model is trained and the predicted accuracies and RMSE are calculated

4.1.1 Case Base Creation

A dataset that includes all the evaluated dataset names along with their meta-features and the computed accuracies for each target classifier on these datasets was considered as a *case base*. For the sake of reproducibility of the *case base*, details about its creation will be provided in the following subsections, however it was created by other PaREn developers.

Classifier	Parameter	Range[Min, Max]	Steps	Scale
Decision Tree	confidence	[1.0E-7, 0.5]	100	linear
LibSVM	γ C	[0.0010, 10.0] [0.0, 50]	100 10	logarithmic logarithmic
K-Nearest Neighbor	K	[1,500]	100	logarithmic
Multilayer Perceptron	Learning rate	[1.0E-5, 1.0]	100	logarithmic
Random Forest Weka	K Depth	[1, 21] [1,2,3,4,5,7,10,20]	10 -	linear -

Table 4.1: The classifiers and their optimized parameters are listed. The grid search parameters (Range, Steps and Scale) are shown for each parameter.

Target Classifiers and Accuracy Computation

The evaluation of the target classifiers was performed in two steps, using RapidMiner pipelines:

1. To achieve more accurate classification models a grid search parameter optimization was applied on important parameters of some target classifiers using a ten-fold cross-validation. A detailed list of the parameters optimized is provided in Table 4.1

2. The accuracy of each of the classifiers was computed for all the datasets using the optimized parameters of the first step. The Decision Trees were set to use Gini index as their classification criterion.

Meta-Features Extraction

The meta-features that were extracted from all the datasets were:

- Simple features: Number of attributes in a dataset, Number of categorical attributes in a dataset, and Number of samples in a dataset.

- Statistical features: Homogeneity of covariances, Kurtosis, Skew, Canonical discriminant correlation (cancor 1, cancor 2), Variation explained by First four canonical discriminants(fract 1, fract 2), Absolute correlation.

- Information theoretic features: Class entropy that was calculated by the METAL DCT tool Petrak (2002).

- Landmarking features : The extraction of the landmarking features was performed using the landmarking operator presented in Section 2.3. The resultant dataset included the evaluated landmarkers' accuracy along with the dataset name as attributes. These attributes were joined with the original *case base* dataset. Different parameter settings for the landmarking operator were experimented, as outlined in Subsection 4.1.3.

Feature	Computed with	Percentage computable
Number of Samples	Python	100%
Number of categorical attributes	Python	100%
Number of numerical attributes	Python	100%
Canonical Correlation 1	R	100%
Canonical Correlation 2	R	100%
Fraction of Canonical Correlation 1	R	100%
Fraction of Canonical Correlation 2	R	100%
Absolute Correlation Coefficient	Python	89.4%
Skewness	Python	100%
Kurtosis	Python	100%
Homogeneity of Covariances (SD Ratio)	Python	80.5%
Class Entropy	DCT	90.2%

Table 4.2: List of data characterization features that were either implemented as part of the PaREn project in R and Python or were available as part of the METAL data characterization toolkit (DCT). The last column, represents the percentage of the datasets for which the corresponding features were extracted usign the specified tool.

The simple, statistical, and information theoretic features were reproduced from the results reported in the StatLog project (King et al., 1995). These features were extracted using either the Python and R algorithms that were implemented as part of the PaREn project, or the DCT toolkit developed by METAL.

In Table 4.2 a list of the most important features and the tools used for their calculation is provided. As illustrated in the last column of this table, the percentage of datasets for which the meta-features are extracted differs from one feature to the other. The features that were not computed are considered as *missing attributes* of the case base, in which all the features and accuracies are stored. In addition, some classifiers' accuracies of different datasets were not computed, mainly due to the unreasonable amount of time they need to be computed. These values that were not computed are called *missing labels*.

4.1.2 Accuracy prediction

To predict the accuracy values of the classifiers, a regression model was trained on the meta-features stored in the *case base* using a *leave-one-out cross-validation*. *LibSVM* of type *ϵ-SVR* and *radial basis function* kernel was used for regression.
To increase the reliability of the regression model, the following steps were performed:

- Missing attributes and labels were filtered from the *case base* dataset. The classifiers' computed accuracies were considered as dataset labels. Any example having

a missing label was excluded from the dataset, as it would not improve the regression model. Furthermore, examples having missing meta-features (represented as dataset attributes) were removed.

- Estimation of the LibSVM parameters **C** and γ was attained by optimization. *Grid search* and *leave-one-out cross-validation* were used for optimization. The search range for C and γ was [1, 500] and [0.0001, 2], respectively. For both parameters, the grid search was performed on a ten step logarithmic scale.

4.1.3 Experiments on Meta-Features

In order to compare the usefulness of meta-features, different experiments were performed:

1. Each feature category described in Subsection 4.1.1 was considered as input features for the regression model.

2. To check which landmarking features are more useful and which settings are better, different settings of the landmarking operator were applied on the datasets. Three experiments were performed using landmarking features extracted by different settings:

   (a) **Experiment 1:** The seven landmakers described in Subsection 2.2.4 were applied on each dataset, using the default settings of the landmarking operator, as outlined in Subsection 2.3.2.

   (b) **Experiment 2:** According to Pfahringer et al. (2000), only four landmarkers were evaluated. These were: One Nearest Neighbor Learner, Decision Node Learner, Randomly Chosen Node Learner and Worst Node Learner. The feature extraction was performed using a ten-fold cross-validation and the datasets were normalized. Note that in this experiment the normal K-NN algorithm is used instead of the the *elite* One Nearest Neighbor that was mentioned in Pfahringer et al. (2000). Table A.2 lists the landmarking features extracted form all the datasets.

   (c) **Experiment 3:** The seven landmarkers were evaluated for all the datasets, using a ten-fold cross-validation and normalization of the datasets.

3. Some combinations of feature categories were provided as meta-features for the regression process. These were:

   - Simple and Statistical Features
   - Simple and Landmarking Features of Experiment 2.
   - All feature categories including the landmarking features extracted in Experiment 2 only.

4.1.4 Evaluation Measurements

Confidence of Prediction

To measure the confidence of prediction of the regression model, two approaches were applied on the results, using RapidMiner functionalities and operators:

1. **Root Mean Squared Error (RMSE):** The RMSE was calculated for each target classifier in each meta-feature category by

$$\text{RMSE} = \sqrt{\frac{\sum_{k=1}^{n}(acc^2_{\text{computed}} - acc^2_{\text{predicted}})}{n}} \tag{4.1}$$

where n is the number of datasets evaluated. In other words, the RMSE gives feedback about how correct the predicted value might be. This measurement was calculated for all the experiments that were performed for the different landmarking features and settings.

2. **Correlation between the Predicted and Computed accuracy:** This measurement gives feedback about the relation between the predicted and computed accuracy. It was measured using the RapidMiner operator *Correlation Matrix*, which gives as a result the correlation matrix between the attributes. In fact, it calculates the *Pearson product-moment Correlation Coefficient (PMCC)*, which returns a value between +1 and -1. The negative values indicate that there is no relation between the attributes. The higher the value of the PMCC, the more correlated are the variables. This coefficient is calculated by:

$$\text{r}_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}} \tag{4.2}$$

where n is the number of variable values from which the correlation is extracted. This correlation was calculated for the experiments performed on the landmarking features only.

Correlation between Landmarkers and Classifiers' Computed Accuracies

An interesting and important point of study is which landmarkers are related to which target classifiers. In this context, the correlation between the landmarking features and computed accuracies of the classifiers was calculated, using the PMCC that is explained above. The case base was an input dataset for the *Correlation Matrix* operator, on which the calculations were performed. A correlation coefficient nearer to the value +1 indicates a high correlation between a landmarker and a classifier. The correlation was computed for each of the experiments that included landmarking features only.

4.2 Results and Discussion

4.2.1 Comparison of all Meta-Features

The first experiment conducted was calculating the RMSE for each meta-features category separately. As a second experiment combinations of meta-features were considered as input features for the regression model as a basis for prediction. The results of these experiments are illustrated in Table 4.3 and the following observations can be drawn:

	RMSE						
Classifier (samples)	Simple[a]	Statistical[a]	Information Theoretic[b]	Landmarking (Pfahringer settings)	Simple and Statistical	Simple and Landmarking	All
Decision Tree ($n = 61$)	0.116	0.111	0.109	**0.070**	0.116	0.117	0.115
LibSVM ($n = 61$)	0.115	0.096	0.109	**0.064**	0.115	0.116	0.116
Naive Bayes ($n = 68$)	0.136	0.121	0.123	**0.079**	0.137	0.132	0.135
Nearest Neighbor ($n = 68$)	0.123	0.105	0.109	**0.053**	0.122	0.122	0.122
Multilayer Perceptron ($n = 57$)	0.118	0.104	0.108	**0.074**	0.118	0.117	0.119
OneR ($n = 68$)	0.162	0.159	0.105	**0.083**	0.167	0.164	0.168
Random Forest ($n = 68$)	0.112	0.101	0.099	**0.051**	0.112	0.109	0.111

[a] Features described in the STATLOG project King et al. (1995).

[b] Features extracted using the METAL data characterization toolkit (DCT) Petrak (2002). More details about the METAL project can be found in http://cordis.europa.eu/esprit/src/26357.htm.

Table 4.3: RMSE for the predicted accuracies based on different meta-feature extraction approaches. Note, that for each classifier the number of samples (datasets) differs due to filtering of missing attributes and labels as explained in Subsection 4.1.2. Each RMSE value is a measure for the confidence of prediction of a classifier. It was found that Landmarking outperforms all other methods with respect to prediction accuracy.

- The number of datasets or samples taken into consideration differs from one classifier to the other. That is a result of the filtering of meta-features and computed accuracies, as explained in Subsection 4.1.2. However, the number of datasets for each classifier was set to a common ground by filtering missing attributes.

- The landmarking features perform best among all the other features for all classifiers, when observing the RMSE values.

- Simple features gave the worst RMSE values compared to the other features, which indicates the ineffectiveness of these features for predicting classifiers' accuracy.

- For the classifiers naive Bayes, K-NN, LibSVM and MLP the statistical features gave the second best confidence of prediction. In addition, information theoretic features showed better RMSE values than simple features for classifiers based on decsion node learners, namely Decision Tree, Random Forest and OneR.

- Combining meta-features from different categories gave less assertive predictions. A possible reason for that may be the increase of the dimensionality of the regression model, due to the increase of the number of features. Therefore, for more input features the number of training datasets should be increased, to have a more reliable regression model. In addition, the more features are combined as input to the regression model, the higher the probability of the occurrance of the problem: curse of dimensionality.

- The meta-feature combinations gave nearly the same RMSE results for all the classifiers. Albeit the best performance of the landmarking features, comparing the results of the landmarking and statistical features combined with simple features we see that the difference between their RMSE values is less than 1%, which was not expected.

This raises up the question of how the features are related to the classifiers and which features of a category affect the prediction's confidence. Further combinations, such as combining statistical and landmarking features, could lead to some interesting results. However, this will deviate from the idea of avoiding complex computations of meta-features by using simple landmarking features.

4.2.2 Evaluation of Landmarking Features

As the results showed that landmarking features gave more promising results than the other features, a more detailed discussion about the landmarking features' experiments will be made in this subsection.

Out of 90 datasets used for the experiments only 86 datasets were included in the *case base* dataset of these experiments, due to the failure of the evaluation of the four datasets: *trains*, *profb*, *splice*, and *mfeat-pixel*. For the trains and profb datasets, the problem occurred when applying the preprocessing operators. For the mfeat-pixel dataset, the landmarking operator was preprocessed successfully only if one landmarker was evaluated at a time. Otherwise, it resulted in an *OutOfMemoryError*. This might have occurred due to the large number of categorical attributes (about 240 categorical attributes) it

	RMSE		
Classifier	Experiment 1 (all landmarkers)	Experiment 2 (four landmarkers, ten-fold cross-validation)	Experiment 3 (all landmarkers, ten-fold cross-validation)
Decision Tree ($n = 76$)	0.075	**0.071**	0.076
LibSVM ($n = 76$)	0.063	**0.061**	0.068
Naive Bayes ($n = 86$)	0.059	0.083	**0.057**
K-NN ($n = 86$)	0.053	0.052	**0.051**
MLP ($n = 70$)	0.072	**0.070**	0.077
OneR ($n = 84$)	0.082	**0.079**	0.084
Random Forest($n = 86$)	0.054	**0.050**	0.053

Table 4.4: RMSE for the predicted accuracies based on different settings of the landmarking operator. The first column lists the classifiers for which the accuracies are predicted and the RMSE is estimated. Note, that for each classifier the number of samples (datasets) differs due to the filtering of missing attributes and labels as explained in Subsection 4.1.2. The RMSE column shows the RMSE, for each classifier evaluated on its corresponding number of datasets. Each RMSE value is a measure for the confidence of prediction of a classifier.

had. The execution of the linear discriminant analysis operator on the splice dataset was not successful neither as an encapsulated operator in the landmarking operator nor independently as a RapidMiner operator; although the preprocessing steps described in Section 4.1 were applied on the dataset in both cases.

From the results shown in Table 4.4 and Figure 4.2.2, the following information can be deduced:

- By filtering the missing labels from the *case base*, the number of datasets taken into consideration differed from one classifier to the other. For example, OneR had two missing labels for the datasets: mfeat-fourier and mfeat-karhunen. For LibSVM, Decision Trees and MLP, 12, 10, and 18 datasets were not evaluated, respectively.

- Comparing the classifiers' confidence of prediction in different experiments, it can be noticed that RMSE ranges are almost equivalent:

  - Experiment 1: [5.3%, 8.4%]
  - Experiment 2: [5.2%, 8.3%]
  - Experiment 3: [5.1%, 8.4%]

- The RMSE values are nearly the same for the classifiers in all the experiments. The difference in the values is less than 1%, except for Naive Bayes.

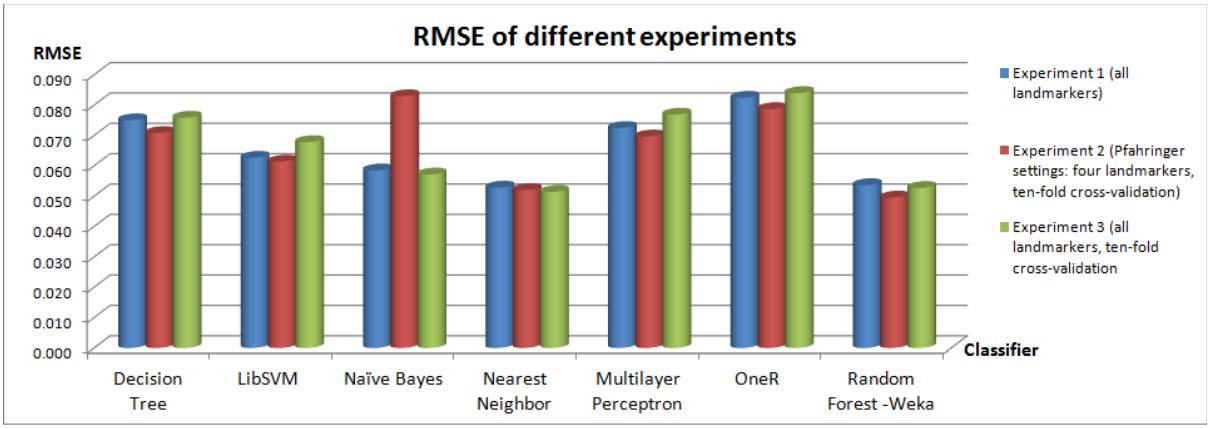


Figure 4.2: The RMSE values of the different experiments of landmarking features are illustrated as bars. Overall, the RMSE values of Experiment 2 are the lowest, indicating the highest confidence of prediction.

- Experiment 1 and Experiment 3, the Naive Bayes classifier has a low RMSE value and the highest correlation coefficient (shown in Table 4.5), indicating the highest confidence of prediction. A possible reason may be its simplicity, due to the independence of attributes or variables taken into consideration in the calculations. It also does not have any parameters that could affect its performance result. Another reason may be due to its inclusion as a landmarker, as its RMSE value compared to the results of Experiment 3 is very low. However, this did not give an RMSE of value zero, because the landmarker is trained and tested on the same data, whereas the evaluation is cross-validated.
- Comparing the results of the different experiments it is observed that Experiment 3 has the lowest RMSE values of the classifiers Naive Bayes and K-NN. For the rest of the classifiers, Experiment 2 showed the most confident predictions.

In Table 4.5 and Figure 4.2.2 the results of the correlation between the computed and predicted accuracies of the classifiers is depicted. Observing these results, the following information can be inferred:

- The computed and predicted accuracies are correlated for all the classifiers.
- The correlation values for all the experiments are nearly the same, having a minimum correlation of 0.78 and a maximum correlation of 0.979.
- For the complex classifiers: LibSVM, MLP, and Random Forest the highest correlations were achieved in Experiment 3 with the values 0.898, 0.859, and 0.898, respectively . However, for the Decision Tree, the computed and predicted accuracies were most correlated in Experiment 1.

	Correlation between predicted and computed acccuracy		
Classifier	Experiment 1 (all landmarkers)	Experiment 2 (four landmarkers, ten-fold cross-validation)	Experiment 3 (all landmarkers, ten-fold cross-validation)
Decision Tree	**0.811**	0.795	0.787
LibSVM	0.882	**0.898**	0.852
Naive Bayes	0.923	0.780	**0.979**
K-NN	0.907	0.907	**0.911**
MLP	0.837	**0.859**	0.827
OneR	0.832	**0.845**	0.820
Random Forest	0.877	**0.898**	0.893

Table 4.5: The correlation between the computed and predicted accuracies for all the classifiers are represented in this table. The correlations were calculated based on the different experiment settings of the landmarker operator. The columns list the Pearson correlation coefficient of the computed and predicted accuracies for each classifier for all landmarking experiments.

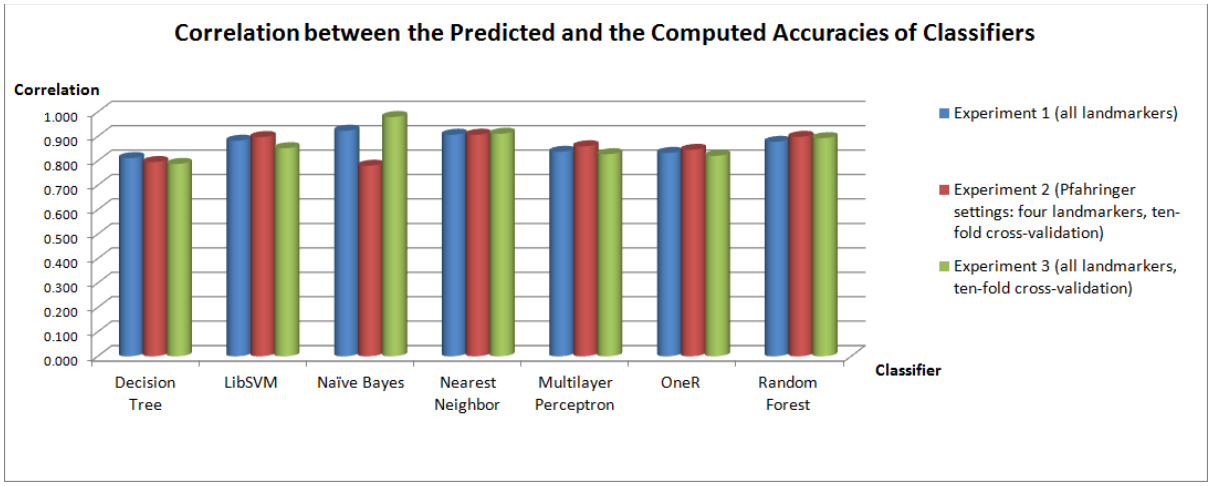


Figure 4.3: The bar chart represents the correlation between the computed and predicted accuracies for the evaluated classifiers. The correlations of the different experiments are all similar. Overall, the correlation of Experiment 2 is the highest compared to the other experiments.

Overall, the RMSE ranges and correlation values of the experiments seem to be promising. The best results for most of the classifiers were achieved by Experiment 2, which was performed based on the report of Pfahringer et al. (2000). As a visualization for the regression results, Figure 4.4 shows a sample of the computed and predicted accuracies for the LibSVM classifier evaluated in Experiment 2.

4.2.3 Suitable Landmarkers for particular Classifiers

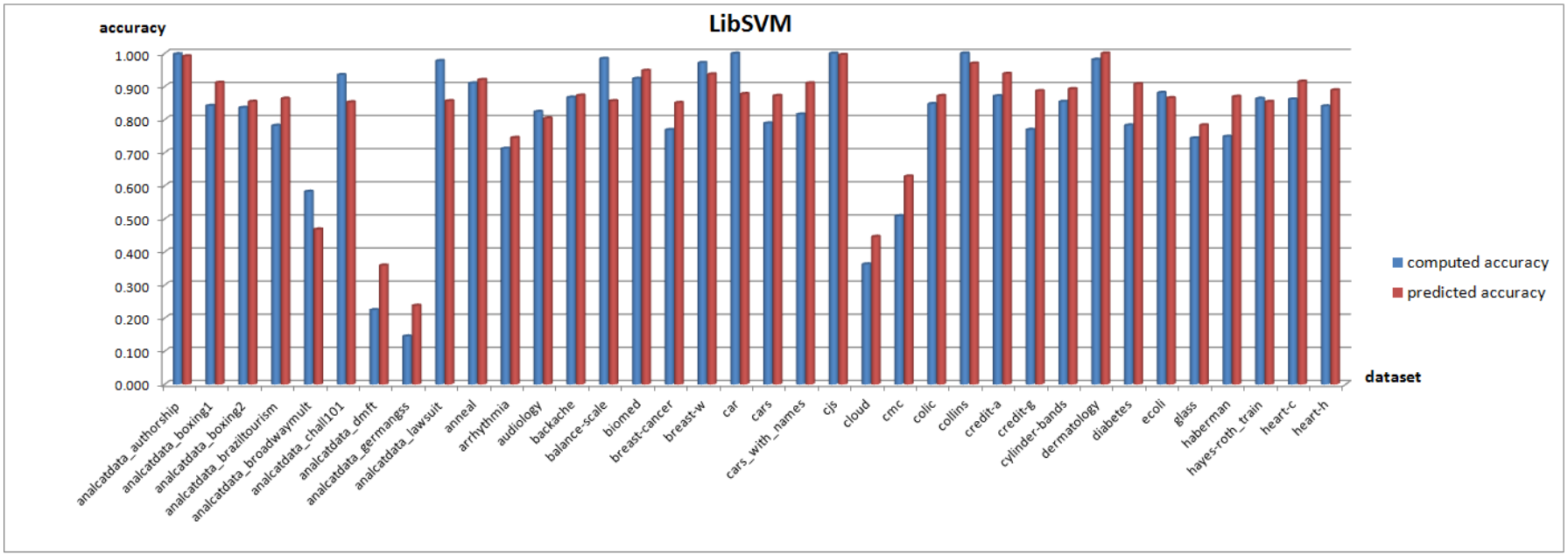
In this subsection, an investigation of which landmarkers are suitable for predicting classifiers accuracy for a given problem is done. As described in Subsection 4.1.4, the correlation between landmarking features and the classifiers acutal accuracies is calculated using the Pearson correlation coefficient. From Table 4.6 and Figure 4.5 the following observations can be deduced:

- Most of the classifiers are highly correlated to One Nearest Neighbor and Naive Bayes. The correlation of the complex classifiers (Decision Tree, LibSVM, MLP, and Random Forest) to Naive Bayes landmarker differs to that of One Nearest Neighbor landmarker by less than 0.1.
  Observing the correlation coefficients of these complex classifiers, it is found that they are most correlated to the One Nearest Neighbor landmarker. A possible reason may be that the accuracy of One Nearest Neighbor classifier indicates the complexity of a given problem. This is generally based on the statement that the *error rate* of One Nearest Neighbor algorithm is approximately asymptotically bound between once and twice the *Bayes error rate*, which is the minimum achievable error rate given the distribution of the data (Cover and Hart, 1967).

- Contrary to our expectations, the landmarkes and classifers based on decision nodes are not highly correlated, except for the OneR classifier. For the Decision Tree and Random Forest, their correlation to the decision node landmarkers (Decision Node, Worst Node, Average Node, and Randomly Chosen Node) is between 0.260 and 0.526. However, for these classifiers, the Decision Node landmarker comes in the third place after One Nearest Neighbor and Naive Bayes, with the correlation values of 0.526 and 0.446.

- The Linear Discriminant landmarker has the lowest correlation to the classifiers compared to the other landmarkers. Its correlation coefficient values are between 0.277 and 0.350, which indicate its inefficiency as input feature for a prediction model.

- The highest correlation is between the Naive Bayes landmarker and classifier having a value of 0.992, which is due to the simplicity of the algorithm, as it has no attributes that should be optimized.



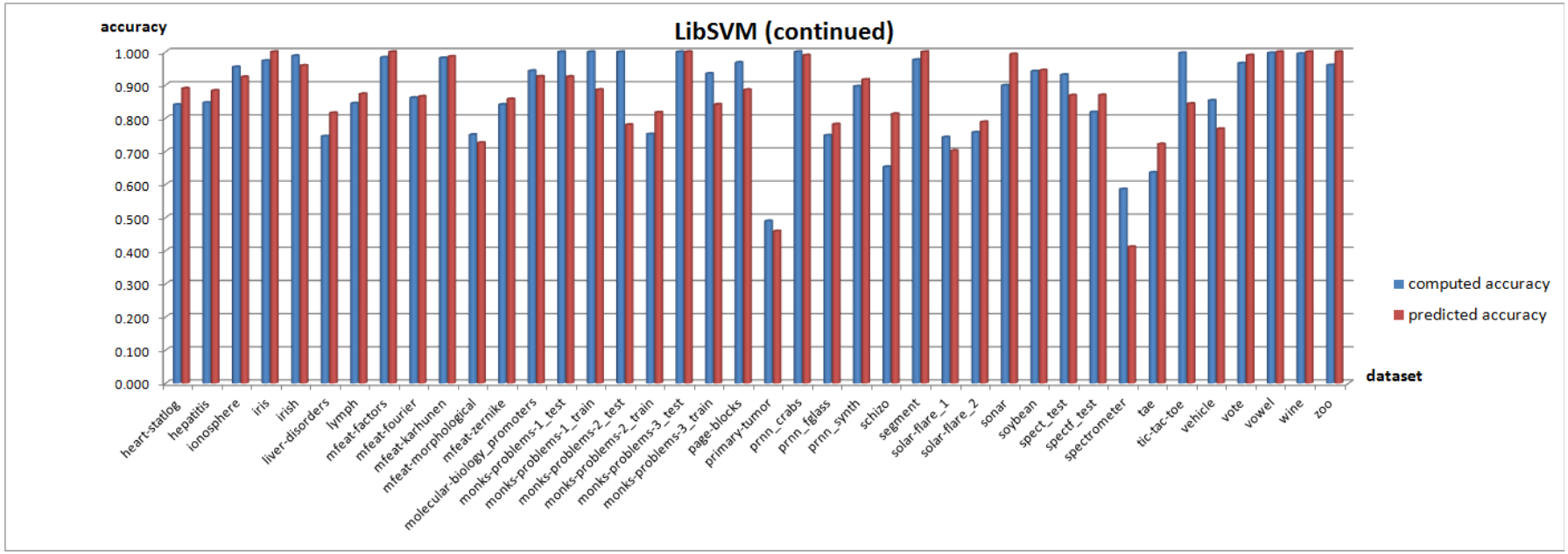


Figure 4.4: Sample charts for LibSVM evaluation in Experiment 2 using regression. The computed and predicted accuracies are plotted for all the datasets using blue and red bars, respectively. These bars show that the computed and predicted accuracy values are close for many datasets. However, for some dataets ,such as, monks-problems-2_test and tic-tac-toe, the differnce between the predicted and computed value is high.

Landmarker	Decision Tree	LibSVM	Naive Bayes	K-NN	MLP	OneR	Random Forest
Linear Discriminant	0.349	0.326	0.277	0.337	0.350	0.319	0.332
One Nearest Neighbor	**0.803**	**0.881**	**0.742**	**0.879**	**0.837**	0.431	**0.878**
Worst Node	0.352	0.316	0.288	0.232	0.334	0.681	0.260
Decision Node	0.526	0.438	0.498	0.406	0.507	**0.835**	0.446
Average Node	0.426	0.343	0.387	0.301	0.400	**0.780**	0.323
Naive Bayes	**0.716**	**0.808**	**0.992**	**0.851**	**0.833**	0.570	**0.839**
Randomly Chosen Node	0.437	0.405	0.401	0.343	0.429	0.734	0.374

Table 4.6: Correlation between landmarkers and the computed accuracy of the classifiers. These results are based on the landmarking features extracted in Experiment 3, where the datasets were normalized and all landmarkers were evaluated using a ten-fold cross-validation. The correlation is calculated using the Pearson correlation coefficient. The table represents the correlation matrix between all the classifiers and land markers specified in the experiment.

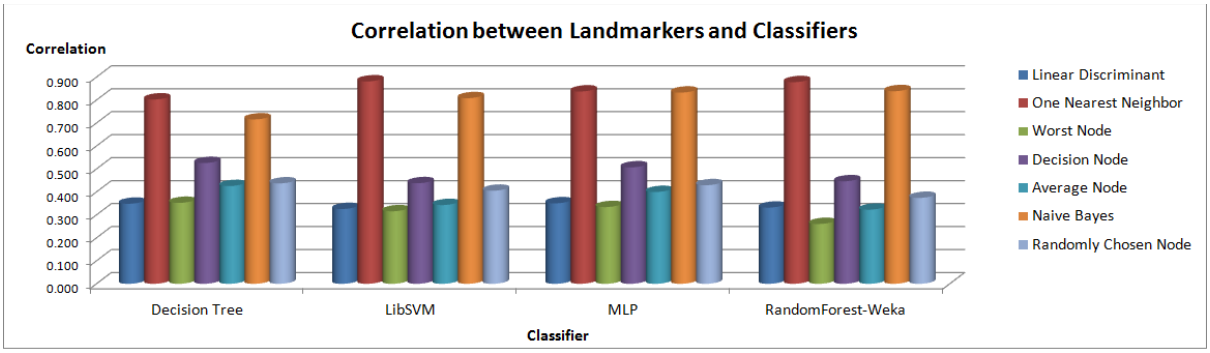


Figure 4.5: The RMSE values of the different experiments of landmarking features are illustrated as bars.

Chapter 5

Conclusion and Outlook

The PaREn project aims to develop tools and algorithms, that make pattern recognition and machine learning easier for non-expert users. In this context, different meta-learning tools were developed. Meta-learning is the process of learning about learning algorithms. Several approaches were developed, trying to map problems to the best performant learning algorithms. In this report, regression was used as a learning model. Meta-features, which are datasets characteristics, were provided as input for the meta-learning model. These meta-features are divided into different categories, namely simple, statistical, information theoretic and landmarking features. The recent experimental features are the landmarking features. The idea behind landmarking is basically trying to relate simple algorithms to more complex ones.

As part of the PaREn project, a landmarking operator was developed using the RapidMiner framework. This operator extracts the landmarking features from a given dataset by basically applying seven fast computable classifiers on it. The landmarkers chosen were: Naive Bayes, Linear Discriminant, One Nearest Neighbor, Decision Node, Randomly Chosen Node, Worst Node and Average Node. Throughout our evaluation illustrated in Chapter 4, it was depicted that landmarking features are well suited for meta-learning. The evaluation was carried out on 90 datasets, mostly from the UCI repository. The meta-features were extracted, in order to predict the accuracy of some classifiers based on regression. By measuring the RMSE and the correlation between the actual and predicted accuracies, it was proved that they had the highest confidence of predictions compared to the other meta-features. As explained in the evaluation, experiments on different landmarking features were performed. Experiment 2, which was based on the experiment described in Pfahringer et al. (2000), had the most assertive prediction for most of the classifiers. In this experiment the RMSE range was [5.2%, 8.3%], which is a highly promising result[1].

In this context, the landmarking features can be considered reasonable for developing

[1] A paper about the developed landmarking operator and the comparison part of the experiments was accepted in the RapidMiner Community Meeting And Conference - RCOMM 2010.

Sarah Daniel Abdelmessih, Faisal Shafait, Matthias Reif, and Markus Goldstein. Landmarking for Meta-Learning using RapidMiner. In RCOMM, Dortmund, Germany, September 2010. (accepted for publication)

systems involving classifier recommendation or automatic classifier selection. In fact, a classifier recommender was developed as part of the PaREn project. This tool was then integrated into a web-interface, called Collaborative User Interface. This application provides different functionalities, such as sharing data and experiments, which may affect the growth of the field of pattern recognition and machine learning positively. Furthermore, it gives the user the possibility to evaluate a problem, by predicting some classifiers' accuracies and visualizing them to the user. The user can then choose the suitable algorithms to be evaluated on a given problem. Currently, the developed application handles XRFF files only. However, it can be simply modified to accept different file types that are supported by RapidMiner as datasets, but it was not implemented due to the lack of time.

An open point of discussion is: which meta-features are related to which classifiers. This correlation could lead to a more reliable approach with confident results. In addition, these correlations can be used to combine different features from different categories, that may result in a regression model with higher confidence of prediction. However, more experiments should be performed to get more meta-data about this correlation.
Another important point of discussion, is finding out the features that are most correlated to the target classifiers and trying to find the best combinations of these features that give a more reliable regression model, for each classifier. Instead of using the same landmarkers for predicting accuracies of each classifier, a subset of suitable landmarkers can be used for that purpose. In this context, automatic feature selection would be a very useful tool. This will make the process dynamic according to the input problem and the available features and algorithms.

Bibliography

D.J. Newman A. Asuncion. UCI machine learning repository, 2007. URL http://www.ics.uci.edu/$\sim$mlearn/{MLR}epository.html.

Shawkat Ali and Kate A. Smith. On learning algorithm selection for classification. *Applied Soft Computing*, 6(2):119–138, 2006.

Hilan Bensusan and Christophe G. Giraud-Carrier. Discovering task neighbourhoods through landmark learning performances. In *PKDD*, pages 325–330, Lyon, France, September 2000.

Hilan Bensusan and Alexandros Kalousis. Estimating the predictive accuracy of a classifier. In *ECML*, volume 2167, pages 25–36, Freiburg, Germany, September 2001.

Pavel B. Brazdil, Carlos Soares, and Joaquim P. da Costa. Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. *Machine Learning*, 50(3): 251–277, 2003.

Ciro Castiello, Giovanna Castellano, and Anna Maria Fanelli. Meta-data: Characterization of input features for meta-learning. In *MDAI*, pages 457–468, Tsukuba, Japan, July 2005.

T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21 – 27, jan. 1967.

Joo Gama and Pavel Brazdil. Characterization of classification algorithms. In *EPIA*, Lecture Notes in Computer Science, pages 189–200, Madeira Island, Portugal, October 1995.

Christophe Giraud-Carrier. Metalearning - a tutorial. 2008.

R.D. King, C. Feng, and A. Sutherland. Statlog: Comparison of Classification Algorithms on Large Real-Worlds Problems. *Applied Artificial Intelligence*, 9(3):289–333, 1995.

Guido Lindner and Rudi Studer. AST: Support for algorithm selection with a cBR approach. In *PKDD*, pages 418–423, Prague, Czech Republic, September 1999.

Donald Michie, David J. Spiegelhalter, and Charles C. Taylor, editors. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, New York, NY, 1994.

Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. Yale: Rapid prototyping for complex data mining tasks. In Lyle Ungar, Mark Craven, Dimitrios Gunopulos, and Tina Eliassi-Rad, editors, *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 935–940, New York, NY, USA, August 2006. ACM. ISBN 1-59593-339-5. doi: http://doi.acm.org/10.1145/1150402.1150531. URL http://rapid-i.com/component/option,com_docman/task,doc_download/gid,25/Itemid,62/.

J. Petrak. The METAL Machine Learning Experimentation Environment V3. 0. 2002.

Bernhard Pfahringer, Hilan Bensusan, and Christophe G. Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *ICML*, pages 743–750, Stanford University, June 2000.

Larry A. Rendell and Howard Cho. Empirical learning as a function of concept character. *Machine Learning*, 5:267–298, 1990.

John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

Kate A. Smith-Miles. Cross-Disciplinary Perspectives on Meta-Learning for Algorithm Selection. *ACM Computing Surveys (CSUR)*, 41(1):25, 2008.

Carlos Soares and Pavel Brazdil. Zoomed ranking: Selection of classification algorithms based on relevant performance information. In *PKDD*, volume 1910, pages 126–135, Lyon, France, September 2000.

Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Reviewww*, 18(2):77–95, 2002.

David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-101, Santa Fe Institute, Santa Fe, NM, 1995.

Appendix A

Appendix

A.1 List of Datasets used

Table A.1: This table lists the names and sources of the datasets used in the evaluation.

Dataset	Source
analcatdata_authorship	StatLib
analcatdata_boxing1	StatLib
analcatdata_boxing2	StatLib
analcatdata_braziltourism	StatLib
analcatdata_broadwaymult	StatLib
analcatdata_chall101	StatLib
analcatdata_creditscore	StatLib
analcatdata_dmft	StatLib
analcatdata_germangss	StatLib
analcatdata_lawsuit	StatLib
anneal	UCI
arrhythmia	UCI
audiology	UCI
backache	StatLib
balance-scale	UCI
biomed	StatLib
breast-cancer	UCI
breast-w	UCI
car	UCI
cars	StatLib
cars with names	StatLib
cjs	StatLib
cloud	StatLib
cmc	UCI
colic	UCI
collins	StatLib
credit-a	UCI

Table A.1: This table lists the names and sources of the datasets used in the evaluation.

Dataset	Source
credit-g	UCI
cylinder-bands	UCI
dermatology	UCI
diabetes	UCI
ecoli	UCI
glass	UCI
haberman	UCI
hayes-roth train	UCI
heart-c	UCI
heart-h	UCI
heart-statlog	UCI
hepatitis	UCI
hypothyroid	UCI
ionosphere	UCI
iris	UCI
irish	StatLib
kr-vs-kp	UCI
letter	UCI
liver-disorders	UCI
lymph	UCI
mfeat-factors	UCI
mfeat-fourier	UCI
mfeat-karhunen	UCI
mfeat-morphological	UCI
mfeat-pixel	UCI
mfeat-zernike	UCI
molecular-biology promoters	UCI
monks-problems-1 test	UCI
monks-problems-1 train	UCI
monks-problems-2 test	UCI
monks-problems-2 train	UCI
monks-problems-3 test	UCI
monks-problems-3 train	UCI
mushroom	UCI
nursery	UCI
optdigits	UCI
page-blocks	UCI
pendigits	UCI
primary-tumor	UCI
prnn_crabs	StatLib
prnn_fglass	StatLib
prnn_synth	StatLib

Table A.1: This table lists the names and sources of the datasets used in the evaluation.

Dataset	Source
profb	StatLib
schizo	StatLib
segment	UCI
sick	UCI
solar-flare_1	UCI
solar-flare_2	UCI
sonar	UCI
soybean	UCI
spambase	UCI
spect_test	UCI
spectf_test	UCI
spectrometer	UCI
splice	UCI
tae	UCI
tic-tac-toe	UCI
vehicle	UCI
vote	UCI
vowel	UCI
waveform-5000	UCI
wine	UCI
zoo	UCI

A.2 Sample of Landmarking Features computed

Table A.2: Landmarking features extracted for all the datasets, using the landmarking operator described in Section 2.3, using its default settings.

Dataset	Linear Discriminant	One NN	Worst Node	Decision Node	Average Node	Naive Bayes	Randomly Chosen Node
analcatdata_authorship	0.999	0.995	0.404	0.553	0.406	0.993	0.377
analcatdata_boxing1	0.467	0.783	0.667	0.833	0.733	0.892	0.700
analcatdata_boxing2	0.833	0.658	0.583	0.833	0.662	0.848	0.583
analcatdata_braziltourism	0.010	0.653	0.774	0.784	0.774	0.786	0.772
analcatdata_broadwaymult	0.782	0.141	0.414	0.509	0.443	0.811	0.414
analcatdata_chall101	0.500	0.726	0.935	0.935	0.935	0.935	0.935
analcatdata_creditscore	0.830	0.710	0.760	0.990	0.790	0.990	0.760
analcatdata_dmft	0.270	0.173	0.227	0.227	0.219	0.276	0.206
analcatdata_germangss	0.388	0.000	0.250	0.275	0.255	0.393	0.250
analcatdata_lawsuit	0.947	0.958	0.932	0.958	0.937	0.970	0.958
anneal	0.762	0.923	0.762	0.802	0.764	0.496	0.762
arrhythmia	0.007	0.537	0.551	0.551	0.548	0.299	0.542
audiology	0.009	0.740	0.257	0.327	0.271	0.973	0.261
backache	0.861	0.794	0.861	0.861	0.861	0.872	0.861
balance-scale	0.870	0.786	0.590	0.590	0.590	0.909	0.590
biomed	0.761	0.900	0.646	0.828	0.719	0.895	0.828
breast-cancer	0.297	0.658	0.703	0.703	0.710	0.759	0.703
breast-w	0.944	0.954	0.790	0.927	0.883	0.959	0.876
car	0.700	0.753	0.700	0.700	0.700	0.874	0.700
cars	0.722	0.746	0.631	0.712	0.648	0.690	0.643
cars_with_names	0.180	0.830	0.631	0.712	0.698	0.938	0.643
cjs	1.000	0.987	0.264	0.365	0.275	0.659	0.244
cloud	0.454	0.316	0.333	0.333	0.310	0.380	0.306
cmc	0.541	0.437	0.427	0.428	0.430	0.508	0.427
colic	0.370	0.750	0.630	0.815	0.654	0.791	0.630

Table A.2: Landmarking features extracted for all the datasets, using the landmarking operator described in Section 2.3, using its default settings.

Dataset	Linear Discriminant	One NN	Worst Node	Decision Node	Average Node	Naive Bayes	Randomly Chosen Node
collins	0.160	1.000	0.162	1.000	0.208	0.998	0.164
credit-a	0.445	0.817	0.555	0.855	0.603	0.783	0.661
credit-g	0.700	0.731	0.710	0.700	0.701	0.772	0.700
cylinder-bands	0.422	0.796	0.580	0.935	0.610	0.985	0.578
dermatology	0.055	0.948	0.358	0.361	0.415	0.992	0.486
diabetes	0.772	0.702	0.656	0.727	0.662	0.762	0.727
ecoli	0.860	0.795	0.440	0.646	0.526	0.860	0.429
glass	0.636	0.701	0.360	0.472	0.390	0.547	0.369
haberman	0.752	0.641	0.742	0.742	0.739	0.765	0.742
hayes-roth_train	0.545	0.705	0.386	0.477	0.455	0.841	0.477
heart-c	0.455	0.742	0.548	0.746	0.649	0.845	0.746
heart-h	0.361	0.762	0.656	0.813	0.675	0.861	0.639
heart-statlog	0.859	0.767	0.559	0.756	0.624	0.859	0.589
hepatitis	0.890	0.793	0.794	0.839	0.803	0.852	0.839
hypothyroid	0.928	0.918	0.923	0.954	0.925	0.957	0.930
ionosphere	0.883	0.878	0.678	0.832	0.713	0.906	0.718
iris	0.867	0.947	0.507	0.667	0.620	0.960	0.640
irish	0.556	0.986	0.556	0.988	0.695	0.982	0.588
kr-vs-kp	0.870	0.851	0.522	0.661	0.551	0.883	0.522
letter	0.655	0.960	0.045	0.045	0.054	0.645	0.045
liver-disorders	0.707	0.629	0.580	0.580	0.588	0.559	0.580
lymph	0.014	0.775	0.547	0.574	0.600	0.865	0.547
mfeat-factors	0.100	0.959	0.101	0.185	0.161	0.934	0.193
mfeat-fourier	0.826	0.799	0.112	0.192	0.119	0.797	0.101
mfeat-karhunen	0.942	0.965	0.154	0.187	0.112	0.948	0.102

Table A.2: Landmarking features extracted for all the datasets, using the landmarking operator described in Section 2.3, using its default settings.

Dataset	Linear Discriminant	One NN	Worst Node	Decision Node	Average Node	Naive Bayes	Randomly Chosen Node
mfeat-morphological	0.741	0.655	0.200	0.196	0.198	0.632	0.199
mfeat-zernike	0.843	0.787	0.102	0.186	0.133	0.763	0.176
molecular-biology_promoters	0.500	0.755	0.528	0.802	0.610	0.991	0.604
monks-problems-1_test	0.653	0.523	0.500	0.750	0.542	0.750	0.750
monks-problems-1_train	0.500	0.649	0.637	0.734	0.591	0.798	0.540
monks-problems-2_test	0.671	0.502	0.671	0.671	0.671	0.671	0.671
monks-problems-2_train	0.639	0.622	0.621	0.621	0.621	0.639	0.621
monks-problems-3_test	0.613	0.905	0.528	0.806	0.616	0.972	0.778
monks-problems-3_train	0.943	0.746	0.541	0.779	0.607	0.934	0.541
mushroom	0.482	1.000	0.564	0.985	0.688	0.995	0.716
nursery	0.333	0.765	0.419	0.710	0.422	0.903	0.419
optdigits	0.939	0.987	0.127	0.195	0.156	0.881	0.193
page-blocks	0.940	0.961	0.900	0.912	0.904	0.904	0.900
pendigits	0.871	0.994	0.187	0.205	0.190	0.858	0.203
primary-tumor	0.569	0.354	0.248	0.248	0.257	0.572	0.248
prnn_crabs	1.000	0.980	0.500	0.615	0.531	0.620	0.505
prnn_fglass	0.636	0.697	0.360	0.472	0.384	0.547	0.369
prnn_synth	0.856	0.880	0.688	0.852	0.770	0.844	0.688
schizo	0.629	0.626	0.600	0.600	0.541	0.588	0.521
segment	0.705	0.969	0.144	0.286	0.234	0.800	0.146
sick	0.951	0.962	0.939	0.966	0.940	0.929	0.939
solar-flare_1	0.728	0.613	0.272	0.526	0.338	0.731	0.303
solar-flare_2	0.219	0.693	0.349	0.614	0.381	0.770	0.614
sonar	0.880	0.865	0.534	0.755	0.553	0.731	0.543

Table A.2: Landmarking features extracted for all the datasets, using the landmarking operator described in Section 2.3, using its default settings.

Dataset	Linear Discriminant	One NN	Worst Node	Decision Node	Average Node	Naive Bayes	Randomly Chosen Node
soybean	0.029	0.915	0.221	0.228	0.214	0.952	0.206
spambase	0.865	0.908	0.606	0.782	0.641	0.821	0.606
spect_test	0.925	0.904	0.920	0.920	0.920	0.845	0.920
spectf_test	0.829	0.706	0.796	0.796	0.796	0.691	0.796
spectrometer	0.002	0.381	0.130	0.105	0.108	0.610	0.109
tae	0.576	0.623	0.411	0.377	0.384	0.583	0.411
tic-tac-toe	0.653	0.674	0.653	0.653	0.659	0.698	0.653
vehicle	0.796	0.693	0.265	0.400	0.301	0.473	0.260
vote	0.959	0.921	0.614	0.956	0.751	0.903	0.614
vowel	0.581	0.992	0.091	0.168	0.107	0.714	0.168
waveform-5000	0.869	0.733	0.339	0.573	0.411	0.802	0.339
wine	1.000	0.950	0.444	0.607	0.567	0.989	0.596
zoo	0.040	0.970	0.406	0.604	0.538	1.000	0.505