A comparison of SVM and Rule- and Decision Tree learning

Vergleich von SVM und Regel- und Entscheidungsbaum-Lernern

Bachelor-Thesis von Chahine Elyes Abid

November 2013

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
The Knowledge Engineering group

A comparison of SVM and Rule- and Decision Tree learning
Vergleich von SVM und Regel- und Entscheidungsbaum-Lernern

Vorgelegte Bachelor-Thesis von Chahine Elyes Abid

1. Gutachten: Johannes Fürnkranz
2. Gutachten: Frederik Janssen

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 14. November 2013

(Chahine Elyes Abid)

Contents

1 Introduction **3**
- 1.1 Motivation . . . 4
- 1.2 Objective of the thesis . . . 4
- 1.3 Structure of the thesis . . . 4

2 Learning Techniques **5**
- 2.1 Decision Tree Learning . . . 5
  - 2.1.1 ID3 Algorithm . . . 5
  - 2.1.2 Pruning . . . 7
  - 2.1.3 C4.5 Algorithm . . . 8
- 2.2 Ensemble learning . . . 9
  - 2.2.1 Random Forest Algorithm . . . 9
- 2.3 Rule Learning . . . 10
  - 2.3.1 Reduced error pruning . . . 11
  - 2.3.2 Incremental reduced error pruning . . . 11
  - 2.3.3 Repeated incremental pruning to produce error reduction . . . 11
- 2.4 Support Vector Machine . . . 12
  - 2.4.1 The maximal margin classifier . . . 12
  - 2.4.2 The non separable case: Soft margin . . . 14
  - 2.4.3 Non Linear Case: Kernels . . . 14

3 Experiments: Tuning algorithms parameters **16**
- 3.1 Datasets . . . 16
- 3.2 Symbolic algorithms . . . 18
  - 3.2.1 J48 . . . 18
  - 3.2.2 RandomForest . . . 21
  - 3.2.3 JRip . . . 23
- 3.3 Support Vector Machine . . . 26
  - 3.3.1 SMO . . . 26
  - 3.3.2 LibSVM . . . 31

4 Comparing the algorithms **35**
- 4.1 Average, standard deviation and variance . . . 35
- 4.2 Default settings and optimized parameters . . . 37
- 4.3 Statistical comparison - Significance test . . . 39

5 Summary and Conclusion **42**

1 Introduction

Machine learning comprises of building programs to solve problems and to improve the performance of solutions based on examples and experiences over time. Implementing such programs is of great importance since some tasks require solutions which change in time and involve human expertise.
For example, problems like speech recognition, medical diagnosis and bioinformatics belong to this category, and we need machine learning algorithms to deal with them, as classical techniques are not able to solve such problems [1][2][3].

A usual task in machine learning is supervised learning. In this task, the information provided to the learner takes the form of a set of examples (or instances), called training set, which are input/output pairs. The input of each instance consists of the values of attributes that measure different aspects of the instance. The output is the associated class (i.e the spam or not spam judgment in the spam filtering problem constitute the class of the example) [4].

In the weather data [5], an instance has the following form :

$$< outlook = sunny,\ temperature = 85,\ humidity = 90,\ windy = TRUE,\ play = no >$$

Each example is described with values for all its attributes: *outlook*, *temperature*, *humidity* and *windy*. The class of this instance is *play* which has the value no. Attributes consist of two main types: nominal or numeric. Nominal attributes (also called categorical) take on values in a prespecified, finite set of values. For example, *outlook* is a nominal attribute, and its values are *sunny*, *overcast* and *rainy*. On the other side, *temperature* and *humidity* are numeric attributes. This kind of attributes (also called continuous attributes) measures numbers (real or integer) [4].

Furthermore, classification and regression are supervised learning problems, where a training set is available, and the task is to learn the mapping from the input to the output. The approach in machine learning is that we assume a model defined up to a set of parameters:

$$y = g(x|\theta)$$

where: $g()$ is the model, and θ its parameters. x is the input (set of attributes).
y represents the class in classification ($0/1$ in binary classification), i.e the objects should be categorized into separate categories (classes). The problem of separating the instances into more than two classes is handled by multi-label classification. In the regression task, y is a number (a real value should be predicted). In classification, $g(.)$ represents the discriminant function which assigns the appropriate class (an example of a classification task is spam filtering). In regression, g(.) is called the regression function [2]. An example of a regression task can be to predict how high the temperature will be tomorrow.

In this thesis, we will rather focus on the classification problem. In this case, $g(.)$ is referred to as the classification function. After processing all the training instances, the learning phase is completed. The model $g()$ is then able to classify new instances (called test set). These instances have not been used in the formation process of the model.

In general training and testing instances are provided together in one set, which will be split randomly into a fixed number of partitions (called folds). The folds have approximately the same size, and each one is used in turn for testing and the rest of the partitions are used for training, in this way every example in the dataset will be used exactly one time during the testing process. This technique is called *cross-validation* and the standard number of folds is 10 [4].

An important procedure used during cross-validation is *stratification*, and it signifies that each class in the dataset should be represented roughly in the same proportion in every fold. This helps the model to perform better results, without having to classify instances whose classes did not take part in the building process of the model.

The machine learning program builds the model $g()$ using the nine-tenths of the data for learning. During testing, the correctly classified instances in the remaining fold are calculated, to compute the values of accuracy and error:

$$Accuracy = \frac{Number\ of\ correctly\ classified\ examples}{Number\ of\ examples} \tag{1}$$

$$Error = 1 - Accuracy \tag{2}$$

The 10 accuracy estimates are averaged to deliver an overall accuracy estimate.
During the learning scheme, the model optimizes its parameters θ, such that the approximation error is minimized. If this is achieved, the accuracy is maximized and the model is said to generalize well to unseen data: this means it is very likely to make correct predictions for novel instances.

The generalization may be affected by incompleteness or anomalies (also called noise) contained in real-world data, from which we learn. Some data cleaning methods can be used, in order to handle these types of anomalies. In general, if there is noise, it will confuse the construction of the model; this will describe the noise instead of just the underlying function, and it may become harder to classify the unseen examples correctly. This phenomenon is called *overfitting*. Therefore, a model that overfits the data will generally have a poor accuracy measure, i.e it won't generalize well on unseen examples. Overfitting may be avoided by building models with a complexity as small as possible [2][4][6]. However, there is a solution to reduce the complexity of the classifier, known as *pruning*. This technique will be described in section 2.1.2.

1.1 Motivation

Support vector machines (SVMs) are supervised learning models used for classification. Generally, they show good performances when their parameters are fitted to the data. Meanwhile, symbolic models (such as algorithms for building decision trees, rule learning or ensemble learning algorithms) are often used in their standard configuration without appropriate adjustment of their parameters; that makes them not perform as good as SVMs.
However, when dealing with real world classification tasks, symbolic approaches are preferred as they are quite interpretable, which is generally not the case of support vector machines. Therefore, we want to know how far the performance of symbolic approaches can be improved when we optimize their parameters in a similar way as they are optimized for SVMs.

1.2 Objective of the thesis

In this thesis, we give an overview and a comparison of several existing classification techniques. The main objective is to see whether decision trees, rule learners and ensembles can be as accurate as support vector machines once their parameters are thoroughly tuned. In order to answer this question, we perform an empirical study which involves optimizing parameters of the different models using the same amount of data.

1.3 Structure of the thesis

In what follows, a short summary of the sections is presented:
Chapter 2 introduces the various learning techniques.
In chapter 3, we optimize the parameters of the algorithms (implementing the techniques described in chapter 2) in order to get the best accuracy values.
Chapter 4 presents a comparison between the different studied approaches.
Finally, in chapter 5, we will give a summary of the results we obtained and conclude this work.

2 Learning Techniques

There are several techniques to perform classification. In this section we explain the basic ideas behind some symbolic approaches used in classification and support vector machine algorithms. We will present their different ways of learning, as well as the characteristics of each model. The overview of the described algorithms is based mainly on the explanations from [4].

2.1 Decision Tree Learning

A decision tree is a kind of supervised learning method; it has a recursive structure consisting of nodes and leaves. Every node has the function of testing a particular attribute, while leaves represent the different classes of the data, so every leaf gives a classification for instances that reach it. To classify an instance, we start from the root and browse through the tree according to the values of the attributes which will be tested in the nodes. When a leaf is reached, its class is assigned to the example [4].
Figure 1 presents an example of decision tree for the weather dataset.

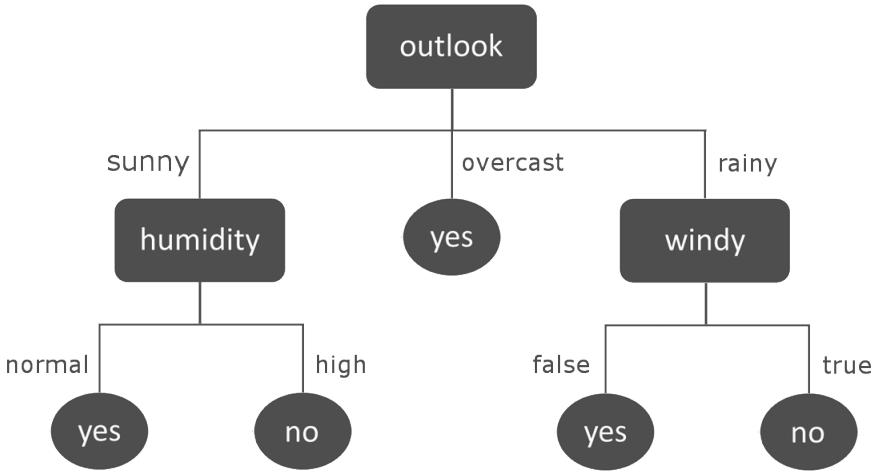

Figure 1: Decision tree for the weather data.

Given a set of instances, it is possible to build different decision trees. However, the target for the classification task is to construct a decision tree which generalizes well to new examples.

2.1.1 ID3 Algorithm

The ID3 method [5] constructs a simple and efficient decision tree. Before explaining how to determine which attributes should be set for each node, some definitions are needed: Entropy measures the quantity of information contained in every node. A high entropy characterizes an equally distribution of the classes.

$$Entropy(S) = \sum_{i=1}^{c} -p_i \, log_2 \, p_i \tag{3}$$

where S is the set of examples, c is the number of the classes of the learning problem, and p_i is the proportion of examples that belong to the i-th class.

As an example, in the weather dataset, we have 14 instances: 9 are positive (classified as Sport = yes) and 5 are negative (Sport = no), so the corresponding entropy is:

[0] [4] p. 101

$$Entropy([+9,-5]) = -\frac{9}{14}\, log_2\,(\frac{9}{14}) - \frac{5}{14}\, log_2\,(\frac{5}{14}) = 0.94$$

To determine which attribute should be tested at a given node, the information gain is calculated for each attribute A, and the one that has the highest value is the one that gains the most information and will be selected.

$$Gain(S,A) = Entropy(S) - \sum_i \frac{S_i}{S} \cdot Entropy(S_i) \tag{4}$$

For the weather data, the *outlook* attribute is the one that gains the most information, and is selected to be the root of the tree. In what follows, we will explain in details the steps for building the rest of the tree. After choosing the root, the tree will have the form presented in figure 2.

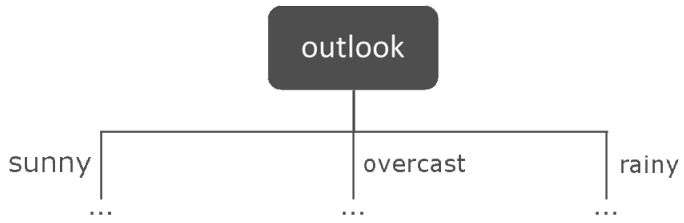

Figure 2: Determining the root of the decision tree for the weather data.

For each value of *outlook*, we obtain a corresponding branch. Table 1 lists the instances for which *outlook* is *rainy*.

Table 1: Examples from the weather data.

outlook	temperature	humidity	windy	play ?
rainy	cool	normal	false	yes
rainy	mild	normal	false	yes
rainy	mild	high	true	no
rainy	cool	normal	true	no
rainy	mild	high	false	yes

There are 3 positive and 2 negative examples, and we calculate the entropy for the case *outlook* = *rainy* as follows:

$$Entropy([3+,2-]) = -\frac{3}{5} \cdot log_2(\frac{3}{5}) - \frac{2}{5} \cdot log_2\,(\frac{2}{5}) = 0.971$$

Now we calculate the information gain for the attributes *temperature*, *humidity*, and *windy*. We obtain the following values:

$$Gain(S_{rainy}, temperature) = Entropy([3+,2-]) - [\,\frac{2}{5} \cdot Entropy([1+,1-]) + \frac{3}{5} \cdot Entropy([2+,1-])] = 0.02$$

$Gain(S_{rainy}, humidity) = 0.02$

$Gain(S_{rainy}, windy) = 0.971$

We notice that *windy* is the attribute that gains the most information, and it is selected at the next node (see figure 3).

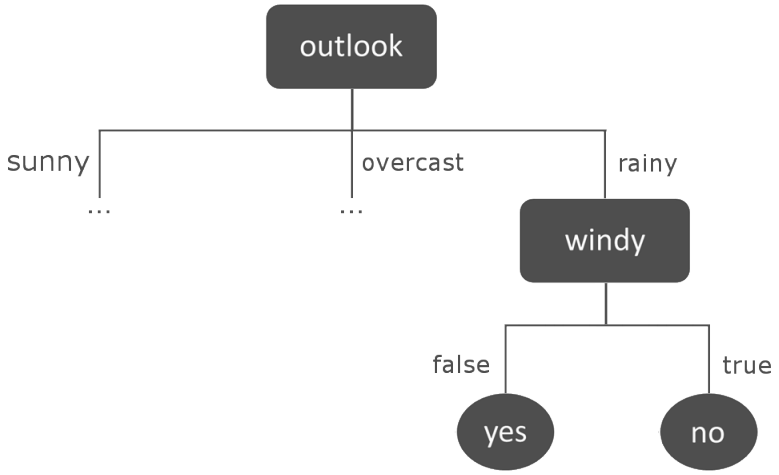

Figure 3: Branch of the decision tree for the weather data.

We obtain too pure nodes (containing instances form the same class). So there is no need to split any further at this branch. We repeat the same process recursively in the other branches until there are no more examples available and we obtain the final tree represented in figure 1.

2.1.2 Pruning

Pruning is a technique to handle the problem of overfitting the data; it reduces the size of the decision tree by removing the branches that may be based on noisy examples, in order to reduce the generalization error on unseen data. There are two approaches to perform tree pruning:
Pre-pruning (also called forward pruning) consists of stopping the tree-construction process early by deciding to not develop a tree partition any further if the information becomes unreliable.
The second strategy to get simpler trees is called post-pruning (or backward pruning). The idea is to grow first a full decision tree, until all leaves are pure and all training examples are classified correctly. Then, two different operations can be performed to postprune. One is subtree replacement: the tree is pruned by replacing some branches with leaf nodes. The most represented class in the subtree being replaced, is assigned to the leaf [2][4][6]. The other option consists of raising a whole subtree to replace another one, it is known as subtree raising (see figure 4). Subtree C is raised to replace node B, and all instances of leaves 4 and 5 are redistributed into the node C [4].
Most decision tree builders use the post-pruning strategy, because it generally leads to the construction of more accurate trees. However, pre-pruning is faster [2][4].

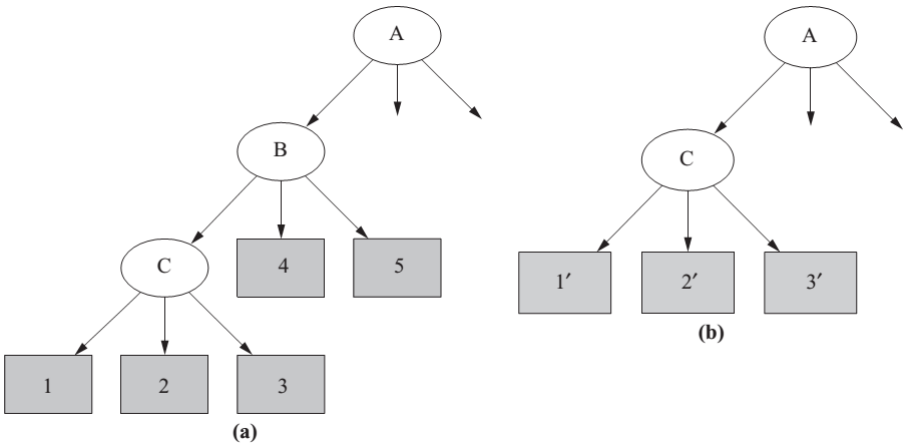

Figure 4: Example of subtree raising.[1]

2.1.3 C4.5 Algorithm

The C4.5 Algorithm [7] is an extension of ID3 which is able of learning from real-world data. It includes methods for handling noisy examples, numeric attributes and missing values for some attributes [4]. This algorithm uses the pruning technique to handle overfitting and remove branches that provide bad results during the classification task.

In order to decide which pruning strategy should be performed, the C4.5 estimates the error at different nodes based on some statistical reasoning. This is known as Error-based Pruning. The user determines a confidence parameter c (default 25%), and a confidence interval is calculated from training data. The upper bound of this interval is used to estimate the error rate e, which leads to a pessimistic error estimate [4]. Starting from the bottom of the tree, we examine each non-leaf subtree. If the combined error estimate of the leaves is greater than the error rate at the parent node, the leaves are pruned away (subtree replacement) [7]. Otherwise, the subtree is still unchanged and we browse back through the tree in the direction of the root, to check if pruning should be realized in another branch. It is also possible that the error estimate of a non-leaf subtree is lower than that of its parent node, in this case the subtree is raised to replace the parent node (see figure 4) [4].

[1] [4] p. 194

2.2 Ensemble learning

Before exposing the characteristic of ensemble learning, the definitions of bias and variance are needed. Bias measures the difference between the model and the learning problem (building a bad model increases the bias). In the other hand, the variance is the part of error caused by the data set. The main aim is to reduce both variance and bias in order to build a model that is adapted to any given decision boundary, without being highly affected by the variations in the data. This will reduce the total expected error (bias and variance) of the model [4].

The idea is to learn multiple classifiers instead of a single one, and then combine their predictions. It is realized by bagging [8]: this technique involves the random generation of a sample of training sets of the same size from the original one, and learning a classifier from each sample. During the testing step, the classifiers are combined: this means the class that is predicted the most from the different classifiers is selected to be the ensemble model class. This method is known as voting and is realized for every test instance [4].

2.2.1 Random Forest Algorithm

The random forest algorithm [9] is based on bagging: after building multiple training sets, the algorithm constructs one tree per training set. During the growing of the trees, the best split at each node is selected among a random subset of features and not among the whole attributes like it is the case in the construction of decision trees. This technique is called random split selection and the trees are known as random trees. Doing so, the attributes, which do not have the highest information gain value, can be selected for splitting at a particular node, and their combination with other attributes may give a better prediction. After constructing a multitude of decision trees, the various outputs are merged into a single prediction, which is used to perform the classification task. By using the right type of randomness, the classifier is likely to achieve good accuracy results [9].

The random forest method deals well with continuous and categorical attributes. Moreover, it helps to avoid overfitting and it reduces the variance by making the decision boundary smoother and more stable when there are variations in the data [10].

2.3 Rule Learning

In this section we present another family of learning methods known as rule learning. A rule has an if-then structure. The if-part (called the rule body) contains a conjunction of conditions in the form of attributes tests, and the then-part (rule head) specifies the predicted class [6]. If the example fulfills the conditions of the rule, we say that the example is covered by this rule. Rules without conditions in their body are called empty rules.

Generally a single rule can not classify all training instances (see figure 5), and other rules have to be learned and added to the rule set in order to formulate the target theory (hypothesis). This represents the concept description and should be complete (all positive examples are covered) and consistent (no negative example should be covered).
Example: Let us suppose we want to find a theory that explains the six instances stated in table 2 from the iris data [11]. It contains four numerical attributes (sepallength, sepalwidth, petallength and petalwidth) and the class that predicts the iris plant (Iris-setosa, Iris-versicolor or Iris-virginica).

Table 2: Examples from the iris data.

sepallength	sepalwidth	petallength	petalwidth	class
4.8	3.4	1.9	0.2	Iris-setosa
5.0	3.5	1.6	0.6	Iris-setosa
6.3	2.9	5.6	1.8	Iris-virginica
7.1	3.0	5.9	2.1	Iris-virginica
5.0	2.3	3.3	1.0	Iris-versicolor
6.3	2.5	4.9	1.5	Iris-versicolor

The following rules can be learned from the examples in table 2:

$$\begin{aligned} &\text{if (petallength} \leq 1.9) && \text{then class = Iris-setosa} \\ &\text{if (petallength} \geq 3.3) \text{ and (petallength} \leq 4.9) && \text{then class = Iris-versicolor} \\ &\text{if (petallength} \geq 5.0) && \text{then class = Iris-virginica} \end{aligned}$$

Figure 5: A set of rules classifying instances from the iris problem.

It is possible to extract a set of if-then rules from a decision tree. For each path from the root to a leaf node, a rule is generated by making a conjunction of the tests in every node. However, most rule learning algorithms are based on a separate and conquer approach. The first step consists of searching one or more rules to cover the examples of each class and separates them from the rest; this is the separate part of the technique. During the conquer part, the algorithm tries recursively to learn other rules that describe the remaining training instances until all instances in the training set are covered [4][6][12].

In general, the obtained rule learning classifier overfits the training data. The model has a high complexity and does not generalize well on unseen instances. As described in section 2.1.2, the pruning technique is a solution to handle the problem of overfitted models. Complex rules will be refined in order to maximize the accuracy on future data [4][12]. As for decision trees, there are two pruning strategies: Pre-pruning consists of simplifying the rules during the learning process, while post-pruning is performed after learning. We will describe some pruning algorithms in the next sections.

2.3.1 Reduced error pruning

Reduced error pruning (REP) is a post-pruning algorithm introduced in [13]. Its basic idea is to split the training set into two parts called a growing set and a pruning set. The first step consists of learning complete and consistent rules in the growing set, based on separate and conquer approach, like it was described above. After that, the hypothesis is simplified by pruning each rule as long as the error does not increase in the pruning set. Then, the rule is added to the rule set and all covered instances are removed from both growing and pruning sets [4][14]. In general, two-thirds of training data are used for growing the rules, and the rest for pruning.

It was shown in [13] that reduced error pruning produces precise theories and is independent of the algorithm used for learning the concept. However, it has a high time complexity ($O(n^4)$) and the fact of splitting the training data into a growing set and a pruning set, may have a negative impact on the performance of the algorithm [14][15].

2.3.2 Incremental reduced error pruning

Incremental reduced-error pruning (IREP) is a rule learning algorithm introduced in [15]. The idea is to integrate pre- and post-pruning to handle the inefficiency problems of REP: instead of learning a complete and consistent thesis from the training data, and prune it after (as is the case in REP), every rule will be pruned right after it has been learned. Then the final rule is added to the theory and all covered examples are removed. Finally, the training set is split once again into two parts (for growing and pruning the rule), a new rule is generated and the process is repeated until there are no instances available. This idea forms the basis for an algorithm called RIPPER, which will be discussed in the next section [15].

Unlike the reduced error pruning, this method ensures that after the pruning of each rule, all examples covered by it are removed from the training set, and don't play a part during the learning of future rules. Moreover the IREP algorithm has a lower complexity ($O(n \cdot log^2 n)$) since it does not grow a complete overfitted theory, but prunes every rule immediately after it has been generated [15][16].

2.3.3 Repeated incremental pruning to produce error reduction

RIPPER, an acronym for repeated incremental pruning to produce error reduction [14], is an algorithm based on the same principle as IREP. However the two algorithms differ in their stopping criterion. In IREP, this criterion is based on the accuracy of the rule on the pruning set: every rule is pruned right after it has been learned, until its accuracy is maximized, but only rules which error don't exceed 50% (rules that cover more positive than negative examples) are added to the rule set [15]. On the other hand, RIPPER has an alternative stopping criterion based on the MDL (minimum description length) principle described in [17]. This heuristic assumes that the best hypothesis is the one that captures the most regularity between the examples in the training set (the data can be well compressed) [17][18].

Moreover, in comparison with IREP, the RIPPER algorithm adds a new post-pruning phase, called rule optimization. The rules are considered in turn in the order in which they were created. For every rule R_i from the hypothesis $(R_1, R_2, ..., R_i, ..., R_n)$, two alternative rules are generated:
One is called the replacement for R_i and is obtained by growing a rule $R_i^{'}$ and pruning it to minimize the error of the hypothesis $(R_1, R_2, ..., R_i^{'}, ..., R_n)$ on the pruning set.
The second one is formed in a similar way by refining R_i and adding new conditions to it, instead of starting of an empty rule. It is called the revision of R_i. After generating the replacement and the revision of the original rule R_i, the algorithm decides based on the MDL heuristic, whether to keep R_i in the rule set or to replace it with one the new rules. It is also possible to iterate the optimization phase by repeatedly optimizing the hypothesis [14].

2.4 Support Vector Machine

Support Vector Machines (SVM) are a supervised learning models used in both classification and regression tasks. These methods are helpful to solve multiple real world applications such as bioinformatics and handwritten digit recognition [3]. In this section, we mainly use the following books: [2], [3] and [19] to describe the SVM technique.

2.4.1 The maximal margin classifier

Let $D = \{(x_i, y_i), \quad i = 1, ..., N\}$ denote the training set. For every example in D, x represents the attribute vector and y is the corresponding class ($y \in \{-1, 1\}$) [20]. In order to classify positive and negative examples, the main idea is to find a hyperplane that performs their separation by splitting the input space X into two half spaces, each containing instances of the same class.
The SVM algorithm operates the classification task, using a function:

$$f(x) = \langle w \cdot x \rangle + b$$

where b is the bias and w represents the vector normal to the separating hyperplane.
The sign function of f represents the decision function that we use for classification: the input vector x is assigned to the positive class if $f(x) \geq 0$, and to the negative class otherwise. In other words, the example (x, y) is classified correctly if: $y \cdot f(x) > 0$ (i.e y and $f(x)$ have the same sign). While $f(x) = 0$ corresponds to the input vectors x that lie on the hyperplane [3].

The separating hyperplane H is defined by the equation $\langle w \cdot x \rangle + b = 0$, so we consider that all examples (x_i, y_i) satisfy the following constraints:

$$\begin{array}{ll} \langle w \cdot x_i \rangle + b \geq +1 & \text{for } y_i = +1 \\ \langle w \cdot x_i \rangle + b \leq -1 & \text{for } y_i = -1 \end{array}$$

These constraints can be rewritten as:

$$y_i \cdot (\langle w \cdot x_i \rangle + b) \geq +1 \qquad \forall i \tag{5}$$

If the examples are linearly separable, it is always possible to find many hyperplanes. However, the support vector machine method looks for an optimal hyperplane H that maximizes the distance of the closest examples from both sides of the space to it. This distance is called margin and the SVM model that aims to maximize the margin is known as the maximal margin classifier. By maximizing the margin, this model guarantees a good generalization [2].
Furthermore the distance that separates an example (x_i, y_i) from the hyperplane is given by the following equation:

$$d_i = \frac{|\langle w \cdot x_i \rangle + b|}{\|w\|}$$

which, when performing binary classification, can be written as:

$$d_i = \frac{y_i \ (\langle w \cdot x_i \rangle + b)}{\|w\|}$$

Hence the margin is: $\dfrac{1}{\|w\|}$
So to find a hyperplane that maximizes the margin, $\|w\|$ should be minimized, which is equivalent to solve the following quadratic optimization problem:

$$\begin{cases} minimize\ \|w\|^2 \\ y_i \cdot (\langle w.x_i \rangle + b) \geq 1 \qquad i = 1, ..., N \end{cases} \tag{6}$$

This constrained problem is a primal optimization problem that can be solved with the help of the method of Lagrange multipliers. This method aims to minimize a function subject to equality constraints by adding the objective function to a linear combination of the constraints:

$$L_p = \frac{1}{2}\langle w.w \rangle - \sum_{i=1}^{N} \alpha_i \left[y_i \left(\langle w.x_i \rangle + b \right) - 1 \right] \tag{7}$$

where $\alpha_i \geq 0$ are the coefficients of the combination and they are called the Lagrange multipliers. L_p is known as the primal Lagrangian. Minimizing L_p is equivalent to maximizing the corresponding dual problem; this can be obtained by differentiating L_p with respect to w and b. We obtain:

$$\sum_{i=1}^{N} y_i \alpha_i = 0$$

$$w - \sum_{i=1}^{N} y_i \alpha_i x_i = 0 \iff w = \sum_{i=1}^{N} y_i \alpha_i x_i$$

After that the weight vector w is described as a linear combination of the training points, the function f can be rewritten as follows:

$$f(x) = \sum_{i=1}^{N} y_i \alpha_i \langle x_i.x \rangle + b$$

The relations obtained by differentiating L_p are plugged into the primal equation to form the corresponding dual Lagrangian:

$$L_d = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j \langle x_i.x_j \rangle \tag{8}$$

The dual L_d is then maximized with respect to Lagrange multipliers α_i subject to the constraints:

$$\sum_{i=1}^{N} y_i \alpha_i = 0 \text{ and } \alpha_i \geq 0$$

The optimal solution α_i of the dual problem must satisfy the following equation:

$$\alpha_i \left[y_i \left(\langle w \cdot x_i \rangle + b \right) - 1 \right] = 0 \qquad i = 1, ..., N$$

By observing the solutions of the optimization problem, we notice that most parameters α_i are zero. These correspond to the x_i for which $y_i \left(\langle w.x_i \rangle + b \right) > 1$, they lie away from the separating hyperplane and have no effect on it.
Meanwhile, just a few solutions are non-zero. The set of x_i whose α_i are non-zero are called support vectors [21]. They lie closest to the hyperplane, as it is shown in figure 6, and satisfy:

$$y_i \left(\langle w.x_i \rangle + b \right) = 1$$

[2] [4] p. 216

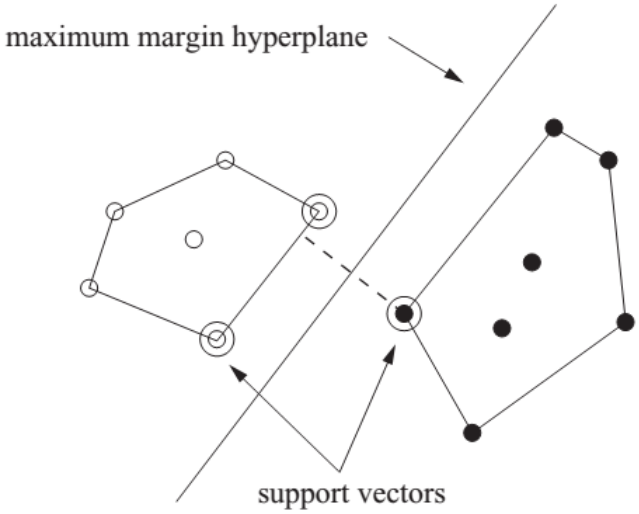

Figure 6: A maximum margin hyperplane.[2]

2.4.2 The non separable case: Soft margin

In most real-world problems, the data is not linearly separable, so the maximal margin model should be modified to allow the misclassification of some examples. We introduce slack variables $\xi_i \geq 0$ which store the degree of misclassification of the data. This allows the constraints to be violated to get a soft margin and we obtain the following optimization problem [2][3][19]:

$$\begin{cases} minimize \ \ \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{N} \xi_i \\ y_i \cdot (\langle w.x_i \rangle + b) \geq 1 - \xi_i \qquad i = 1, ..., N \end{cases} \tag{9}$$

C is the penalty factor, it determines the trade-off between the error (number of misclassified examples) and the width of the soft margin. The same dual Lagrangian L_d should be maximized subject to the new constraints:

$$\sum_{i=1}^{N} y_i \alpha_i = 0 \text{ and } 0 \leq \alpha_i \leq C$$

The parameter C represents the upper bound on α_i and is determined by the user. In general, choosing a large C makes the margin smaller and allows just a few number of misclassified examples, while a small value of C will have the opposite effect.

2.4.3 Non Linear Case: Kernels

In most cases, it is not possible to separate the data linearly, so the techniques described should be extended to solve the classification problem by creating nonlinear decision boundaries. To achieve that, it is possible to map the data into a high dimensional feature space F in which the linear classifiers can be used [3][19].
We consider again the function:

$$f(x) = \sum_{i=1}^{N} w\phi(x) + b$$

where ϕ is the mapping function that transforms the data into a feature space F. By expressing the weight vector w with the use of the mapping function, the function f can be rewritten as a combination of inner products between the test point and the training points [3]:

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i \langle \phi(x_i) \cdot \phi(x) \rangle + b \tag{10}$$

Therefore to use the function, we need to map the examples to the feature space and compute the inner product in this space. This kind of calculations can be extremely complicated in a high dimensional space. The idea in SVM, is to replace the inner product $\langle \phi(x) \cdot \phi(y) \rangle$ by a function of the original samples in the input space, and this avoids performing any type of mapping or calculation in the new space [2][19]. This function is called the kernel function. For all $x, y \in X$, it is defined as:

$$K(x, y) = \langle \phi(x) \cdot \phi(y) \rangle \tag{11}$$

There are different kernel functions that express the similarity from the inner product in the high dimensional feature space. During the optimization of the parameters of the SVM algorithms presented in section 3.3, one of the most popular kernel functions may be considered:

Polynomial kernel: $K(x, y) = (\langle x \cdot y \rangle + c)^{d}$ where c is a positive constant and the parameter d is the degree of the kernel.

Radial-basis function kernel: $K(x, y) = e^{-\gamma \|x-y\|^2}$ where γ is the parameter that controls the width of the kernel.

Sigmoid kernel: $K(x, y) = tanh\,(\gamma \langle x \cdot y \rangle + c)$

3 Experiments: Tuning algorithms parameters

In this section, we evaluate the implementations of the learning techniques presented in section 2. This is operated by optimizing the parameters of the algorithms in order to get the best accuracy values for a set of fixed learning problems obtained. We will use the Weka Experiment Environment [22], which enables the user to run different learning algorithms with various parameter settings on a set of different learning problems [4]. We will explain the role of the relevant parameters in the next sections. The remaining parameters will be used with their default settings, and they will be removed when exposing the results.

In the next section, we will introduce the datasets used during the experiments [3]. The algorithms as well as the results of the different parameter settings will be presented in the following sections. The results are stated in tables 4, 5, 6, 7 and 8: for every dataset, we give the best accuracy value obtained during the evaluation along with the corresponding parameter setting.

3.1 Datasets

The experiments are performed on a set of 21 learning problems (datasets), presented in table 3. Most of the datasets contain both numerical and nominal attributes. Six datasets don't have numeric attributes: *car, contact-lenses, dbworld_subjects_stemmed, glass, solar-flare-c, solar-flare-m*.
Seven datasets have a large number of numeric attributes (10 or more): *heart-statlog, segment, sonar, spectrometer2, vehicle, vowel*.

As we focus on binary as well as multiclass classification, both learning problems should be represented in our evaluation set: we have 6 binary and 15 multilabel datasets.
Three Datasets have a large number of classes (10 or more): *spectrometer2, vowel, yeast*.

[3] These data sets are available from [23] and https://research.cs.wisc.edu/dbworld/

Table 3: Datasets used for the parameters optimization.

Dataset	# Instances	# Classes	# Nominal Att.	# Numeric Att.
anneal	898	6	33	6
balance-scale	625	3	1	4
car	1728	4	7	0
contact-lenses	24	3	5	0
german_credit	1000	2	14	7
dbworld_subjects_stemmed	64	2	230	0
pimadiabetes	768	2	1	8
glass	214	7	10	0
heart-statlog	270	2	1	13
ionosphere	351	2	1	34
iris	150	3	1	4
lymphography	148	4	16	3
segment	2310	7	1	19
solar-flare-c	1712	6	11	0
solar-flare-m	1389	6	11	0
sonar	208	2	1	60
spectrometer2	531	48	3	100
vehicle	846	4	1	18
vowel	990	11	4	10
yeast	1484	10	1	8
zoo	101	7	17	1

3.2 Symbolic algorithms

In general the symbolic algorithms that implement the decision tree or rule learning schemes are used with their default settings, and in most cases this does not achieve the best results. In this section we optimize the important parameters of each algorithm in order to improve the accuracy.

3.2.1 J48

The J48 algorithm is a weka implementation of the C4.5 decision tree learner described in section 2.1.3. It has many options, and we will try to optimize the following ones by combining all the values below:
The parameter M represents the minimum number of instances per leaf. Choosing a small number has the effect of obtaining a specific tree, while increasing it produces more general trees. This parameter will have 11 different values during the experiments: 1, 2, 3, 4, 5, 6, 10, 20, 30, 60 and 100.
The parameter C corresponds to the confidence factor used for pruning. The confidence factor is used for pruning. The default value for the C4.5 decision tree learner is 0.25. Smaller values incur more pruning and produce more generalized trees. We will use 12 values for this parameter: 0.005, 0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.4, 0.5, 0.75 and 0.99.
The parameter B determines whether to use binary splits on nominal attributes when building the trees. There is no difference for nominal binary attribute. For nominal attributes that have multiple values, this option will creates only two branches for each attribute, if it is set to true. The parameter A determines whether counts at leaves are smoothed based on Laplace. We prevent the predicted probabilities from being calculated as zero, by using Laplace smoothing when setting A to true [4].
The default setting is false for both parameters A and B. If one of them is turned on, the corresponding letter will written in the fourth column of table 4, which exposes the results for the whole set of our learning problems.

Figures 7 and 8 show the variation of accuracy depending on values of M and C, for the data vowel and heart. A and B are both set to false on these examples.

[4] https://monk.library.illinois.edu/cic/public/analytics/decisiontree.html

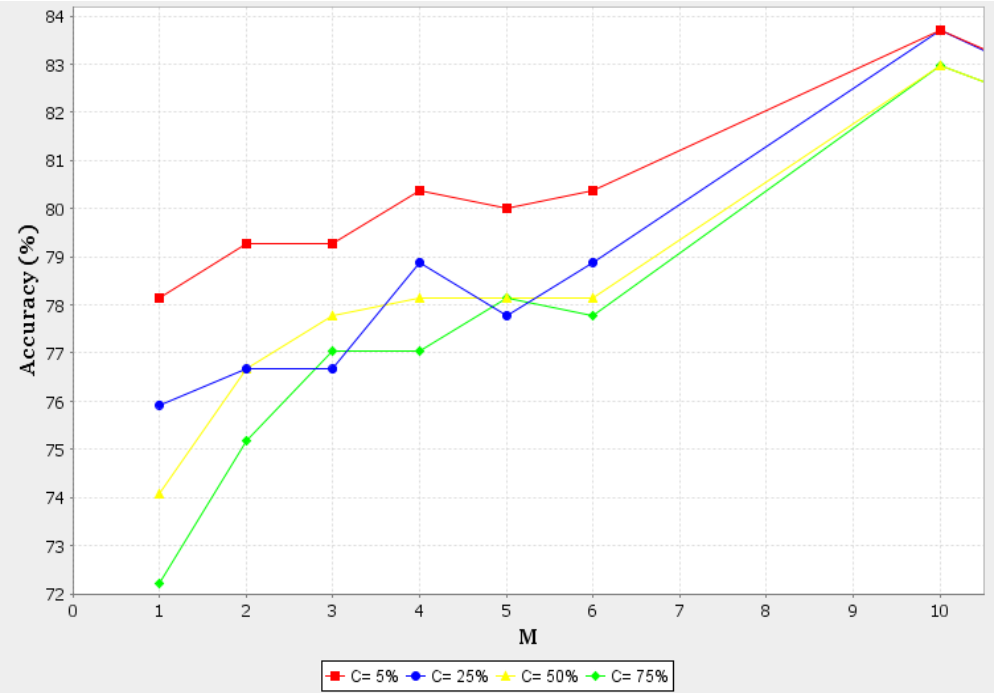

Figure 7: Variation of the accuracy depending on the C and M parameters for the heart data.

Figure 7 shows that a higher accuracy is reached when performing more pruning on the heart data. Figure 8 indicates that for the vowel data, the best accuracy is obtained with a high value of C (equivalent to less pruning). In both examples, the accuracy decreases for M greater than 10.

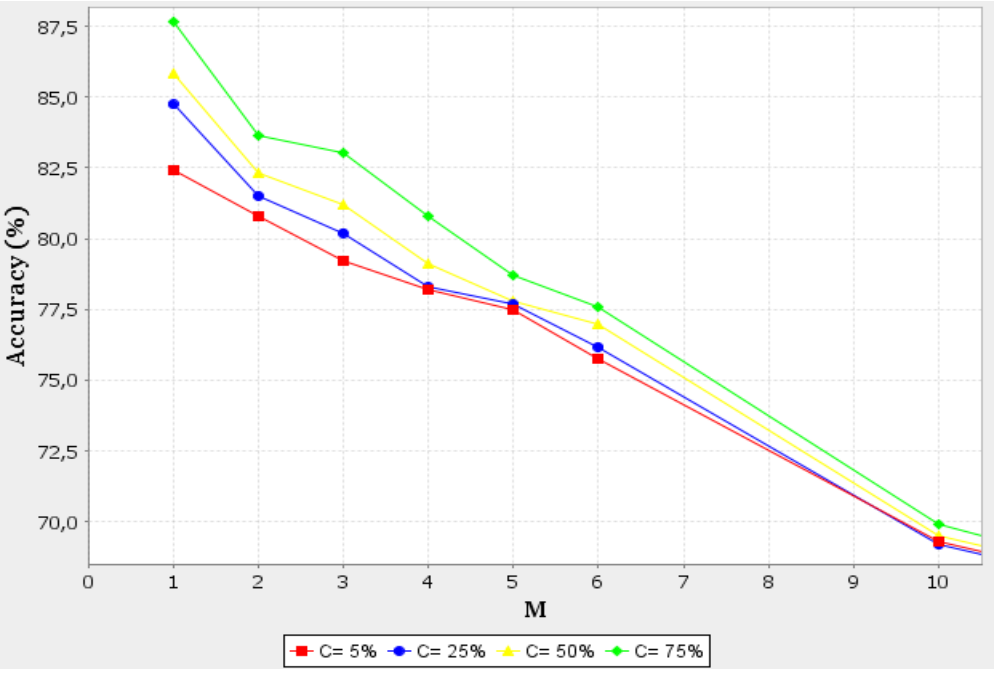

Figure 8: Variation of the accuracy depending on the C and M parameters for the vowel data.

Table 4: Best accuracy values obtained by J48, with the corresponding parameter settings. The columns represent in the following order: the datasets, the confidence factor (C), the minimum number of instances per leaf (M) and the binary split (B) or Laplace (A) options. The last column corresponds to the best accuracy value.

Datasets	C	M	B / A	Accuracy (%)
anneal	0.75	1		99.888
balance-scale	0.5	1		80.002
car	0.75	1	B	98.553
contact-lenses	0.01	3		85
german_credit	0.2	20	B	73.2
dbworld_subjects_stemmed	0.3	4		88.095
pima_diabetes	0.75	60		76.57
glass	0.25	1		69.523
heart-statlog	0.05	10		83.703
ionosphere	0.25	2		91.46
iris	0.2	2		96
lymphography	0.3	6	B	81
segment	0.5	1		97.532
solar-flare-c	0.3	3	B	85.573
solar-flare-m	0.01	1		95.104
sonar	0.05	6		75.476
spectrometer2	0.1	6		49.531
vehicle	0.4	10		74.47
vowel	0.75	1		87.676
yeast	0.01	3		58.55
zoo	0.1	1		94.181

3.2.2 RandomForest

RandomForest is the Java implementation for generating a forest of random trees. We will try to optimize the following parameters by combining all the values listed below:
The parameter I sets the number of trees to be constructed. We will use 11 values for this parameter during the experiments: 5, 10, 20, 30, 50, 75, 100, 250, 350, 500 and 700. The default value is 10.
The maximum depth of the trees is determined by the parameter depth. The following values will be tested for this parameter: 0 (which is the default setting and corresponds to unlimited depth), 2, 3, 5, 10, 20 , 30, 50, 75 and 100.
We will also optimize the parameter K that defines the number of attributes considered in random selection. 12 values will be tested during the experiments: 0 (default value), 1, 2, 3, 4, 5, 7, 10, 15, 20, 50 and 100. If this parameter is left to its default setting, $\log(M)+1$ attributes will be used, where M is the number of inputs. Table 5 exposes the results for the whole dataset.

Figure 9 shows the variation of accuracy depending on the number of trees (I) and their depth (Depth) for the data vowel and heart. The number of features (K) is left to default. When using a depth value greater than 10, the accuracy does not change for this data. We notice that using more trees produces better accuracy than when using the default value (10). In this example, the best results are obtained when running the algorithm with the default depth of the trees or for depths greater than 5.

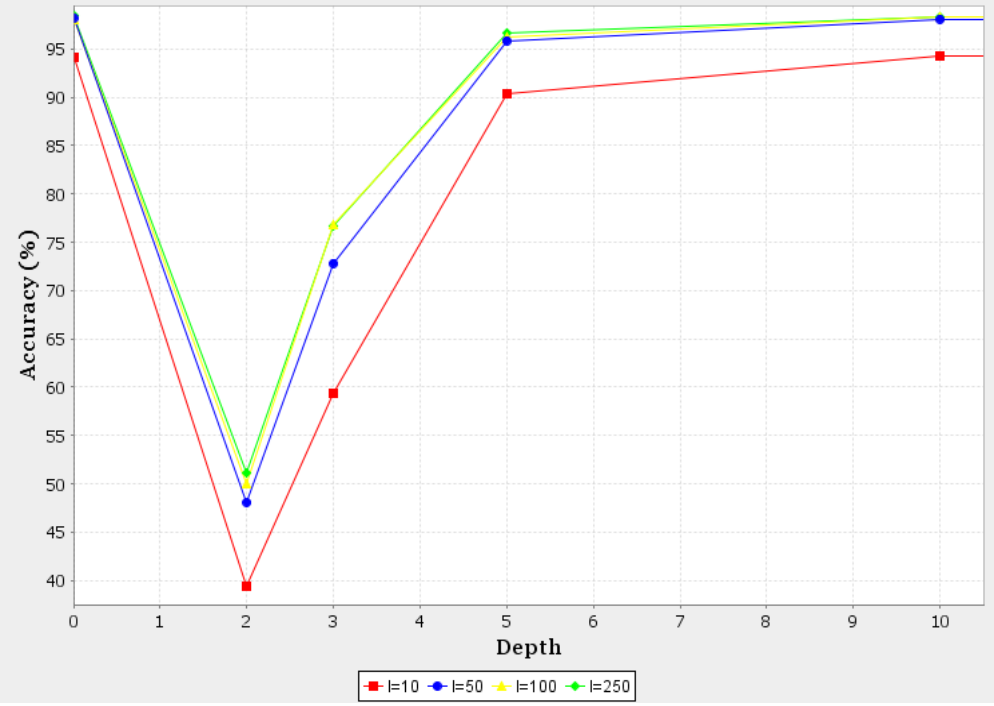

Figure 9: Variation of the accuracy depending on the I and depth parameters for the vowel data.

Table 5: Best accuracy values obtained by RandomForest, with the corresponding parameter settings.

Datasets	# Trees	# Features	Depth	Accuracy (%)
anneal	350	10	20	99.776
balance-scale	100	1	5	88.476
car	75	2	10	95.66
contact-lenses	5	0	2	86.666
german_credit	250	10	10	77.5
dbworld_subjects_stemmed	20	20	3	92.619
pima_diabetes	250	2	30	77.479
glass	350	3	10	81.32
heart-statlog	50	0	2	85.185
ionosphere	20	5	5	95.452
iris	20	4	5	95.999
lymphography	700	4	0	87.047
segment	100	0	10	98.181
solar-flare-c	350	0	5	85.981
solar-flare-m	350	0	3	95.104
sonar	75	1	10	87.999
spectrometer2	250	0	20	58.567
vehicle	75	0	10	77.072
vowel	250	3	10	98.686
yeast	500	1	20	63.409
zoo	20	2	10	97.09

3.2.3 JRip

JRip is the Java implementation of the RIPPER rule learner described in section 2.3.3. It has many options, and we will try to optimize the following ones by combining all the values listed below:
The parameter F determines how the data will be split to form the growing and the pruning sets. The default value is 3, which means that one fold is used for pruning and 2 folds for growing the rules. We will use 4 values for F: 2, 3, 5 and 10.
The number of optimizations is determined by the parameter O. It will be tested with 12 values: 1, 2, 3, 5, 10, 20, 50, 80, 100, 200, 400 and 600. The default setting of this parameter is 2.
The parameter N sets the minimum total weight of the instances in a rule. We will use 9 values for this parameter: 1, 2, 3, 4, 5, 6, 10, 20 and 30. The default setting is 2.
The stopping criterion of the Ripper algorithm includes the error rate test by default (accepting just the rules which for the error rates don't exceed 50%). This test can be excluded by setting the parameter E to false.
The parameter P determines whether to perform pruning. Its default value is true. If it is turned to false, the parameter F will not be relevant.
During the experiments, the check error rate parameter E and the pruning parameter P will not be set both to false in the same test. As their default setting is true, if one of them is turned off, the word No will be written in the corresponding cell in table 6, which exposes the results for the whole set of our learning problems.

We notice that for more than the half of our dataset (11 data), the maximum accuracy is obtained after 5 or less optimizations.
Figure 10 shows the variation of the accuracy for 4 data depending on the number of optimizations O. For the dbworld_subjects_stemmed data, the maximum accuracy is reached after one optimization. The same result is observed for 7 data (the third of the data used).

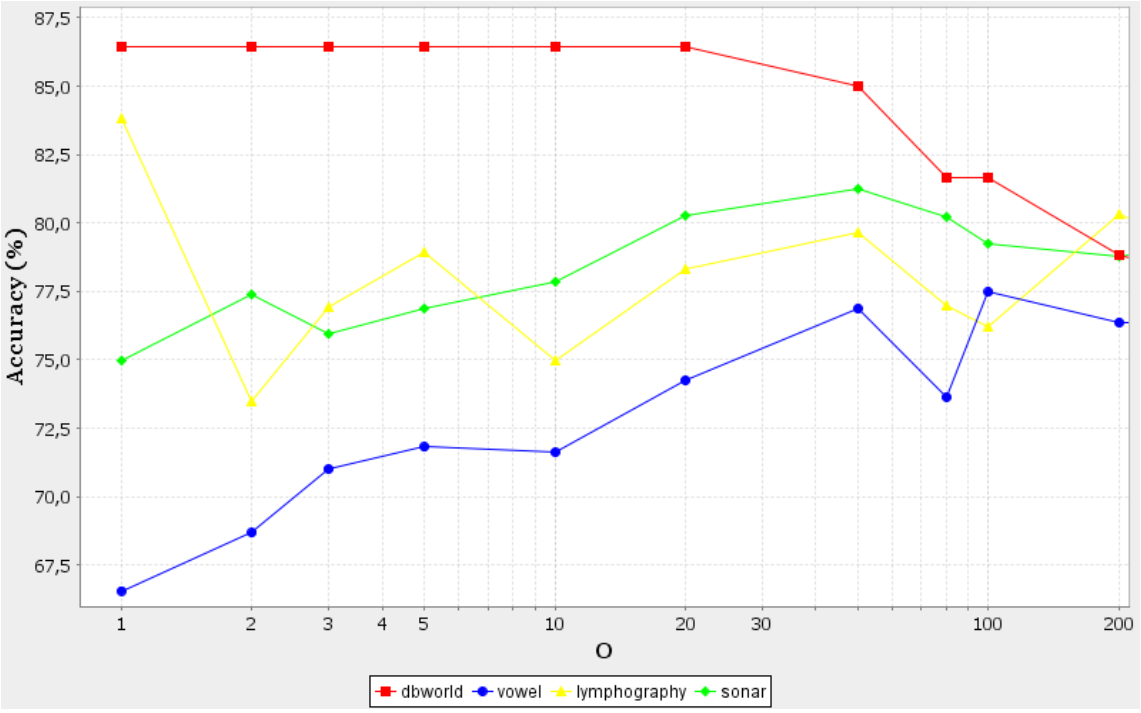

Figure 10: Variation of the accuracy depending on the O parameter for different data.

Figure 11 shows the variation of accuracy depending on the values of F and N for the contact-lenses data, where the optimization parameter is set to 1. We notice that performing less pruning (equivalent to a high F) on the contact-lenses data produces a higher accuracy. The maximum accuracy is obtained when no pruning is used. In this example, the accuracy does not change for N greater than 10.

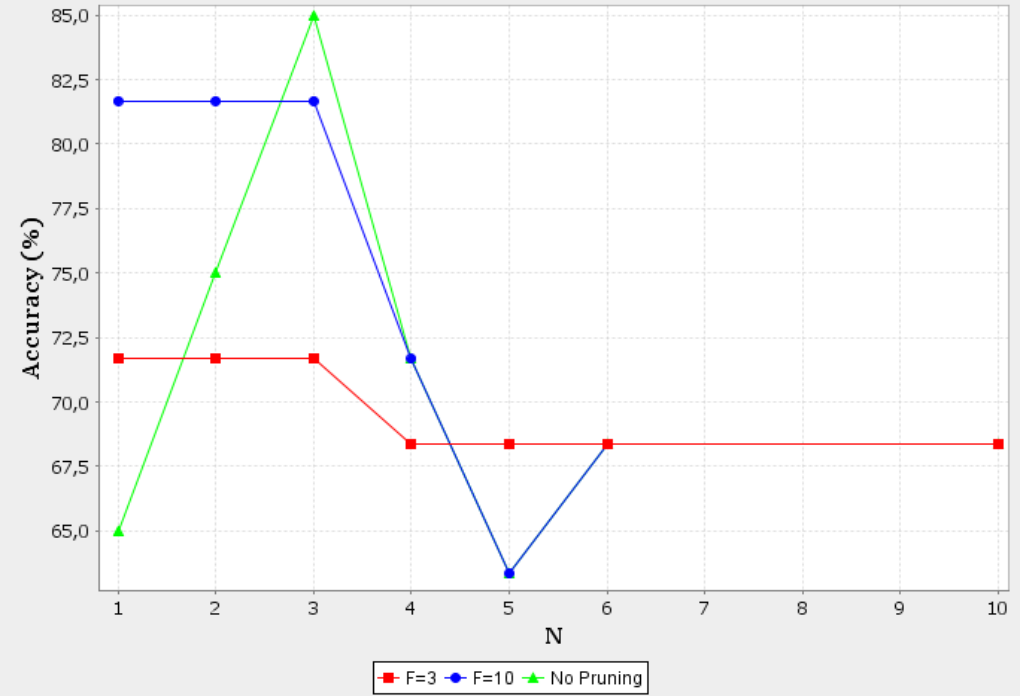

Figure 11: Variation of the accuracy depending on the F and N parameters for the contact-lenses data.

Table 6: Best accuracy values obtained by JRip, with the corresponding parameter settings. The columns represent in the following order: the datasets, the number of folds (F), the minimum total weight of the instances in a rule (N), the number of optimizations (O), the check error rate option (E) and the pruning option. The last column corresponds to the best accuracy value.

Datasets	F	N	O	E	Pruning	Accuracy (%)
anneal	2	1	1		No	99.441
balance-scale	2	2	1			82.096
car	3	1	600	No		90.277
contact-lenses	2	3	1		No	84.999
german_credit	2	20	20			75
dbworld_subjects_stemmed	3	1	1			86.428
pima_diabetes	2	30	10	No		77.73
glass	5	2	200			75.259
heart-statlog	2	10	400	No		82.222
ionosphere	3	4	1			91.476
iris	2	1	3			97.333
lymphography	5	1	1			83.809
segment	5	3	200	No		96.666
solar-flare-c	2	1	1			85.573
solar-flare-m	3	4	5	No		95.32
sonar	2	2	50			81.238
spectrometer2	5	1	400			43.871
vehicle	10	2	100	No		74.116
vowel	2	1	600	No		79.393
yeast	5	1	2			60.036
zoo	2	1	3			91.181

3.3 Support Vector Machine

In this section, we focus on two algorithms used for training vector machines: the SMO algorithm suggested by [24], and the LibSVM library described in [25]. The penalty factor C (see section 2.4.2) is the most important parameter for our classification task.
We will use 14 values for C in our experiments:
10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 0.5, 1, 2, 10, 100, 1000, 2000, 5000 and 8000.

3.3.1 SMO

Sequential minimal optimization (SMO) is an SVM learning algorithm that divides the quadratic programming problem (see (9)) into multiple small problems, which makes the algorithm faster [24]. Moreover, some modifications are proposed by [20] to improve the performances of SMO.

In what follows, we define some kernels used by the SMO algorithm, and we determine which values will be tested for the parameters of every kernel during the experiments. The parameter N determines whether to normalize (N=0 which is the default setting), to standardize (N=1) the training data, or if no transformation is performed (N=2).

RBF Kernel:

$$K(x, y) = e^{-\gamma \|x-y\|^2}$$

γ is the gamma parameter (corresponding to the parameter G). It will be tested with 20 different values: $0, 2^{-13}, 2^{-12}, ..., 2^4, 2^5$.

Figure 12 shows the variation of the accuracy depending on the parameters C and G for the anneal data. In this example, N is left to its default setting, so the training data are normalized.

The first thing to notice is that for all the values of G, the obtained curves have similar behavior. First, there is a phase where the value of the accuracy remains constant and equal for all curves. In this figure, for $C \leq 10^{-2}$, we see that the accuracy is equal to 76.169% for all values of G. Then there is a transition phase, where the accuracy grows with the value of C. We point out here that this transition starts as soon as C becomes higher than a certain threshold, which is different for each value of G. For instance, this threshold is equal to 10^{-1} for G=2^{-7} and G=2^5 and greater than 1 for G=2^{-11}. Moreover, the transition phase is more or less sharp according to the value of G. In this example, the transition is smoother for G=2^{-11}, G=2^{-7} and G=2^{-3} than for G=2^1 and G=2^5.

Finally all curves are stabilized for C greater than a certain threshold. The value of this threshold is significantly dependent on G. It is equal to 10^3 for G=2^{-7} and G=2^{-3} whereas it is less than 10 for G= 2^1 and G=2^5. Furthermore, the value at which the accuracy is stabilized differs from one value of G to another. For G $\in \{2^{-11}, 2^{-7}, 2^{-3}\}$, the best accuracy is almost reached, while it is stabilized to 96.546% and 84.856% for G=2^1 and G=2^5 respectively.

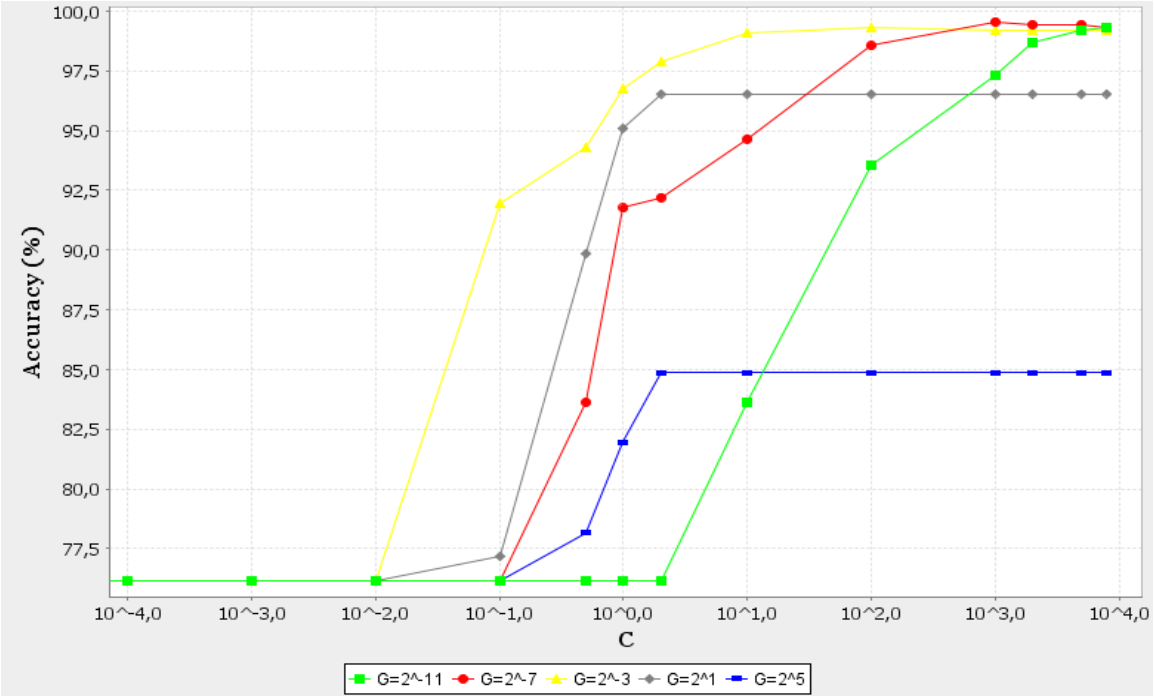

Figure 12: Variation of the accuracy depending on the C and G parameters of the RBF kernel for the anneal data.

Polykernel:

$$K(x,y) = \langle x \cdot y \rangle^{E} \text{ or } K(x,y) = (\langle x \cdot y \rangle + 1)^{E}$$

The parameter E sets the exponent of the kernel. We will use 5 values for E during the experiments: 1, 2, 3, 4 and 5. The parameter N will be set to zero for this kernel; that means that the training data will be normalized.

Figure 13 shows the variation of the accuracy depending on the parameters C and E for the glass data. We notice that there is a phase where the value of the accuracy remains constant and the same for all the curves. In this figure, for $C \leq 10^{-3}$, we see that the accuracy is equal to 35.519% for all the values of E. Then there is transition phase, where the accuracy grows with the value of C. We point out here that this transition starts as soon as C becomes higher than a certain threshold, which is different from one value of E to another. For instance, this threshold is equal to 10^{-3} for E=4 and E=5 and to 10^{-2} for smaller values of E.

Furthermore, the transition phase is more or less sharp according to the value of E. In this example, the transition is smoother for small exponents (E=1, E=2 and E=3) than for the curves corresponding to E=4 and E=5 where it changes sharply.

Unlike the RBF kernel in figure 12, the curves are not stabilized when using the polykernel on the glass data. Mostly all curves attain their maximum at a certain C threshold and start to decrease afterwards. We also notice that the curve corresponding to E=1 attains worse results than the other exponents. The best accuracy in this example (75.606%) is reached by E=2 when C=2000.

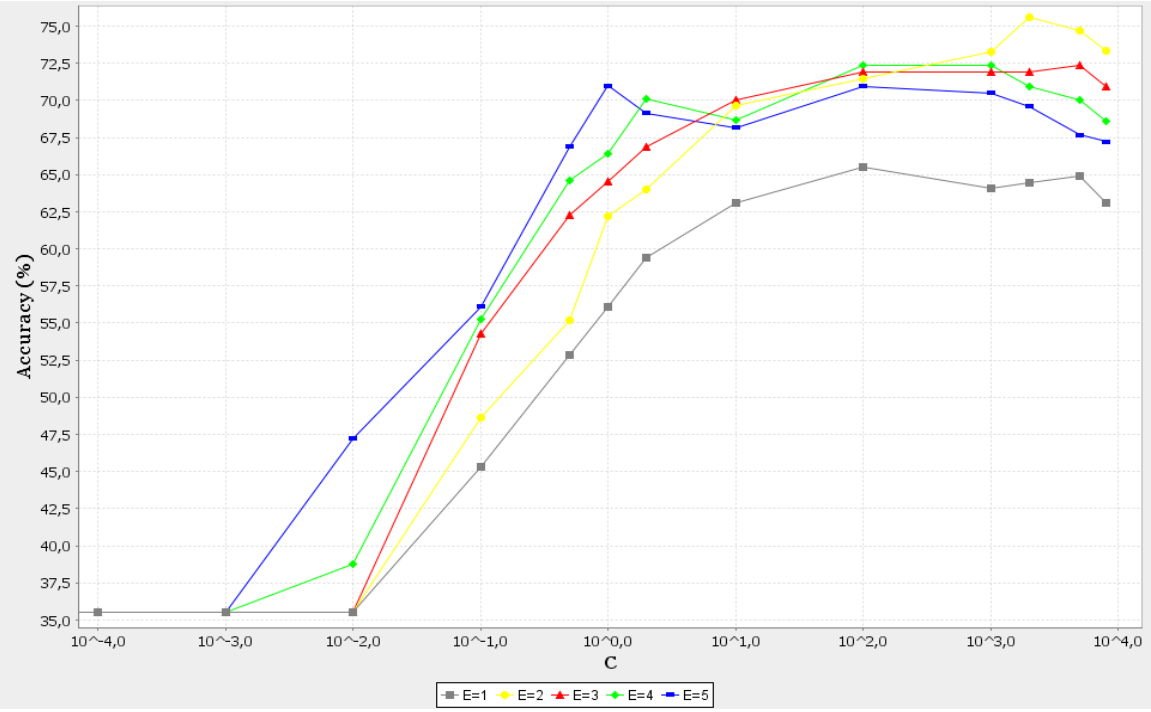

Figure 13: Variation of the accuracy depending on the C and E parameters of the polykernel for the glass data.

Normalized Polykernel:

$$K(x,y) = \frac{<x,y>}{\sqrt{(<x,x>.<y,y>)}} \; where < x,y >= PolyKernel(x,y)$$

The polykernel is used in the normalized polykernel. Seven values will be tested for the exponent parameter E: 2, 3, 4, 5, 6, 7 and 8.

Figure 14 shows the variation of the accuracy depending on the parameters C and E for the sonar data. In this example, the parameter N is left to its default setting, so the training data are normalized.

The first thing to notice is that for all the values of E, the obtained curves have similar behavior. First, there is a phase where the value of the accuracy remains constant and the same for all the curves. In this figure, for $C \leq 10^{-2}$, we see that the accuracy is equal to 53.38% for all the values of E.
Then there is transition phase, where the accuracy grows with the value of C. We point out here that this transition starts as soon as C becomes higher than a certain threshold, which is the same for all the values of E. For instance, this threshold is equal to 10^{-2}. We notice that the transition phase is more or less sharp for the different exponents.

Finally all the curves are stabilized for C greater than 10^2. The value at which the accuracy is stabilized differs from one value of E to another. For E=7 and E=8 this value is clearly higher than for smaller exponents. In this example, the best accuracy (89.88%) is reached by E=7 when C=100.

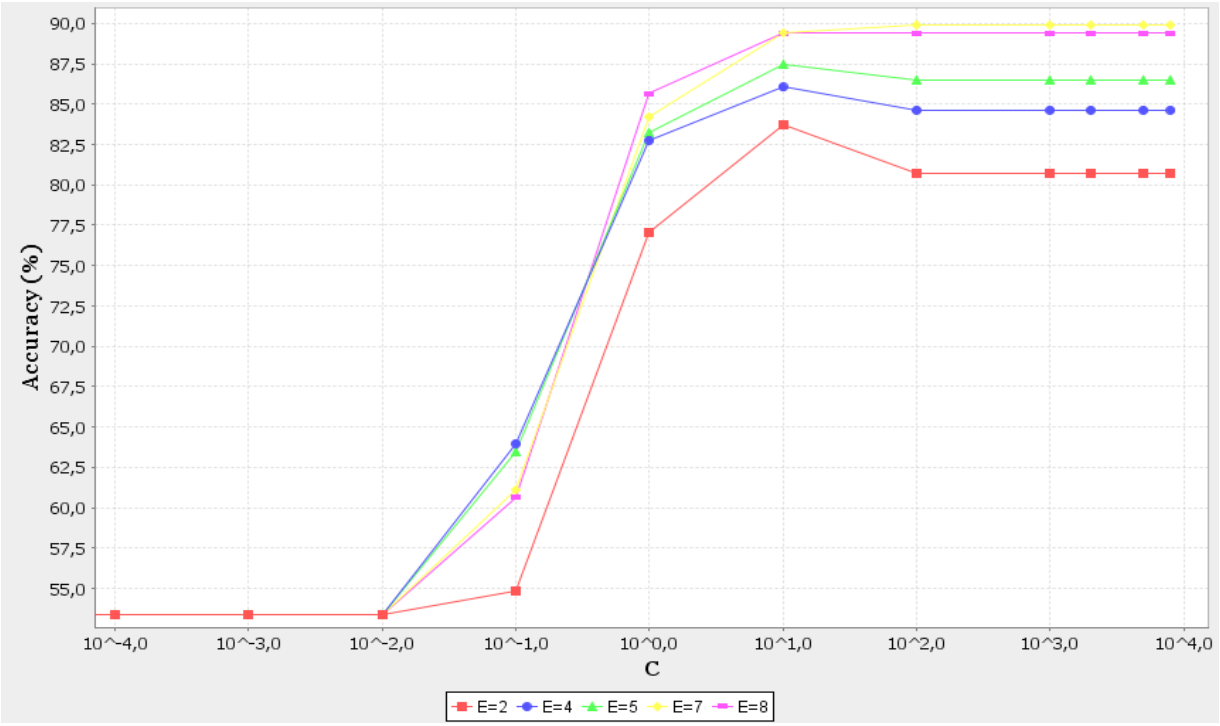

Figure 14: Variation of the accuracy depending on the C and E parameters of the normalized polykernel for the sonar data.

In table 7, we present the results of the experiments run by the SMO algorithm on the whole dataset.

Table 7: Best accuracy values obtained by SMO, with the corresponding parameter settings. The columns represent in the following order: the datasets, the soft margin parameter (C), the kernel type, the gamma or the exponent parameter (G or E) and the filter type (N). The last column corresponds to the best accuracy value.

Datasets	C	Kernel type	G / E	N	Accuracy (%)
anneal	100	RBF	2^{-7}	0	99.555
balance-scale	2000	RBF	2^{-3}	0	100
car	10	Normalized polykernel	3	1	100
contact-lenses	100	RBF	2^{-3}	0	78.333
german_credit	1000	RBF	2^{-11}	1	78.3
dbworld_subjects_stemmed	2	RBF	2^{-9}	1	92.619
pima_diabetes	5000	RBF	2^{-8}	0	78.253
glass	2000	Polykernel	2	0	75.606
heart-statlog	0.1	RBF	2^{-5}	1	85.185
ionosphere	0.5	Normalized polykernel	8	2	96.293
iris	10	RBF	2^{-6}	2	98.666
lymphography	10	RBF	2^{-10}	1	88.523
segment	1000	RBF	2^{-12}	2	97.619
solar-flare-c	10^{-4}	Polykernel	5	0	86.098
solar-flare-m	10^{-4}	RBF	2^{-11}	0	95.104
sonar	100	Normalized polykernel	7	0	89.88
spectrometer2	1000	Normalized polykernel	2	2	71.942
vehicle	1000	RBF	2^{-2}	0	86.176
vowel	10	RBF	2^{-1}	2	99.797
yeast	2	RBF	2^{4}	2	60.912
zoo	10	RBF	2^{-3}	2	98

3.3.2 LibSVM

LibSVM (Library for Support Vector Machines) is a software for SVM developed by [25]. It can be used for both classification and regression. The integration of the LibSVM classifier into Weka Environment is realized by [26].

In what follows, we define some kernels used by the LibSVM algorithm, and we determine which values will be tested for the parameters of every kernel during the experiments.
The parameter N determines whether to normalize the input data. Its default value is false.
G sets the gamma parameter. If it is left to its default value (which is 0), $\frac{1}{maxindex}$ is used instead.

Linear kernel:

$$K(u, v) = u' \cdot v$$

Sigmoid kernel:

$$K(u, v) = \tanh(\gamma \cdot u' \cdot v + R)$$

The gamma parameter G will be tested with 20 different values: $0, 2^{-13}, 2^{-12}, ..., 2^{4}, 2^{5}$.
The parameter R (which default value is 0) will be tested with 3 values: -1000, 0 and 1000 for this kernel.

Polykernel:

$$K(u, v) = (\gamma \cdot u' \cdot v + R)^D$$

The parameter D sets the degree of the kernel. We will use 5 values for this parameter during the optimization: 1, 2, 3, 4 and 5.
As it is the case for the Sigmoid kernel, the parameter R will be tested with 3 values: -1000, 0 and 1000, while G will be left to its default setting.

Figure 15 shows the variation of the accuracy depending on the parameters C and for the balance-scale data. In this example, the input data are not normalized and R is set to 0.

We notice that there is a phase where the accuracy grows with the value of C. This transition occurs sharply for the curve corresponding to D=1 in comparison with the other curves. We also notice that the accuracy is dependent on D: for small values of C, higher degrees attain a better accuracy. However, as the value of C increases, this is reversed: a better accuracy is reached by small values of D (except D=1 which has clearly the worst results for all values of C).

After the transition phase, the curves are stabilized for C greater than a certain threshold which is equal to 10 for all the curves. The value at which the accuracy is stabilized differs from one value of D to another. It is stabilized to 100% for D=2, while it is stabilized to 91.671% for D=1.

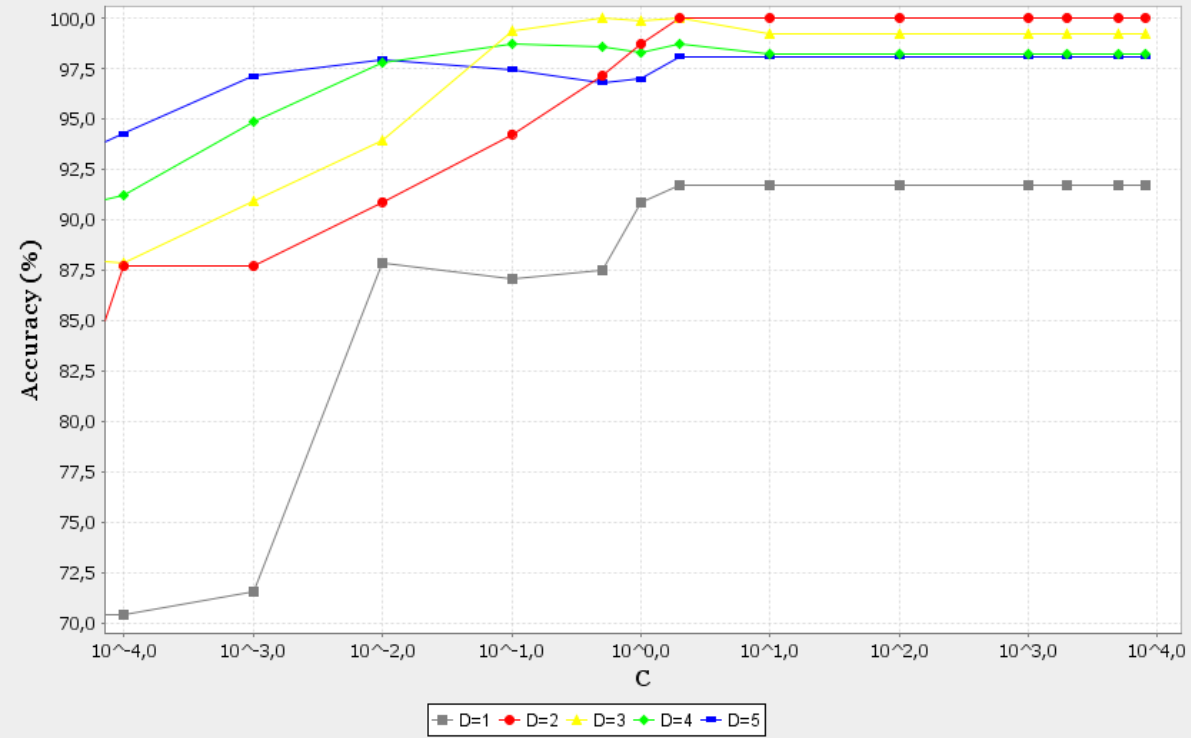

Figure 15: Variation of the accuracy depending on the C and D parameters of the polykernel for the balance-scale data.

RBF kernel:

$$K(u, v) = \exp(-\gamma \cdot |u - v|^2)$$

As it is the case for the Sigmoid kernel, we will use 20 values for the gamma parameter G.

Figure 16 shows the variation of the accuracy depending on parameters C and G for the lymphography data. In this example, the input data are normalized.

The first thing to notice is that for all the values of G, the obtained curves have similar behavior. First, there is a phase where the value of the accuracy remains constant and the same for all the curves. In this figure, for $C \leq 10^{-2}$, we see that the accuracy is equal to 54.761% for all the values of G. Then there is transition phase, where the accuracy grows with the value of C. We point out here that this transition starts as soon as C becomes higher than a certain threshold, which is different from one value of G to another. The transition phase is rather sharp for the different values of G.

Finally all the curves are stabilized for C greater than a certain threshold. The value of this threshold is significantly dependent on G. It is equal to 10 for G=2^{-7} and $G = 2^{-3}$ whereas it is equal to 5000 for G=2^{-13} and G=2^{-11}. Furthermore, the value at which the accuracy is stabilized differs from one value of G to another. The best accuracy (89.857%) is reached by G=2^{-11} when C=5000, while the curves corresponding to bigger values of G are stabilized to worse accuracy.

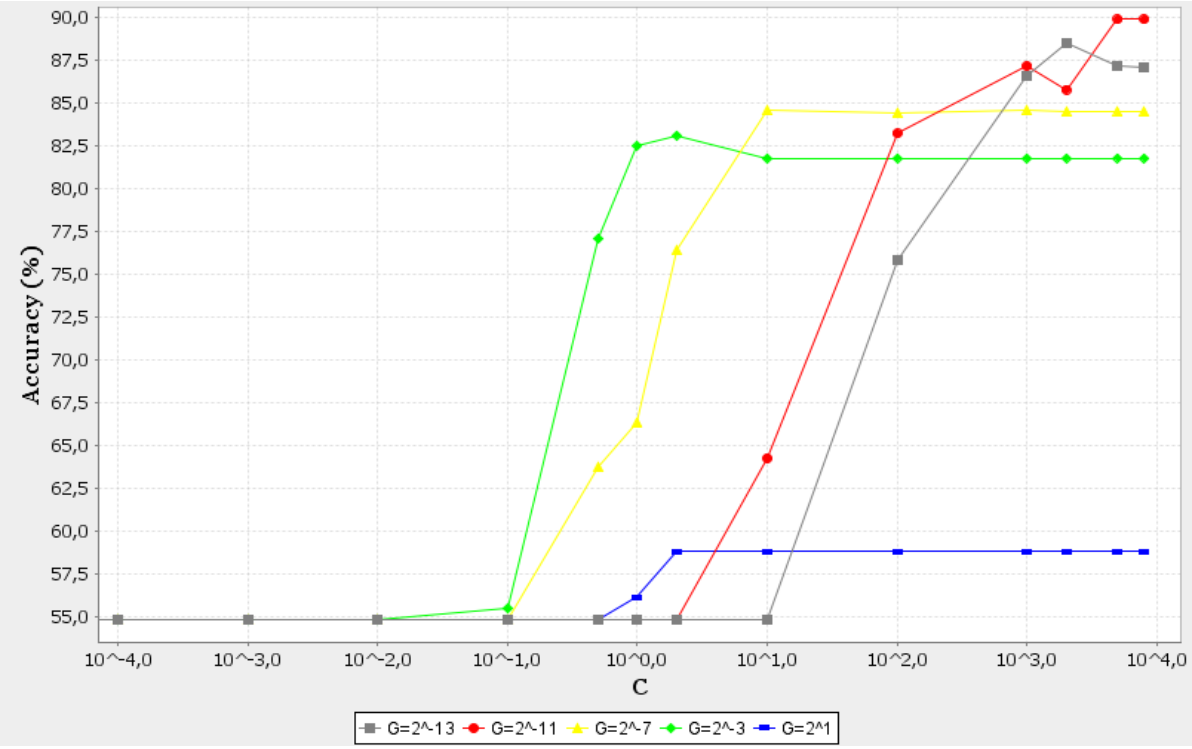

Figure 16: Variation of the accuracy depending on the C and G parameters of the rbf kernel for the lymphography data.

In table 8, we present the results of the experiments run by the LibSVM algorithm on the whole dataset.

Table 8: Best accuracy values obtained by LibSVM, with the corresponding parameter settings. The columns represent in the following order: the datasets, the soft margin parameter (C), the kernel type, the gamma or the degree parameter (G or D) and the normalize option. The last column corresponds to the best accuracy value.

Datasets	C	Kernel type	G / D	Normalize	Accuracy (%)
anneal	2000	Polykernel	2	Yes	99.444
balance-scale	10	Polykernel	2		100
car	10	RBF	2^{-1}		99.884
contact-lenses	100	Polykernel	2		83.333
german_credit	5000	Polykernel	1	Yes	77
dbworld_subjects_stemmed	1	Sigmoid	2^{2}		94.285
pima_diabetes	10	Polykernel	1		78.257
glass	5000	RBF	2^{-1}	Yes	74.653
heart-statlog	100	Polykernel	3	Yes	85.185
ionosphere	100	RBF	2^{-1}		95.174
iris	10	RBF	2^{-6}		98.666
lymphography	5000	RBF	2^{-11}	Yes	89.857
segment	100	RBF	2^{-12}		97.532
solar-flare-c	1	RBF	2^{2}		85.922
solar-flare-m	1	Polykernel	2		95.104
sonar	10	RBF	2^{-2}	Yes	89.404
spectrometer2	10^{-4}	Polykernel	1		71.757
vehicle	1000	RBF	2^{-2}	Yes	86.294
vowel	2	RBF	2^{0}		99.797
yeast	2	RBF	2^{4}		60.845
zoo	0.5	linear	-		96.09

4 Comparing the algorithms

4.1 Average, standard deviation and variance

In this section, we compare the sensitivity of the different algorithms. First, the accuracy measures obtained from the different parameter settings of each algorithm for the whole set of data will be averaged. Then the variance Var and the standard deviation SD measures will be computed:

$$Var = \frac{\sum_{i=1}^{N}(X - X_i)^2}{N-1} \tag{12}$$

where N is the number of parameter settings (P) tested for a given algorithm (A) multiplied by the number of datasets (21), X_i is the accuracy of the i^{th} experiment run by A, and X is the average of all accuracy measures obtained for A.
The standard deviation is the square root of the variance. It shows the variation from the average: In our comparison, a high standard deviation value means that the accuracy measures obtained spread out over a large range, while a low deviation indicates that these measures are close to the average. The results are stated in table 9.

Table 9: Calculation of the average (X), the variance (Var) and the standard deviation (SD) of every algorithm over the whole dataset.

	J48	**Random Forest**	**JRip**	**SMO**	**LibSVM**
P	528	1320	1296	1204	2688
N	11088	27720	27216	25284	56448
X (in %)	75.853	80.223	75.796	68.774	58.3
Var	245.078	222.619	264.655	505.177	634.991
SD	15.654	14.92	16.268	22.476	25.199

Moreover, we will compute the standard deviation between the accuracy measures obtained from the different parameter settings of each algorithm for every dataset. The results are stated in table 10: The columns represent the different algorithms. For every dataset, we give the maximal accuracy in the first row, the minimal in the second row, the average value in the third row and the standard deviation in the last one.

Table 10: An overview of the maximal-, the minimal-, the average accuracy and the standard deviation of every algorithm for the single data.

Datasets	J48	Random Forest	JRip	SMO	LibSVM
anneal	99.888 81.962 95.827 4.611	99.776 76.169 96.823 5.813	99.441 91.423 96.994 1.86	99.555 76.058 86.189 9.716	99.444 64.795 79.135 8.481
balance-scale	80.002 64.493 73.325 4.486	88.476 72.956 81.38 2.019	82.096 71.208 78.376 1.998	100 19.859 80.537 16.529	100 4.966 64.22 21.577
car	98.553 76.565 88.398 6.6	95.66 70.023 90.918 6.454	90.277 74.133 82.878 4.23	100 70.023 82.314 13.605	99.884 49.592 73.738 11.026
contact-lenses	85 53.333 75.024 7.907	86.666 58.333 72.28 3.623	84.999 58.333 70.115 5.297	78.333 38.333 68.481 4.287	83.333 49.999 68.682 4.283
german_credit	73.2 65.7 70.423 1.413	77.5 69.8 74.47 1.83	75 68.4 72.415 0.995	78.3 61.2 70.857 2.624	77 36.5 68.374 5.934
dbworld_subjects_stemmed	88.095 54.523 72.091 13.789	92.619 55.952 82.324 7.588	86.428 54.523 76.864 10.578	92.619 43.809 63.663 13.811	94.285 15.238 62.455 15.581
pima_diabetes	76.57 72.13 74.392 0.98	77.479 68.62 74.916 1.117	77.73 71.883 74.32 1.094	78.253 60.415 68.224 4.495	78.257 37.889 65.257 7.065
glass	69.523 35.519 60.506 9.121	81.32 57.64 74.885 5.151	75.259 41.623 64.228 6.876	75.606 35.519 51.714 14.981	74.653 21.515 42.418 12.695
heart-statlog	83.703 72.222 76.694 3.095	85.185 75.555 80.462 2.162	82.222 68.518 77.682 2.202	85.185 55.555 65.897 11.291	85.185 23.333 60.093 10.565
ionosphere	91.46 74.087 88.32 4.684	95.452 84.333 92.609 1.254	91.476 80.373 89.342 1.761	96.293 64.103 78.517 13.38	95.174 39.587 69.534 12.154
iris	95.999 33.333 86.418 18.617	95.999 87.999 95.136 0.73	97.333 33.333 93.479 8.773	98.666 33.333 86.955 14.059	98.666 0 59.472 32.025

Datasets	J48	Random Forest	JRip	SMO	LibSVM
lymphography	81	87.047	83.809	88.523	89.857
	54.761	71.523	54.761	54.761	26.999
	72.998	81.068	72.617	67.599	60.541
	7.157	3.173	4.577	14.005	12.441
segment	97.532	98.181	96.666	97.619	97.532
	89.09	69.437	89.004	14.285	0
	94.674	94.34	94.865	77.206	43.103
	2.441	6.421	1.268	23.734	35.399
solar-flare-c	85.573	85.981	85.573	86.098	85.922
	84.055	83.47	83.703	83.877	30.804
	85.076	84.55	85.098	85.193	83.941
	0.268	0.496	0.352	0.273	5.5
solar-flare-m	95.104	95.104	95.32	95.104	95.104
	92.729	92.945	94.313	92.153	48.041
	95.017	93.833	95	94.433	94.16
	0.313	0.694	0.158	0.873	4.349
sonar	75.476	87.999	81.238	89.88	89.404
	53.38	62.999	64.357	53.38	33.571
	70.146	80.919	73.88	67.857	59.418
	5.627	3.599	3.738	14.465	10.979
spectrometer2	49.531	58.567	43.871	71.942	71.757
	13.19	10.359	10.359	10.359	5.461
	42.659	43.345	31.027	21.079	16.75
	7.577	13.851	8.538	17.245	17.56
vehicle	74.47	77.072	74.116	86.176	86.294
	65.488	54.844	55.326	25.648	15.721
	71.079	72.99	66.99	51.589	38.207
	2.947	4.457	3.238	24.558	21.475
vowel	87.676	98.686	79.393	99.797	99.797
	33.838	25.858	27.07	9.09	6.06
	67.523	85.989	66.288	69.405	30.707
	16.125	18.932	10.998	29.642	32.97
yeast	58.55	63.409	60.036	60.912	60.845
	51.547	40.763	50.405	31.131	13.072
	56.167	58.359	55.34	45.408	37.666
	1.803	4.42	2.988	12.742	11.814
zoo	94.181	97.09	91.181	98	96.09
	40.636	42.636	40.636	40.636	23.727
	76.154	73.077	73.911	61.139	46.43
	19.685	23.01	19.871	25.041	17.532

4.2 Default settings and optimized parameters

In this section, we compare the accuracy values obtained when running the algorithms with their default settings and after the optimization. The results are stated in table 12: the columns represent the different algorithms. For every dataset, we give the maximal accuracy in the first row, and the accuracy corresponding to the default settings of each algorithm in the second row.

Table 12: An overview of the maximal accuracy and the accuracy corresponding to the default settings of every algorithm for the single data.

Datasets	J48	Random Forest	JRip	SMO	LibSVM
anneal	99.888 98.439	99.776 99.332	99.441 98.329	99.555 97.439	99.444 90.087
balance-scale	80.002 76.653	88.476 82.237	82.096 79.05	100 87.677	100 89.754
car	98.553 92.362	95.66 93.519	90.277 86.457	100 93.75	99.884 96.354
contact-lenses	85 81.666	86.666 81.666	84.999 74.999	78.333 71.666	83.333 63.333
german_credit	73.2 70.5	77.5 74.3	75 71.7	78.3 75.1	77 70
dbworld_subjects_stemmed	88.095 75.714	92.619 86.666	86.428 86.428	92.619 87.857	94.285 54.523
pima_diabetes	76.57 73.834	77.479 73.438	77.73 76.037	78.253 77.344	78.257 65.105
glass	69.523 66.753	81.32 73.831	75.259 68.658	75.606 56.125	74.653 68.766
heart-statlog	83.703 76.666	85.185 81.111	82.222 78.888	85.185 84.074	85.185 55.925
ionosphere	91.46 91.46	95.452 94.595	91.476 89.753	96.293 88.603	95.174 93.46
iris	95.999 95.999	95.999 94.666	97.333 95.333	98.666 95.999	98.666 96.666
lymphography	81 76.952	87.047 80.333	83.809 77.761	88.523 86.428	89.857 79.809
segment	97.532 96.926	98.181 97.662	96.666 95.411	97.619 93.073	97.532 65.367
solar-flare-c	85.573 85.105	85.981 84.346	85.573 85.397	86.098 85.163	85.922 85.163
solar-flare-m	95.104 95.104	95.104 93.736	95.32 94.745	95.104 95.104	95.104 95.104
sonar	75.476 71.166	87.999 80.238	81.238 73.071	89.88 75.952	89.404 65.904
spectrometer2	49.531 47.449	58.567 49.346	43.871 31.656	71.942 51.418	71.757 10.359
vehicle	74.47 72.469	77.072 75.417	74.116 69.04	86.176 74.364	86.294 30.495
vowel	87.676 81.515	98.686 94.141	79.393 70.101	99.797 71.414	99.797 88.484
yeast	58.55 55.991	63.409 58.619	60.036 58.081	60.912 57.075	60.845 43.262
zoo	94.181 92.181	97.09 91.09	91.181 87.272	98 96.181	96.09 49.636

4.3 Statistical comparison - Significance test

In this section we use Friedman and Nemenyi tests to compare the algorithms. This procedure is based on average ranks and is described in detail in [27]. During our experiments, we used $k = 5$ algorithms and $N = 21$ data sets. First, the algorithms are ranked for each dataset according to their accuracy. Average ranks are assigned in case of ties between some algorithms. The results are stated in table 13. For each dataset, we give the accuracy in the first row and the rank of each algorithm in the second row. After that, the average ranks of the different algorithms are computed (see table 14).

The null hypothesis of the Friedman Test assumes that the different algorithms perform equally well, so the average ranks should not be significantly different.
We start with calculating the critical value of the F-distribution with $(k-1) = 4$ and $(k-1)(N-1) = 80$ degrees of freedom at different alpha levels, and compare them with the Friedman test statistic F_F from [28]. We obtain $F_F = 15.08$. The critical value of the F-distribution for level $\alpha = 0.01$ is 3.563, so we reject the null hypothesis and then we perform the post-hoc Nemenyi test [29] by computing the critical difference:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$$

where q_α is the critical value at level α.
If the average ranks of two algorithms differ by at least the critical difference, we consider that their performances are significantly different. Figure 17 shows that we can identify two groups of classifiers that are not significantly different in our test ($CD = 1.588$ at $\alpha = 0.01$). Classifiers that belong to the same group are connected together.

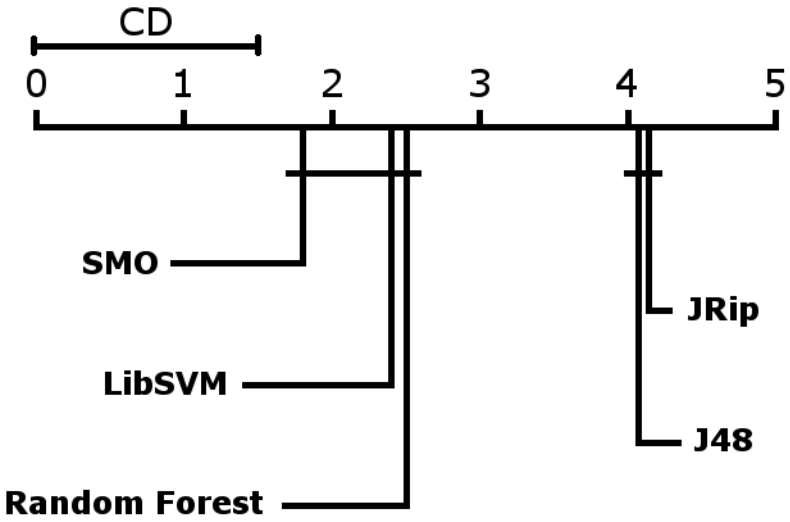

Figure 17: Results of the significance test.

Table 13: Ranking of the algorithms for each data.

Datasets	J48	Random Forest	JRip	SMO	LibSVM
anneal	99.888 1	99.776 2	99.441 5	99.555 3	99.444 4
balance-scale	80.002 5	88.476 3	82.096 4	100 1.5	100 1.5
car	98.553 3	95.66 4	90.277 5	100 1	99.884 2
contact-lenses	85 2	86.666 1	84.999 3	78.333 5	83.333 4
german_credit	73.2 5	77.5 2	75 4	78.3 1	77 3
dbworld_subjects_stemmed	88.095 4	92.619 2.5	86.428 5	92.619 2.5	94.285 1
pima_diabetes	76.57 5	77.479 4	77.73 3	78.253 2	78.257 1
glass	69.523 5	81.32 1	75.259 3	75.606 2	74.653 4
heart-statlog	83.703 4	85.185 2	82.222 5	85.185 2	85.185 2
ionosphere	91.46 5	95.452 2	91.476 4	96.293 1	95.174 3
iris	95.999 4.5	95.999 4.5	97.333 3	98.666 1.5	98.666 1.5
lymphography	81 5	87.047 3	83.809 4	88.523 2	89.857 1
segment	97.532 3.5	98.181 1	96.666 5	97.619 2	97.532 3.5
solar-flare-c	85.573 4.5	85.981 2	85.573 4.5	86.098 1	85.922 3
solar-flare-m	95.104 3.5	95.104 3.5	95.32 1	95.104 3.5	95.104 3.5
sonar	75.476 5	87.999 3	81.238 4	89.88 1	89.404 2
spectrometer2	49.531 4	58.567 3	43.871 5	71.942 1	71.757 2
vehicle	74.47 4	77.072 3	74.116 5	86.176 2	86.294 1
vowel	87.676 4	98.686 3	79.393 5	99.797 1.5	99.797 1.5
yeast	58.55 5	63.409 1	60.036 4	60.912 2	60.845 3
zoo	94.181 4	97.09 2	91.181 5	98 1	96.09 3

Table 14: Average ranking of the algorithms.

Algorithms	J48	Random Forest	JRip	SMO	LibSVM
Average rank	4.095	2.5	4.119	1.88	2.404

5 Summary and Conclusion

In this thesis, we presented various existing learning methods used for classification (decision trees, random forests, rule learners and support vector machines). Then, we attempted to increase the accuracy of the algorithms implementing these approaches by optimizing the relevant parameters of each algorithm; this was achieved by running the different parameter settings on a fixed set of data and comparing the results. During the experiments, we noticed that it is possible to reach better accuracy when the parameters are carefully optimized. This is indicated in detail in table 12 which states, for every data, both the best accuracy obtained after the optimization process and the accuracy obtained when running the algorithms with their default settings.

When observing the results from table 10, we stated that the accuracy values and the variance highly depend on the datasets. We also noticed that the symbolic approaches perform better on average for the single data. The deviation from the average is also lower; this means that these algorithms are more stable on the datasets, which is not the case for SVM methods (for example, LibSVM has a high variance for all the data). We made a similar observation when analyzing the results from table 9, where we take the whole dataset into account: we concluded that the SVM algorithms have a high variance and perform worse on average than the other algorithms. Furthermore, we noticed that increasing the SVM cost parameter C generally produces better results. But the SVM technique was time consuming for our dataset which contains 21 data: the smallest one contains 24 instances and the biggest one consists of 2310 examples. Therefore, it would be worthwhile to see how the different algorithms perform on bigger data.

During our experiments, the best results were obtained by the random forest algorithm: On the one hand, when observing the resuls for single data (see table 10), this algorithm has a low deviation (similar to the decision tree learner and the rule learner) and produces good accuracy values (similar to support vector machines). On the other hand, when taking into account the whole data set (see table 9), the random forest algorithm has the lowest variance value and the best average accuracy.

We did also perform a significance test to compare the different approaches. The results of this test are represented in figure 17. It shows that the classifiers can be divided into two groups. The first groupe consists of the RandomForest method and the SVM algorithms, which were not significantly different in our experiments. Also, we did notice that J48 and JRip have similar results, and are generally less accurate than the algorithms of the first group.

In conclusion, we expect that if we combine an ensemble of multiple classifiers, we might attain similar results with support vector machines and have a lower variance as well.

References

[1] Tom M. Mitchell. *Machine Learning*. New York: McGraw-Hill, 1997.

[2] Ethem Alpaydin. *Introduction to Machine Learning, Second Edition*. The MIT Press Cambridge, Massachusetts, London, England, 2010.

[3] John Shawe-Taylor Nello Cristianini. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[4] Eibe Frank Ian H. Witten. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. Morgan Kaufmann, 2005.

[5] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[6] Micheline Kamber Jiawei Han. *Data Mining: Concepts and Techniques, Second Edition*. Morgan Kaufmann, 2006.

[7] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[8] Leo Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[9] Leo Breiman. Random Forests. *Machine Learning*, 45:5–32, 2001.

[10] Carolin Strobl, James Malley, and Gerhard Tutz. An Introduction to Recursive Partitioning: Rationale, Application, and Characteristics of Classification and Regression Trees, Bagging, and Random Forests. *Psychological Methods*, 14:323–348, 2009.

[11] R. A. Fisher. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, 7:179–188, 1936.

[12] Johannes Fürnkranz. Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13:3–54, 1999.

[13] Clifford Brunk and Michael J. Pazzani. An Investigation of Noise-Tolerant Relational Concept Learning Algorithms. In *International Conference on Machine Learning*, pages 389–393, 1991.

[14] William W. Cohen. Fast effective rule induction. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[15] Johannes Fürnkranz and Gerhard Widmer. Incremental Reduced Error Pruning. In *International Conference on Machine Learning*, pages 70–77, 1994.

[16] Johannes Fürnkranz. Pruning Algorithms for Rule Learning. *Machine Learning*, 27:139–172, 1997.

[17] J. Rissanen. Modeling by the shortest data description. *Automatica*, 14:465–471, 1978.

[18] Peter Grünwald. A tutorial introduction to the minimum description length principle. In *in Advances in Minimum Description Length: Theory and Applications. 2005*. MIT Press.

[19] Ivica Kopriva Te-Ming Huang, Vojislav Kecman. *Kernel Based Algorithms for Mining Huge Data Sets*. Springer-Verlag Berlin Heidelberg, 2006.

[20] S. Sathiya Keerthi, Shirish Krishnaj Shevade, Chiranjib Bhattacharyya, and K. R. K. Murthy. Improvements to Platt's SMO Algorithm for SVM Classifier Design. *Neural Computation*, 13:637–649, 2001.

[21] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[22] Geoffrey Holmes Bernhard Pfahringer Peter Reutemann Ian H. Witten Mark Hall, Eibe Frank. The WEKA data mining software: an update. *SIGKDD Explorations*, 11:10–18, 2009.

[23] K. Bache and M. Lichman. UCI machine learning repository. `http://archive.ics.uci.edu/ml` University of California, Irvine, School of Information and Computer Sciences., 2013.

[24] J. Platt. Fast training of support vector machines using sequential minimal optimization. 1998.

[25] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[26] Yasser EL-Manzalawy and Vasant Honavar. *WLSVM: Integrating LibSVM into Weka Environment*, 2005. Software available at `http://www.cs.iastate.edu/~yasser/wlsvm`.

[27] Janez Demsar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[28] R. L. Iman and J. M. Davenport. Approximations of the critical region of the Friedman statistic. *Communications in Statistics*, pages 571–595, 1980.

[29] P. B. Nemenyi. *Distribution-free multiple comparisons*. PhD thesis, Princeton University, 1963.

List of Tables

1 Examples from the weather data. 6
2 Examples from the iris data. 10
3 Datasets used for the parameters optimization. 17
4 Best accuracy values obtained by J48, with the corresponding parameter settings. The columns represent in the following order: the datasets, the confidence factor (C), the minimum number of instances per leaf (M) and the binary split (B) or Laplace (A) options. The last column corresponds to the best accuracy value. 20
5 Best accuracy values obtained by RandomForest, with the corresponding parameter settings. 22
6 Best accuracy values obtained by JRip, with the corresponding parameter settings. The columns represent in the following order: the datasets, the number of folds (F), the minimum total weight of the instances in a rule (N), the number of optimizations (O), the check error rate option (E) and the pruning option. The last column corresponds to the best accuracy value. 25
7 Best accuracy values obtained by SMO, with the corresponding parameter settings. The columns represent in the following order: the datasets, the soft margin parameter (C), the kernel type, the gamma or the exponent parameter (G or E) and the filter type (N). The last column corresponds to the best accuracy value. 30
8 Best accuracy values obtained by LibSVM, with the corresponding parameter settings. The columns represent in the following order: the datasets, the soft margin parameter (C), the kernel type, the gamma or the degree parameter (G or D) and the normalize option. The last column corresponds to the best accuracy value. 34
9 Calculation of the average (X), the variance (Var) and the standard deviation (SD) of every algorithm over the whole dataset. 35
10 An overview of the maximal-, the minimal-, the average accuracy and the standard deviation of every algorithm for the single data. 36
12 An overview of the maximal accuracy and the accuracy corresponding to the default settings of every algorithm for the single data. 38
13 Ranking of the algorithms for each data. 40
14 Average ranking of the algorithms. 41

List of Figures

1 Decision tree for the weather data. . . . 5
2 Determining the root of the decision tree for the weather data. . . . 6
3 Branch of the decision tree for the weather data. . . . 7
4 Example of subtree raising. . . . 8
5 A set of rules classifying instances from the iris problem. . . . 10
6 A maximum margin hyperplane. . . . 14
7 Variation of the accuracy depending on the C and M parameters for the heart data. . . . 19
8 Variation of the accuracy depending on the C and M parameters for the vowel data. . . . 19
9 Variation of the accuracy depending on the I and depth parameters for the vowel data. . . . 21
10 Variation of the accuracy depending on the O parameter for different data. . . . 23
11 Variation of the accuracy depending on the F and N parameters for the contact-lenses data. 24
12 Variation of the accuracy depending on the C and G parameters of the RBF kernel for the anneal data. . . . 27
13 Variation of the accuracy depending on the C and E parameters of the polykernel for the glass data. . . . 28
14 Variation of the accuracy depending on the C and E parameters of the normalized polykernel for the sonar data. . . . 29
15 Variation of the accuracy depending on the C and D parameters of the polykernel for the balance-scale data. . . . 32
16 Variation of the accuracy depending on the C and G parameters of the rbf kernel for the lymphography data. . . . 33
17 Results of the significance test. . . . 39