Università
Ca'Foscari
Venezia

Master's Degree programme – Second Cycle
(*D.M. 270/2004*)
in Computer Science

Final Thesis

Ca' Foscari
Dorsoduro 3246
30123 Venezia

Mining Top-K Classification Rules

Supervisor
Ch. Prof. Salvatore Orlando

Graduand
Cristian De Zotti
Matriculation Number 815356

Academic Year
2014 / 2015

Abstract

In this thesis we present a classifier that uses the associative classification approach. Specifically, we exploit the mined top-k patterns to extract classification rules from a transactional dataset, where each transaction is associated with a class attribute. The top-k patterns extracted are approximate and are aimed to concisely describe the dataset. The top-k pattern discovery problem is commonly stated as an optimization one, where the goal is to minimize a given cost function, watching the accuracy of the data description.

Indeed, for generating of candidate rules, we used a greedy algorithmic framework named PaNDa+, which extracts top-k patterns directly from training data. The pattern extraction is performed by partitioning the training set according to the attribute class. On each disjoint partition so obtained, we apply PaNDa+ by firstly removing the attribute class, thus obtaining a set of patterns that are strictly related to a specific class label. This allows us to naturally derive so-called classification rules, in turn exploited to classify. Once extracted the rules, however, we also calculate the prediction power of each rules by using many measures. Such measures are used to rank the rules that match a given test transaction, and permit us to identify the best rule to apply for classifying the transaction.

Finally, we evaluate the goodness of the our rule-based classifier by measuring the quality and the accuracy of the extracted rules. The evaluation was conducted on UCI data sets, and the results are compared with other classifiers, such as JCBA, CPAR, Weighted Classifier, SVM, and C4.5.

Contents

List of Tables

Chapter 1

Introduction

1.1 Overview

In this era of data overflow, Knowledge Discovery and Data Mining (KDD) is playing an important role in extracting knowledge. KDD consists of many methods and techniques that can be applied to different data to extract knowledge. Some of the methods include association, classification, and clustering. In this thesis, we primarily focus on association and classification.

Association rule mining was introduced to extract associations from market basket data [1]. Association rule mining has proven useful in many other domains (e.g. microarray data analysis, recommender systems, and network intrusion detection), and it can be used, in general, to find out the association relationships among a set of items in a dataset. Association rule mining has become an important data mining technique due to the descriptive and easily understandable nature of the rules. In the domain of market basket analysis, data consists of transactions where each is a set of items purchased by a customer. A method to measuring the usefulness of association rules, is to use the support that is the percentage of transactions that contain all the items of the rule, and the confidence, that is the percentage of the transactions that carry all the items of the rule among those transactions that contain the items of the antecedent of the rule. This method was introduced by [1].

The problem of association rule mining can be stated as: Given a dataset of transactions, a threshold support (minsupport), and a threshold confidence (minconfidence); Generate all association rules from the set of transactions that have support greater than or equal to minsupport and confidence greater

than or equal to minconfidence.

Classification is another method of data mining. Classification can be defined as learning a function that maps (classifies) a data instance into one of several predefined class labels [2]. Classification function or model is learned from a training set of data containing observations (or instances) whose class label membership is known, for identifying to which of a set of categories (sub-populations) a new observation belongs. To test the classifying ability of the learned model or function is used a testing set, where it is checked if the function or model correctly classifies instances. Examples of classification models include decision trees, Bayesian models, and neural nets. Classification rules are of the form $P \rightarrow c$, where P is a pattern in the training data and c is a predefined class label.

In this thesis, we study a different way of association rules extraction, based on the extraction of top-k pattern, and compare different ways of building models or classifier rules extracted from this, and with other existing classifier. In this work top-k pattern mining is an alternative approach to pattern extraction. It aims at discovering the (small) set of k patterns that best describes, or models, the input data. State-of-the-art algorithms differ in the formalization of the concept of dataset description. The goodness of a description is measured with some cost function, and the top-k mining task is casted into an optimization of such cost [3]. The pattern top-k extracted by the set of data are approximated pattern that are able to briefly describe the input data, and are then used as a classification rule in prediction, within our rule-based classifier.

For generating of candidate rules, we used a greedy algorithmic framework named PaNDa+, that extract top-k pattern directly from training data. With this method, a pattern set Π_k is extracted from a training set where the class labels have been removed in advance. For what regard the test set, we say that an approximate pattern $P \in \Pi_k$ occurs in an unseen test transaction t if and only if P intersection ratio is greater or equal than the minimum intersection ratio calculated for every training transaction. The rationale is to accept a pattern P for a test transaction t if it does not generate more noise than what it has been observed in the training set [4]. The previous operations allows us to determine whether a given pattern extracted by PaNDa is a valid rule, i.e., it is a possibile candidate rule to be applied for classifyng test transaction t. At the end, we have a set of valid rules for every test set transaction t. After mapping the transactions into such pattern space, the class labels are restored in the transformed training set, which is used

to train our classifiers. Classifiers named PaNDa$_{\eta}$, PaNDa$_{conf}$, PaNDa$_{supp}$, and PaNDa$_{laplace}$ were implemented with 4 different approaches, based on 4 different scoring functions that sort the valid rules, for every test set transaction, using measures like intersection ratio, confidence, support and Laplace accuracy. Finally, the classifiers are evaluated with accuracy measure on the mapped test set and results collected compared with each other and with other existing classifiers like JCBA, CPAR, Weighted Classifier, SVM and C4.5.

1.2 Problem Statement

The overall goal of this thesis is to design and develop a classification system that is based on Association Rule with using a Mining top-k algorithm. Our problem can be further broken down as follows:

- Adapt the training set data to generate classification association rules with PaNDa.
- Build Classification Models:
  - Select the association rule valid for each test set transaction.
  - For each test set transaction, apply the association rule valid and calculate the relative measure.
  - For each test set transaction, select as class the class pointed to the association rule that has the highest measure.
- Calculate the Accuracy of the Classification Model and the time of the execution.
- Compare our Classification Model with existing classifier like JCBA, CPAR, Weighted Classifier, SVM and C4.5.

Chapter 2

Background

In this chapter we present briefly what is the association rule, the classification, the classification association rule and the pattern top-k.

2.1 Association Rules

Association rule mining was introduced in [1] for discovered interesting relations between products in market basket data. Market basket data consist of transactions where a transaction is a set of products purchased by a customer. The motivation for applying this data mining approach on market basket data was that such information can be used as the basis for decisions about marketing activities, catalog design, promotional pricing or store layout design. Association rules are studied and employed today in many other domains including web usage mining, credit card fraud, intrusion detection, genetic data analysis. In every domain, there is a need to analyze data to identify patterns associating different attributes. Association rule mining, addresses this need.

Let $\mathcal{I}$ be a set of n attributes called items, $\mathcal{D}$ be a multiset of transactions (subset of the items in $\mathcal{I}$) and t be a transaction. A rule extracted from $\mathcal{D}$ is defined as an implication of the form: $X \Rightarrow Y$, where $X, Y \subseteq \mathcal{I}$ and $X \cap Y = \emptyset$. To illustrate the concepts, we use a small example from the supermarket domain. The set of items is $\mathcal{I} = \{\text{milk}, \text{bread}, \text{butter}, \text{beer}, \text{diapers}\}$ and in the table is shown a small dataset containing the items, where, in each entry, the value 1 or 0 means the presence or the absence of the item in the corresponding transaction.

An example rule for the supermarket could be $\{\text{butter}, \text{bread}\} \Rightarrow \{\text{milk}\}$ meaning that if butter and bread are bought, customers also buy milk. To

ID	milk	bread	butter	beer	diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

select interesting rules from the set of all possible rules, constraints on support and confidence measures are used. The support value of the rule X is the percentage of transactions t in dataset $\mathcal{D}$ which contains the item-set X. In our supermarket example, the item-set $\{\text{milk}, \text{bread}, \text{butter}\}$ has a support of $1/5 = 0.2$ since we can easily see it occurs in 20% of all transactions (1 out of 5 transactions). The confidence value of a rule, $X \Rightarrow Y$ is, among all the transactions that contain X, the percentage that contain Y as well. Confidence is defined as: $\text{conf}(X \Rightarrow Y) = \text{supp}(X \cup Y)/\text{supp}(X)$. For our example, the rule $\{\text{butter}, \text{bread}\} \Rightarrow \{\text{milk}\}$ has a confidence of $0.2/0.2 = 1.0$ in the dataset, which means that for 100% of the transactions containing butter and bread the rule is correct (100% of the times a customer buys butter and bread, milk is bought as well). Given a minimum support and minimum confidence threshold, a dataset of transaction where each transaction is a set of items, we can define a problem of association rule mining as a problem of find all association rules R that satisfy given support and confidence threshold.

2.2 Classification

Classification is the problem of identifying a function or a model from a data set (training set) where the class membership is known, so as to predict to which class a new observation belongs.
Classification models are frequently represented as rules of this form: $P \rightarrow c$ where P is a pattern in the training data (P forms the set of predicting attribute(s)) and c is the class label.

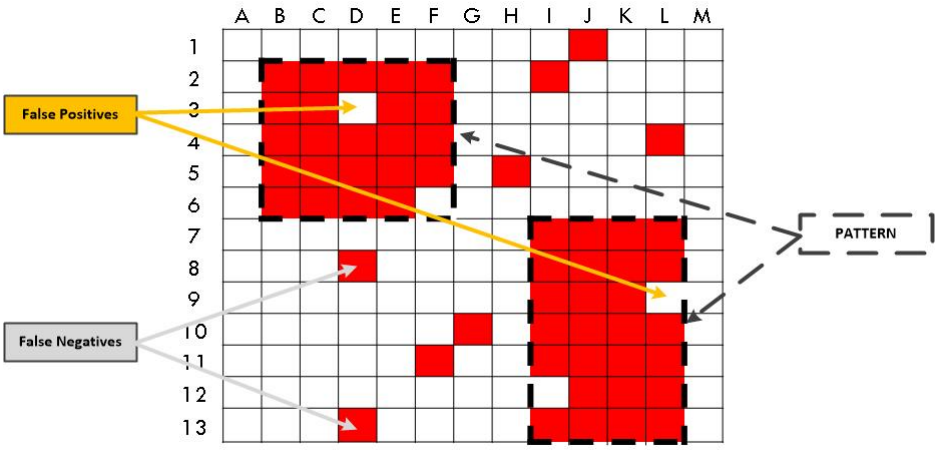
2.3 Classification Association Rules

Associative classification differs from general association rule mining by introducing a constraint as the attribute that must appear on the consequent of the rule. This association rules are produced with only a particular attribute in the consequent [5]. This particular case of constrained association rules

are called class association rules (CARs), and can be used to build a model or classifier. There has been several studies in this field concerning the integration of these constraints into the mining phase rather than filtering the enormous number of rules produced using the constraints as post-processing filters, with the goal of obtaining general advantages like faster execution and lower memory utilization. Many methods of classification rules extraction, once extracted all the frequent rules, which are all the rules that fulfill the constraints of support and confidence threshold, performing a pruning, to avoid that during the application of the rules are too many candidates applicable to a test transaction. In practice, only keep the rules "cover" quite the dataset from which they are extracted. In this way we have the best class association rules to use for the classification of new instances.

2.4 Top-k Pattern

A top-k pattern is an approximate pattern extracted from a matrix thus corresponds to a pair of sets, items and transactions, where the items of the former set are mostly included in all the transactions of the latter set. Top-k pattern mining is an alternative approach to pattern enumeration. It aims at discovering the (small) set of k patterns that best describes, or models, the input dataset. The goodness of a description is measured with some cost function, and the top-k mining task is casted into an optimization of such cost. There are greedy strategies adopted to solve this optimization problem. At each iteration, the pattern that best optimizes the given cost function is added to the solution, and this is repeated until k patterns have been found or until it is not possible to improve the cost function [3].



A simple example of patterns extraction, we can see in the figure above. Given a dataset with some noise all around, that simply flipped some of the bits in our representation, we can simply see that the patterns extracted are two: items $\{BCDEF\}$ occurring in transactions $\{2, 3, 4, 5, 6\}$ and items $\{IJKL\}$ occurring in transactions $\{7, 8, 9, 10, 11, 12, 13\}$. Patterns are represented as a hyper-rectangle, and may contain false positives, i.e. holes, which are missing items in the transactions supporting a given pattern. Inside the dataset, due to the noise, there may be false negatives, spurious occurrences, which may belong to any patterns, that are ignored in the pattern extraction.

Chapter 3

Contribution

In this chapter, we discuss the structure of our project, starting from the algorithm to extract patterns and rules, up to the exploitation of these rules to build a classifier. The rest of this chapter is organized as follows: Section 3.1 presents PaNDa, the algorithm to extract important patterns from a dataset, and Section 3.2 discusses how PaNDa is used in the project to classify unseen transactional records.

3.1 PaNDa Algorithm

An alternative approach to pattern enumeration, which has been very popular in the field of Market Basket Analysis, is top-k pattern mining. The set of top-k patterns extracted aims to best describes, or models, the input data, namely a transactional dataset where each record stores a subset of items. The goodness of the dataset description, made possible by the top-k patterns, is measured in terms of some cost function, and the objective of the top-k mining task is to optimize such cost.

This family of top-k pattern mining algorithms usually adopts a greedy strategy and adds, at each iteration, a pattern to the solution set if and only if it best optimizes (either minimize or maximize) the given cost function. This operation is repeated until either k patterns have been found, or it is no more possible to improve the cost function.

PaNDa[3] is a new algorithmic framework for mining top-k patterns, which improves the accuracy of each mined pattern, and can deal with a variety of cost functions. In addition, it is capable to deal with error noise tolerance thresholds. To extract patterns, PaNDa adopts a greedy strategy that consists in extracting an ordered sequence of patterns, by progressively increasing the degree of coverage of the dataset. We can decompose the

problem of pattern detection into two simpler problems:

1. Discovering of a noise-less pattern.

2. Pattern extension by allowing false positives to occur in the pattern.

PaNDa assumes that, even in presence of noise, it is possible to detect a *core* noise-less pattern, composed of a set of items actually occurring in a group of transactions. The core pattern, which is indeed an *exact* pattern, is then transformed into an *approximate* pattern P, where P identifies a group of transactions that mostly include a given itemset.

In order to obtain P, PaNDa starts from the core pattern, and considers all the items and transactions to further enlarge the core, to finally obtain its largest extension that minimizes the cost function. If the pattern obtained from this last operation reduces the current cost of the model, P is added to the result pattern set Π. PaNDa stops generating further patterns when:

- The one under examination does not improve the cost of the model.

- It has reached the maximum value of user provided parameter k.

Concerning the detection of core patterns, PaNDa at each iteration starts to discover them from portions of the dataset (residual dataset) not yet covered by any previous pattern.

3.2 Using PaNDa to extract classification rules

In this thesis we aim at using the pattern extracted by PaNDa to build a classifier based on *classification rules*. We start from a transactional dataset where every transaction is labeled with a discrete class attribute, i.e., a *training set*. The pattern extraction is performed by first partitioning the training set according to the attribute class. On each disjoint partition so obtained, we apply PaNDa by firstly removing the attribute class, thus obtaining a set of patterns that are related to the specific class label, and can be used to derive so-called *classification rules*. Once we got the patterns, we use them to make the classification.

More formally, let $\mathcal{D}$ be the training set, which indeed is a multiset of labelled transactions. Specifically, each element of $\mathcal{D}$ is a pair (t, c), where transaction t is a subset of items, $t \subseteq \mathcal{I}$, and c is a class label, $c \in C$. Symbol $\mathcal{I}$ corresponds to the collection of all the possible items occurring in a transaction, while C is the set of class attributes. Let $\mathcal{D}_c$ be the subset of $\mathcal{D}$ whose transactions are labeled by c ($\cup_{c \in C} \mathcal{D}_c = \mathcal{D}$), and let $\overline{\mathcal{D}}_c$ be the same

set $\mathcal{D}_c$ from which the class attribute have been eliminated for the purpose of extracting patterns. Similarly, we denote by $\overline{\mathcal{D}}$ the complete training set from which we have got rid of all the class labels.

Let Π_c be the set of *approximate patterns* extracted by PaNDa from $\overline{\mathcal{D}}_c$. Each patterns is indeed a pairs $\langle I, T\rangle$, where I is an itemset, and $T \subseteq \overline{D}_c$ is the set of transactions where I "occurs". However, since the pattern $\langle I, T\rangle$ is approximate, in principle we can have that given a transaction $t \in T$, we can have that $I \not\subseteq T$. Finally, the classification rules extracted are of the form $I \to c$, where $\langle I, T\rangle \in \Pi_c$.

Since $\langle I, T\rangle$ is an approximate pattern, we need a quantitive method to redefine the concept of inclusion of a pattern within a transaction. First we define the *minimum intersection ratio*, and then the *approximate inclusion* concept.

Definition 1. *[Minimum intersection ratio [4]]*
$\forall P = \langle I, T\rangle \in \Pi_c$, i.e., for each pattern extracted by PaNDa from the training set $\overline{\mathcal{D}}_c$, let ρ_P be the minimum intersection ratio for every transaction $t \in T$ defined as follows:

$$\rho_P = \min_{t \in T} \frac{|I \cap t|}{|I|}$$

Note that $\rho_P = 1.0$ iff $\forall t \in T$, we have that $I \subseteq t$.

Definition 2. *[Approximate inclusion]*
Given a transaction t' of the test set, and given a rule $I \to c$ derived from the pattern $P = \langle I, T\rangle \in \Pi_c$, the left hand of the rule is included *in t', denoted by $I \widetilde{\subseteq} t'$, iff $|I \cap t'|/|I| \geq \rho_P$.*

The previous definition allows us to determine whether a given classification rule $I \to c$ is a *valid rule*, i.e., it is a possible candidate rule to be applied for classifying t' as belonging to class c. According to this schema, given $\mathcal{R} = \{I \to c\}$ defined as the set of all the rules extracted by PaNDa, and given a test transaction t', the set of all the *candidate rules* for t' is

$$\mathcal{R}' = \{(I \to c) \in \mathcal{R} \mid I \widetilde{\subseteq} t'\}$$

Finally, from this set of candidate rules $\mathcal{R}'$ for test transaction t', we need to select the rule that most "fits" t'. To this end we can employ many scores for ranking the candidate rules, to finally select the best rule, and assign to t' the class c of the selected rule. In case of tie, the selected rule is the rule with the greater lenght.

In the following we discuss the possible scoring functions used in this work. Since some of the methods need to exploit measures based on the training set, in particular classical support and confidence of a rule, in the following we redefine these measures taking into account the approximate nature of each rule.

Definition 3. *[Approximate support and confidence]*
Given a rule $I \rightarrow c$ derived from the pattern $P = \langle I, T \rangle \in \Pi_c$, we define $\widetilde{supp}(I)$ and $\widetilde{supp}(I \cup c)$ as follows:

$$\widetilde{supp}(I) = \frac{|\{t \in \overline{\mathcal{D}} \mid I \widetilde{\subseteq} t\}|}{|\overline{\mathcal{D}}|}$$

$$\widetilde{supp}(I \cup c) = \frac{|\{t \in \overline{\mathcal{D}}_c \mid I \widetilde{\subseteq} t\}|}{|\overline{\mathcal{D}}|}$$

where for the second definition we limit ourself to $\mathcal{D}_c$, i.e., the transactions labeled by c.

Finally, the confidence is defined as follows:

$$\widetilde{conf}(I \rightarrow c) = \frac{\widetilde{supp}(I \cup c)}{\widetilde{supp}(I)}$$

3.2.1 PaNDa$_\eta$

Given a test transaction, a very simple method to score the various candidate valid rules, and thus guessing their prediction power, is to resort to a simple *intersection ratio*, denoted by η. Specifically, given a valid classification rule $r \equiv I \rightarrow c$ for a test transaction t', we define $\eta(r, t')$, $0 \leq \eta(r, t') \leq 1$, as the intersection ratio between the antecedent of the rule and t':

$$\eta(r, t') = \frac{|I \cap t'|}{|I|}$$

Therefore, if many valid rules there exist for t', we eventually select the rule with the *maximum value* of η. More formally, if $R(t')$ is the set of all the valid rules for t', we select the rule $\overline{r} \in R(t')$ such that:

$$\overline{r} = \underset{r \in R(t')}{\text{argmax}} \ \eta(r, t')$$

3.2.2 PaNDa$_{conf}$

Like in PaNDa$_\eta$, for making a prediction we calculate the intersection ratio η for every rule. In addition, for each rule we also calculate the confidence, which measures the reliability of the inference made by a rule. Given a rule $r \equiv I \rightarrow c$, the traditional definition of confidence is the ratio between the proportion of transactions in the training set which contains itemset $I \cup c$ and the proportion of transactions in the training set which contains itemset I only.

$$conf(r) = \frac{supp(I \cup c)}{supp(I)}$$

where $supp(I \cup c)$ is given by the number of transactions of the training set that contain all the elements of the rule r, and also the attribute class c, and $supp(I)$ is the number of train set transaction that contain all the elements of the itemset I.

For our purpose, however, since itemsets detected by PaNDa are approximate, we exploit the definition above of confidence $\widetilde{conf}(r)$. Therefore, given a test transaction t' for which rule r is valid, the measure of the prediction power of r is given by:

$$\eta(r, t') * \widetilde{conf}(r)$$

If many valid rules there exist for t', we eventually select the rule with the *maximum value* of the above measure. More formally, if $R(t')$ is the set of all the valid rules for t', we select the rule $\overline{r} \in R(t')$ such that:

$$\overline{r} = \underset{r \in R(t')}{\operatorname{argmax}} \ \eta(r, t') * \widetilde{conf}(r)$$

3.2.3 PaNDa$_{supp}$

This measure is similar to the one used in PaNDa$_{conf}$. We only use the approximate support $\widetilde{supp}(I)$ rather than $\widetilde{conf}(r)$. Given a test transaction t', if $R(t')$ is the set of all the valid rules for t', where every rule is in the form $r \equiv I \rightarrow c$, we select the rule $\overline{r} \in R(t')$ such that:

$$\overline{r} = \underset{r \in R(t')}{\operatorname{argmax}} \ \eta(r, t') * \widetilde{supp}(I)$$

3.2.4 PaNDa$_{laplace}$

In this method we use as another measure in addition to η, the Laplace expected error estimate [6], to predict the accuracy of rules. Given the rule

$r \equiv I \rightarrow c$, we have:

$$LaplaceAccuracy(r) = \frac{(n_c + 1)}{(n_{tot} + |C|)}$$

where $|C|$ is the number of classes, and n_{tot} is the total number of transactions in the training set satisfying the rule's body, and n_c are the subset of these transactions belonging to class c. Therefore $n_{tot} = \widetilde{supp}(I)$ and $n_c = \widetilde{supp}(I \cup c)$.

So, our measure of prediction power given a test transaction t' and a valid rule r is given by:

$$\eta(r, t') * LaplaceAccuracy(r)$$

Like in the other methods, given a test transaction t', if $R(t')$ is the set of all the valid rules for t', we select the rule $\overline{r} \in R(t')$ such that:

$$\overline{r} = \underset{r \in R(t')}{\operatorname{argmax}} \ \eta(r, t') * LaplaceAccuracy(r)$$

Chapter 4

Other Classification Algorithm

In this chapter we present briefly the other existing classification algorithms, used in our thesis to make comparisons with the performance of our implementations.

4.1 JCBA

JCBA is the java class implenting a java version of the CBA algorithm using a CrTree with Apriori carMiner algorithm. The CBA algorithm is described in: [5]. CBA first generates all the association rules with certain support and confidence thresholds as candidate rules. The rules extracted are in the form $< condset; y >$ where *condset* is a set of items, and y is the class labels. For all rules that have the same *condset*, the one with the highest confidence is selected as the representative of those rules, thus obtaining a small set of rules, to form a classifier. When predicting the class label for an example, the best rule (with the highest confidence) whose body is satisfied by the example is chosen for prediction.

4.2 CPAR

CPAR (Classification based on Predictive Association Rules) is classification approach which combines the advantages of both associative classification and traditional rule-based classification. CPAR, in comparison with associative classification, has the following advantages:

1. Generates a much smaller set of high-quality predictive rules with lower redundacy directly from the dataset;

2. To avoid generating redundant rules, CPAR generates each rule by considering the set of "already-generated" rules;

3. When predicting the class label of an example, CPAR uses expected accuracy to evaluate rules, and uses the best k rules that this example satisfies.

CPAR results to be much more time-efficient in both rule generation and prediction but achieves as high accuracy as associative classification. [7]

4.3 Weighted Classifier

Weighted Classifier is a java class implenting three different weighted classifiers for class association rules:

1. All rules are weighted equally with weight one (default behaviour).

2. All rules are weighted linearly.

3. All rules are weighted using the inverse function 1/position in sort order of mining algorithm.

The pruning step is omitted. All mined rules are used for classifcation, if not specified otherwise with the ruleLimit option. The extraction of the rule is performed by the Apriori carMiner java class implemented in Weka.

4.4 SVM

SVM in Weka is implemented by LIBSVM, that is a library for support vector classification, (C-SVC, nu-SVC), regression (epsilon-SVR, nu-SVR) and distribution estimation (one-class SVM). It supports multi-class classification [8]. SVM (Support Vector Machines) is an algorithm that analyze data and recognize patterns. Given a trainig set examples, each with a class label, SVM algorithm build a model, that is a representation of the examples as points in space, divided by a clear gap that is as wide as possible. New examples are assigned into one class or the other, based on which side of the gap they fall on.

4.5 C4.5

C4.5 is a decision tree algorithm [9], implemented in Weka by java class J48 [10]. Decision trees are a very effective method of supervised learning.

It aims at partitioning of a dataset into groups as homogeneous as possible in terms of the variable to be predicted. It takes as input a set of classified data, and outputs a tree that resembles to an orientation diagram where the root nodes are the top node of the tree and it considers all samples and selects the attributes that are most significant, each end node (leaf) is a decision (a class) and each non-final node (internal) represents a test. Each leaf node represents the decision of belonging to a class, and the rule are generated by illustrating the path from the root node to leaf node. C4.5 constructs a very big tree by considering all attribute values and finalizes the decision rule by pruning. Pruning consists in: if a sub-tree can only lead to a unique solution, then all sub-tree can be reduced to the simple conclusion, this simplifies the process and does not change the final result. Dealing huge data with computational efficiency is one of the major problem of C4.5.

Chapter 5

Experimental setting

We have conducted an extensive performance study to evaluate accuracy and execution time of our implementation and compare it with that of CPAR [7], JCBA, Weighted Classifier, SVM and C4.5. As in [7], 10 datasets from UCI Machine Learning Repository are used. All the experiments are performed on an Intel Core i7 PC with 16GB main memory.

5.1 Dataset

We tested the classification system with the following datasets obtained from the UCI Machine Learning Repository [11], where datasets continuous valued attributes were discretized for pattern mining [12].:

- abalone.
- ecoli.
- glass.
- ionosphere.
- iris.
- penDigits.
- segment.
- vehicle.
- wine.
- zoo.

Table 5.1 shows the properties of these datasets. As part of pre-processing, continuous valued attributes were discretized for pattern mining [12], to allow the use of the datasets by the algorithms used.

	# classes	# items	# transaction
abalone	3	8	4177
ecoli	8	7	336
glass	7	9	214
ionosphere	2	33	351
iris	3	4	150
penDigits	10	16	10992
segment	7	19	2310
vehicle	4	18	846
wine	3	13	178
zoo	7	16	101

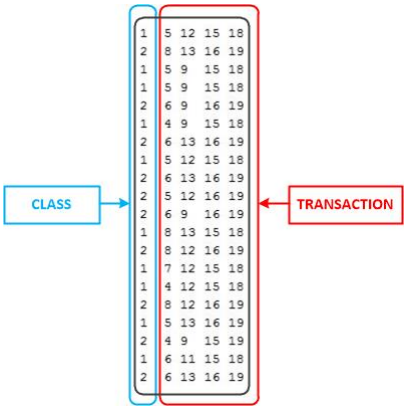
Table 5.1: Characteristics of the UCI datasets used for the experiments

5.2 Data structure for the experiments

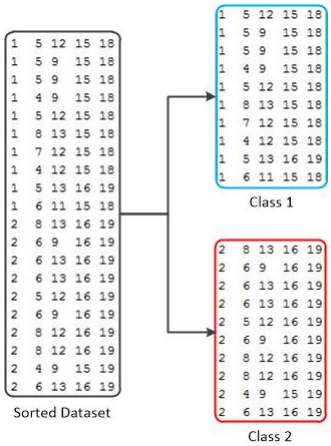
A modified version of 10-fold cross validation is used for every dataset. For the construction of the train and test set, we following these steps:

- The rows of the dataset are composed of a first element which corresponds to the class, and all of the other elements that make up the transaction. First of all we sorted rows according to the attribute class, and then subsequently we divided the data set according to the attribute class.
- Each newly created partition is divided into 10 parts. We take one part for each class partition and we merge to create the test set. With the remainder of each partition we create the training set, which consists of a nine parts for each class.
- The process is then repeated 10 times (the folds). We used for each class a different test partition among the 10 available, thus obtaining that each partition is used only once. For each run we also changes the training set, which is constructed using the remaining partitions of each class that is not used for the test.

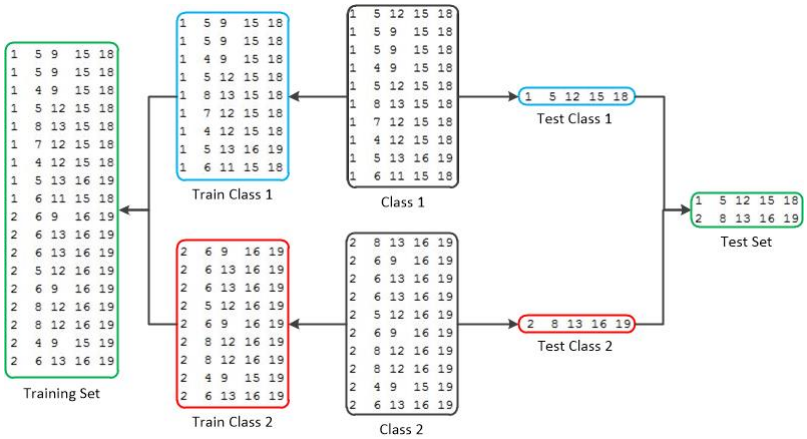
Let's take a simple example to understand better the operations. We take the case of a dataset with two classes:



As we can see in the figure below, we sort the dataset and then partition it according to the attribute class. We get two partitions of the initial dataset.



We split each partition into 10 parts. Take 1 part for the test set, and the remaining we use for the training. In our case, we have 10 transactions per class. We take one transaction per class to form the test set and the remaining 9 we use for the training.



Repeat the previous step, changing at each iteration the transaction that was used as a test, until the completion of the planned 10 fold.

The advantage of this method over repeated random sub-sampling is that all observations are used for both training and test, and each observation is used for test exactly once. Also by selecting the same amount of elements for each class, in each sub-sampling, there is a greater uniformity of the data in each experiment performed.

5.3 Weka

"WEKA" stands for the Waikato Environment for Knowledge Analysis, which was developed at the University of Waikato in New Zealand. Weka is extensible and has become a collection of machine learning algorithms for solving real-world data mining problems. It is written in Java and runs on almost every platform. Weka is easy to use and to be applied at several different levels. You can access the Weka class library from your own Java program, and implement new machine learning algorithms.

Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. Weka is open source software issued under the GNU General Public License and is used for research, education, and applications.

Weka, offering a wide range of classification algorithms, is used in our thesis for testing comparison of our implementations with other classification algorithms existing tools to obtain data about the quality of our implementations. In the particular case we use implementations in Weka of CBA, Weighted Classifier, SVM and C4.5.

5.4 Algorithms setting

In this section we show how are the setting of the various algorithms that were used to make our tests, showing the parameters used and the commands to perform the execution.

5.4.1 PaNDa

For the execution of PaNDa algorithm we can specify this series of parameter:

- **-d <dataset>**: input data file.
- **-k <# patterns>**: maximum number of patterns to be extracted (default: -1 → infinite).
- **-s <strategy>**: strategy for extraction **f** - for frequency, **c** - for child frequency, **o** - for correlated, **h** - for charm (default: f).
- **-r <# rnd iter>**: number of random iteration (0: no randomness), (default: 0).
- **-c <cost f>**: **1** - norm 1, **w** - norm s with weight, **2** - norm 2, **x** - typed xor, **n** - naive xor (default: 1).
- **-w <weight>**: weight to be used for the "-c w" option.
- **-o <output>**: default: non output file.
- **-a <data struct>**: **f** - for fptree, **v** - for full vertical (default: f).
- **-y**: row tolerance ratio.
- **-t**: column tolerance ratio.
- **-v**: verbose mode. Outputs cost per iteration.

For make our tests we use this four series of options to set PaNDa algorithm. For the test with default parameters we execute PaNDa we use the following sequence options:

```
./panda/panda -d <dataset> -k 5 -o out.w1
```

Instead to run with the cost function typed xor, we specify the parameter to option -c:

```
./panda/panda -d <dataset> -k 5 -c x -o out.w1
```

To run with error rate values, we specify the parameter to option -y for the row tolerance, and -t for the column tolerance:

```
./panda/panda -d <dataset> -k 5 -y 0.3 - t 0.3 -o out.w1
```

Instead to run with randomization value, we specify the parameter to option -r:

```
./panda/panda -d <dataset> -k 5 -r 10 -o out.w1
```

For the final tests we mix every parameter to obtain the maximun value of accuracy:

```
./panda/panda -d <dataset> -k 5 -r 10 -c x -y 0.3 -t 0.3 -o out.w1
```

5.4.2 CPAR

For the test with the algorithm CPAR use an implementation in java as indicated in [7], which has the following preset parameters that can not be changed:

- Rules per class label (k): 6
- totalWeightFactor: 0.001
- decayFactor: 0.25
- minGain: 1.0

To make a prediction on our test set, starting from our training set, we should be using the following command, which produces in output our test set with the label predicted by CPAR:

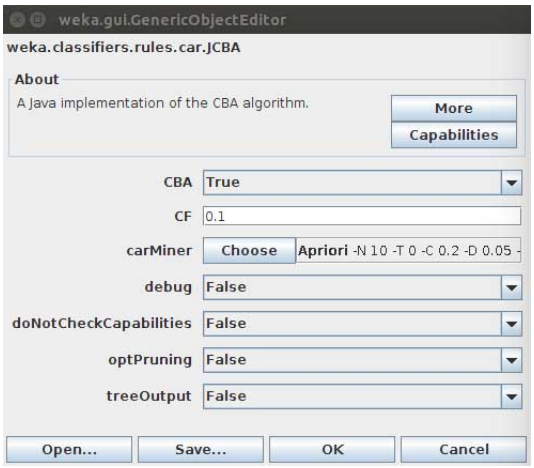
```
./predict train_set.w1 test_set.w1 label_test_set.w1
```

5.4.3 JCBA

Weka on the various algorithms classification association rule we chose to use JCBA which is an implementation of Java in CBA. We used by setting it with the following parameters to get the best performance:

- **CF**: Sets the confidence value for the optional pessimistic error rate based pruning step.
- **CBA**: If set to false, a simple decision list classification without pruning (no opt. and no obligatory pruning step) is performed, otherwise JCBA performs at least its obligatory pruning step.
- **carMiner**: The class association rule miner with its options.

- **debug**: If set to true, classifier may output additional info to the console.
- **doNotCheckCapabilities**: If set, classifier capabilities are not checked before classifier is built.
- **optPruning**: Enables or disabled the optional pessimistic error rate based pruning step.
- **treeOutput**: Enables/Disables the output of the mined rule set.



The parameters relating to the confidence value to make the pruning (CF) have been lowered from 0.25 to 0.1 to thus obtain the performace further by the algorithm.

As CAR miner within JCBA, Apriori algorithm it is used with the parameters that you see in the figure:

- **car**: If enabled class association rules are mined instead of (general) association rules.
- **classIndex**: Index of the class attribute. If set to -1, the last attribute is taken as class attribute.
- **delta**: Iteratively decrease support by this factor. Reduces support until min support is reached or required number of rules has been generated.
- **doNotCheckCapabilities**: If set, associator capabilities are not checked before asssociator is built.

- **lowerBoundMinSupport**: Lower bound for minimun support.

- **metricType**: Set the type of metric by which to rank rules. Confidence is the proportion of the examples covered by the premise that are also covered by the consequence (Class association rules can only be mined using confidence). Lift is confidence divided by the proportion of all examples that are covered by the consequence. This is a measure of the importance of the association that is independent of support. Leverage is the proportion of additional examples covered by both the premise and consequence above those expected if the premise and consequence were independent of each other. The total number of examples that this represents is presented in brackets following the leverage. Conviction is another measure of departure from independence. Conviction is given by $P(premise)P(!consequence)/P(premise, !consequence)$.

- **minMetric**: Minimun metric score. Consider only rules with scores higher than this value.

- **numRules**: Number of rules to find.

- **outputItemSets**: If enabled the itemset are output as well.

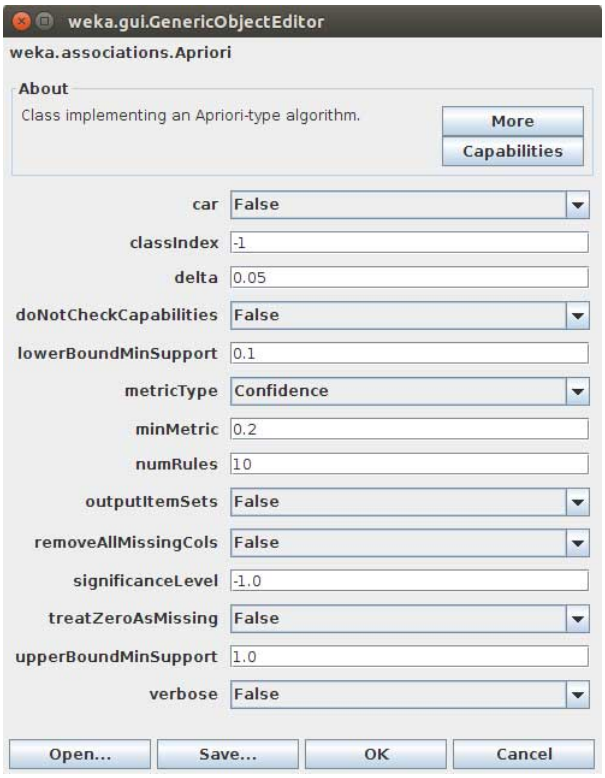
- **removeAllMissingCols**: Remove columns with all missing values.

- **significanceLevel**: Significance level.

- **treatZeroAsMissing**: If enabled, zero (that is, the first value of a nominal) is treated in the same way as a missing value.

- **upperBoundMinSupport**: Upper bound for a minimun support. Start iteratively decreasing minimun support from this value.

- **verbose**: If enabled the algorithm will be run in verbose mode.



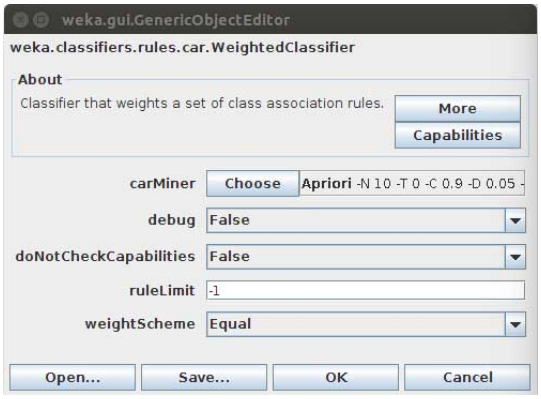
It been lowered the value of metric score (minMetric) that consider only rules with scores higher than this value, from 0.9 to 0.2, so that we get so many more rules for classification.

5.4.4 Weighted Classifier

Another classification association rule algorithm used for comparison is the weighted classifier that is located between the various algorithms available on Weka. The algorithm was set to perform our tests with the default parameters that you see in the figure below:

- **carMiner**: The class association rule miner with its options.
- **debug**: If set to true, classifier may output additional info to the console.
- **doNotCheckCapabilities**: If set, classifier capabilities are not checked before classifier is built (Use with caution to reduce runtime).
- **ruleLimit**: If set to -1 or 0, all rules in the pruned rule set are used for classification (default), otherwise it uses the specified number of top ranked rules in the pruned rule set.

- **weightScheme**: Specify the type of the weighting scheme: equal(default), linear, invers. Equal weights all instances equally with 1, linear uses a linear weighting function depending on the sort order of the mining algorithm, and inverse uses the inverse function 1/(rank in the sort order).



As in the case of JCBA, also the Weighted Classifier uses Apriori algorithm as CAR miner. In this case it is used with the following parameters:

- **car**: If enabled class association rules are mined instead of (general) association rules.
- **classIndex**: Index of the class attribute. If set to -1, the last attribute is taken as class attribute.
- **delta**: Iteratively decrease support by this factor. Reduces support until min support is reached or required number of rules has been generated.
- **doNotCheckCapabilities**: If set, associator capabilities are not checked before associator is built.
- **lowerBoundMinSupport**: Lower bound for minimum support.
- **metricType**: Set the type of metric by which to rank rules. Confidence is the proportion of the examples covered by the premise that are also covered by the consequence (Class association rules can only be mined using confidence). Lift is confidence divided by the proportion of all examples that are covered by the consequence. This is a measure of the importance of the association that is independent of support. Leverage is the proportion of additional examples covered by both the

premise and consequence above those expected if the premise and consequence were independent of each other. The total number of examples that this represents is presented in brackets following the leverage. Conviction is another measure of departure from independence. Conviction is given by $P(premise)P(!consequence)/P(premise, !consequence)$.

- **minMetric**: Minimum metric score. Consider only rules with scores higher than this value.

- **numRules**: Number of rules to find.

- **outputItemSets**: If enabled the itemsets are output as well.

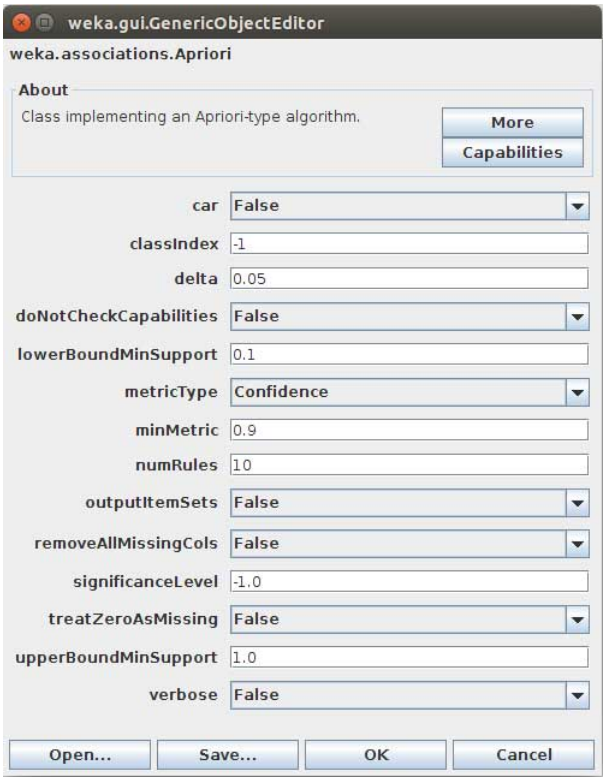
- **removeAllMissingCols**: Remove columns with all missing values.

- **significanceLevel**: Significance level.

- **treatZeroAsMissing**: If enabled, zero (that is, the first value of a nominal) is treated in the same way as a missing value.

- **upperBoundMinSupport**: Upper bound for minimum support. Start iteratively decreasing minimum support from this value.

- **verbose**: If enabled the algorithm will be run in verbose mode.



5.4.5 SVM

To perform another comparison with classification algorithms of different type use SVM within Weka, running it with the following parameters as shown in the figure below:

- **SVMType**: The type of SVM to use.
- **cacheSize**: The cache size in MB.
- **coef0**: The coefficient to use.
- **cost**: The cost parameter C for C-SVC, epsilon-SVR and nu-SVR.
- **debug**: If set to true, classifier may output additional info to the console.
- **degree**: The degree of the kernel.
- **doNotCheckCapabilities**: If set, classifier capabilities are not checked before classifier is built.

- **doNotReplaceMissingValues**: Whether to turn off automatic replacement of missing values.

- **eps**: The tolerance of the termination criterion.

- **gamma**: The gamma to use, if 0 then $1/max_indexisused$.

- **kernelType**: The type of kernel to use.

- **loss**: The epsilon for the loss function in epsilon-SVR.

- **modelFile**: The file to save the libsvm-internal model to; no model is saved if pointing to a directory.

- **normalize**: Whether to normalize the data.

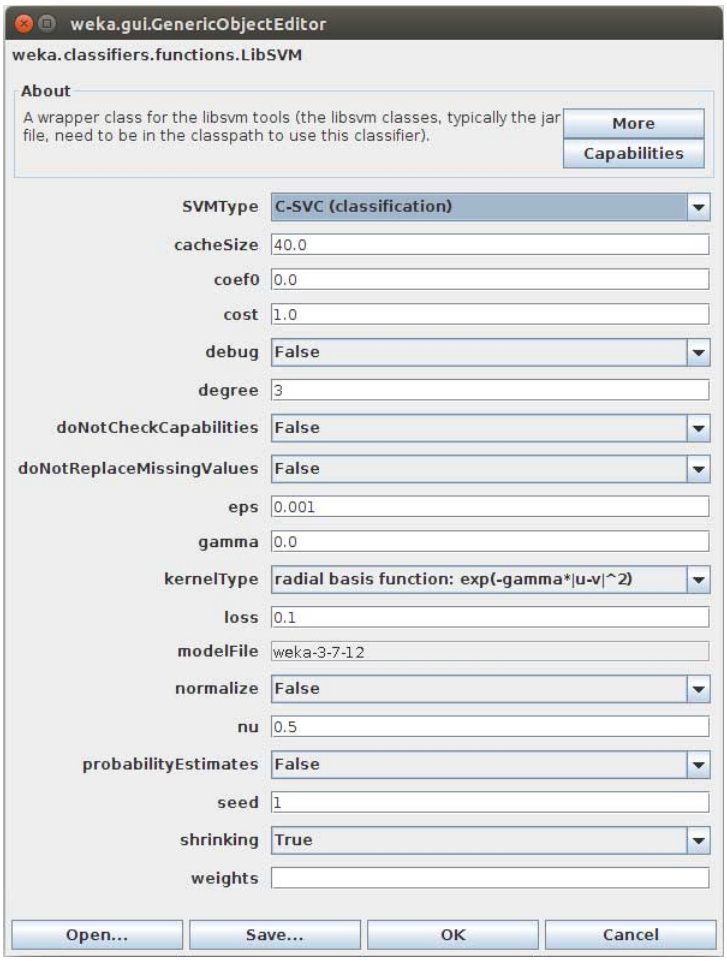
- **nu**: The value of nu for nu-SVC, one-class SVM and nu-SVR.

- **probabilityEstimates**: Whether to generate probability estimates instead of -1/+1 for classification problems.

- **seed**: The random number seed to be used.

- **shrinking**: Whether to use the shrinking heuristic.

- **weights**: The weights to use for the classes (blank-separated list, eg, "1 1 1" for a 3-class problem), if empty 1 is used by default.



5.4.6 C4.5

As a final test, to diversify the types of classifiers and have a broader comparison, we use the implementation of C4.5 tree classifier in Weka. How we use the following parameters in C4.5:

- **binarySplits**: Whether to use binary splits on nominal attributes when building the trees.
- **collapseTree**: Whether parts are removed that do not reduce training error.
- **confidenceFactor**: The confidence factor used for pruning (smaller values incur more pruning).
- **debug**: If set to true, classifier may output additional info to the console.

- **doNotCheckCapabilities**: If set, classifier capabilities are not checked before classifier is built.

- **doNotMakeSplitPointActualValue**: If true, the split point is not relocated to an actual data value. This can yield substantial speed-ups for large datasets with numeric attributes.

- **minNumObj**: The minimum number of instances per leaf.

- **numFolds**: Determines the amount of data used for reduced-error pruning. One fold is used for pruning, the rest for growing the tree.

- **reduceErrorPruning**: Whether reduced-error pruning is used instead of C.4.5 pruning.

- **saveInstanceData**: Whether to save the training data for visualization.

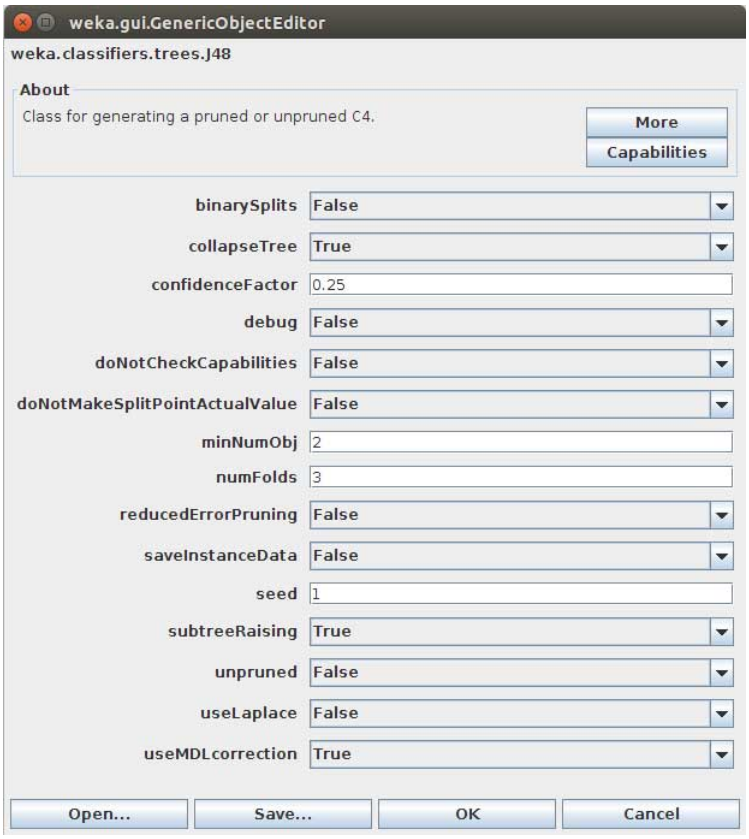
- **seed**: The seed used for randomizing the data when reduced-error pruning is used.

- **subtreeRaising**: Whether to consider the subtree raising operation when pruning.

- **unpruned**: Whether pruning is performed.

- **useLaplace**: Whether counts at leaves are smoothed based on Laplace.

- **useMDLcorrection**: Whether MDL correction is used when finding splits on numeric attributes.



With Weighted Classfier, SVM, and C4.5 we use the default parameters because it is the better solutions to obtain the best results of accuracy with this algorithms.

5.5 Experimental design

From our training set just created with the 10-fold cross validation, and set all the parameters of the various algorithms as shown in the previous sections, we start with the execution of the various tests. First we begin to run PaNDa with the default parameters, do the validation pattern extracted and send them to the vaults PaNDa$_{\eta}$, PaNDa$_{conf}$, PaNDa$_{supp}$ and PaNDa$_{laplace}$. After finishing the classification, the accuracy is calculated according to the test set that we have available. It also calculated the running time from the start of that part of pandas and finishes by calculating the accuracy.

We repeat the test by changing the parameters of the cost function, error rate and randomization of PaNDa to see the performance difference. Later we modify the training set by adding at the end of each transition the attribute

class and perform the algorithm CPAR with the training set just modified, calculating on completion the value of accuracy and execution time. Once completed the execution of CPAR, we send the training set and the test set to Weka where we execute the algorithms JCBA, Weighted Classifier, SVM and C4.5, to collecting more data on the accuracy and execution time. Once finished the execution of various algorithms, and repeated for the 10 fold, the final results are calculated with the arithmetic mean of the different values of accuracy, and execution time and stored on a text file.

Chapter 6

Experimental evaluation

In this chapter, we describe the experiments carried out to compare and evaluate our classification system. We break down this section into data description, evaluation metrics and experimental results. We show the performance related to accuracy and execution time of our classifiers and classifiers JCBA, CPAR, Weighted Classifier, SVM and C4.5.

6.1 Evaluation Metrics

We evaluate the classifier based on accuracy rate, and we also evaluate the execution time to execute the model. The accuracy rate signifies the number of correct predictions over the total number of predictions, while the execution time is calculated from the start of generation of the association rules, until the completion of the classification of the test set transaction.

$$accuracy = \frac{number\ of\ correct\ classifications}{total\ number\ of\ classifications\ made}$$

A prediction involves selecting an appropriate class label for a case whose class label is unknown. For example, let $< x_1, x_2, ...x_k, ? >$ be a data instance whose class label is unknown (denoted by a question mark). x_i represents the value of attribute i of the instance. If this data instance is given as an input to a model, the rule that "covers" this instance (the rule with the highest cover measure related to this instance) will determine the class label for the data instance.

6.2 Experimental Results

In this set of experiments, we compared the performances of PaNDa$_{\eta}$, PaNDa$_{conf}$, PaNDa$_{supp}$, and PaNDa$_{laplace}$ with well-know classifiers such as JCBA, CPAR, Weighted Classifer, SVM, and C4.5. JCBA, Weighted Classifer, SVM, and C4.5 are performed using Weka, while our implementations and CPAR are performed directly using the source code.

Below we are presented the results collected relating to accuracy:

	JCBA	**CPAR**	**Weighted**	**SVM**	**C4.5**
abalone	23.48	**46.27**	20.79	31.06	31.33
ecoli	43.80	2.26	23.03	**77.92**	75.36
glass	16.66	23.33	26.00	59.44	**60.93**
ionosphere	80.27	2.35	37.14	**86.79**	79.15
iris	**94.00**	33.33	50.00	89.00	91.00
penDigits	20.49	37.57	10.15	**98.48**	95.51
segment	13.33	63.38	15.15	**93.99**	93.48
vehicle	32.86	42.93	20.63	**62.86**	60.00
wine	49.54	71.25	40.74	**89.02**	85.68
zoo	60.00	**100.00**	50.00	**100.00**	**100.00**
Average	43.44	42.27	29.36	**78.56**	77.24

Table 6.1: Accuracy (%)

	PaNDa$_{\eta}$	**PaNDa$_{conf}$**	**PaNDa$_{supp}$**	**PaNDa$_{laplace}$**
abalone	**50,10**	48,89	50,02	50,07
ecoli	80,00	78,71	79,68	**80,32**
glass	48,89	48,33	**50,00**	**50,00**
ionosphere	72,06	70,59	71,47	**72,35**
iris	94,00	94,00	94,00	94,00
penDigits	**61,89**	41,69	61,87	**61,89**
segment	60,65	56,62	**60,74**	60,65
vehicle	**50,12**	47,32	49,76	50,00
wine	60,63	61,25	61,88	**62,50**
zoo	67,50	67,50	**70,00**	**70,00**
Average	64,58	61,49	64,94	**65,18**

Table 6.2: Accuracy (%) Default Parameters

Table 6.1 shows the accuracy results for the UCI datasets using JCBA, CPAR, Weighted Classifier, SVM and C4.5, while Table 6.2 shows the accu-

racy results of our implementations with default parameters on PaNDa. We can see, using this dataset that the best of our implementations is that of PaNDa$_{laplace}$, followed closely by PaNDa$_{supp}$ and then by PaNDa$_{\eta}$ and finally PaNDa_$conf$. Compared to CPAR, JCBA and Weighted Classifier, our implementations have an accuracy much higher. In the case of SVM and C4.5 our classifiers have a value of accuracy lowest.

6.2.1 Cost function

In the following tables we show how the behavior of accuracy is modified, by changing the cost function of PaNDa algorithm:

	PaNDa$_{\eta}$	**PaNDa$_{conf}$**	**PaNDa$_{supp}$**	**PaNDa$_{laplace}$**
abalone	0,00	0,00	0,00	0,00
ecoli	45,16	45,16	45,16	45,16
glass	0,00	0,00	0,00	0,00
ionosphere	**64,71**	0,00	0,00	0,00
iris	0,00	0,00	0,00	0,00
penDigits	10,41	10,41	10,41	10,41
segment	**14,29**	0,00	0,00	0,00
vehicle	0,00	0,00	0,00	0,00
wine	31,25	31,25	31,25	31,25
zoo	**36,25**	1,25	1,25	1,25
Average	**20,21**	8,81	8,81	8,81

Table 6.3: Accuracy (%) Norm 2 Cost Function

	PaNDa$_{\eta}$	**PaNDa$_{conf}$**	**PaNDa$_{supp}$**	**PaNDa$_{laplace}$**
abalone	**49,71**	48,75	49,64	**49,71**
ecoli	79,03	**79,35**	78,71	**79,35**
glass	58,33	53,89	58,33	**59,44**
ionosphere	72,35	70,29	71,76	**72,65**
iris	90,67	90,67	90,67	90,67
penDigits	59,82	41,98	**59,83**	59,82
segment	**60,65**	47,23	**60,65**	**60,65**
vehicle	**52,32**	48,17	51,83	**52,32**
wine	**75,63**	68,75	75,00	**75,63**
zoo	42,50	42,50	42,50	42,50
Average	64,10	59,16	63,89	**64,27**

Table 6.4: Accuracy (%) Naive XOR Cost Function

	PaNDa$_{\eta}$	**PaNDa$_{conf}$**	**PaNDa$_{supp}$**	**PaNDa$_{laplace}$**
abalone	**48,75**	45,67	48,68	48,70
ecoli	78,06	78,06	77,74	**78,39**
glass	**51,11**	**51,11**	46,11	50,00
ionosphere	**75,00**	71,18	73,24	**75,00**
iris	91,33	**95,33**	**95,33**	**95,33**
penDigits	51,74	38,12	**51,75**	51,74
segment	**37,75**	36,32	**37,75**	**37,75**
vehicle	**52,44**	45,24	51,95	52,32
wine	78,13	69,38	77,50	**78,75**
zoo	**98,75**	96,25	97,50	97,50
Average	66,31	62,67	65,76	**66,55**

Table 6.5: Accuracy (%) Typed XOR Cost Function

Table 6.3, 6.4, 6.5, shows the accuracy obtained using PaNDa with Norm 2 cost function, Naive XOR and Typed XOR cost function. The worst solution between the three cost function is the Norm 2, because it obtained the lowest values of accuracy with respect to any other.
Compared to the version with the default parameter, Naive XOR have a decrease in the average of accuracy of all our implementation. Typed XOR instead it achieved an increase in the average of accuracy of all our implementations respect to the version with the default parameters.
Ultimately, cost functions better than the default (Norm 1) for PaNDa$_{\eta}$, PaNDa$_{supp}$, PaNDa$_{laplace}$ and PaNDa$_{conf}$ is the Typed XOR cost function.

6.2.2 Error Rate

In the following tables we show how the behavior of accuracy is modified, by changing the error rate parameter of PaNDa algorithm. Error rate was varied to the rows and columns with identical values, starting from 0,0 up to 1,0 with steps of 0,1:

	0,0	**0,1**	**0,2**	**0,3**	**0,4**	**0,5**	**0,6**	**0,7**	**0,8**	**0,9**	**1,0**
abalone	36,78	36,78	46,56	48,58	47,98	**49,28**	49,01	48,61	48,73	48,75	48,75
ecoli	68,06	68,06	70,00	70,32	73,87	**78,06**	**78,06**	**78,06**	**78,06**	**78,06**	**78,06**
glass	56,11	56,11	**59,44**	57,78	52,22	52,78	51,67	52,22	51,67	51,11	51,11
ionosphere	**82,06**	80,59	81,76	77,35	73,24	75,29	66,18	75,88	74,71	74,71	75,00
iris	90,67	92,67	92,67	92,67	92,67	**93,33**	**93,33**	**93,33**	**93,33**	**93,33**	**93,33**
penDigits	73,50	73,53	74,32	**74,37**	74,05	72,35	62,66	58,21	54,75	52,84	51,74
segment	**70,04**	**70,04**	65,37	62,81	57,14	43,29	38,44	37,71	37,71	37,75	37,75
vehicle	42,80	42,80	47,80	50,12	50,37	52,44	52,56	**52,93**	52,80	52,44	52,44
wine	76,88	76,88	77,50	**85,63**	80,63	76,88	78,13	77,50	77,50	78,13	78,13
zoo	96,25	96,25	**98,75**	97,50	97,50	**98,75**	**98,75**	**98,75**	**98,75**	**98,75**	**98,75**
Average	69,32	69,37	71,42	**71,71**	69,97	69,24	66,88	67,32	66,80	66,59	66,51

Table 6.6: PaNDa$_\eta$ Accuracy (%) Error Rate variation

Table 6.6, show the accuracy of PaNDa$_\eta$ obtained using PaNDa with error rate variation. Looking at the average of the values for accuracy, we can see that for error values of 0,3, we have the best accuracy performance.

	0,0	**0,1**	**0,2**	**0,3**	**0,4**	**0,5**	**0,6**	**0,7**	**0,8**	**0,9**	**1,0**
abalone	41,20	41,20	44,11	47,26	**48,41**	48,12	47,00	45,84	45,65	45,67	45,67
ecoli	68,39	68,39	70,00	70,32	73,87	**78,06**	**78,06**	**78,06**	**78,06**	**78,06**	**78,06**
glass	52,78	52,78	56,67	**57,22**	56,11	53,89	51,67	51,67	51,11	51,11	51,11
ionosphere	**82,06**	80,59	**82,06**	77,35	72,94	64,41	72,06	70,29	71,18	71,18	71,18
iris	94,67	94,67	94,67	94,67	94,67	**95,33**	**95,33**	**95,33**	**95,33**	**95,33**	**95,33**
penDigits	73,53	73,56	**74,13**	73,58	70,61	61,10	45,39	38,98	38,52	38,49	38,12
segment	**70,09**	**70,09**	65,41	62,90	57,14	41,73	36,32	36,41	36,36	36,41	36,32
vehicle	45,12	45,12	46,95	50,24	**51,59**	50,61	46,34	45,73	45,49	45,12	45,24
wine	76,88	76,88	77,50	**85,00**	76,88	72,50	69,38	69,38	69,38	69,38	69,38
zoo	96,25	96,25	**98,75**	96,25	95,00	96,25	96,25	96,25	96,25	96,25	96,25
Average	70,10	69,95	71,02	**71,48**	69,72	66,20	63,78	62,79	62,73	62,70	62,67

Table 6.7: PaNDa$_{conf}$ Accuracy (%) Error Rate variation

In the case of PaNDa$_{conf}$, the average value of the accuracy shown in table 6.7, achieves the best accuracy by using value 0,3 of error rate.

	0,0	**0,1**	**0,2**	**0,3**	**0,4**	**0,5**	**0,6**	**0,7**	**0,8**	**0,9**	**1,0**
abalone	41,20	41,20	44,11	47,14	48,13	49,18	**49,25**	48,51	48,65	48,68	48,68
ecoli	65,48	65,48	67,42	68,06	71,61	**77,74**	**77,74**	**77,74**	**77,74**	**77,74**	**77,74**
glass	50,56	50,56	**54,44**	53,33	50,00	48,33	46,67	47,22	46,67	46,11	46,11
ionosphere	**79,12**	77,65	**79,12**	74,71	72,06	74,71	64,12	74,12	72,94	72,94	73,24
iris	94,67	94,67	94,67	94,67	94,67	**95,33**	**95,33**	**95,33**	**95,33**	**95,33**	**95,33**
penDigits	73,56	73,59	**74,37**	**74,37**	74,03	72,31	62,64	58,23	54,76	52,86	51,75
segment	**70,13**	**70,13**	65,45	62,94	57,23	43,20	38,44	37,75	37,71	37,75	37,75
vehicle	45,24	45,24	47,68	49,88	50,12	52,07	52,20	**52,44**	52,32	51,95	51,95
wine	73,13	73,13	75,00	**81,88**	77,50	76,88	77,50	76,88	76,88	77,50	77,50
zoo	96,25	96,25	**98,75**	96,25	96,25	97,50	97,50	97,50	97,50	97,50	97,50
Average	68,93	68,79	70,10	**70,32**	69,16	68,73	66,14	66,57	66,05	65,84	65,76

Table 6.8: PaNDa$_{supp}$ Accuracy (%) Error Rate variation

	0,0	**0,1**	**0,2**	**0,3**	**0,4**	**0,5**	**0,6**	**0,7**	**0,8**	**0,9**	**1,0**
abalone	41,20	41,20	44,11	47,12	48,13	49,18	**49,28**	48,56	48,68	48,70	48,70
ecoli	68,71	68,71	70,65	71,29	74,84	**78,39**	**78,39**	**78,39**	**78,39**	**78,39**	**78,39**
glass	53,33	53,33	**57,22**	**57,22**	53,89	52,22	50,56	51,11	50,56	50,00	50,00
ionosphere	**82,06**	80,59	**82,06**	77,65	74,41	75,59	66,76	75,88	74,71	74,71	75,00
iris	94,67	94,67	94,67	94,67	94,67	**95,33**	**95,33**	**95,33**	**95,33**	**95,33**	**95,33**
penDigits	73,53	73,56	74,35	**74,37**	74,05	72,35	62,65	58,21	54,75	52,84	51,74
segment	**70,09**	**70,09**	65,41	62,90	57,14	43,29	38,44	37,71	37,71	37,75	37,75
vehicle	45,12	45,12	47,44	49,63	50,00	52,32	52,56	**52,93**	52,80	52,32	52,32
wine	76,88	76,88	77,50	**85,00**	80,63	78,13	78,75	78,13	78,13	78,75	78,75
zoo	96,25	96,25	**98,75**	96,25	96,25	97,50	97,50	97,50	97,50	97,50	97,50
Average	70,18	70,04	71,22	**71,61**	70,40	69,43	67,02	67,37	66,85	66,63	66,55

Table 6.9: PaNDa$_{laplace}$ Accuracy (%) Error Rate variation

To PaNDa$_{supp}$ and PaNDa$_{laplace}$ instead, observing the tables 6.8 and 6.9, we can see that they get the maximum of accuracy average of 70,32 and 71,61 respectively, using error rates of 0,3 for PaNDa$_{supp}$ and 0,3 for PaNDa$_{laplace}$.

6.2.3 Randomization

In this section we see how it changes the value of accuracy for PaNDa$_{\eta}$, PaNDa$_{conf}$, PaNDa$_{supp}$ and PaNDa$_{laplace}$, by changing the randomization parameter of PaNDa algorithm. The randomization was varied using 1, 5, 10, 15, 20 as values:

	1	**5**	**10**	**15**	**20**
abalone	48,58	**48,89**	44,30	41,92	42,79
ecoli	**80,32**	77,74	78,71	79,68	79,03
glass	**61,67**	54,44	58,89	59,44	57,78
ionosphere	70,29	**76,18**	74,41	71,18	72,06
iris	91,33	92,67	**93,33**	**93,33**	**93,33**
penDigits	**64,56**	55,31	57,37	57,53	53,81
segment	**57,88**	52,60	53,55	55,89	56,84
vehicle	48,66	51,10	53,54	53,17	**53,90**
wine	83,75	**85,00**	81,88	82,50	81,25
zoo	**98,75**	96,25	**98,75**	98,75	97,50
Average	**70,58**	69,02	69,47	69,34	68,83

Table 6.10: PaNDa$_{\eta}$ Accuracy (%) Randomization variation

	1	**5**	**10**	**15**	**20**
abalone	42,64	**45,94**	40,60	41,66	40,79
ecoli	**80,65**	78,71	78,39	78,71	79,03
glass	**56,67**	51,11	52,78	53,89	51,67
ionosphere	72,06	74,71	74,12	**75,00**	73,24
iris	**95,33**	**95,33**	94,67	94,67	**95,33**
penDigits	**35,84**	26,72	24,86	25,53	24,47
segment	**49,83**	43,90	43,29	43,90	44,63
vehicle	46,83	45,49	45,73	43,90	**47,20**
wine	**81,88**	81,25	79,38	81,25	**81,88**
zoo	**96,25**	93,75	**96,25**	96,25	95,00
Average	**65,80**	63,69	63,01	63,48	63,32

Table 6.11: PaNDa$_{conf}$ Accuracy (%) Randomization variation

Observing the values of the average accuracy in Tables 6.10 and 6.11 we can see that we get the maximum value with value 1 of randomization for PaNDa$_{\eta}$ and 1 for PaNDa$_{conf}$.

	1	**5**	**10**	**15**	**20**
abalone	48,63	**49,11**	44,25	41,90	42,76
ecoli	**80,00**	77,42	78,39	79,35	78,71
glass	**56,11**	51,67	53,89	55,56	53,33
ionosphere	69,41	**75,29**	73,24	69,71	70,29
iris	**95,33**	**95,33**	94,67	94,67	**95,33**
penDigits	**64,58**	55,31	57,39	57,55	53,82
segment	**57,58**	52,47	53,42	55,71	56,58
vehicle	50,98	51,22	53,66	53,05	**54,39**
wine	**82,50**	**82,50**	79,38	79,38	78,75
zoo	**97,50**	95,00	**97,50**	**97,50**	96,25
Average	**70,26**	68,53	68,58	68,44	68,02

Table 6.12: PaNDa$_{supp}$ Accuracy (%) Randomization variation

	1	**5**	**10**	**15**	**20**
abalone	48,65	**49,16**	44,30	41,92	42,76
ecoli	**80,65**	78,06	79,03	80,00	79,35
glass	**59,44**	54,44	56,67	57,78	56,11
ionosphere	70,88	**76,47**	74,41	71,18	72,06
iris	**95,33**	**95,33**	94,67	94,67	**95,33**
penDigits	**64,55**	55,31	57,37	57,53	53,81
segment	**57,88**	52,60	53,55	55,89	56,84
vehicle	51,10	50,98	53,41	53,17	**54,27**
wine	**83,75**	83,13	81,88	82,50	80,00
zoo	**97,50**	95,00	**97,50**	**97,50**	96,25
Average	**70,97**	69,05	69,28	69,21	68,68

Table 6.13: PaNDa$_{laplace}$ Accuracy (%) Randomization variation

For PaNDa$_{supp}$ the value of randomization for the accuracy average maximum is 1 and also for PaNDa$_{laplace}$ is 1. The data collected are shown in Tables 6.12 and 6.13 above.

6.2.4 Randomization+Error

This section shows the results of accuracy for every our implementation, using for each one, the parameters of error rate and randomization, which obtained in the above tables the maximum value of average accuracy. The following table contains the values of error rate and randomization used for the tests:

	Error Rate	**Randomization**
PaNDa$_{\eta}$	0,3	1
PaNDa$_{conf}$	0,3	1
PaNDa$_{supp}$	0,3	1
PaNDa$_{laplace}$	0,3	1

Using the same parameters of error rate and randomization, we did a test for each of the three cost functions with best average values of accuracy (Norm 1, Naive XOR and Typed XOR).

In the following tables we can see the results of accuracy obtained from the tests described above. As we can see in the three tables, the best result of accuracy average gets PaNDa$_{laplace}$, followed by PaNDa$_{\eta}$, then by PaNDa$_{supp}$ and finally PaNDa$_{conf}$. PaNDa$_{laplace}$ gets the highest average accuracy using the PaNDa algorithm with the following parameters: Typed XOR cost function, error rate of 0,3 and randomization value of 10. PaNDa$_{\eta}$ however, gets his best result with default cost function, error rate of 0.3 and randomization 1, while for PaNDa$_{supp}$ we get it with Typed XOR cost function, error rate 0,3 and randomization 1, and PaNDa$_{conf}$ finally, gets the best result with default cost function, error rate 0,3 and randomization 1.

	PaNDa$_{\eta}$	**PaNDa**$_{conf}$	**PaNDa**$_{supp}$	**PaNDa**$_{laplace}$
abalone	**47,81**	47,31	47,33	47,28
ecoli	77,10	76,13	75,48	**77,74**
glass	**57,22**	**57,22**	54,44	56,11
ionosphere	76,18	75,29	74,41	**76,47**
iris	93,33	**94,00**	**94,00**	**94,00**
penDigits	**76,32**	74,80	76,26	76,30
segment	63,68	62,77	63,77	**63,85**
vehicle	48,41	**49,15**	48,78	49,02
wine	**80,00**	79,38	76,88	79,38
zoo	**98,75**	97,50	**98,75**	**98,75**
Average	71,88	71,35	71,01	**71,89**

Table 6.14: Accuracy (%) Error Rate, Randomization and Norm 1 Cost Function

	PaNDa$_{\eta}$	**PaNDa**$_{conf}$	**PaNDa**$_{supp}$	**PaNDa**$_{laplace}$
abalone	40,75	45,26	**45,36**	**45,36**
ecoli	77,10	76,77	75,81	**78,06**
glass	57,78	**58,89**	57,78	**58,89**
ionosphere	75,29	74,41	74,12	**75,88**
iris	94,00	**94,67**	**94,67**	**94,67**
penDigits	**73,29**	72,38	73,26	**73,29**
segment	67,71	67,53	**67,88**	**67,88**
vehicle	46,46	**46,95**	46,46	46,59
wine	82,50	**83,13**	**83,13**	82,50
zoo	53,75	53,75	53,75	53,75
Average	66,86	67,37	67,22	**67,69**

Table 6.15: Accuracy (%) Error Rate, Randomization and Naive XOR Cost Function

	PaNDa$_{\eta}$	**PaNDa**$_{conf}$	**PaNDa**$_{supp}$	**PaNDa**$_{laplace}$
abalone	48,75	**49,54**	49,42	49,45
ecoli	79,35	78,39	78,71	**79,68**
glass	60,56	62,22	**62,78**	**62,78**
ionosphere	75,00	73,82	73,53	**75,59**
iris	93,33	**94,00**	**94,00**	**94,00**
penDigits	**75,42**	73,06	75,41	**75,42**
segment	**71,47**	70,17	71,34	**71,47**
vehicle	50,73	**51,71**	50,24	50,61
wine	**83,75**	81,25	82,50	83,13
zoo	73,75	73,75	77,50	**78,75**
Average	71,21	70,79	71,54	**72,09**

Table 6.16: Accuracy (%) Error Rate, Randomization and Typed XOR Cost Function

6.2.5 Execution Time

Below we presented the results collected relating to execution time. Table 6.17, 6.18 and 6.19 shows respectively the execution time for JCBA, CPAR, Weighted Classifier, SVM, C4.5, our implementations with default parameter on PaNDa and our implementation with Typed XOR cost function on PaNDa, error rate value set to 0,3 and randomization value set to 10. The execution time was measured starting from the training set newly generated, until completion of the prediction operation. As you can see from the tables, the execution time of our implementations is much higher than other classifiers. The execution time so high is given by the time taken to perform the extraction of the pattern top-k, with which, however, is possible to obtain high values of accuracy. By changing parameters on Panda and using as cost function the Typed XOR, set value of error rate and randomization we get the execution times slightly lower compared to the test performed using the default parameters, thus obtaining also the values of higher accuracy in all of our implementation PaNDa$_{\eta}$, PaNDa$_{conf}$, PaNDa$_{supp}$, and PaNDa$_{laplace}$.

	JCBA	**CPAR**	**Weighted**	**SVM**	**C4.5**
abalone	0.129	0.000076	0.07	1.225	0.069
ecoli	0.011	0.000076	0.01	0.025	0.001
glass	0.01	0.000083	0.01	0.023	0.001
ionosphere	0.137	0.000094	0.13	0.035	0.017
iris	0.002	0.000084	0.002	0.011	0.001
penDigits	1.463	0.000094	1.4	2.002	0.172
segment	0.486	0.000077	0.4	0.218	0.03
vehicle	0.05	0.000090	0.04	0.116	0.019
wine	0.0089	0.000070	0.003	0.016	0.001
zoo	0.0095	0.000091	0.001	0.054	0.001
Average	0.23064	0.000083	0.20660	0.3725	0.0213

Table 6.17: Execution Time (sec)

	PaNDa$_{\eta}$	**PaNDa**$_{conf}$	**PaNDa**$_{supp}$	**PaNDa**$_{laplace}$
abalone	0,872	0,880	0,873	0,873
ecoli	0,051	0,051	0,051	0,051
glass	0,073	0,072	0,073	0,073
ionosphere	0,091	0,090	0,090	0,091
iris	0,027	0,028	0,027	0,028
penDigits	7,417	7,410	7,431	7,474
segment	1,181	1,184	1,186	1,168
vehicle	0,260	0,263	0,262	0,263
wine	0,060	0,060	0,060	0,060
zoo	0,025	0,025	0,025	0,025
Average	1,006	1,006	1,008	1,011

Table 6.18: Execution Time (sec) Default Parameter

	PaNDa$_{\eta}$	**PaNDa**$_{conf}$	**PaNDa**$_{supp}$	**PaNDa**$_{laplace}$
abalone	0,777	0,785	0,773	0,780
ecoli	0,067	0,064	0,066	0,064
glass	0,037	0,037	0,037	0,037
ionosphere	0,078	0,078	0,079	0,078
iris	0,028	0,027	0,027	0,027
penDigits	6,122	6,119	6,118	6,116
segment	0,976	0,987	0,982	0,979
vehicle	0,194	0,196	0,193	0,195
wine	0,032	0,032	0,032	0,031
zoo	0,016	0,016	0,016	0,016
Average	0,833	0,834	0,832	0,832

Table 6.19: Execution Time (sec) Error Rate, Randomization and Typed XOR Cost Function

Chapter 7

Conclusion

We developed a different approach to classify with using a classifier association rule. When using associative classification to produce an accurate model, we require a large number of rules be generated, with the consequence of having a large number of unclassified instances. To solving this problem, we mine itemsets for each class label separately, thereby using different minimum supports for the classes. This ensure the presence of a good number of strong rules for each class label.

To reduce the memory usage, and execution time of extraction of class association rule we used a greedy algorithmic framework named PaNDa+ to generate rules directly from training data. The greedy strategy, adopted in PaNDa, consists in extracting an ordered sequence of patterns, by progressively increasing the degree of coverage of the dataset, achieving an improved accuracy of each mined pattern.

We developed four classification systems to handle the prediction of class attributes: PaNDa$_{\eta}$, PaNDa$_{conf}$, PaNDa$_{supp}$ and PaNDa$_{laplace}$ and we measure performance in terms of accuracy and execution time. We did further testing to get the best performance of accuracy by changing the parameters of noise tolerance, type of cost function and random iteration, of class association rule extraction algorithm PaNDa. To have a parameter on the goodness of our achievements, we compared our implementations with other classification algorithms such as JCBA, CPAR, Weighted Classifier, SVM, C4.5.

The performance of accuracy that we got from our tests have shown that, for our set of datasets used, the better implementation of the four that we realized, was PaNDa$_{laplace}$, reaching an accuracy average of 72.09%. Com-

paring our implementation with other classification algorithms examined, we see that the values of average accuracy of PaNDa$_{laplace}$ far exceed those of JCBA (43.44%), CPAR (42.27%) and Weighted Classifier (29.36%), while in the case of C4.5 (77.24%) and SVM (78.56%), the values of average accuracy are greater, and PaNDa$_{laplace}$ approaches to their performance.

Further work should be done to reduce the execution time. We think this problem can be solved by optimizing both the rules extraction algorithm PaNDa, that the step of prediction for the classification. Further work should also be done to test the performance on large datasets, or on real-world datasets to see that behavior would in these cases our achievements, even in comparison to other existing classifiers.

Bibliography

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, June 1993.

[2] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[3] Salvatore Orlando Claudio Lucchese and Raffaele Perego. A unifying framework for mining approximate top-k binary patterns, 2014.

[4] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. A unifying framework for greedy mining approximate top-k binary patterns and their evaluation.

[5] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proceedings of the 4th international conference on Knowledge Discovery and Data mining (KDD'98)*, pages 80–86. AAAI Press, August 1998.

[6] Peter Clark and Robin Boswell. Rule induction with cn2: Some recent improvements. In *Proceedings of the European Working Session on Machine Learning*, EWSL '91, pages 151–163, London, UK, UK, 1991. Springer-Verlag.

[7] Xiaoxin Yin and Jiawei Han. Cpar: Classification based on predictive association rules, 2003.

[8] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011.

[9] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[10] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[11] M. Lichman. UCI machine learning repository, 2013.

[12] Frans Coenen. KDD, 2012.