FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Ensemble Methods in Ordinal Data Classification

João David Pereira da Costa

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Jaime S. Cardoso

Second Supervisor: Ricardo Sousa

July 23, 2014

Ensemble Methods in Ordinal Data Classification

João David Pereira da Costa

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Eugénio Oliveira (Full Professor at Universidade do Porto)

External Examiner: Paulo Cortez (Associate Professor at Universidade do Minho)

Supervisor: Jaime S. Cardoso (Assistant Professor at Universidade do Porto)

July 23, 2014

Abstract

Ordinal classification problems can be found in various areas, such as product recommendation systems, intelligent health systems and image recognition. These problems have the goal of learning how to classify certain instances (*e.g.* a movie) in an ordinal scale (*e.g.* good, average, bad).

The performance of supervised learned problems (such as ordinal classification) can be improved by using ensemble methods, where various models are combined to perform better decisions. While there are various ensemble methods for nominal classification, ranking and regression, ordinal classification has not received the same level of attention.

The goal of this dissertation is, therefore, to introduce novel ensemble methods for the classification of ordinal data. To do this, first a new ordinal classification algorithm based on decision trees and the data replication method is presented, whose results show a competitive performance when compared to other ordinal and non-ordinal classifiers. Then, the main ideas of this method are exploited to try and improve ensembles whose models share similarities with decision trees (*i.e.* ADABOOST with Decision Stumps and Random Forests).

Resumo

Problemas de classificação ordinal podem ser encontrados nas mais diversas áreas, tais como sistemas de recomendação de produtos, sistemas inteligentes de saúde e reconhecimento de imagem. Estes problemas têm como objectivo aprender a classificar uma determinada instância (*e.g.* um filme) numa escala ordinal (*e.g.* bom, médio, mau).

Uma forma de melhorar o desempenho de problemas de aprendizagem supervisionada (como é o caso da classificação ordinal) é usando métodos de ensemble, onde vários modelos são combinados para tomar melhores decisões. Embora existam diversos métodos de ensemble desenvolvidos para problemas de classificação nominal, ranking e regressão, a classificação ordinal não tem recebido a mesma atenção.

O objectivo desta dissertação é, assim, introduzir novos métodos de ensemble para dados ordinais. Para isso, em primeiro lugar é apresentado um novo algoritmo de classificação baseado em árvores de decisão e no método de replicação dos dados, cujos resultados revelam um desempenho competitivo com outros classificadores ordinais enão ordinais. Depois as ideias principais deste classificador são aproveitadas para melhorar ensembles cujos modelos gerados possuem semelhanças com árvores de decisão (*i.e.* ADABOOST com Decision Stumps e Random Forests).

Acknowledgements

I would like to thank both my supervisor Jaime Cardoso and my co-supervisor Ricardo Sousa for their scientific guidance and constant support. Without their help, this dissertation would not be possible.

I would also like to thank Daniel Moura for all the help with *Weka* and the whole VCMI group at INESC, one of the best groups I could ever hope to work with.

Finally, I would like to dedicate this work to my late friend Eduardo Jesus, one of the most brilliant persons I have ever met.

João David Costa

"The question of whether Machines Can Think
is about as relevant as the question of whether Submarines Can Swim."

Edsger W. Dijkstra

Contents

List of Figures

LIST OF FIGURES

List of Tables

List of Algorithms

Abbreviations

ANN	Artificial Neural Network
CLI	Command-line Interface
DT	Decision Tree
GUI	Graphical User Interface
JVM	Java Virtual Machine
KNN	K-Nearest Neighbors
MAE	Mean Absolute Error
MER	Mean Error Rate
MSE	Mean Squared Error
OCR	Optical character recognition
RF	Random Forest
SVM	Support Vector Machine

Chapter 1

Introduction

This chapter presents the context and motivation, with a quick introduction to ordinal data classification and ensemble methods, and the goals of this dissertation. It also presents the structure of this document.

1.1 Context and Motivation

Machine learning is the area of artificial intelligence responsible for the study of systems that are able to learn from data. This field has seen its usage spread rapidly during the last decade, with applications in various areas where it is usually very attractive to build models automatically (*e.g.* when the amount of data is too much for a human to handle). Some of those applications include web search, spam filters, fraud detection, ad placement, drug design and language processing [Die97, Dom12].

Machine learning is also a very broad area, as there are many learning tasks that can be applied to different types of data. In this work we will focus supervised learning, which is considered one of the most mature and widely used machine learning tasks [Dom12]. Supervised learning is the task of inferring a function from previously labeled training data (*i.e.* learn rules from a set of examples). One could, for example, use a set of images labeled as "Car" and "Not Car" to learn a function that automatically identifies cars on images.

It is possible to imagine many varied applications of supervised learning: One could use previous player performances to predict tennis results, use GPS and weather information to predict the time a bus would take to go from point A to point B or even predict how much someone will enjoy a movie based on their previous reviews.

We can see that there are many types of supervised learning problems. Different problems need different approaches and, therefore, supervised learning problems are usually subdivided in various tasks, usually related to the type of function one wants to learn, for example:

Binary Classification

In binary classification one wants to learn a function that returns a class from a set of two elements, such as: given some information about a person (*e.g.* height, weight, age...) determine its gender.

Nominal Classification

In nominal classification one wants to learn a function that returns a class from a finite set, for example: given some information about an animal (*e.g.* color, number of limbs, size...) determine if it is a cat, a dog or a mouse.

Regression

In regression, one wants to learn a function that returns a real value – this is usually applied to learn a mathematical function such as $\hat{f} : \mathbb{R}^d \to \mathbb{R}$. Note that, unlike classification, in regression our co-domain is infinite.

Ordinal Classification

In ordinal classification (sometimes referred to as ordinal regression), one wants to learn a function that returns a class from a finite set where its elements, while not having a numerical interpretation, have an order (*e.g.* classify a movie as bad, average or good). Ordinal classification, therefore, shares similarities with both nominal classification and regression.

Ranking

In ranking one has an order relation between pairs of items and wants to learn a function that is able to rank new elements. A common example of a ranking problem is ranking web pages (for example by search engines).

This dissertation will focus only on the problem of ordinal data classification, although many of the used approaches will be based on the methods used for other supervised learning tasks. Ordinal classification has various applications:

- Content Recommendation Systems:
  - Predict which products a consumer might like or not, and recommend them.
  - Use music ratings to automatically generate playlists shaped to the user's taste.
  - Use previous movie reviews to find out what a user values in a movie (*i.e.* a certain actor or a certain genre) and recommend the rental of other movies accordingly.
- Personal Skill Tracking:
  - Keep track of personal statistics to classify someone's performance (*e.g.* practicing sports).
- Stock Market Prediction:
  - Stock market prediction can be seen as a ordinal classification problem, where one wants to either sell, keep or buy a stock.

- Image Recognition:
  - Classifying a someone's age in a photo in an ordinal scale (*e.g.* baby, child, teenager, young adult, adult, old person).
  - Classify a picture's quality using an ordinal classifier and, with that information, apply a model (*e.g.* a nominal classifier) that performs better in low-quality or high-quality images accordingly.
- Health Care:
  - Classify the results of a medical operation automatically (this approach is currently being used by INESC[1] to classify the results of breast cancer operations from an aesthetically point of view).
  - Automatically classify someone's exercise routine based on their performance, physical condition and needs.

Various methods have been developed to tackle this problem, using the ordering relation between classes to try to obtain better results (usually by imposing restrictions on the final hypothesis). Some of those methods reduce the problem to multiple binary classification problems and combine their results [FH01] [WB06], others reduce it to a single binary classification problem on a feature space with extended dimensionality [CDC07] and others use variations of nominal classification algorithms, such as SVMs [SL02] [CK05] and decision trees [PB00].

One way to improve the results of supervised learning tasks is by combining various models via ensemble methods, which have been shown to usually have better results than any single classifier [Die00]. Ensembles also solve the problem of scaling supervised learning algorithms to large databases, learn from multiple physically distributed datasets (this is important on some cases where data cannot be stored in a single site due to legal reasons) and are useful for learning concept-drifting data streams [TPV08].

It is interesting to note, however, the lack of ensemble methods developed with ordinal classification tasks in mind (with only few exceptions, such as ADABOOST.OR [LL09] and the work of Sousa and Cardoso with Decision Trees [SC11]). Therefore, it appears to be interesting to study and develop such methods.

1.2 Objectives and Contributions

In this dissertation we will introduce modifications to various popular supervised learning algorithms, in order to improve their performance on ordinal tasks. With this, we intend to present the following contributions:

- Introduce the concept of local parallelism as a weak constraint that can be applied to ordinal classification;

[1] Instituto de Engenharia de Sistemas e Computadores.

- Present a new decision tree algorithm based on the data replication method [CDC07] that exploits local parallelism;
- Explore the ideas used on our decision tree to present a new ADABOOST variant;
- Study simple ways to improve the performance of Random Forests for ordinal classification;
- Provide open source implementations of the new proposed algorithms.

Our algorithms were developed on top of *Weka* [WFT+99], which is an open source machine learning toolkit developed in Java. This allows us our work to be used in various ways, and therefore reach a broader audience, namely:

- It is possible to install our algorithms as packages for *Weka*, that can then be used via *Weka*'s GUI or CLI;
- Since our all our algorithms were developed in Java, they can be easily integrated with various languages that run on the JVM (*e.g.* Java, Scala and Clojure).

1.3 Structure

This dissertation will be presented in the following way: Chapter 2 presents some of the background in supervised learning and formally introduces the problem of ordinal classification, ensemble learning and the notation that will be used on this dissertation. Chapter 3 describes the state of the art on both ordinal data classification and ensemble methods. Chapter 4 describes a novel algorithm for ordinal data classification based on decision trees and data replication. Chapter 5 presents our work on ordinal ensemble methods, namely a new ADABOOST variant and some possible modifications to Random Forests. Finally, Chapter 6 presents the conclusions and future work.

Chapter 2

Background

On this chapter the problem of ordinal classification and ensemble learning will be formally introduced. A quick introduction to supervised learning is given on Section 2.1, ordinal classification will be defined on Section 2.2, followed by a quick overview of evaluation metrics on Section 2.3. Finally, ensemble methods are presented on Section 2.4. The notation introduced in this chapter will be the one used on the rest of the dissertation. A quick notation reference is available on Appendix A.

2.1 Supervised Learning

As stated in the introduction (Chapter 1), supervised learning is the task of inferring a certain function from previously labeled data. There are various tasks in supervised learning – such as binary classification, nominal classification, ordinal classification, regression and ranking – depending on the function that we want to learn.

To achieve this task, we first need a dataset composed of various examples that have been previously labeled. We say that each example is composed of a feature vector with various attributes (*e.g.* the age and height of a person) and a label, which is the attribute we want to learn (*e.g.* if that person is tall or short).

We will also need a learning algorithm that is able to learn our function. These algorithms can either generate our function from simple models or from very complex ones.

2.1.1 Formal Definition

Formally, we assume a feature space $\mathcal{X}$ composed of feature vectors $\mathbf{x}_i = (x_{i,1}, x_{i,2}, ..., x_{i,n})$ and a label space $\mathcal{Y}$ composed of single elements. For reading simplicity, $\mathbf{x}_i = (x_{i,1}, x_{i,2}, ..., x_{i,n})$ will sometimes be represented as $\mathbf{x} = (x_1, x_2, ..., x_n)$ when the index i is implicit. For example, we could be trying to classify a person as tall or short based on their age and height. Here, our feature vectors could be of the form $\mathbf{x_i} = (age_i, height_i)$ with $\mathcal{Y} = \{short, tall\}$.

To be able to learn a certain function, we need to train a classifier with various examples already classified – a dataset. Given a dataset $\mathcal{D} = (D, f)$, where $D \subseteq \mathcal{X}$ is our example set and $f : D \to \mathcal{Y}$ is the class labeling of the elements of set D, we want to learn a function $\hat{f} : \mathcal{X} \to \mathcal{Y}$ such that $\hat{f}$ is able to model our problem correctly. We will also assume that there exists an unknown function $g : \mathcal{X} \to \mathcal{Y}$ that represents the perfect classifier.

Coming back to our previous example, we could have a dataset where $D = \{(10, 1.40), (20, 1.50)\}$ and f is defined as a mapping $((10, 1.40) \mapsto tall, (20, 1.50) \mapsto short)$.

To learn our function $\hat{f}$ we need a learning algorithm. This algorithm will try to find the best function $\hat{f} \in \mathcal{H}$, where $\mathcal{H}$ is our hypothesis space (*i.e.* the space that contains all the functions that our classifier can learn). For example, in many binary classification approaches, we want to define a boundary that separates our two classes (a visual example can be seen on Figure 2.1, where we have an hyperplane separating the white from the black region). To do this, our elements of $\mathcal{Y}$ are mapped to $\{-1, 1\}$ (*e.g.* $white = -1$ and $black = 1$) and we define our hypothesis space as $\mathcal{H} = \{h | h(\mathbf{x}) = sign(\mathbf{w} \cdot \mathbf{x} + b)\}$ [1] , where $\mathbf{w}$ is a vector of weights associated to each attribute of $\mathbf{x}$ and b is a constant factor.

Note that it is possible that $g \notin \mathcal{H}$ (*e.g.* if there was no hyperplane that was able to separate the white points from the black points in Figure 2.1), therefore our algorithm might not be able to find the optimum solution.

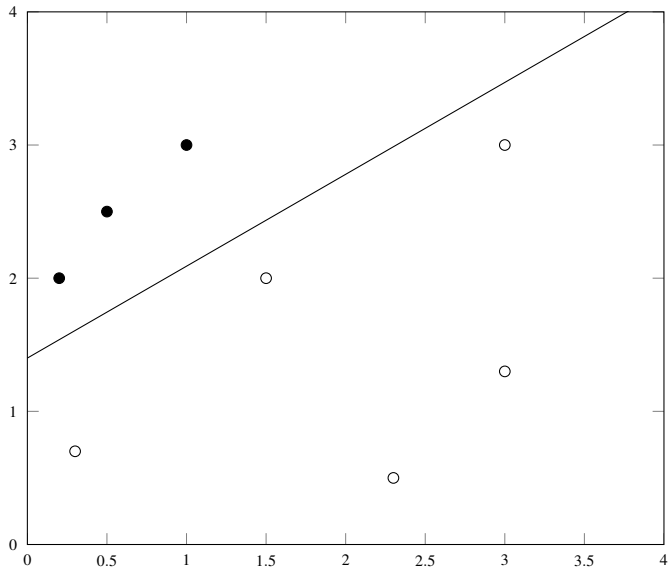


Figure 2.1: Example of a binary classifier

[1] Note that $\mathbf{w} \cdot \mathbf{x} + b = 0$ is the equation for a generic hyperplane. For example, on a 3D space, we would have the plane equation $w_1 \cdot x_1 + w_2 \; cdot x_2 + w_3 \cdot x_3 + b = 0$.

2.1.2 Learning Algorithms

As stated previously, to learn our function $\hat{f}$ we need a learning algorithm. There are various learning algorithms, and therefore presenting them all in detail is out of the scope of this dissertation. With that in mind, the main families of learning algorithms discussed in this dissertation are the following:

Bayesian inference

Bayesian inference is a statistical principle used in various learning algorithms that uses Bayes' rule to estimate the probabilities of certain hypothesis. From a machine learning point of view, one can see the task of learning as computing $P(g(\mathbf{x}) = y|\mathbf{x})$, where we can use our dataset to estimate $P(\mathbf{x}|g(\mathbf{x}) = y)$.

K-Nearest Neighbors

Our model classifies a new point by considering the label of the K closest points. Even though these models are very easy to implement, they can model very complex functions with ease (although this also makes them prone to overfit).

Decision Trees

Our model consists of a set of hierarchical cuts on each attribute, that can be represented in a tree-like structure (Figure 2.2). Since the problem of constructing the optimal tree is known to be NP-complete [HR76], most learning algorithms build the tree by in a top-down fashion, using a greedy heuristic to recursively split the data [MS95].

Note that if we keep building our tree in this fashion, we will probably overfit (on the limit, we could subdivide our tree so that each leaf corresponded to one example). This is usually solved by pruning our tree. This operation is usually divided in two steps: Pre-pruning (pruning executed while we construct our tree, such as stopping subdividing a node if it has less than n examples) and Post-pruning (pruning executed after the tree was constructed, for example if our error function in not expected to grow much by transforming a node into a leaf).

One interesting property of Decision Trees is that they are usually easy to interpret. This has many advantages, such as allowing our model to be validated by humans and, if needed, executed by them. It can also help one to extract interesting properties of our problems.

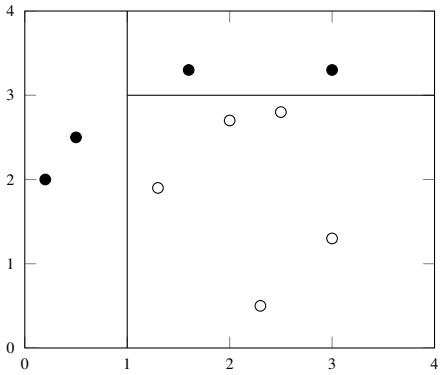
Support Vector Machines

The task of learning a classifier can usually be reduced to finding the optimal boundaries that separates our points into various classes. Support Vector Machines assume that the optimal boundaries are those that separate the various classes with the largest margin (see Figure 2.3), and obtains them by solving a quadratic programming problem.

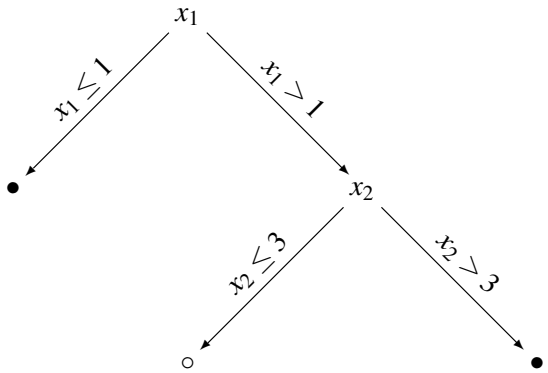
Artificial Neural Networks

An Artificial Neural Network is a directed graph where each node represents an artificial neuron (a variable) and each edge represents a connection between two neurons (a weight).

Usually, these networks are organized in layers, where the first layer is the input layer (every node represents one attribute) and the last layer is the output layer (every node represents one output variable). The value of each node (excluding the input) is usually calculated by applying a transfer function (*e.g.* a sigmoid or a step function) to the weighted average of the nodes from the previous layer.



(a) Cuts on the feature space



(b) Decision tree structure

Figure 2.2: Example of a decision tree classifier

2.1.3 Nonlinear Transformations

The usage of an hyperplane might seem limited at first, since most problems cannot be solved by using linear models. One way to bypass this problem is by by applying a nonlinear transformation $\Phi : \mathcal{X} \rightarrow \mathcal{Z}$ to our feature space. Note that an hyperplane on our nonlinear space $\mathcal{Z}$ will correspond to a nonlinear model on our original space $\mathcal{X}$.

As an example, imagine that our feature vectors are of the form $\mathbf{x} = (x_1, x_2)$ and that our goal is to learn the function $g(\mathbf{x}) = sign(x_2 + x_1 x_2 + x_1^2)$. Using a simple hyperplane would lead us to a very rough approximation of this function (as we cannot model neither x_1^2 nor $x_1 x_2$). To solve this, we can apply the transformation $\mathbf{z} = \Phi(\mathbf{x}) = (x_1, x_2, x_1 x_2, x_1^2, x_2^2)$ to our feature vectors. In this new space, our problem has a linear solution $\hat{f}(\mathbf{z}) = sign(z_2 + z_3 + z_4) = sign(x_2 + x_1 x_2 + x_1^2)$. While

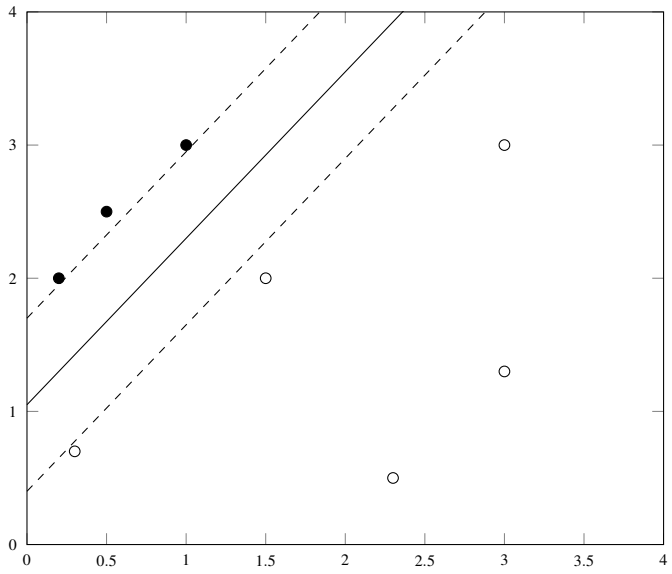
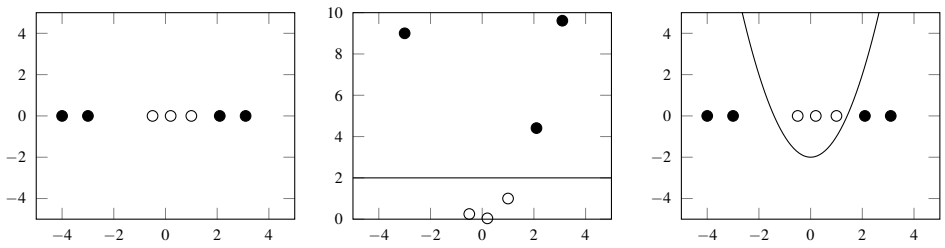


Figure 2.3: Example of a SVM

our model can still be interpreted as an hyperplane in our extended space $\mathcal{Z}$, it represents a curved surface in our original space $\mathcal{X}$. A simple example of this can be seen on Figure 2.4.



(a) Original problem

(b) Problem after the nonlinear transformation $\Phi(\mathbf{x}) = (x, x^2)$

(c) Solution on the original space

Figure 2.4: Example of a nonlinear transformation

Some classifiers such as SVMs are able to efficiently implement this type of transformations, using the so called *kernel trick*.

Note, however, that by doing this, we are increasing the size of $\mathcal{H}$, and we might need more training examples to avoid overfitting [Dom12].

2.2 Ordinal Data Classification

Ordinal data classification is a task of supervised learning where our labels have a certain order between them. In this task, we have that $\mathcal{Y} = \{\mathcal{C}_1, \mathcal{C}_2, ...\mathcal{C}_n\}$ and there is an ordering relation

$\prec$[2] on the members of $\mathcal{Y}$ such that $\mathcal{C}_1 \prec \mathcal{C}_2 \prec ... \prec \mathcal{C}_n$ Also, our classes cannot be interpreted as numbers. For example, we could want to classify a movie as bad, average or good. In this example $\mathcal{Y} = \{bad, average, good\}$ and we have the order $bad \prec average \prec good$.

It is important to be able to distinguish nominal classification, ordinal classification and regression problems, as the concepts of order and numerical interpretation might not be trivial. Here are some examples:

- A person's age - Regression
  - While a person's age can be seen as a finite and discrete problem (*e.g.* an integer between 0 and 120), using a classification approach would need a large number of examples (in classification we need at least one example per class, although usually a lot more are needed to achieve good results). Also, even if all our training examples have integer labels, an age of 12.5 years is still valid, since our classes have a numerical interpretation.
- Search engine results - Ranking
  - The task of ordering search engine results according to their relevance can be seen as a problem where we want to assign and individual rank to each instance. Note that, even though our classes can be represented as numbers (*e.g.* 0 for the worst result, 1 for the second worst result...), they lack a numerical interpretation, and therefore a regression approach is not suitable to solve this problem. Also, our classes do not have a concrete value, as their assigned integer is only used to represent their order. Therefore this is a ranking problem and not an ordinal one.
- A restaurant's 5-star rating - Ordinal classification
  - While it is possible to assign a number to each class and apply regression, this assumes that there is a real numerical value to each one of our labels, which might not be true (*e.g.* is a 4-star restaurant the double of a 2-star restaurant?).
- Number detection using Optical Character Recognition - Nominal classification
  - While there is an obvious mapping from each one of our classes and a number, it is obvious that neither their value nor their ordering is in any way related to the number's shape, and therefore this is a nominal classification problem.

2.2.1 Ordinal Datasets

For our experiments we will use two synthetic datasets and six real datasets. A more detailed description of each dataset is presented on table 2.1

The synthetic datasets are the following:

[2] $A \prec B$ means that our value A precedes B. For example, one could claim that $bad \prec average \prec good$.
[3] Excluding the class attribute.
[4] One of the labels has no examples.

Table 2.1: Datasets

Name	Points	Attributes[3]	Labels
Synthetic 1 (Circle)	100	2	3
Synthetic 2 (Non-monotonic)	5000	2	5
Arie Ben David ERA	1000	3	9
Arie Ben David ESL	488	4	9
Arie Ben David LEV	1000	4	5
Arie Ben David SWD	1000	10	4(5)[4]
Balance-Scale	625	4	3
BCCT	1143	30	4

1. 1000 points calculated via $\lfloor 6 \times ((x-0.5)^2 + (y-0.5)^2) \rfloor$ with $x, y \in [0,1]$ (the radius of a circle centered on $(0.5, 0.5)$ multiplied by 6 so that it goes from 0 to 2).

2. 1000 points from the dataset presented by Ricardo Sousa [SYdCC13] (this dataset is also shown on Figure 3.1).

The Balance-Scale dataset is available on the UCI repository (`https://archive.ics.uci.edu/ml/`) and the Arie Ben David datasets are available on the MLData Repository (`https://mldata.org/`).

2.3 Evaluation Metrics

Once a classifier is trained, it is important to be able to measure its performance. However, performance is a subjective concept, as one might consider some metrics more important than others (*e.g.* if one wants to rank planes based on their performance, they might value speed over capacity or vice-versa, depending on the plane's main task).

There are many evaluation metrics for supervised learning tasks that depend on the nature of the task itself. This evaluation is usually done using a test set – a dataset whose points were not used during the classifier's training – in order to achieve unbiased metrics.

In classification tasks one of the most common metrics is the percentage of misclassified points in our test set (this metric is usually referred to as Misclassification Error Rate). One would usually expect that a classifier that misclassifies 10% of our test set examples has a better performance than one that misclassifies 30%.

However, this might not always be the case. As an example, imagine a system that automatically detects whether someone boarding a plane is carrying a weapon. It is easy to see that, in this case, just comparing the number of misclassifications is not enough, it is also important to consider whether the those are false positives (accusing an innocent passenger) or false negatives (let someone board the plane carrying a gun). In this case its usually preferred to use a cost matrix, where we can assign custom costs to each type of error (see Figure 2.5) and obtain the average cost.

	Innocent	Not Innocent
Predicted as Innocent	0	100
Predicted as Not Innocent	1	0

Figure 2.5: Example of a cost matrix

Note that cost matrices can also be easily extended to problems with more than two classes. Unfortunately, the costs of each outcome are not usually obvious, and therefore cost matrices should only be designed by experts on the problem. For example, on Figure 2.5 we claimed that a false negative is 100 times worse than a false positive, although this value is arguably too small or too large (should 100 innocents be accused and investigated to avoid a disaster?).

The previous metrics, however, cannot be used for regression tasks, as an example cannot be simply considered as misclassified (*e.g.* if a car takes 45 minutes to go from point A to point B, and our model predicted that the trip would take 44 minutes, was the example misclassified?). Nevertheless, we can use the difference between the prediction and the true value to measure our performance. This is usually done by calculating either the Mean Absolute Error ($\frac{1}{N}\sum_{\mathbf{x}}|f(x)-\hat{f}(x)|$) or the Mean Squared Error ($\frac{1}{N}\sum_{\mathbf{x}}(f(x)-\hat{f}(x))^2$).

Ordinal data classification also has various evaluation metrics. Since this is also a classification task, one can apply most metrics used in classification (*e.g.* MER and cost matrices). One can also apply the same metrics used in regression by mapping our ordinal classes to integers ($\mathcal{C}_1 \mapsto 1, \mathcal{C}_2 \mapsto 2, \ldots$), or by generating the appropriate cost matrix (on Figure 2.6 we present an example of a cost matrix that implements the MAE).

	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$
$\mathcal{C}_1$	0	1	2	3
$\mathcal{C}_2$	1	0	1	2
$\mathcal{C}_3$	2	1	0	1
$\mathcal{C}_4$	3	2	1	0

Figure 2.6: Example of a cost matrix that implements the MAE on a 4 class ordinal problem

Be that as it may, those approaches to ordinal evaluation have their limitations:

- Classification evaluation metrics such as MER do not take into account the relation between classes (*e.g.* classifying a 1-star movie as 5-stars is considered as bad as classifying a 4-star movie as 5-stars).

- Regression evaluation metrics such as MAE assume that our classes have a numerical interpretation, which is usually false on ordinal problems.

Many ordinal classification metrics have been proposed to solve these limitations [PdCAC08,

CS11, BVh13], however, since their usage is still not widespread, in this work we will only compare the MER and the MAE[5].

2.4 Ensemble Methods

An ensemble of classifiers is a set of classifiers whose results are combined in some way in order to achieve better results when classifying new instances. Assume a set of classifiers $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, ..., \hat{f}_n\}$. One can combine the results of each of these classifiers (*e.g.* via majority voting) in order to create a new classifier $\hat{f}_{\mathcal{F}}$.

In order for our new classifier $\hat{f}_{\mathcal{F}}$ achieve a better performance than any classifier $\hat{f}_i \in \mathcal{F}$ it is necessary and sufficient that all members of $\mathcal{F}$ are diverse and accurate [Die00].

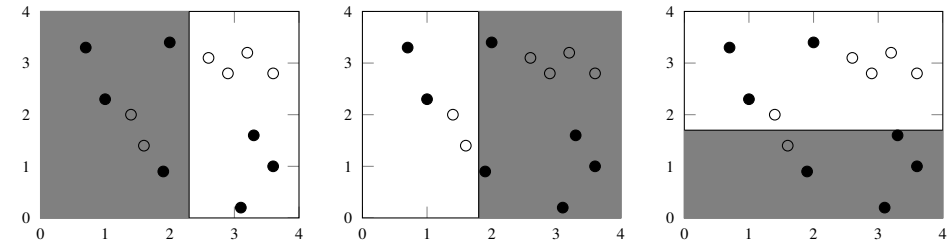
A set of classifiers $\mathcal{F}$ is considered diverse if all classifiers are uncorrelated. As an example, assume a set of three classifiers $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \hat{f}_3\}$. If $\hat{f}_1 \simeq \hat{f}_2 \simeq \hat{f}_3$, then when one of the classifiers is wrong, all the other classifiers will be wrong as well, and there will be no point in combining their results. However, if this is not the case, it is possible that when $\hat{f}_1$ is wrong both $\hat{f}_2$ and $\hat{f}_3$ are right, and therefore it is probable that $\hat{f}_{\mathcal{F}}$ will be right as well (depending on the method used to combine the different classifiers).

Also, a set of classifiers $\mathcal{F}$ is considered accurate if its classifiers perform better than a random classifier. In other words, assuming a classification problem with K classes and a perfect classifier g, the members of $\mathcal{F}$ are accurate if and only if $\forall \hat{f}_i \in \mathcal{F} : P(\hat{f}_i(\mathbf{x}) = g(\mathbf{x})) \geq \frac{1}{K}$. This is obviously necessary, as otherwise our combination of classifiers would only make results worse.

A simple example of an ensemble of 3 binary classifiers can be seen on Figure 2.7.

It is interesting to note that, even if $\mathcal{F} \subseteq \mathcal{H}$, it is possible that $\hat{f}_{\mathcal{F}} \notin \mathcal{H}$ and therefore we can obtain classifiers that would be impossible to obtain using our learning algorithm [Die00].

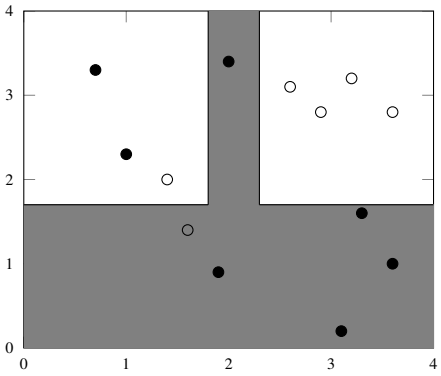
[5] Some machine learning tools (such as *Weka*) come with an implementation of MAE for classification. This implementation is different from the one we presented and is not suited for ordinal tasks, therefore, special care should be taken when attempting to reproduce our results.



(a) Classifier $\hat{f}_1$
(Accuracy $\simeq 61.5\%$)

(b) Classifier $\hat{f}_2$
(Accuracy $\simeq 53.8\%$)

(c) Classifier $\hat{f}_3$
(Accuracy $\simeq 69.2\%$)



(d) Final Result $\hat{f}_{\mathcal{F}}$
(Accuracy $\simeq 84.6\%$)

Figure 2.7: Example of an ensemble of 3 binary classifiers

Chapter 3

State of the Art

On this chapter the state of the art will be presented. On section 3.1 several proposed methods and approaches for ordinal data classification will be presented. Section 3.2 will present some of the most used ensemble methods. Finally, section 3.3 will present some conclusions about the state of the art.

3.1 Ordinal Data Classification

This section will describe various approaches to ordinal data classification. For simplicity, similar approaches have been grouped together in various subsections and therefore they will not be presented in chronological ordering.

Also, the main focus of this section will be to summarize the different types of approaches to ordinal data classification. A more detailed presentation of most of this methods is available in the work of Sousa *et al.* [SYdCC13].

3.1.1 Naïve Approaches

There are two very simple approaches to ordinal data classification:

- Treat the problem as a nominal classification problem.
- Assign a number to each class and treat the problem as a regression problem.

While this approaches might be intuitive and the implementation is trivial, they are incorrect ways to treat the problem [SYdCC13]. Namely:

- If we treat the problem as a nominal one, we will be ignoring the order information, therefore we will need more training examples.
- If we use regression, our classifiers will be more sensible to our arbitrary value assignment than to the ordering.

Therefore, it is expected that the results obtained via naïve approaches to be inferior to more complex ones.

3.1.2 Monotonic Data Approaches

One of the simplest assumptions that can be applied in ordinal data classification is the monotonicity of the data, where it is assumed an ordering relation between the elements of $\mathcal{X}$. Usually it is assumed that our feature vectors are ordered according to $\mathbf{x}_i \preceq \mathbf{x}_j \iff \forall k : x_{i,k} \leq x_{j,k}$ (*e.g.* $(0,0,0) \preceq (0,1,0) \preceq (0,1,1) \preceq (1,1,1)$). Then, we assume that our function $\hat{f}$ is monotone, so that larger values of $\mathbf{x}$ cannot have smaller values of $\hat{f}(\mathbf{x})$. Formally, $\mathbf{x}_i \preceq \mathbf{x}_j \Rightarrow \hat{f}(\mathbf{x}_i) \preceq \hat{f}(\mathbf{x}_j)$.

Some of the proposed algorithms based on this constraints include an ordinal Decision Tree [PB00] and a KNN implementation [DF08]. The ordinal Decision Tree implementation [PB00] inserts new elements in the dataset during the training, in order to enforce a minimum and maximum value for each branch. With this, they are able to enforce the monotonicity constraint. The KNN implementation [DF08] relabels the training data so that the monotonicity constraints are kept and, when classifying a new point, enforces that it is value is kept in a valid interval.

Monotonicity is a very strong assumption and, while in some applications monotonic rules are acceptable by experts [PMS01] and these methods might need fewer examples than other approaches, there are some problems that are clearly ordinal, yet do not respect this constraints [SYdCC13], therefore it would be impossible to model them correctly using this assumptions. An example of an ordinal problem that does not respect the monotonicity constraint can be seen on Figure 3.1.

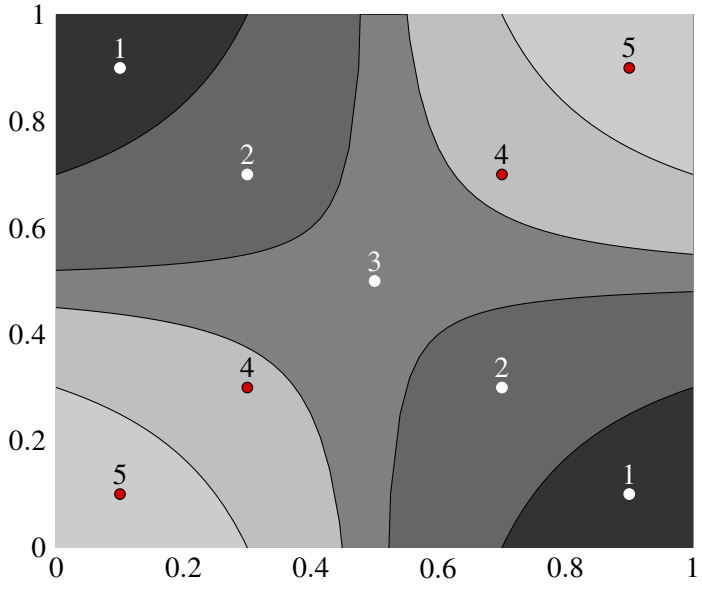
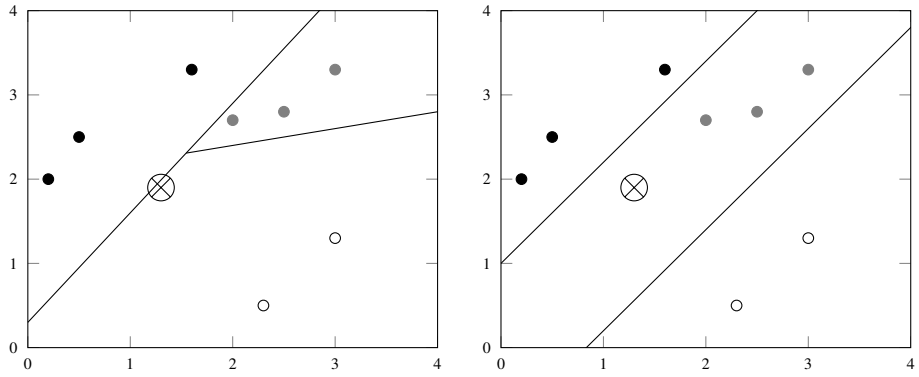


Figure 3.1: Example of a nonmonotonic ordinal problem

3.1.3 Parallel Boundaries using SVMs

Another approach to ordinal data classification is to assume that the boundaries from our classifiers cannot cross each other, as that would lead to some possible point where a small variation of one feature could lead to a huge variation in our label, which is not expected (*e.g.* it is not expected that a small change in a feature makes a product go from "bad" to "good" without going through "average").

One simple way to impose that restriction is to force the hyperplanes used for classification to be parallel to each other. In Figure 3.2 it is possible to see two different classifiers, one without the parallelism constraint and another one with it. In the first one, the point marked with $\otimes$ can make an abrupt jump between two very distinct classes ("black" and "white"), while on the second classifier that is not possible.



(a) Simple multiclass classifier (not consistent) (b) Classifier with parallelism constraints (consistent)

Figure 3.2: Comparison of a multiclass classifier vs. a multiclass classifier with parallelism constraints

One very important aspect of this constraint is that, even though we are using parallelism, this does not mean that we are limited to linear models (*i.e.* hyperplanes). Recal that, by using nonlinear transformations (as presented on section 2.1.3), it is possible to generate our parallel hyperplanes in a nonlinear space, and therefore obtain non-intersecting nonlinear boundaries. Using this intuition, various authors have developed SVMs that, for a K class ordinal problem, generate $K-1$ parallel hyperplanes [SL02] [CK05] [WB06] (usually on the extended space $\mathcal{Z}$), where each boundary separates a group of classes respecting their order (*e.g.* one boundary separates $\{\mathcal{C}_1\}$ from $\{\mathcal{C}_2,\mathcal{C}_3\}$ and another boundary separates $\{\mathcal{C}_1,\mathcal{C}_2\}$ from $\{\mathcal{C}_3\}$).

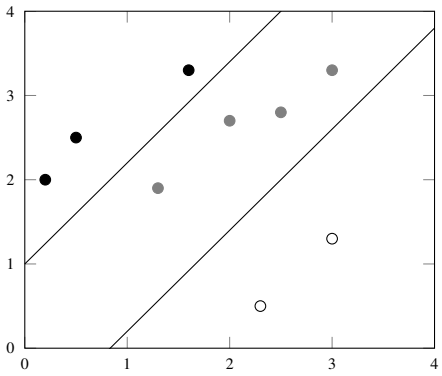
While the parallelism constraint is simple and powerful, the previous approaches are limited to SVMs. Even though SVMs usually present good results, they lack the interpretability of other methods (*e.g.* decision trees).

3.1.4 Subdivision into Binary Problems

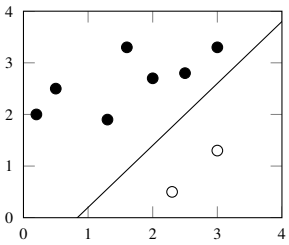
Another approach to ordinal data classification is based on the intuition that a K class ordinal problem can be reduced to $K-1$ binary classification problems, with classes $\mathcal{C}_-$ and $\mathcal{C}_+$, where, for a classifier k, the points are labeled as:

$$f_k(\mathbf{x}) = \begin{cases} \mathcal{C}_- & \text{if } f(\mathbf{x}) \preceq \mathcal{C}_k \\ \mathcal{C}_+ & \text{if } f(\mathbf{x}) \succ \mathcal{C}_k \end{cases}$$

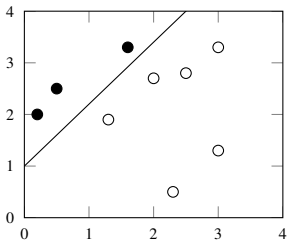
An example of this subdivision can be seen on Figure 3.3.



(a) Original Problem: $\mathcal{C}_1 \prec \mathcal{C}_2 \prec \mathcal{C}_3$



(b) Division 1: $\mathcal{C}_1 \prec \{\mathcal{C}_2, \mathcal{C}_3\}$



(c) Division 2: $\{\mathcal{C}_1, \mathcal{C}_2\} \prec \mathcal{C}_3$

Figure 3.3: Example of a subdivision of a 3 class ordinal problem into 2 binary classification problems

The approach proposed by Frank and Hall [FH01] consists on applying a classifier to each of the binary problems and then combining their results in the following way:

First, note that each of the $K-1$ classifiers will give us the probabilities $P(\hat{f}_i(\mathbf{x}) = \mathcal{C}_+) \simeq P(g(\mathbf{x}) \succ \mathcal{C}_k), k \in \{1, ..., K-1\}$. With that, we can try to calculate $P(g(\mathbf{x}) = \mathcal{C}_k)$:

$$P(g(\mathbf{x}) = \mathcal{C}_k) = \begin{cases} 1 - P(g(\mathbf{x}) \succ \mathcal{C}_1) & \text{if } k = 1 \\ P(g(\mathbf{x}) \succ \mathcal{C}_{i-1}) - P(g(\mathbf{x}) \succ \mathcal{C}_i) & \text{if } k \in [2, K-1] \\ P(g(\mathbf{x}) \succ \mathcal{C}_{K-1}) & \text{if } k = K \end{cases}$$

Then our classification function simply needs to return the label with the largest probability, therefore $\hat{f}(\mathbf{x}) = \arg\max_{\mathcal{C}_k} P(g(\mathbf{x}) = \mathcal{C}_k)$.

While this method allows for an easy application of various binary classifiers to ordinal classification, it is important to note that it will need more memory (due to the data replication) and usually more time than other methods. Compared to the other approaches, this is the one that places the weakest constraint on the classification problem (such as monotonicity or parallelism), which might be a advantageous if none of the constraints hold for our problem (on the other hand, it might present a worst performance if the hard constraints of the other classifier holds).

Even though our conversion from ordinal to binary guarantees that $f_k(\mathbf{x}) = \mathcal{C}_- \Rightarrow f_{k+1}(\mathbf{x}) = \mathcal{C}_-$ and $f_k(\mathbf{x}) = \mathcal{C}_+ \Rightarrow f_{k-1}(\mathbf{x}) = \mathcal{C}_+$, those rules do not always hold for our function $\hat{f}$. In practice, this means that it is possible that the combination rule proposed by Frank and Hall returns a negative probability for some classes. One solution to this problem is setting that negative probabilities to zero, although other smoothing operations could be applied.

Another possible way to combine our binary classifiers is by a simple counting method: $\hat{f}(\mathbf{x}) = \mathcal{C}_i$, with $i = 1 + \sum_k^{K-1} [\![f_k(\mathbf{x}) = \mathcal{C}_+]\!]$[1]. We will use this approach on our algorithms, as it seems to present the best results. Nevertheless, changing them to use the previous solution is trivial.

3.1.5 Data Replication Approaches

One interesting variant of the Frank & Hall method is the data replication method proposed by Cardoso and Pinto da Costa [CDC07]. In this method, the feature space is extended such that each replica has its own dimension. [2]

Assume a feature space of a K class ordinal problem. Also, for simplification, assume $\mathbf{e}_0$ as a vector composed of $(K-2)$ zeros and a vector $\mathbf{e}_q$ as a vector composed by $(K-3)$ zeros and a constant larger than 0 (*e.g.* 1) on the q-th position. Every feature vector $\mathbf{x} = (x_1, x_2, ..., x_n)$ is transformed into $(K-1)$ vectors of the form $\mathbf{z}_q = (\mathbf{x}, \mathbf{e_q})$ labeled as:

$$f(\mathbf{z}_q) = \begin{cases} \mathcal{C}_- & \text{if } f(\mathbf{x}) \leq \mathcal{C}_{q+1} \\ \mathcal{C}_+ & \text{if } f(\mathbf{x}) > \mathcal{C}_{q+1} \end{cases}$$

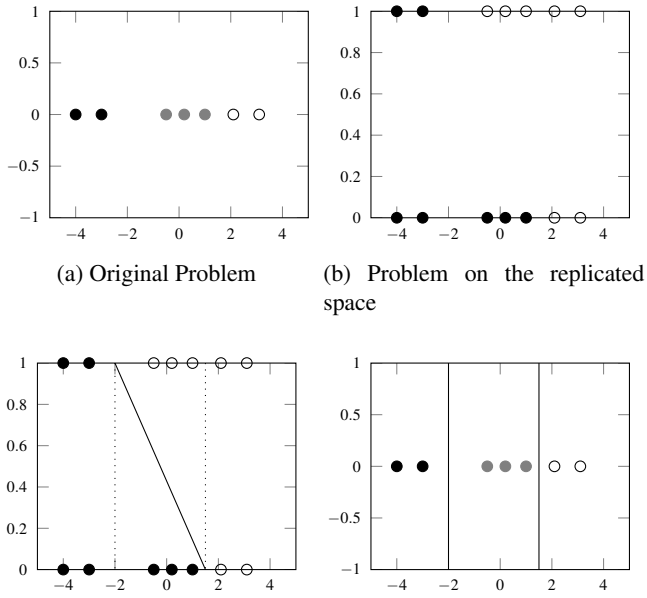
The reason to do this is that, once this new binary classification problem is solved, the intersections of our separating hyperplane and our extra dimensions can be projected on the original space, resulting on a set of parallel hyperplanes dividing our classes (in a similar fashion to the approaches shown on Section 3.1.3).

The best way to understand this approach is to look at a graphical example. Figure 3.4 shows a 1-dimensional dataset with 3 classes (Figure 3.4a). This dataset is then separated in 2 binary replicas, each on it is own dimension (Figure 3.4b). The binary classification problem is then solved (Figure 3.4c) and the hyperplane intersections are projected on the original space (Figure 3.4d),

[1] $[\![\cdot]\!]$ is the indicator function. $[\![\cdot]\!]$ is 1 if the inner condition is true, 0 otherwise.

[2] While the original paper proposes a more general solution that allows the use of smaller replicas, in this dissertation only the simplest use case will be considered, where data is completely replicated.

resulting in a set of two parallel hyperplanes. Note that, since our original space was 1 dimensional, it is impossible to see the parallelism in this example. A 3-class 2-dimensional example can be seen on Figure 3.5.



(a) Original Problem

(b) Problem on the replicated space

(c) Solution to the problem on the replicated space

(d) Result on the original space

Figure 3.4: Toy example of the data replication method

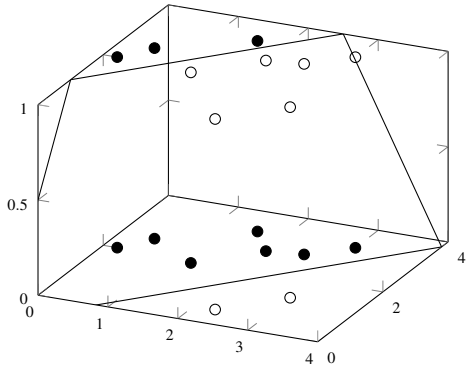


Figure 3.5: Toy example of the data replication method for a 3-class 2-dimensional problem

Like the Frank & Hall approach, this method allows for an easy application of various binary classifiers to ordinal classification. It will also need more memory and take more time than other methods.

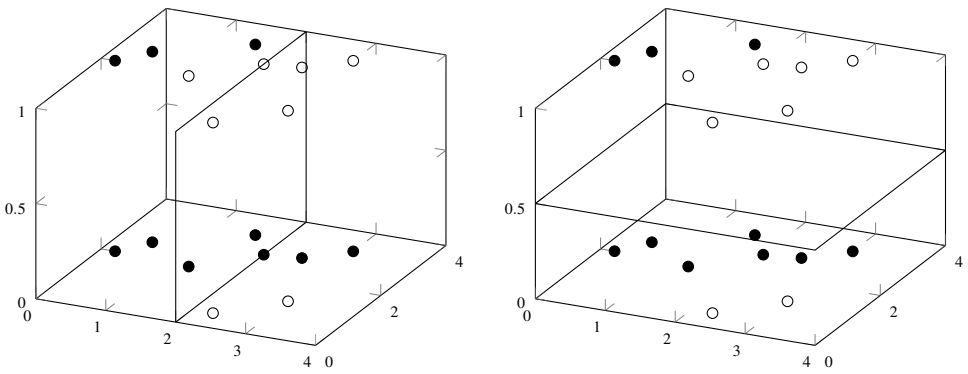
This method allows one to apply the parallelism constraint to various classifiers, and has already been mapped to SVMs, ANNs and LDA [CDC07, CSD12]. Nevertheless, it has some problems when applied to decision trees, since they only use one attribute on each cut.

To see why this is the problematic, assume that we have a 3-class ordinal problem with two attributes. After replicating the data, every point $\mathbf{x} = (x_1, x_2)$ will be transformed in two points of the form $\mathbf{z}_q = (x_1, x_2, e_1)$ (in this case, $\mathbf{z}_0 = (x_1, x_2, 0)$ and $\mathbf{z}_1 = (x_1, x_2, 1)$). If our algorithm learned a linear model (*e.g.* a SVM with a linear kernel), we would obtain a solution corresponding to the plane $w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot e_1 + b = 0$, which, when projected to the original space, would correspond to two parallel lines: $w_1 \cdot x_1 + w_2 \cdot x_2 + b = 0$ and $w_1 \cdot x_1 + w_2 \cdot x_2 + (w_3 + b) = 0$. On a decision tree, since we can only pick one attribute for each cut, this is the same as saying that only one of weights will be equal to 1 and all the others will be equal to 0. Therefore, two types of cuts can happen:

- If the cut happens on one of the original attributes (x_1 or x_2, in this case), we will obtain a cut on the plane $x_1 + b = 0$ (or $x_2 + b = 0$). Note that, since e_1 is not present on our equation, projecting this plane on the original space would result in two lines on the same position (which is not desirable) lines.

- If the cut happens on one of the new attributes (e_1 in our example), then the cut will separate a replica from the others, and the decisions on each replica will become independent, therefore defeating the purpose of the data replication method (On the limit, we would end up with a Frank & Hall approach)

A graphical explanation of this problem is presented on Figure 3.6, where both types of cuts can be seen.



(a) Cut on one of the original attributes

(b) Cut on one of the new attributes

Figure 3.6: Problems with decision trees and the data replication model

3.1.6 Unimodal Distribution Constraint

Another assumption that can be made for ordinal classification is that, for each point $\mathbf{x}$, the probability $P(\hat{f}(\mathbf{x}) = C_k)$ follows an unimodal distribution, therefore it only exists a maximum in our distribution.

This means that, if $\mathcal{C}_k$ is the most probable class, then:

$$\forall i, j : |i| < |j| \Rightarrow P(\hat{f}(\mathbf{x}) = \mathcal{C}_{k+i}) \geq P(\hat{f}(\mathbf{x}) = \mathcal{C}_{k+j})$$

As an example, assume $\mathcal{Y} = \{bad, average, good\}$. If $P(\hat{f}(\mathbf{x}) = good) \geq P(\hat{f}(\mathbf{x}) = average)$, then it is also expected that $P(\hat{f}(\mathbf{x}) = good) \geq P(\hat{f}(\mathbf{x}) = bad)$.

This constraint was proposed by Pinto da Costa *et al.* [PdCAC08], where 3 approaches were proposed:

- A parametric approach, where SVMs and ANNs were used to learn parameters of monotonic distributions (binomial distribution and Poisson distribution).
- A nonparametric approach, where the output of an ANN is forced to output an unimodal distribution.
- A parametric approach, where the parameters of the monotonic distributions are learned via regression.

This approach has shown to perform better than other ordinal classifiers [PdCAC08], however, this only happens when the right distribution is chosen, which is a problem of its own.

3.1.7 Globally Consistent Approaches

Some of the previous approaches applied some constraints to our final model, such as monotonicity, parallelism or unimodality. Cardoso and Sousa proposed a more general concept, the concept of consistency [CS10].

A function is said to be consistent if, for every continuous decision region, it holds that adjacent regions have adjacent classes (see Figure 3.2). It is interesting to note that monotonicity, parallelism and unimodality guarantee this property, even though they are not necessary conditions.

While this appears to be the one of the best approaches, imposing this constraint is not trivial and may lead to computational intensive solutions, therefore it has only been applied to small problems.

3.2 Ensemble Methods

The usage of ensemble methods can be divided in 3 phases [MMSJS12]:

1. Ensemble generation;
2. Ensemble pruning;
3. Ensemble integration.

In the ensemble generation phase, the various classifiers $\hat{f}_i \in \mathcal{F}$ are generated. Note that sometimes this step generates redundant models. The second phase is ensemble pruning, where some of the models generated are removed. Some authors skip this phase, using a direct approach. The final phase is ensemble integration, where the our classifiers $\hat{f}_i \in \mathcal{F}$ are combined to generate our final model $\hat{f}_{\mathcal{F}}$.

3.2.1 Ensemble Generation

3.2.1.1 Bayes Optimal Classifier

This ensemble tries to predict the value of $g(\mathbf{x})$ by computing the probability $P(g(\mathbf{x}) = y|\mathcal{D}, \mathbf{x})$ via:

$$P(g(\mathbf{x}) = y|\mathcal{D}, \mathbf{x}) = \sum_{h \in \mathcal{H}} P(h(\mathbf{x}) = y|\mathbf{x}) P(h|\mathcal{D})$$

Then, we can simply define $\hat{f}(\mathbf{x}) = \arg\max_y P(g(\mathbf{x}) = y|\mathcal{D}, \mathbf{x})$.

This can be interpreted as an ensemble that combines all the hypothesis in $\mathcal{H}$ weighted by the probability $P(h|\mathcal{D})$ – the probability of obtaining a certain hypothesis using our dataset.

While, in theory, this ensemble should be optimal, this only holds for very limited number of problems, where $\mathcal{H}$ is finite, $g \in \mathcal{H}$ and we are able to calculate $P(h|\mathcal{D})$. Unfortunately this 3 assumptions do not hold on most practical applications [Die00]

3.2.1.2 Dataset Subsampling

A simple way to build various different hypothesis is to use a subsample of our dataset for training the various functions in $\mathcal{F}$. Since each classifier is trained with different examples, it is expected that each classifier will also be different. This works particularly well with unstable algorithms, such as decision trees and neural networks, where a small change in the input can lead to major changes in the classifier [Die00].

This can be done in various ways:

- Bagging
- Cross-Validated Committee
- Boosting

The simplest way to do this is via bagging (**B**ootstrap **Agg**regat**ing**). In bagging, our classifiers are trained in various runs, where in each run i we train a classifier $\hat{f}_i$ using a dataset $\mathcal{D}' = (D', f)$, where D' is obtained by drawing N examples randomly with repetition from D (where N is the number of elements in D). The training set $\mathcal{D}'$ is called a bootstrap replicate of $\mathcal{D}$, and contains, on average, 63.2% of the original training set.

A cross-validated committee is generated in a similar way to cross-validation. For example, on a 10-fold cross-validated committee, we divide our dataset $\mathcal{D}$ in 10 disjoint subsets $\mathcal{D}'_1, \mathcal{D}'_2, ..., \mathcal{D}'_{10}$. Then we create 10 overlapping training sets by dropping one of those 10 datasets from our original

subset. Therefore, $\hat{f}_i$ would be generated using the dataset $\mathcal{D} - \mathcal{D}'_i$. An example can be seen on Figure 3.7.

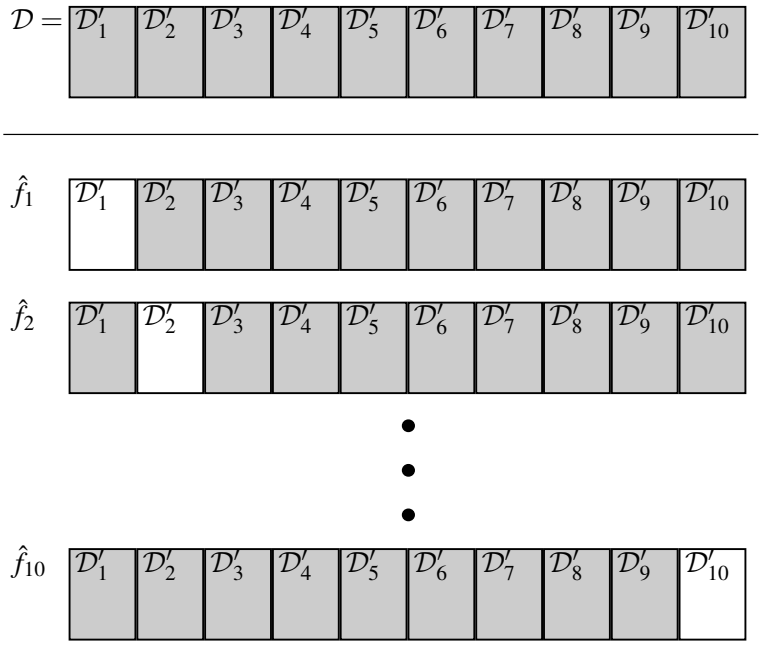


Figure 3.7: Example of a 10-fold cross committee

Our third method is boosting. This method was first presented by Schapire in 1989 [Sch90] and the idea behind it is to train various classifiers in an iterative fashion by taking it into account the errors of previous classifiers. The original algorithm can be seen on Algorithm 3.1 [Fer07].

Data: A dataset $\mathcal{D} = (D, f)$ with N elements
Result: A classifier $\hat{f}_{\mathcal{F}}$
$\mathcal{D}'_1 :=$ Subset of $\mathcal{D}$ with N'_1 elements selected without replication;
$\hat{f}_1 := train(\mathcal{D}'_1)$;
$\mathcal{D}'_2 :=$ Subset of $\mathcal{D}$ with N'_2 examples where $\frac{N'_2}{2}$ samples where misclassified by $\hat{f}_1$;
$\hat{f}_2 := train(\mathcal{D}'_2)$;
$\mathcal{D}'_3 :=$ Subset of $\mathcal{D}$ where $\hat{f}_1(\mathbf{x}) \neq \hat{f}_2(\mathbf{x})$;
$\hat{f}_3 := train(\mathcal{D}'_3)$;
$\hat{f}_{\mathcal{F}}(\mathbf{x}) = sign(\hat{f}_1(\mathbf{x}) + \hat{f}_2(\mathbf{x}) + \hat{f}_3(\mathbf{x}))$; `// Chooses a class according to a majority voting`

Algorithm 3.1: Boosting algorithm for binary classification

Later, Freund and Schapire proposed the AdaBoost (**Ada**ptive **Boost**ing) method [FS95]. This method manipulates our training examples by associating a weight to each one and then uses that weight to define a weighted error function for each classifier [3]. After training each classifier, the algorithm updates the weight of each example, increasing the weight of misclassified instances (intuitively, this makes "harder" instances more important). It also associates a weight to each

[3] If the classifier does not support weighted errors, then the examples are sampled randomly according to their weight.

classifier based on it is performance, using it to combine the classifiers via weighted voting. The details of the method can be seen on Algorithm 3.2.

Data: A dataset $\mathcal{D} = (D, f)$ with N elements
Result: A classifier $\hat{f}_{\mathcal{F}}$
Initialize the example weights as $w_j := \frac{1}{N}$;
$i := 1$;
while $\neg EndingCondition$ **do**
  $\hat{f}_i := \texttt{train}(\mathcal{D})$;
  $error_i = \sum_{j=1}^{N} w_j [\![\hat{f}_i(\mathbf{d}_j) \neq f(\mathbf{d}_j)]\!]$;
  $\alpha_i = 0.5 log(\frac{error_i}{1-error_i})$; // Classifier weight
  forall the j **do**
    $w_j := w_j \alpha_i^{1-[\![\hat{f}_i(\mathbf{d}_j) \neq f(\mathbf{d}_j)]\!]}$; // Updates the example weights
  end
  Normalize the weights so that $\sum_j w_j = 1$;
  $\mathcal{F} := \mathcal{F} \cup \{\hat{f}_i\}$;
  $i := i + 1$
end
$\hat{f}_{\mathcal{F}}(\mathbf{x}) = sign(\sum_i \alpha_i \hat{f}_i(\mathbf{x}))$; // Chooses the best class according to a weighted voting

Algorithm 3.2: Discrete AdaBoost

Note that both boosting and discrete AdaBoost, as stated on algorithms 3.1 and 3.2 only work on binary classification problems (assuming $\mathcal{Y} = \{-1, 1\}$) and with discrete classifiers (*e.g.* it does not work with classifiers that return a probability distribution). There are, however, many variants of the AdaBoost method [Fer07], that can be applied to other supervised learning tasks such as:

Real AdaBoost/Gentle AdaBoost/Modest AdaBoost
Variants that works with real classifiers (*e.g.* Naïve Bayes).

AdaBoost.M1/AdaBoost.M2/AdaBoost.M1W/BoostMA
Variants that works with multiclass problems.

AdaBoost.OC/AdaBoost.ECC
Variants based on ensemble of binary classifiers with output codes and error correcting codes to solve multiclass problems (similar to the method presented on Section 3.2.1.4).

RankBoost
A variant designed for Ranking problems.

3.2.1.3 Manipulating Input Features

Another way to alter our input to build different ensembles is to manipulate the input features.

For example, imagine that our feature vectors are of the form $\mathbf{x} = (age, height, weight, gender)$, we can train a classifier using only $\mathbf{x}' = (age, height)$, another one using only $\mathbf{x}' = (height, weight)$

and so on. We could also train a classifier using linear or nonlinear combinations of this features, for example $\mathbf{x}' = (age, \frac{weight}{height^2}, gender)$. The feature selection can be done either by an expert on the problem domain or via search heuristics, such as genetic algorithms [MMSJS12].

Another way to manipulate the input features is via feature discretization [MMSJS12], where numerical values are replaced by discretized versions. By repeating this process with different discretization parameters, it is possible to generate different datasets and promote diversity.

Feature selection methods may present better results than bagging and boosting approaches, however, our input features must be highly redundant [Die00], as feature selection is not a trivial task, and removing some features might cripple our classifiers' performance so much that our voted ensemble will not perform as well as expected.

3.2.1.4 Manipulating Output Targets

Instead of manipulating our examples features, we can change their labels to add diversity to our classifiers.

One way to do this is via error-correcting output codes [DB95] where, for each of our classifiers $\hat{f}_i$, we divide our set $\mathcal{Y}$ into two subsets, converting our problem into a binary classification problem.

Here, we assume that each one of our labels is a L-bit keyword. For example, if we have that $\mathcal{Y} = \{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4, \mathcal{C}_5, \mathcal{C}_6, \mathcal{C}_7, \mathcal{C}_8\}$, we can map each class to a 3 bit binary number ($\mathcal{C}_1 \mapsto 000, \mathcal{C}_2 \mapsto 001, ..., \mathcal{C}_8 \mapsto 111$). We then need to create L binary classifiers, one for each bit. Going back to our example, we need three classifiers $\hat{f}_1$, $\hat{f}_2$, $\hat{f}_3$. $\hat{f}_1$ will give us the value of the first bit, $\hat{f}_2$ the value of the second bit and $\hat{f}_3$ the value of the third bit. Using these classifiers, we can obtain our 3-bit keyword, and then find out the corresponding label.

Note that we can actually use more than $log_2(|\mathcal{Y}|)$ classifiers. By doing this, we will have bigger keywords and it is not guaranteed that our classifiers will give us a valid keyword. To fix this, we simply pick the label whose keyword is closer (in Hamming distance) to our classifiers' results.

It is interesting to note that the ordinal classifier proposed by Frank & Hall [FH01] works in a similar way to this method, although the keyword mapping and the final classification are obtained in a different way.

3.2.1.5 Injecting Randomness

The last general purpose method for generating ensembles is to inject randomness into our learning algorithm.

For example, some classifiers such as neural networks are initialized with random weights. By repeating the training with a different random seed, we are expected to obtain a different classifier.

Another way to inject randomness is through input smearing, where Gaussian noise is added to each input value [MMSJS12].

3.2.1.6 Algorithm Specific Methods

Finally, we can have algorithm specific methods.

One family of such methods are Random Forests [Bre01], which only apply to decision trees. The original definition of Random Forests is the following:

> "A random forest is a classifier consisting of a collection of tree structured classifiers $\{h(\mathbf{x}, \Theta_k), k = 1, ...\}$ where the Θ_k are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input $\mathbf{x}$." [Bre01]

Note that this is a broad definition that encompasses many ensembles of decision trees that use randomness in some way and are whose results are combined with a simple metric. The most common types of Random Forests are:

Tree Bagging

The simplest form of Random Forests is the application of bagging to decision trees. Note that, while this is a Random Forest by definition, it is usually preferable to avoid that terminology to avoid confusion.

Random Forests using Random Input Selection

This is the most common form of Random Forests and are the ones we will study in this work. Here, ate each node, our algorithm picks a random subset of attributes[4]. and performs the split on the most promising of those attributes.

Random Forests using Linear Combination of Inputs

This is a more complex variant of Random Forests, where instead of simply using a random combination of features, one uses a random linear combination of random features.

As previously stated, in this work we will use the term Random Forests to refer to "Random Forests using Random Input Selection".

Other algorithms, such as SVMs, ANNs and KNN, have various parameters that can be altered to generate different models, even if they are suboptimal [MMSJS12].

While these methods can achieve good results, they obviously only work with a specific learning algorithm.

3.2.2 Ensemble Pruning

The goal of ensemble pruning is, after obtaining our set $\mathcal{F}$, obtain a subset $\mathcal{F}' \subseteq \mathcal{F}$ that reduces the size of our ensemble without a significant performance loss.

Tsoumakas, Partalas and Vlahavas [TPV08] propose the following taxonomy for ensemble pruning methods:

- Search based methods

[4] usually $\lfloor log_2(M) + 1 \rfloor$, where M is the number of attributes

- Clustering based methods
- Ranking based methods
- Other methods

3.2.2.1 Search based Methods

Search based methods are the most direct approach to ensemble pruning. This methods consist of performing a heuristic search in the space of possible subsets, according to some evaluation metric (usually the performance or diversity of the ensemble [TPV08]).

This search can either be a greedy search or a stochastic search (*e.g.* genetic algorithms). While the former is among the most popular categories of ensemble pruning, stochastic search can achieve better results by avoiding getting stuck in local optima [TPV08].

Greedy search methods can occur in two directions: Forward selection (from $\varnothing$ to $\mathcal{F}$) and Backward elimination (from $\mathcal{F}$ to $\varnothing$). An example a search space and search directions can be seen on Figure 3.8.

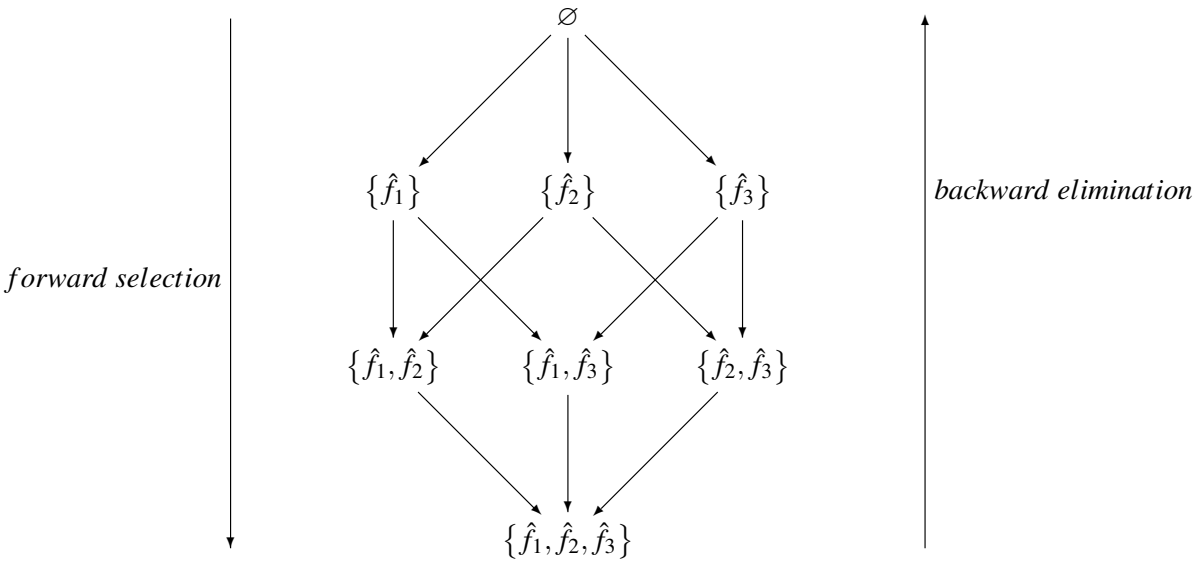


Figure 3.8: Example of an ensemble search space

3.2.2.2 Clustering based Methods

Clustering based methods can be divided in two phases:

1. Group the models using clustering techniques
2. Prune each cluster

To employ these methods, a distance measure between two classifiers must be defined, for example, the probability that two classifiers do not make coincident errors (estimated from a validation set) [TPV08].

3.2.2.3 Ranking based Methods

Ranking based methods order the classifiers in $\mathcal{F}$ according to some evaluation metric (*e.g.* their performance) and select them in this order.

Using an approach called Orientation Ordering it is possible to order the classifiers in a way that gives preference to classifiers that correctly classify examples misclassified by the ensemble [TPV08].

3.2.2.4 Other Methods

Other pruning methods include the usage of statistical procedures to determine if the differences in predictive performance among the classifiers are significant, only retaining the significant classifiers, or formulate the pruning problem as a quadratic integer programming problem that looks for a fixed size subset of classifiers that minimizes the misclassification and maximizes the divergence. [TPV08]

3.2.3 Ensemble Integration

Nominal classifiers are usually combined in one of 3 ways:

- Majority voting
- Weighted voting
- Borda count

Of those methods, majority voting is the simplest one. It is important to note that, usually, it is possible to predict which classifiers have a better performance, and therefore it might be interesting to give more importance to the opinion of better classifiers, and therefore use a weighted voting. On the other hand, some classifiers (*e.g.* Naïve Bayes) are able to say what are the most probable decisions, assigning a probability to each one. With this, it is possible to use a borda count, to give more importance to more certain decisions.

Regressions can be combined in many ways, as can be expected, for example:

- Mean
- Weighted Mean
- Percentile (Minimum, Median, Maximum...)
- Weighted Percentile

The interest of using weighted combinations is the same in regression as in nominal classification.

3.2.4 Ensemble Methods for Ordinal Data Classification

3.2.4.1 AdaBoost.OR

One of the few ensemble methods for ordinal classification is AdaBoost.OR [LL09]. This method uses a primal-dual approach to solve an ordinal problem both in the binary space and the ordinal space.

To do this, they propose a relation between an ordinal classifier $\hat{f}^{ord}(\mathbf{x})$ and either $K-1$ closely related binary classifiers $\hat{f}_k(\mathbf{x})$ or one binary classifier $\hat{f}^{bin}(\mathbf{x},k)$, where each point on the ordinal space has a cost vector $\mathbf{c}$ associated to it and each binary point has an associated weight of $w=(K-1)|\mathbf{c}[k+1]-\mathbf{c}[k]|$. Using this relation, we are able to train our algorithms on the binary space (as each point has an associated weight) and then use the cost vector on the ordinal space to calculate the error and update the costs of each point. Finally, our ordinal classifiers are combined using a weighted median, as it is shown to be equivalent as performing the boosting on the binary space and convert the results back to the ordinal space.

A quick presentation of the method can be seen on Algorithm 3.3.

Data: A dataset $\mathcal{D}=(D,f)$ with N elements
Result: A classifier $\hat{f}_{\mathcal{F}}$
Initialize the cost vectors $\mathbf{c}_j^1$ (*e.g.* as MAE cost vectors)
$i:=1$;
while $\neg EndingCondition$ **do**
  $\hat{f}_i^{ord} :=$ `train(`$\mathcal{D}$`)` ; `// Train the ordinal classifier (train and combine the binary classifiers)`
  $error_i = \left(\sum_{j=1}^{N} \mathbf{c}_j^i[\hat{f}_i(\mathbf{d}_j)]\right) / \left(\sum_{j=1}^{N} \mathbf{c}_j^i[1]+\mathbf{c}_j^i[K]\right)$;
  $\alpha_i = 0.5log(\frac{1-error_i}{error_i})$; `// Classifier weight`
  $\delta_i = \frac{1-error_i}{error_i} - 1$; **forall the** j **do**
    if $\hat{f}_i > f_i$ **then**
      $$\mathbf{c}_j^{i+1}[k] = \mathbf{c}_j^i[k] + \begin{cases} 0 & \text{if } k \leq f(\mathbf{x}_j) \\ \delta_i \cdot \mathbf{c}_j^i[\hat{f}_i(\mathbf{x}_j))] & \text{if } k > \hat{f}_i(\mathbf{x}_j) \\ \delta_i \cdot \mathbf{c}_j^i[k] & \text{otherwise} \end{cases}$$
    else
      (same as above, but switch > to < and vice versa)
    end
  end
  $\mathcal{F} := \mathcal{F} \cup \{\hat{f}_i^{ord}\}$;
  $i := i+1$
end
$\hat{f}_{\mathcal{F}}(\mathbf{x}) =$ `weightedMedian(`$\mathcal{F},\alpha$`)`

Algorithm 3.3: AdaBoost.OR

One of the main limitations of this model is that it only works with ordinal classifiers, and those must work on the binary space (such as the Frank and Hall method or the Data Replication Method).

3.2.4.2 Ensembles of Globally Consistent Classifiers

As previously stated, one interesting restriction that can be applied to ordinal models is to force our model to be globally consistent [CS10].

One interesting property about consistent classifiers is that, if our ensemble of classifiers $\mathcal{F}$ is composed only of consistent classifiers, then if $\hat{f}_{\mathcal{F}} = \texttt{median}(\mathcal{F})$, it is guaranteed that $\hat{f}_{\mathcal{F}}$ will also be consistent. Sousa and Cardoso [SC11] exploited this property to build ensembles of consistent decision trees that guarantee a consistent final result.

3.2.4.3 Possibilities to be Explored

Those are the few examples of ensemble methods for ordinal data, although it is possible to see that there are more ideas that can be explored.

For example, during the ensemble generation phase, it is interesting to explore what would happen if restrictions such as parallelism were imposed on the final result. It is also noteworthy that on the ensemble integration phase one can apply both regression and nominal approaches, for example:

- Majority voting
- Weighted voting
- Borda count
- Percentile (Minimum, Median, Maximum...)
- Weighted Percentile

Note that, since ordinal data does not have a numerical interpretation, the mean of a set of values cannot be calculated, although it is still possible to calculate any percentile (such as the median).

Actually, both ensembles previously presented used the median as a combination rule due to its interesting properties: Besides the guarantee of consistency and the duality between the binary space and the ordinal space, if there is an absolute majority, then the median will behave majority voting (the same applies to the weighted voting and the weighted mean).

Another interesting idea to be explored is how can boosting on the binary space be used to improve ordinal classification. While some of this ideas are explored in ADABOOST.OR, it would be interesting to explore other ideas (such as enforcing parallelism in our weak classifiers).

3.3 Conclusions

In this chapter we presented the state of the art in both ordinal classification and ensemble methods, including ensembles designed for ordinal tasks.

We shown that there are various algorithms for ordinal classification and how most of them use the order relationship to assume properties of our data (and exploit those properties), such as monotonicity, parallelism or consistency.

We also shown the main types of ensemble methods and how they are currently being used for ordinal classification. It was possible to see that there is a lack of ensembles designed specifically for ordinal data classification, and we may be able to build on the previously presented ideas to build new types of ensembles.

Chapter 4

Ordinal Decision Tree using the Data Replication Method

While the focus of this dissertation is on ensemble methods for ordinal classification, it is interesting to develop a new learning algorithm based on decision trees and the data replication method for various reasons:

- Decision trees can be interpreted as boosting algorithms [KM96], and therefore some of the insights from a new type of decision tree might be applicable to new ADABOOST variants.
- Some ensemble methods are exclusive to decision trees (*e.g.* Random Forests).
- The original data replication method does not support decision trees, and therefore this algorithm will make the data replication method a more powerful framework, making the development of ensembles of classifiers on the replicated space more versatile.

4.1 Limitations of the Data Replication Method

First of all, it is important to understand why the data replication method does not work with decision trees.

As stated on the state of the art (Section 3.1.5), the data replication method works on an extended space and then projects the hyperplane intersections on the original space, to obtain parallel hyperplanes (a graphical example can be seen on Figure 3.5). However, when we use a decision tree, our cut will only take one attribute in consideration, and therefore two things can happen:

- If our cut is applied on one of the original attributes, then it will be parallel to all of the replicated axis. Therefore, when our cut is projected back into the original space, all our parallel lines will be equal.

- If our cut is applied on one of the new attributes, then we will get a replica separated from the others – on the limit, we would be reduced to the Frank and Hall approach.

Those problems can be visualized on Figure 3.6. Again, a more detailed description of this problem can be found on Section 3.1.5.

4.2 Proposed Solution

To understand our solution, it is interesting to first imagine what would happen if our replicas were decoupled (such as in the Frank and Hall method). In this case, we would build $K-1$ binary trees, where each branch node could be defined by the tuple $\langle att, \theta, left, right \rangle$, where att is the attribute were the split will take place, θ the position of the split and $left$ and $right$ the left and right sons. On the other hand, a leaf node would simply be composed of a binary label y[1].

Our goal is to use the data replication method to build correlated trees that are able to exploit parallelism. From this, we can extract some intuitions:

1. We want our trees to be correlated, so the attributes of their nodes should share some values.

2. Since our cuts can only have one attribute att, two cuts are parallel if and only if they are made in the same attribute.

Based on this ideas, instead of growing $K-1$ trees, we will build a single tree where each branch node is composed only of parallel cuts. That is to say, our branch nodes would have one attribute, but different cut position (depending on the replica). Formally, our branch nodes now become $\langle att, [\theta_1, ...\theta_{K-1}], left, right \rangle$. Note that now our leaf nodes must also be composed of $K-1$ binary labels $[y_1, ..., y_{K-1}]$.

With this, we assure that some of the parallelism constraints are kept, and we are able to apply our decision tree on the replicated space. We will refer to this property as local parallelism, since we only enforce parallelism on each node. An example of our tree can be seen on Figure 4.1.

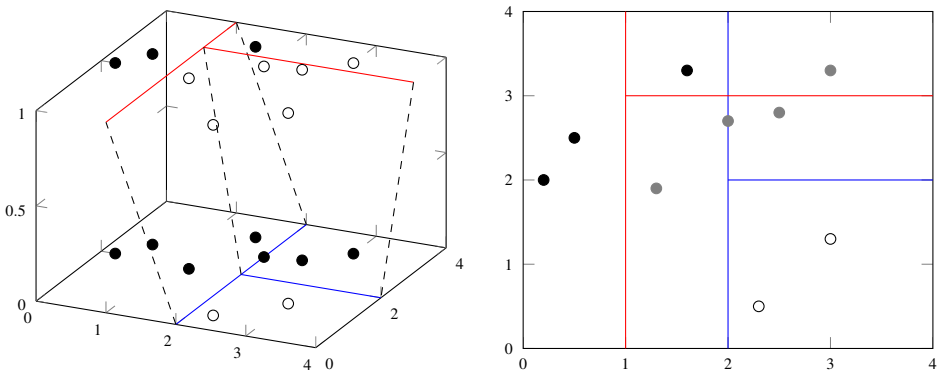
Note our ordinal decision tree diagram in Figure 4.1c. Here it is possible to see that our ordinal decision tree can be interpreted in two different ways:

- As a single decision tree, where each node has one attribute and $K-1$ split points and each leaf has $K-1$ binary labels.

- As $K-1$ decision trees that share the same structure.

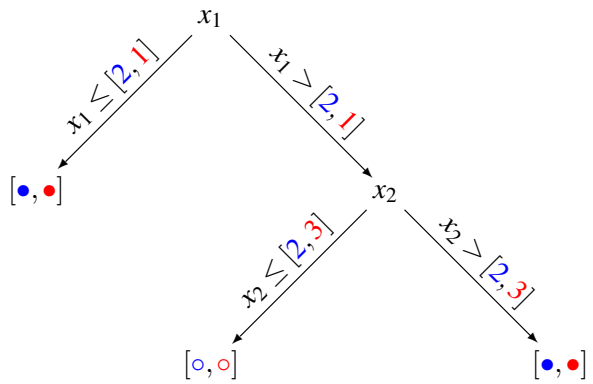
As a side note, decision tree algorithms (such as C4.5) also allow cuts on nominal attributes. Since those attributes have no explicit order, such splits should be done as on a normal decision tree (*i.e.* $\theta_1 = \theta_2 = ... = \theta_{K-1}$).

[1] For now, we will assume this for simplicity, although on most DT implementations leaf nodes contain a probability distribution.



(a) Cut on the replicated space

(b) Cut on the original space



(c) Ordinal Decision Tree

Figure 4.1: Example of our decision tree on the data replicated space

4.3 Growing the Tree

To grow our tree, it is usually easier to see our model as a single tree. First, we need to find the best attribute $\hat{att}$ and the best sequence of splits $[\hat{\theta}_1, ..., \hat{\theta}_{K-1}]$. To do this, for each attribute att we find the values for $[\hat{\theta}_1^{att}, ..., \hat{\theta}_{K-1}^{att}]$ by simply picking the θ_k that maximize the gain ratio of each replica k. Then, to pick the best attribute to split on, we combine our information gain/gain ratio values using a combination heuristic (*e.g.* the average), and pick the attribute that maximizes that value.

One problem of this model is that, sometimes, one replica might need less splits than others (or one replica might need more splits than the rest). Also, it is possible that it is useful for some replicas to make a split on a certain attribute and useless to others. Therefore, we need some way to allow a replica to not make a cut on a certain attribute if there's no reason to. We solve this problem by allowing each replica to make a split at ∞ if the information gain is too small for a cut to be made. Note that a cut at ∞ is the same as no cut at all, since all points will be classified as $< \infty$.

This brings us to another problem: it is not clear what combination heuristic should be used to combine the various performance metrics (*e.g.* information gain and gain ratio). One of the main

problems being how to deal with cuts at infinity. For example, imagine that, as a combination criteria, we pick the average information gain:

- If we use the infinity values in our metric, good cuts might be penalized.
- If we ignore the infinity values, them our metric will usually prefer splits that only occur in a few replicas, and therefore not take advantage of the correlation.

For post-pruning we are using the same methods used in *Weka*'s J48 implementation with some slight modifications. Since each node now has one distribution of points per replica, to get the number of misclassified points we simply sum the number of misclassified points in each replica.

With this in mind, we compared various metrics by averaging the results of 10 experiments using 10-fold cross validation. Assume $S = \{\theta_1, ..., \theta_{K-1}\}$ the set of all $K-1$ splits, $AS = \{\theta | \theta \in S \wedge \theta \neq \infty\}$ the set of all active splits and $value(s)$ the value of the we want to maximize (*e.g.* the information gain or the gain ratio of that split). The metrics used are the following:

- Sum: $\sum_{s \in S} value(s)$
  - This is equivalent to the average of all splits.
- Average: $\frac{\sum_{as \in AS} value(as)}{|AS|}$
- Product: $1 - \prod_{s \in S}(1 - value(s))$
  - Note that if the metric was simply the product, if one of the splits was done at infinity ($value(s) = 0$) that branch would be pruned.
- Geometric Mean: $\sqrt[|AS|]{\prod_{as \in AS}(value(as))}$
- Minimum: $\min(value(as))$
- Maximum: $\max(value(as))$
- Euclidean distance to the minimum possible value: $\sqrt{\prod_{as \in AS}((0 - value(as))^2)}$
- Euclidean distance to the maximum possible value: $\sqrt{|AS|} - \sqrt{\prod_{as \in AS}((1 - value(as))^2)}$
  - This metric is actually the difference between the maximum possible distance and the euclidean distance to the maximum, so that it can still be used as a maximization criterion.

The results are presented on Table 4.1.

While there is not much variation on the results, the sum appears to be one of the best combination rules, and therefore will be the one used in the next experiments.

Table 4.1: Comparison of various combination methods (oDT)

(a) Percentage of incorrect classifications: mean of 10 experiments

Dataset	Sum	Average	Product	Geometric	Min.	Max.	Dist. Min.	Dist. Max.
Circle	5.48	5.46	5.46	6.07	6.08	5.45	5.44	5.56
Non-mon.	19.92	19.88	19.97	19.97	20.24	19.78	19.82	19.87
ERA	74.48	74.41	74.48	74.73	74.88	74.53	74.46	74.38
ESL	34.75	34.74	34.63	35.33	38.71	38.42	35.50	38.36
LEV	38.42	38.98	38.46	38.96	39.99	38.74	38.73	42.46
SWD	42.13	42.20	42.14	42.93	45.24	42.58	42.10	43.67
Balance	22.77	22.85	22.79	22.97	22.66	22.79	22.73	24.22
BCCT	10.41	10.44	10.47	10.19	11.00	9.77	9.85	10.54

(b) Mean Absolute Error: mean of 10 experiments

Dataset	Sum	Average	Product	Geometric	Min.	Max.	Dist. Min.	Dist. Max.
Circle	0.05	0.05	0.05	0.06	0.06	0.05	0.05	0.06
Non-mon.	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20
ERA	1.24	1.24	1.24	1.24	1.26	1.24	1.24	1.24
ESL	0.38	0.38	0.38	0.38	0.43	0.42	0.39	0.42
LEV	0.42	0.43	0.42	0.43	0.44	0.42	0.42	0.48
SWD	0.45	0.45	0.45	0.45	0.48	0.46	0.45	0.47
Balance	0.29	0.29	0.29	0.29	0.29	0.29	0.29	0.31
BCCT	0.11	0.11	0.11	0.11	0.11	0.10	0.10	0.11

(c) Tree Size: mean of 10 experiments

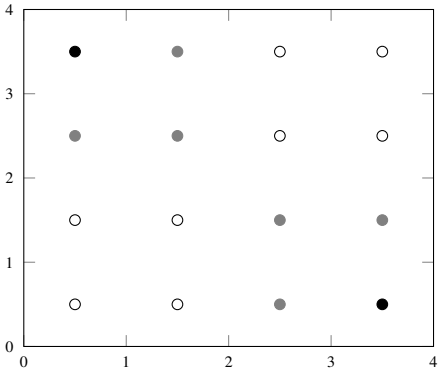
Dataset	Sum	Average	Product	Geometric	Min.	Max.	Dist. Min.	Dist. Max.
Circle	54.64	54.54	54.56	54.34	58.06	54.50	54.48	54.52
Non-mon.	75.20	74.96	75.30	74.94	74.32	75.74	75.66	74.92
ERA	11.74	11.86	11.74	11.68	14.16	11.96	11.80	11.92
ESL	29.86	29.14	30.20	29.92	28.98	26.98	29.12	28.50
LEV	23.16	26.70	23.28	26.30	28.88	23.82	23.26	41.24
SWD	26.68	27.88	26.84	28.24	27.20	27.10	27.38	34.40
Balance	75.26	77.56	75.36	77.24	77.46	77.32	75.80	85.52
BCCT	106.86	106.74	106.68	104.50	109.30	102.58	104.24	106.72

4.3.1 The XOR problem

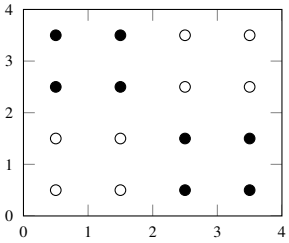
During the conversion from a K class problem to multiple $K-1$ binary problems, it is possible that, for one of the replicas, every possible cut has no (or very little) information gain. A toy example of such problem can be seen on Figure 4.2, where every possible cut on the first replica has no information gain. A more complex example of that presents this problem Figure 3.1.

In practice, since there's usually small amounts of noise in our data and our points are not perfectly placed (such as in Figure 4.2), this problem usually only leads to cuts at "random"[2] locations. Fortunately, one interesting property of decision trees is that random cuts do not have a big effect on their accuracy, as this is compensated by a larger tree size [Min89].

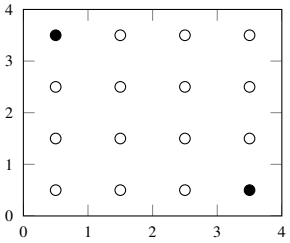
[2]Note that the cuts are note really random, but their position will be heavily dependent on the noise.



(a) Original ordinal problem



(b) First replica (problematic case)



(c) Second replica

Figure 4.2: Example of our decision tree on the data replicated space

Note, however, that some pruning optimizations that operate directly on the information gain might be problematic (this is the case of *Weka*'s MDL correction). Due to this our results were obtained by disabling this optimization.

Nevertheless, it is actually possible to detect when this problem has occurred during training. To do this, after we compute the optimal cuts for each replica, if one of the cuts was done at ∞ and has a similar (and significant) number of points classified as $\mathcal{C}_-$ and $\mathcal{C}_+$, then that replica is a XOR. We can then apply some heuristics to recompute the cut and attempt to solve the problem, such as:

- If we assume that a cut k should be placed between the cuts $k-1$ and $k+1$ we can simply use the point between this two cuts.
- Use a different metric for the information gain that prefers distributions with the same size. For example, assuming that our cut splits our set of points D in two sets, L and R, we could calculate the modified information gain $H(D) - \left(\frac{|L|}{|D|}\right)^2 \cdot H(L) - \left(\frac{|R|}{|D|}\right)^2 H(R)$.
- Unbalance the replica and recompute the cut. One could, for example, remove from our replica k all the points $\mathcal{C}_+$ that are also classified as $\mathcal{C}_+$ in $k+1$.

It is not clear which of these is the best option, as all have their limitations, therefore we will not be using any of them in this work.

4.4 Classifying a Point

To classify a point, we can assume that our model is a set of $K-1$ decision tree classifiers $\hat{f}_k$, where each decision tree is composed by branch nodes $\langle att, \theta_k, left, right \rangle$ and leaf nodes $\langle y_k \rangle$. Then we simple combine our binary labels as we would do on the Data Replication Method to obtain the final classification.

Note that these trees can also be easily used by humans without needing $K-1$ separate diagrams. To do this, one needs to parse the tree $K-1$ times, considering only the k^{th} attributes and k^{th} labels on each iteration. The final result can then be obtained by counting the $\mathcal{C}_+$ labels (represented as a $\bullet$ on Figure 4.1c).

On our results, we are combining our results via the simple formula $\hat{f}(\mathbf{x}) = \mathcal{C}_i$, with $i = 1 + \sum_k^{K-1} [\![f_k(\mathbf{x}) = \mathcal{C}_+]\!]$ although other combinations (such as the one proposed by Frank and Hall) can be used.

4.5 Results

We compared our method (oDT) to the C4.5 implementation from *Weka* (J48) and the Frank and Hall method using C4.5 decision trees. All classifiers were run with MDL correction disabled (as it cannot be used by oDT and degraded the performance of the other classifiers). Our experiments were executed 10 times, each one with 10-fold cross validation.

The results were compared with a paired t-test (corrected) with a confidence of 0.05. Results marked $\bullet$ were significantly worse than our algorithm and results marked $\circ$ were significantly better. They are presented on Table 4.2.

4.6 Conclusion

In this chapter we presented a new decision tree for ordinal classification (oDT). Our experiments show that this model is competitive and on par with the state of the art (performing better than C4.5 in some cases).

While our results were not superior to Frank and Hall's approach, we claim that our method has its advantages, namely on interpretability, as we believe that our trees are easier to read and comprehend than the set of $K-1$ uncorrelated trees generated by the Frank and Hall method.

Nevertheless, this method still has some possible improvements, which are discussed on Section 6.2.1.

Table 4.2: Comparison of the oDT with C4.5 and the Frank & Hall method

(a) Percentage of incorrect classifications: mean (standard deviation) of 10 experiments

Dataset	oDT	C4.5	Frank & Hall
Circle	5.48(2.29)	5.22(2.08)	5.39(2.33)
Non-mon.	19.92(4.16)	20.10(4.33)	19.54(3.79)
ERA	74.48(4.21)	72.22(3.78)	72.67(3.96)
ESL	34.75(6.68)	35.17(6.04)	34.84(5.93)
LEV	38.42(4.41)	39.56(4.13)	38.96(4.29)
SWD	42.13(4.86)	43.18(4.29)	41.64(4.34)
Balance	22.77(4.57)	22.56(3.33)	24.02(4.36)
BCCT	10.41(2.99)	9.62(2.79)	10.04(2.98)

(b) Mean Absolute Error: mean (standard deviation) of 10 experiments

Dataset	oDT	C4.5	Frank & Hall
Circle	0.05(0.02)	0.05(0.02)	0.05(0.02)
Non-Mon.	0.20(0.04)	0.20(0.04)	0.20(0.04)
ERA	1.24(0.11)	1.32(0.11)•	1.22(0.10)
ESL	0.38(0.08)	0.39(0.07)	0.37(0.07)
LEV	0.42(0.05)	0.43(0.05)	0.42(0.05)
SWD	0.45(0.05)	0.46(0.05)	0.44(0.05)
Balance	0.29(0.06)	0.33(0.06)•	0.29(0.06)
BCCT	0.11(0.03)	0.11(0.03)	0.11(0.03)

$\circ, \bullet$ statistically significant improvement or degradation

Chapter 5

Ensemble Methods for Ordinal Data Classification

In this chapter a we present some changes to AdaBoost and Random Forests, to show how ensemble methods can take advantage of the order relation in our data.

5.1 AdaBoost

One interesting property of AdaBoost is its similarities with Decision Trees [KM96], especially when a Decision Stump[1] is used as a weak learner.

With this, we can try and apply the same principles from our ordinal decision tree to AdaBoost. We can do this by independently boosting each replica while forcing that, at every iteration, all weak classifiers use the same attribute:

1. As before, we create $K-1$ binary replicas of our dataset;
2. Initialize the weights such as the sum of the weights of each replica is 1;
3. We train a weak classifier for each attribute of each replica;
4. Pick the best attribute (from the original attributes) $\hat{att}$ based on a combination of the errors of each replica;
5. For this iteration, use only the classifiers trained with attribute $\hat{att}$.
6. Calculate the weight of each replica k at iteration i;
7. If a replica has an error superior to 50%, stop boosting that replica;

[1] A Decision Stump is a very simple classification algorithm that only makes a cut in one attribute. It is equivalent to a Decision Tree with only one branch node.

8. The training stops if every replica has an error superior to 50% or after a set number of iterations.

A more detailed explanation of this algorithm can be seen on algorithm 5.1.

Data: A dataset $\mathcal{D} = (D, f)$ with N elements
Result: A classifier $\hat{f}_{\mathcal{F}}$
Replicate the dataset in $K-1$ binary replicas $\mathcal{D}_k = (D_k, f_k)$
Initialize the example weights as $w_k^j := \frac{1}{N}$;
Initialize $active_k = true$; $i := 1$;
while $\neg EndingCondition$ **do**
  forall the $att \in Attributes$ **do**
    forall the $k : 0 \leq k < K-1 \wedge active_k$ **do**
      $\hat{f}^i_{k,att} = \texttt{train}(\mathcal{D}_k, att)$;
      $error^i_{k,att} = \sum_{j=1}^{N} w_k^j [\![\hat{f}^i_{k,att}(\mathbf{d}_k^j) \neq f(\mathbf{d}_k^j)]\!]$;
    end
  end
  $\hat{att}_i = \arg\min_{att} \texttt{combination}(error^i_{k,att})$; ; `// The attribute to split on`
  forall the $k : 0 \leq k < K-1 \wedge active_k$ **do**
    $\hat{f}^i_k = \hat{f}^i_{k,\hat{att}_i}$;
    $error^i_k = error^i_{k,\hat{att}_i}$;
    if $error^i_k > 0.5$ **then**
      $active_k := false$; `// Stop boosting this replica`
    else
      $\alpha^i_k = 0.5 log(\frac{error^i_k}{1-error^i_k})$; `// Classifier weight`
      forall the j **do**
        $w_k^j := w_k^j (\alpha^i_k)^{1-[\![\hat{f}^i_k(\mathbf{d}_j) \neq f_k(\mathbf{d}_j)]\!]}$; `// Updates the example weights`
      end
      $\mathcal{F}_k := \mathcal{F}_k \cup \{\hat{f}^i_k\}$; `// Binary ensemble for relica` k
    end
  end
  Normalize the weights so that $\forall k : \sum_j w_k^j = 1$;
  $i := i + 1$
end
forall the k **do**
  $\hat{f}_{\mathcal{F}_k}(\mathbf{x}) = sign(\sum_i \alpha^i_k \hat{f}^i_k(\mathbf{x}))$; `// Binary classifier for relica` k
end
$\hat{f}_{\mathcal{F}} = \texttt{getOrdinalClassifier}(\hat{f}_{\mathcal{F}_1}, ..., \hat{f}_{\mathcal{F}_{K-1}})$; `// Combines the binary results`

Algorithm 5.1: Ordinal AdaBoost

In our implementation, we implement the function `train(`*dataset*,*att*`)` by creating a new dataset with only two attributes: *att* and the label. We then train our function in this dataset. Nevertheless, any function that compels a classifier to use a specific attribute could be used.

As was previously the case, there are many ways to combine our error function in order to pick the best attributes. To select the best combination method, we ran our AdaBoost variant

with the various alternatives and limited the maximum number of iterations to 10[2]. The results are presented on Table 5.1.

Table 5.1: Comparison of various combination methods (oADABOOST)

(a) Percentage of incorrect classifications: mean of 10 experiments

Dataset	Sum	Average	Product	Geometric	Min.	Max.	Dist. Min.	Dist. Max.
Circle	20.43	20.43	20.43	20.38	21.41	22.08	20.59	20.25
Non-mon.	74.42	74.42	74.52	74.52	74.45	74.28	74.48	74.49
ERA	75.12	75.12	74.97	74.93	75.17	75.23	75.16	74.88
ESL	35.41	35.41	35.44	36.31	36.04	34.65	33.91	35.60
LEV	38.40	38.40	38.24	38.33	39.96	38.40	38.42	38.35
SWD	43.84	43.84	43.83	43.38	44.42	43.14	43.82	43.96
Balance	15.40	15.40	15.43	15.19	15.27	15.76	15.43	15.19
BCCT	28.75	28.75	29.13	29.00	29.88	32.49	29.33	29.12

(b) Mean Absolute Error: mean of 10 experiments

Dataset	Sum	Average	Product	Geometric	Min.	Max.	Dist. Min.	Dist. Max.
Circle	0.21	0.21	0.21	0.21	0.22	0.23	0.21	0.21
Non-mon.	1.01	1.01	1.02	1.02	1.01	1.03	1.02	1.02
ERA	1.23	1.23	1.23	1.23	1.24	1.23	1.23	1.23
ESL	0.38	0.38	0.38	0.39	0.39	0.38	0.37	0.39
LEV	0.42	0.42	0.42	0.42	0.44	0.42	0.42	0.42
SWD	0.46	0.46	0.46	0.45	0.46	0.45	0.46	0.46
Balance	0.21	0.21	0.21	0.20	0.20	0.21	0.21	0.20
BCCT	0.30	0.30	0.30	0.30	0.31	0.34	0.30	0.30

Considering this, we will use the sum as a combination method on our next comparisons.

In Table 5.2 we show a comparison of the following ADABOOST variants, instantiated with Decision Stumps, limited to 100 iterations:

ADABOOST.M1 One of the most common ADABOOST variants with support for multiple classes [FS+96].

ADABOOST.M1W A small variant of the ADABOOST.M1 algorithm designed for with weak learners such as Decision Stumps [EP02].

ADABOOST.OR A variant of the ADABOOST.M1 designed for ordinal classification [LL09]. Since it needs an ordinal classifier as the weak learner, we used our decision tree limited to one cut.

OADABOOST Our ADABOOST variant.

It is possible to see that our method presents favorable results when compared to other boosting algorithms. Nevertheless, the current version only works with Decision Stumps, and therefore there's some interesting future work that could be done to make it more versatile.

[2] This limit is purposely small so that different combination methods have a larger impact on the final result.

Table 5.2: Comparison of oADABOOST with ADABOOST variants

(a) Percentage of incorrect classifications: mean (standard deviation) of 10 experiments

Dataset	oADABOOST	ADABOOST.M1	ADABOOST.M1W	ADABOOST.OR
Circle	6.87(2.61)	39.58(3.07)•	55.03(1.28)•	16.16(3.79)•
Non-mon.	66.30(3.14)	69.99(2.38)•	60.97(4.97)○	76.26(1.79)•
ERA	75.09(3.87)	78.19(2.32)	77.94(3.50)	78.10(2.31)
ESL	33.02(6.08)	56.97(2.89)•	46.77(6.05)•	44.86(5.48)•
LEV	37.63(4.44)	57.60(2.85)•	42.14(4.72)•	50.34(4.19)•
SWD	43.09(5.01)	48.20(3.90)•	48.26(5.13)•	48.20(3.90)•
Balance	2.57(2.14)	28.23(4.24)•	8.29(2.40)•	16.78(7.99)•
BCCT	12.80(2.76)	37.01(2.81)•	37.82(5.04)•	31.94(3.01)•

(b) Mean Absolute Error: mean (standard deviation) of 10 experiments

Dataset	oADABOOST	ADABOOST.M1	ADABOOST.M1W	ADABOOST.OR
Circle	0.07(0.03)	0.44(0.03)•	0.55(0.01)•	0.16(0.04)•
Non-Mon.	0.99(0.07)	1.30(0.08)•	1.19(0.14)•	1.03(0.04)
ERA	1.24(0.10)	1.43(0.07)•	1.44(0.12)•	1.43(0.07)•
ESL	0.35(0.07)	0.73(0.06)•	0.56(0.08)•	0.51(0.07)•
LEV	0.41(0.05)	0.71(0.03)•	0.46(0.06)•	0.57(0.05)•
SWD	0.45(0.05)	0.50(0.04)•	0.54(0.06)•	0.50(0.04)•
Balance	0.03(0.02)	0.49(0.09)•	0.08(0.02)•	0.18(0.09)•
BCCT	0.13(0.03)	0.38(0.03)•	0.40(0.07)•	0.33(0.03)•

○, • statistically significant improvement or degradation

5.2 Random Forests

While Sousa and Cardoso shown that combinations of consistent classifiers using the median results in a consistent classifier [SC11], they only studied the combination of consistent Decision Trees via bagging.

Our oDT algorithm however, while not guaranteed to be globally consistent, has a tendency to be consistent on each cut, with the exception of some edge cases (see appendix B for a more detailed proof). Also, since our new proposed algorithm is a Decision Tree, it is interesting to see if there's any way to improve ensembles designed for Decision Trees, such as random forests.

In Table 5.3 we present a quick comparison of random forests (each composed of 100 trees and picking from $\lfloor log_2(M)+1 \rfloor$ attributes on each node) to see if it is favorable to use the median or/and to use our ordinal Decision Tree when building the forests.

While there was not much variations on our results, it appears that the median is a better choice when combining ordinal classifiers. On the other hand, while the usage of our Decision Tree also appears to improve the results of normal Random Forests (especially on the MAE), the results are not as convincing.

5.3 Conclusion

In this chapter we presented a new AdaBoost variant, oAdaBoost, which presents a competitive performance when compared to other boosting algorithms. From this, it appears that boosting ensembles designed for ordinal classification might benefit of imposing restrictions, such as local parallelism, on its classifiers during training.

We also studied if Random Forests would benefit from using an ordinal classifier (in this case, our Decision Tree) and using the median to combine the results. From this, it seems that by using the median one can achieve better results on ordinal tasks.

Even though the results were positive, there are some changes that could lead to improvements of these methods, and therefore would be interesting to study. Those ideas are presented later on Sections 6.2.2 (oAdaBoost) and 6.2.3 (Random Forests).

Table 5.3: Comparison of Random Forest variants

(a) Percentage of incorrect classifications: mean (standard deviation) of 10 experiments - Comparison with normal Random Forests

Dataset	Random Forest	R. For. (Median)	R. For. (oDT)	R. For. (oDT + Median)
Circle	4.63(2.16)	4.62(2.18)	5.01(2.09)	4.94(2.04)
Non-mon.	17.74(4.17)	17.67(4.11)	16.99(3.48)	17.03(3.44)
ERA	73.87(4.41)	75.46(3.65)	73.95(4.53)	74.01(4.42)
ESL	33.43(5.95)	33.76(5.92)	32.20(5.86)	31.93(5.89)
LEV	37.69(4.00)	36.58(4.06)	37.36(4.53)	37.28(4.58)
SWD	42.79(4.34)	41.07(4.44)	41.57(4.32)	41.50(4.33)
Balance	18.47(3.54)	18.28(4.24)	17.94(3.99)	14.91(4.99)○
BCCT	1.61(1.22)	2.24(1.49)●	4.44(2.04)●	4.75(2.11)●

(b) Mean Absolute Error: mean (standard deviation) of 10 experiments - Comparison with normal Random Forests

Dataset	Random Forest	R. For. (Median)	R. For. (oDT)	R. For. (oDT + Median)
Circle	0.05(0.02)	0.05(0.02)	0.05(0.02)	0.05(0.02)
Non-Mon.	0.18(0.04)	0.18(0.04)	0.17(0.03)	0.17(0.03)
ERA	1.36(0.13)	1.28(0.10)○	1.22(0.11)○	1.22(0.11)○
ESL	0.36(0.06)	0.36(0.06)	0.35(0.07)	0.34(0.07)
LEV	0.41(0.05)	0.40(0.05)	0.41(0.05)	0.41(0.05)
SWD	0.46(0.05)	0.43(0.05)○	0.43(0.05)	0.43(0.05)
Balance	0.20(0.04)	0.19(0.05)	0.20(0.05)	0.16(0.05)○
BCCT	0.02(0.01)	0.02(0.02)	0.04(0.02)●	0.05(0.02)●

(c) Percentage of incorrect classifications: mean (standard deviation) of 10 experiments - Comparison with modified Random Forests

Dataset	R. For. (oDT + Median)	Random Forest	R. For. (Median)	R. For. (oDT)
Circle	4.94(2.04)	4.63(2.16)	4.62(2.18)	5.01(2.09)
Non-mon.	17.03(3.44)	17.74(4.17)	17.67(4.11)	16.99(3.48)
ERA	74.01(4.42)	73.87(4.41)	75.46(3.65)	73.95(4.53)
ESL	31.93(5.89)	33.43(5.95)	33.76(5.92)	32.20(5.86)
LEV	37.28(4.58)	37.69(4.00)	36.58(4.06)	37.36(4.53)
SWD	41.50(4.33)	42.79(4.34)	41.07(4.44)	41.57(4.32)
Balance	14.91(4.99)	18.47(3.54)●	18.28(4.24)●	17.94(3.99)●
BCCT	4.75(2.11)	1.61(1.22)○	2.24(1.49)○	4.44(2.04)

(d) Mean Absolute Error: mean (standard deviation) of 10 experiments - Comparison with modified Random Forests

Dataset	R. For. (oDT + Median)	Random Forest	R. For. (Median)	R. For. (oDT)
Circle	0.05(0.02)	0.05(0.02)	0.05(0.02)	0.05(0.02)
Non-Mon.	0.17(0.03)	0.18(0.04)	0.18(0.04)	0.17(0.03)
ERA	1.22(0.11)	1.36(0.13)●	1.28(0.10)●	1.22(0.11)
ESL	0.34(0.07)	0.36(0.06)	0.36(0.06)	0.35(0.07)
LEV	0.41(0.05)	0.41(0.05)	0.40(0.05)	0.41(0.05)
SWD	0.43(0.05)	0.46(0.05)	0.43(0.05)	0.43(0.05)
Balance	0.16(0.05)	0.20(0.04)●	0.19(0.05)●	0.20(0.05)●
BCCT	0.05(0.02)	0.02(0.01)○	0.02(0.02)○	0.04(0.02)

○, ● statistically significant improvement or degradation

Chapter 6

Conclusions and Future Work

6.1 Overview and Conclusions

In this dissertation we presented the state of the art on both ordinal data classification and ensemble methods (including those for ordinal tasks).

We then presented a new decision tree and the data replication method, which presents better results on ordinal classification tasks than C4.5.

Finally, we presented variations of two popular ensemble methods: AdaBoost and Random Forests, that not only appear to perform better then the original methods, but also shed some light on possible improvements for ordinal ensembles.

With this, we conclude that the results of ensemble methods for ordinal classification tasks can be improved by two simple modifications:

- Enforcing local parallelism on each weak classifier (as seen in oDT and oAdaBoost)
- Combining the results using a measure that takes the order into account (such as the median)

6.2 Future Work

While our algorithms presented good results, there are many ideas that could not be explored due to time constraints.

6.2.1 Future Work on oDT

One of the main modifications that could be made to our decision tree is the choice of the thresholds θ_k. As shown in appendix B, there are some cases where the order of our cuts is not guaranteed. It would, therefore, be interesting to study if there is any way to enforce this order (using reasonable computational resources) and if it improves our classifier's performance.

Another problem that our current implementation has is the XOR problem. While we already presented some solutions to this problem, there might be other options. One interesting idea is that, if we apply the ordering constraint to our thresholds our θ_k, it should be impossible that $\theta_k = \infty$

if $(\theta_{k-1} < \infty \wedge \theta_{k+1} < \infty)$. Therefore, it might be possible to enforce this restriction in such a way that solves this problem (or makes it even rarer). On the other hand, this could have the opposite result, and propagate the XOR problem to other replicas, therefore special care should be taken when implementing such constraint.

Also, we used a very simple approach to post-pruning, which might not be the best one. It might be interesting to see if executing the post-pruning on the ordinal space achieves better results. One simple way to do this could be to convert our final ordinal tree into a normal decision tree, and perform the pruning on that tree. Note that this transformation can be achieved in the following way:

1. Assume that our ordinal Decision Tree are $K-1$ binary trees T_k;
2. Replace the values of the leaves according to the mapping $(\mathcal{C}_- \mapsto 0, \mathcal{C}_+ \mapsto 1)$;
3. Initiate your Decision Tree as a single tree with a single leaf (with value 0);
4. Start with $k = 1$;
5. Substitute each leaf (that has label y) by a copy of T_k, however, add y to the value of each leaf of this new tree;
6. Repeat until $k = K-1$;
7. Assign a ordinal variable $\mathcal{C}_{y+1}$ to each leaf.

While this new decision tree is equivalent to our ordinal tree, pruning it should lead to different results. The pruned tree, however, will lose some of the correlation of our ordinal tree and might be much larger than our ordinal tree[1] (although this problem should be minimized by the post-pruning). Therefore, it might also be interesting to explore ways to prune our tree in its ordinal representation.

Finally, it would be interesting to apply the post-processing steps of Cardoso and Sousa [CS10] to our ordinal tree, to see if the restrictions during the tree growth can lead to improvements (either on the performance or optimization time) compared to other Decision Trees.

6.2.2 Future Work on oADABOOST

While our ADABOOST variant presented good results, our choice of attribute makes it unsuitable to be instantiated with more powerful algorithms (*e.g.* SVMs), as it does not allow them to make cuts that are not orthogonal to our attribute axes. Therefore, it might be interesting to study the possibility of, instead of forcing our algorithm to be trained along an attribute, allow it to be trained along a vector/combination of attributes. Unfortunately, the choice of these combinations does not appear to be trivial, and they might need to be randomly generated.

[1] Trees generated by this transformation have a maximum depth of $(K-1)*d$, where d is the maximum depth of our ordinal tree and have many nodes that are impossible to reach.

6.2.3 Future Work on Random Forests

Unfortunately, there was not enough time in this dissertation to explore Random Forests in more detail. Nevertheless, it would be interesting to see the results of extending the work of Sousa and Cardoso [SC11] to Random Forests. Note that simply applying their approach to each tree in the forest would result in a computationally expensive problem. Therefore it would be interesting to see if one could generate a Random Forest that simplified each sub-problem, for example by imposing that each tree used a certain maximum number of attributes.

References

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[BVh13] Mohamad Hasan Bahari and Hugo Van hamme. Normalized ordinal distance; a performance metric for ordinal, probabilistic-ordinal or partial-ordinal classification problems. In *Proceedings*, pages 1–7, 2013.

[CDC07] Jaime S Cardoso and Joaquim F Pinto Da Costa. Learning to classify ordinal data: The data replication method. *Journal of Machine Learning Research*, 8(1393-1429):6, 2007.

[CK05] Wei Chu and S Sathiya Keerthi. New approaches to support vector ordinal regression. In *Proceedings of the 22nd international conference on Machine learning*, pages 145–152. ACM, 2005.

[CS10] Jaime S Cardoso and Ricardo Sousa. Classification models with global constraints for ordinal data. In *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on*, pages 71–77. IEEE, 2010.

[CS11] Jaime S Cardoso and Ricardo Sousa. Measuring the performance of ordinal classification. *International Journal of Pattern Recognition and Artificial Intelligence*, 25(08):1173–1195, 2011.

[CSD12] Jaime S Cardoso, Ricardo Sousa, and Inês Domingues. Ordinal data classification using kernel discriminant analysis: A comparison of three approaches. In *Machine Learning and Applications (ICMLA), 2012 11th International Conference on*, volume 1, pages 473–477. IEEE, 2012.

[DB95] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *arXiv preprint cs/9501101*, 1995.

[DF08] Wouter Duivesteijn and Ad Feelders. Nearest neighbour classification with monotonicity constraints. In *Machine Learning and Knowledge Discovery in Databases*, pages 301–316. Springer, 2008.

[Die97] Thomas G Dietterich. Machine-learning research. *AI magazine*, 18(4):97, 1997.

[Die00] Thomas G Dietterich. Ensemble methods in machine learning. In *Multiple classifier systems*, pages 1–15. Springer, 2000.

[Dom12] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[EP02] Günther Eibl and Karl Peter Pfeiffer. How to make adaboost. m1 work for weak base classifiers by changing only one line of the code. In *Machine Learning: ECML 2002*, pages 72–83. Springer, 2002.

[Fer07] Artur Ferreira. Survey on boosting algorithms for supervised and semi-supervised learning. *Institute of Telecommunications*, 2007.

[FH01] Eibe Frank and Mark Hall. *A simple approach to ordinal classification*. Springer, 2001.

[FS95] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pages 23–37. Springer, 1995.

[FS+96] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156, 1996.

[HR76] Laurent Hyafil and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[KM96] Michael Kearns and Yishay Mansour. On the boosting ability of top-down decision tree learning algorithms. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 459–468. ACM, 1996.

[LL09] Hsuan-Tien Lin and Ling Li. Combining ordinal preferences by boosting. In *Proceedings ECML/PKDD 2009 Workshop on Preference Learning*, pages 69–83, 2009.

[Min89] John Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine learning*, 3(4):319–342, 1989.

[MMSJS12] João Mendes-Moreira, Carlos Soares, Alípio Mário Jorge, and Jorge Freire De Sousa. Ensemble approaches for regression: A survey. *ACM Computing Surveys (CSUR)*, 45(1):10, 2012.

[MS95] Sreerama K Murthy and Steven Salzberg. Decision tree induction: How effective is the greedy heuristic? In *KDD*, pages 222–227, 1995.

[PB00] Rob Potharst and Jan C Bioch. Decision trees for ordinal classification. *Intelligent Data Analysis*, 4(2):97–111, 2000.

[PdCAC08] Joaquim F Pinto da Costa, Hugo Alonso, and Jaime S Cardoso. The unimodal model for the classification of ordinal data. *Neural Networks*, 21(1):78–91, 2008.

[PMS01] MJ Pazzani, S Mani, and WR Shankle. Acceptance of rules generated by machine learning among medical experts. *Methods of information in medicine*, 40(5):380–385, 2001.

[SC11] Ricardo Sousa and Jaime S Cardoso. Ensemble of decision trees with global constraints for ordinal classification. In *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*, pages 1164–1169. IEEE, 2011.

[Sch90] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[SL02] Amnon Shashua and Anat Levin. Ranking with large margin principle: Two approaches. In *Advances in neural information processing systems*, pages 937–944, 2002.

[SYdCC13] Ricardo Sousa, Iryna Yevseyeva, Joaquim F Pinto da Costa, and Jaime S Cardoso. Multicriteria models for learning ordinal data: A literature review. In *Artificial Intelligence, Evolutionary Computing and Metaheuristics*, pages 109–138. Springer, 2013.

[TPV08] Grigorios Tsoumakas, Ioannis Partalas, and Ioannis Vlahavas. A taxonomy and short review of ensemble selection. In *ECAI 2008, workshop on supervised and unsupervised ensemble methods and their applications*, 2008.

[WB06] Willem Waegeman and Luc Boullart. An ensemble of weighted support vector machines for ordinal regression. *Transactions on Engineering, Computing and Technology*, 12:71–75, 2006.

[WFT+99] Ian H Witten, Eibe Frank, Leonard E Trigg, Mark A Hall, Geoffrey Holmes, and Sally Jo Cunningham. Weka: Practical machine learning tools and techniques with java implementations. 1999.

REFERENCES

Appendix A

Quick Notation Reference

$\mathcal{X}$	Feature Space
$\mathbf{x}_i$	Feature Vector $\mathbf{x}_i = (x_{i,1}, x_{i,2}, ..., x_{i,n}) \in \mathcal{X}$ – The i subscript might be omitted
$\mathcal{Y}$	Label space
$\mathcal{D}$	Dataset $\mathcal{D} = (D, f)$;
D	Set of examples $D \subseteq \mathcal{X}$ – $D = (d_1, d_2, ..., d_n)$
f	Example labeling $f : D \to \mathcal{Y}$
$\mathcal{D}'$	A subset of a dataset $\mathcal{D} = (D, f)$ – $\mathcal{D}' = (D', f)$ and $D' \subset D$
$\hat{f}$	Learned classifier $\hat{f} : \mathcal{X} \to \mathcal{Y}$
g	Perfect classifier $g : \mathcal{X} \to \mathcal{Y}$
$\mathcal{H}$	Hypothesis space $\mathcal{H} = \{h_1, h_2, ..., h_n\}$ – $\hat{f} \in \mathcal{H}$
$\mathbf{w}$	Weight vector $\mathbf{w} = (w_1, w_2, ..., w_n)$ – usually used to define an hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$
$\mathcal{Z}$	Non-linear feature space
Φ	Non-linear transformation $\Phi : \mathcal{X} \to \mathcal{Z}$ of the feature space
$\prec$	Order relation on an ordinal classification problem – $a \prec b$ stands for "a precedes b"
$\mathcal{C}_i$	Label of an ordinal classification problem where $\mathcal{Y} = \{\mathcal{C}_1, \mathcal{C}_2, ...\mathcal{C}_n\}$ – if our problem is ordinal, we assume that $\mathcal{C}_1 \prec \mathcal{C}_2 \prec ... \prec \mathcal{C}_n$
$\mathcal{C}_+$ and $\mathcal{C}_-$	Label of a binary classification problem
$[\![\cdot]\!]$	Indicator function. $[\![\cdot]\!]$ is 1 if the inner condition is true, 0 otherwise.
$\mathcal{F}$	Ensemble of classifiers $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, ..., \hat{f}_n\}$
$\hat{f}_{\mathcal{F}}$	Classifier generated from an ensemble $\mathcal{F}$
$\mathbf{e}_0$	Vector filled with zeros
$\mathbf{e}_q$	Vector filled with zeros except on the q-th position
θ	Split point (*e.g.* position of a cut made by a Decision Tree)

Appendix B

Proof of local consistency

Cardoso and Sousa [CS10] proposed the definition of consistency for ordinal functions. Intuitively, a function is said to be consistent in the ordinal setting if adjacent contiguous decision regions have adjacent classes (an example of a consistent and a non-consistent function can be seen on figure 3.2).

In this chapter, we will show that, for most K-class ordinal problems transformed with the data replication method, if our learning algorithm chooses the $K-1$ thresholds $\hat{\theta}_k$ on an attribute that maximize the information gain on each replica k, it holds that $\hat{\theta}_1 \leq \hat{\theta}_2 \leq ... \leq \hat{\theta}_{K-1} \vee \hat{\theta}_1 \geq \hat{\theta}_2 \geq ... \geq \hat{\theta}_{K-1}$. Note that, if our thresholds are ordered, then that set of splits is generating a consistent function.

B.1 Assumptions and Definitions

For the purposes of this proof we will have to make some assumptions about our dataset. First, we will assume assume that our feature vectors $\mathbf{x}_i \in D$ only have one attribute $x_i \in \mathbb{R}$. This is a valid assumption for our algorithms, as they pick various thresholds from one attribute only. Due to this assumption, we will use $f(\mathbf{x}_i)$ and $f(x_i)$ interchangeably.

We will also assume that $x_i \in [0,1]$ to be able to avoid normalizing constants, and therefore keep our expressions simpler. Since our algorithms do not use the absolute value of each point (only its relative value), this is also a valid assumption (the results would be the same if our dataset was normalized to fit this range).

We will assume that our dataset has an infinite number of points and that our dataset is noiseless. We claim that this assumption is not as strong as it might seem, as most datasets do not have that much noise and have a large number of points.

Finally, we will assume that $\mathcal{C}_-$ and $\mathcal{C}_+$ behave as if there was a mapping $(\mathcal{C}_- \mapsto 0, \mathcal{C}_+ \mapsto 1)$, namely:

- $\mathcal{C}_- \prec \mathcal{C}_+$.
- $\mathcal{C}_- + \mathcal{C}_- = \mathcal{C}_-$, $\mathcal{C}_- + \mathcal{C}_+ = \mathcal{C}_+$ and so on.

- $\mathcal{C}_+ + \mathcal{C}_+$ and $\mathcal{C}_- - \mathcal{C}_+$ have undefined results.

This assumptions allow us to treat our problem in an analytical fashion using the following two Lemmas:

Lemma 1. *For a binary replica, we can calculate the information gain of a split at point $\hat{\theta}$ via:*

$$
\begin{aligned}
&-\mathfrak{D} \cdot log_2(\mathfrak{D}) - (1-\mathfrak{D}) \cdot log_2(1-\mathfrak{D}) \\
&+\mathfrak{L} \cdot log_2(\mathfrak{L}) + (\hat{\theta}-\mathfrak{L}) \cdot log_2(\hat{\theta}-\mathfrak{L}) \\
&+\mathfrak{R} \cdot log_2(\mathfrak{R}) + (1-\hat{\theta}-\mathfrak{R}) \cdot log_2(1-\hat{\theta}-\mathfrak{R}) \\
&-\hat{\theta} \cdot log_2(\hat{\theta}) - (1-\hat{\theta}) \cdot log_2(1-\hat{\theta})
\end{aligned}
$$

Where $\mathfrak{D} = \int_0^1 f(x)dx$, $\mathfrak{L} = \int_0^{\hat{\theta}} f(x)dx$ and $\mathfrak{R} = \int_{\hat{\theta}}^1 f(x)dx$.

Lemma 2. *The data replication method guarantees that $f_k(\mathbf{x}) = \mathcal{C}_- \Rightarrow f_{k+1}(\mathbf{x}) = \mathcal{C}_-$ and $f_k(\mathbf{x}) = \mathcal{C}_+ \Rightarrow f_{k-1}(\mathbf{x}) = \mathcal{C}_+$.*

B.2 Monotonic Problems

Definition 1. *A dataset $\mathcal{D} = (D, f)$ is said to be monotonic if $x_i < x_j \Rightarrow f(x_i) \preceq f(x_j)$ (monotonically increasing) or $x_i < x_j \Rightarrow f(x_i) \succeq f(x_j)$ (monotonically decreasing).*

If we assume that our dataset is monotonic, it is trivial to prove that the ordering constraints are satisfied. Without loss of generality, we will assume that our dataset is increasingly monotonic.

Definition 2. *The function $u(\cdot)$ will be used to represent the Heavyside step function, where*

$$
u(x) = \begin{cases} \mathcal{C}_- & x < 0 \\ \mathcal{C}_+ & x \geq 0 \end{cases}
$$

Recall that the sum and subtraction operations on the binary labels $\mathcal{C}_-, \mathcal{C}_+$ are defined, allowing the definition of a descending step function ($\mathcal{C}_+ - u(x)$) and the combination of step functions.

Lemma 3. *If a dataset is increasingly monotonic, all its replicas are increasingly monotonic (assuming $\mathcal{C}_- \prec \mathcal{C}_+$), then our functions f_k behave like a step function and are defined by $f_k(\mathbf{x}) = u(x - \theta_k)$.*

Proof. The labeling of a replica k is given by:

$$
f_k(\mathbf{x}) = \begin{cases} \mathcal{C}_- & \text{if } f(\mathbf{x}) \preceq \mathcal{C}_k \\ \mathcal{C}_+ & \text{if } f(\mathbf{x}) \succ \mathcal{C}_k \end{cases}
$$

Therefore, $f(\mathbf{x_i}) \preceq f(\mathbf{x_j}) \Rightarrow f_k(\mathbf{x_i}) \preceq f_k(\mathbf{x_j})$ which allows us to replace $f(\cdot)$ with $f_k(\cdot)$ on Definition 1.

Since f_k is binary and monotonically increasing, then there must be a point θ_k such that:

$$f_k(\mathbf{x}) = \begin{cases} \mathcal{C}_- & x < \theta_k \\ \mathcal{C}_+ & x \geq \theta_k \end{cases}$$

With this, we have that $f_k(\mathbf{x}) = u(x - \theta_k)$. □

Lemma 4. *If a dataset is increasingly monotonic, then $f_k(\mathbf{x}) = u(x - \theta_k)$ and it holds that $\theta_1 \leq \theta_2 \leq ... \leq \theta_{K-1}$.*

Proof. Assume that f is increasingly monotonic where $f_1(\mathbf{x}) = u(x - \theta_1)$ and $f_2(\mathbf{x}) = u(x - \theta_2)$. As a counter example, assume that $\theta_2 < \theta_1$ (making our initial claim false).

Assume now a point $\mathbf{p} = (p)$ where $\theta_2 < p < \theta_1$. We know that $f_1(\mathbf{p}) = \mathcal{C}_-$ and $f_2(\mathbf{p}) = \mathcal{C}_+$.

However, such point cannot exist, since according to Lemma 2 $f_2(\mathbf{x}) = \mathcal{C}_+ \Rightarrow f_1(\mathbf{x}) = \mathcal{C}_+$, which is not true. Since there can be no points $\mathbf{p} = (p)$ where $\theta_2 < p < \theta_1$, it must hold that $\theta_1 \leq \theta_2 \leq ... \leq \theta_{K-1}$. □

Theorem 1. *If a dataset is increasingly monotonic and we pick the splits $\hat{\theta}_k$ that maximize the information gain in each replica, then it holds that $\hat{\theta}_1 \leq \hat{\theta}_2 \leq ... \leq \hat{\theta}_K$.*

Proof. By Lemma 3, we have that $f_k(\mathbf{x}) = u(x - \theta_k)$. By applying this to Lemma 1, we have that:

$$\mathfrak{D} = 1 - \theta$$

$$\mathfrak{L} = \begin{cases} 0 & \text{if } \hat{\theta} < \theta \\ 0 & \text{if } \hat{\theta} = \theta \\ \hat{\theta} - \theta & \text{if } \hat{\theta} > \theta \end{cases}$$

$$\mathfrak{R} = \begin{cases} 1 - \theta & \text{if } \hat{\theta} < \theta \\ 1 - \theta & \text{if } \hat{\theta} = \theta \\ 1 - \hat{\theta} & \text{if } \hat{\theta} > \theta \end{cases}$$

It is then possible to see that, when $\hat{\theta}_k = \theta_k$, our information gain is $-\theta \cdot log_2(\theta) - (1 - \theta) \cdot log_2(1 - \theta)$, which is equal to our initial entropy, and therefore its the maximum possible information gain. It is also easy to see that for $\hat{\theta}_k < \theta_k$ and $\hat{\theta}_k > \theta_k$ our information gain will be smaller. Therefore, we know that our cuts $\hat{\theta}_k$ will occur at positions θ_k. □

B.3 Symmetric Problems

Definition 3. *A dataset $\mathcal{D} = (D, f)$ is said to be symmetric if $\forall x \in [0,1] : f(x) = f(1-x)$*

Unfortunately, if our dataset is perfectly symmetric, it is impossible to guarantee that our thresholds $\hat{\theta}_k$ are ordered.

Lemma 5. *If a dataset is symmetric then, for every possible split $\hat{\theta}$, there will be another possible split $\hat{\theta}'$ with exactly the same information gain.*

Proof. For every split $\hat{\theta}$ there is another possible split $\hat{\theta}' = 1 - \hat{\theta}$.

Recall Lemma 1. For our first split $\hat{\theta}$, we have that:

- $\mathfrak{D} = \int_0^1 f(x)dx$
- $\mathfrak{L} = \int_0^{\hat{\theta}} f(x)dx$
- $\mathfrak{R} = \int_{\hat{\theta}}^1 f(x)dx$

While for our second split $\hat{\theta}'$, we have:

- $\mathfrak{D}' = \int_0^1 f(x)dx = \mathfrak{D}$
- $\mathfrak{L}' = \int_0^{\hat{\theta}'} f(x)dx = \int_0^{1-\hat{\theta}} f(x)dx = \int_0^{1-\hat{\theta}} f(1-x)dx = \int_{\hat{\theta}}^1 f(x)dx = \mathfrak{R}$
- $\mathfrak{R}' = \int_{\hat{\theta}'}^1 f(x)dx = \int_{1-\hat{\theta}}^1 f(x)dx = \int_{1-\hat{\theta}}^1 f(1-x)dx = \int_0^{\hat{\theta}} f(x)dx = \mathfrak{L}$

With this, it is easy to show that the information gain for both splits will be the same. □

Theorem 2. *If a dataset is symmetric and we pick the splits $\hat{\theta}_k$ that maximize the information gain in each replica, then we cannot guarantee that the splits $\hat{\theta}_k$ are ordered,*

Proof. Assume that our dataset has a set of thresholds $\hat{\theta}_1 \leq \hat{\theta}_2 \leq ... \leq \hat{\theta}_{K-1}$, which maximize the information gain, and a symmetric set of thresholds $\hat{\theta}'_{K-1} \leq \hat{\theta}'_{K-2} \leq ... \leq \hat{\theta}'_1$, where $\forall k : \hat{\theta}_k < \hat{\theta}'_k$.

Since according to Lemma 5 we know that the information gain for a split at point $\hat{\theta}_k$ is the same as the $\hat{\theta}'_k$, then it is possible that our method will pick splits from different sets, and therefore the chosen splits will not be ordered. □

B.4 Concave/Convex Problems

Definition 4. *A dataset $\mathcal{D} = (D, f)$ is said to be concave if it behaves as a monotonically increasing dataset $\mathcal{D}^{inc} = (D^{inc}, f^{inc})$ until a point p and behaves as a monotonically decreasing dataset $\mathcal{D}^{dec} = (D^{dec}, f^{dec})$ from point p onwards.*

(The Definition for a convex dataset can be obtained by switching the order of $\mathcal{D}^{inc}$ and $\mathcal{D}^{dec}$)

If we assume that our dataset is concave or convex and non-symmetric, it is possible to prove that the ordering constraints can be satisfied. Without loss of generality, we will assume that our dataset is concave.

Lemma 6. *If a dataset is concave, all its replicas are concave, which in turn guarantees that our functions f_k are defined by $f_k(\mathbf{x}) = u(x - \theta_k^{inc}) - u(x - \theta_k^{dec})$, with $\theta_k^{inc} < p < \theta_k^{dec}$.*

Proof. The labeling of a replica k is given by:

$$f_k(\mathbf{x}) = \begin{cases} f_k^{inc}(\mathbf{x}) & \text{if } x < p \\ f_k^{dec}(\mathbf{x}) & \text{if } x > p \end{cases} = \begin{cases} u(x-\theta_k^{inc}) & \text{if } x < p \\ 1-u(x-\theta_k^{dec}) & \text{if } x > p \end{cases}$$

From Definition 4 we know that $\theta_k^{inc} < \theta_k^{dec}$, therefore $f_k(\mathbf{x}) = u(x-\theta_k^{inc}) - u(x-\theta_k^{dec})$ □

Lemma 7. *Every replica of a concave dataset can be represented as a symmetric replica with a shift δ_k, where $f_k(\mathbf{x}) = u(x-(\theta_k+\delta_k)) - u(x-(1-\theta_k+\delta_k))$ and $\theta_k \in [0,0.5]$ and $\delta_k \in [-\theta_k, \theta_k]$.*

Proof. From Lemma 6, we know that $f_k(\mathbf{x}) = u(x-\theta_k^{inc}) - u(x-\theta_k^{dec})$.

Assume that we have $\theta = \frac{1-(\theta^{dec}-\theta^{inc})}{2}$ and a symmetric point $\theta' = 1-\theta = \frac{1+(\theta^{dec}-\theta^{inc})}{2}$. We claim that there must exist a shift δ that makes the following equalities true:

$$\begin{cases} \theta^{inc} &= \theta+\delta \\ \theta^{dec} &= \theta'+\delta \end{cases}$$

By solving this, we obtain the solution $\delta = \frac{\theta^{dec}+\theta^{inc}-1}{2}$.

Therefore, one can conclude that $f_k(\mathbf{x}) = u(x-(\theta_k+\delta_k)) - u(x-(1-\theta_k+\delta_k))$, with $\theta_k \in [0,0.5]$. Also, since $\theta_k+\delta \geq 0$ and $1-\theta_k+\delta \leq 1$,it must hold that $\delta_k \in [-\theta_k, \theta_k]$ □

Lemma 8. *If $\delta_k > 0$, then the maximum information gain will be at θ_k^{inc} and vice versa.*

Proof. Recall Lemma 1 and Lemma 7.

Now, we have that:

$$\mathfrak{D} = 1-2\cdot\theta$$

$$\mathfrak{L} = \begin{cases} 0 & \text{if } \hat{\theta} < \theta+\delta \\ 0 & \text{if } \hat{\theta} = \theta+\delta \\ \hat{\theta}-(\theta+\delta) & \text{if } \theta+\delta < \hat{\theta} < 1-\theta+\delta \\ 1-2\cdot\theta & \text{if } \hat{\theta} = 1-\theta+\delta \\ 1-2\cdot\theta & \text{if } \hat{\theta} > 1-\theta+\delta \end{cases} = \begin{cases} 0 & \text{if } \hat{\theta} \leq \theta+\delta \\ \hat{\theta}-(\theta+\delta) & \text{if } \theta+\delta < \hat{\theta} < 1-\theta+\delta \\ 1-2\cdot\theta & \text{if } \hat{\theta} \geq 1-\theta+\delta \end{cases}$$

$$\mathfrak{R} = \begin{cases} 1-2\cdot\theta & \text{if } \hat{\theta} < \theta+\delta \\ 1-2\cdot\theta & \text{if } \hat{\theta} = \theta+\delta \\ (1-\theta+\delta)-\hat{\theta} & \text{if } \theta+\delta < \hat{\theta} < 1-\theta+\delta \\ 0 & \text{if } \hat{\theta} = 1-\theta+\delta \\ 0 & \text{if } \hat{\theta} > 1-\theta+\delta \end{cases} = \begin{cases} 1-2\cdot\theta & \text{if } \hat{\theta} \leq \theta+\delta \\ (1-\theta+\delta)-\hat{\theta} & \text{if } \theta+\delta < \hat{\theta} < 1-\theta+\delta \\ 0 & \text{if } \hat{\theta} \geq 1-\theta+\delta \end{cases}$$

By applying Lemma 1 to our expressions and by limiting the domain of θ and δ according to 7, it can be shown that the first part of our expression (*i.e.* when $hat\,\theta \leq \theta+\delta$) is monotonically

increasing, our middle part (*i.e.* when $\theta + \delta < \hat{\theta} < 1 - \theta + \delta$) is convex and the last part (*i.e.* when $\hat{\theta} \geq 1 - \theta + \delta$) is monotonically decreasing. Therefore, our function has two local maxima: one at $\theta + \delta$ and another one at $1 + \theta + \delta$.

By substituting $\hat{\theta}$ for $\theta + \delta$ and $1 - \theta + \delta$, it is possible to show that the information gain at $\theta + \delta$ is superior to the one at $1 - \theta + \delta$ if and only if:

$$\begin{aligned} & -(\theta+\delta)log_2(\theta+\delta) - (1-(\theta+\delta))log_2(1-(\theta+\delta)) \\ > \\ & -(\theta-\delta)log_2(\theta-\delta) - (1-(\theta-\delta))log_2(1-(\theta-\delta)) \end{aligned}$$

Note that both sides of this inequality are actually entropies of binary events: The first one of an event with probability $p_1 = (\theta + \delta)$ and the second one of an event with probability $p_2 = (\theta - \delta)$. Knowing that, from Lemma 7, $\theta \in [0, 0.5]$, $\delta \in [-\theta, \theta]$ and that the maximum value of the Shannon entropy for a binary event occurs at probability $p = 0.5$, it is easy to see that the first part will dominate when $\delta > 0$ and vice versa.

Also, it can be seen that, when $\delta = 0$, both sides will be the same, as should be expected since that makes our dataset symmetric. □

Theorem 3. *If* $\forall k : \delta_k > 0$ *or* $\forall k : \delta_k < 0$*, then* $\hat{\theta}_1 \leq \hat{\theta}_2 \leq ... \leq \hat{\theta}_{K-1} \vee \hat{\theta}_1 \geq \hat{\theta}_2 \geq ... \geq \hat{\theta}_{K-1}$.

Proof. If $\delta_k > 0$, then all splits θ_k will take place on the monotonically increasing part of the concave function, and therefore will be ordered (and vice versa). □