Université de Liège

University of Liege

PhD Thesis

Computer science

Trimming the complexity of Ranking by Pairwise Comparison

Samuel Hiard

2012

Abstract

In computer science research, and more specifically in bioinformatics, the size of databases never stops to increase. This can be an issue when trying to answer questions that imply algorithms in nonlinear polynomial time with regards to the number of objects in the database, the number of attributes or the number of associated labels per objects.

This is the case of the Ranking by Pairwise Comparison (RPC) algorithm. This algorithm builds a model which is able to predict the label preference for a given object, but the computation needs to be performed in an order of $\frac{N(N-1)}{2}$ in terms of the number N of labels. Indeed, a pairwise comparator model is needed for each possible pair of labels.

Our hypothesis is that a significant part of the set of comparators often contains redundancy and/or noise, so that trimming the set could be beneficiary. We implemented several methods, starting from the simplest one, which merely chooses a set of T comparators ($T < \frac{N(N-1)}{2}$) at random, to a more complex approach based on partially randomized greedy search.

This thesis will provide a detailed overview of the context we are working in, provide the reader with required background, describe existing preference learning algorithms including RPC, investigate on possible trimming methods and their accuracy, then will conclude on the relevance and robustness of the trimming approximation.

After implementing and executing the procedure, we could see that using between $\frac{N}{2}$ and 2N comparators was sufficient to keep up with the original RPC algorithm, as long as a smart trimming method is used, and sometimes even

outperforms it on noisy datasets. Also, comparing the use of base models in regression mode vs. classification mode showed that models built in regression mode may be more robust when using the original RPC.

We thus empirically show that, in the particular case of RPC, reducing the complexity of the method gives similar or better results, which means that problems that could not be addressed by this algorithm, or at least not in an acceptable period of time, now can be. We also found that the regression mode yields RPC to be often more robust regarding its base learner parameters, meaning that the quest of optimality, which can also be time-consuming, is less difficult.

Yet research on this topic is not over, and we could think of different means to further improve the RPC algorithm or investigate other innovative approaches, which will be discussed in the future work section. Also, the trimming method is not limited to RPC and could be applied to other algorithms which aggregate information provided by a set of models, e.g. the whole multitude of ensemble models used in machine learning.

Frequent notations

In this thesis, we will refer to mathematical objects or concepts using symbols. We hereby provide the correspondence between our notations and what they represent.

#	The number of items in a set. For instance, $\#Q$ is the number of objects in set Q
a	An object attribute
A	The set of attributes (features) used to build a model
$\mathcal{A}$	An algorithm
$\mathbf{A}$	A computer software
$\mathbf{B}$	A human being
$\mathbf{C}$	A human being different than $\mathbf{B}$
c	A class
C	A set of conditions or constraints
d	The depth of a node in a tree structure if d transitions are required to reach this node from the root node
D	A domain, i.e. a field of study that defines a set of common requirements, terminology, and functionality for any software program constructed to solve a problem in that field
$f()$	A mathematical function
G	A set of goal states
h	A solution, a hypothesis
H	The set of all possible solutions, the hypothesis space
$\mathcal{H}$	The heuristic knowledge

i, j, k, l	Used to represent index values
K	The number of randomly selected pairs (attribute,threshold) in the node split procedure of the Extra-Trees algorithm
$\mathbf{LS}$	The learning set (i.e. the set of objects used to construct a model)
M	The number of trees in a tree ensemble
$\mathcal{M}$	A tree structure
n	The size of the learning set (thus $n = \#\mathbf{LS}$)
$n_{\min}$	The number of required objects in a given node in order to perform a node split
N	The total number of class labels in a given preference learning dataset
o	An object or an observation
O	A set of objects
$P(...)$	A probability distribution
$\mathcal{P}$	A prediction procedure
π	An ordering (also quoted “ranking”) or a permutation
$\overline{\pi}$	A mean ordering
$\hat{\pi}$	A predicted complete ordering
$\Pi(\mathbf{Y})$	The set of all possible orderings on $\mathbf{Y}$
$\overline{\Pi(\mathbf{Y})}$	The set of all possible orderings on all subsets of $\mathbf{Y}$
q_{kl}	A comparison, comparing label y_k to label y_l (also quoted as $\succ_{kl}$)
$f_{kl}(x)$	A comparator built on a q_{kl} comparison
Q	A set of comparisons
Q^{Full}	The set of all $N(N-1)/2$ possible comparisons
r	A reward
S	A score
$\mathbf{S}$	A state space
s_i	The i^{th} state
s_0	The initial state
T	The number of comparisons (or their corresponding comparators) in a trimmed subset of comparisons (comparators)
$\mathcal{T}$	A state transition graph
$\mathbf{TS}$	The test set (i.e. the set of objects used to estimate the model accuracy)

u	An action
U	A set of possible actions
v	A real-valued variable which can be used in particular to represent a vote
x	The attribute vector of an object
X	A set of objects
$\mathbf{X}$	The set of all possible attribute vectors (the input space)
y	An output (a value, a class label, a ranking,...)
$\mathbf{Y}$	The set of all possible outputs (the output space)

Contents

1 Introduction **15**

1.1 Context and History 15
1.2 Definition of AI 17
1.3 Problem solving 18
1.4 Knowledge representation 19
1.5 Machine Learning 20
1.6 Objective 21
1.7 Thesis outline 22

2 A gentle introduction to Machine Learning **25**

2.1 Machine Learning 26
2.2 Types of ML protocols 27
2.2.1 Supervised learning 27
2.2.2 Unsupervised learning 28
2.2.3 Semi-supervised learning 31
2.2.4 Reinforcement learning 31
2.2.5 Interoperability of ML techniques 33
2.3 Supervised learning algorithms 36
2.3.1 Definition of "model" 37
2.3.2 Problem settings 37
2.3.3 Definition of "good model" 39
2.3.4 Predictive accuracy 39
2.3.5 Examples of SL algorithms 45

3 Preference Learning **55**

3.1 Preference representations 57
3.1.1 Utility function 57

3.1.2 Pairwise information 58
3.1.3 Orderings 58
3.1.4 Interchangeability 58
3.2 Learning to rank 60
3.2.1 Label ranking 60
3.2.2 Object ranking 68
3.2.3 Instance ranking 69
3.3 Combining (partial) orderings 71
3.3.1 Difficulty of combining orderings 72
3.3.2 Minimizing pairwise disagreements 72
3.3.3 FAS-PIVOT 73
3.3.4 PTAS 74
3.3.5 Mallows model 75
3.4 Similar domains 80
3.4.1 Multi-label classification (MLC) 80
3.4.2 Multi-label ranking (MLR) 86
3.5 Existing reduction techniques 86
3.5.1 QWeighted algorithm 87
3.5.2 QWeighted for multi-label classification 87
3.6 Summary and outlook 88

4 Methods and algorithms 91
4.1 Formal description of the problem 93
4.1.1 Complexity reduction 94
4.1.2 Sparsity in partially ranked datasets 95
4.2 Scoring a ranking scheme on a sample 96
4.3 Dealing with sparsely ranked datasets 98
4.3.1 Modeling empty comparisons by dummy models 98
4.3.2 Dropping empty comparisons 98
4.4 Comparison selection algorithms 99
4.4.1 Evaluating a set of comparisons during optimization . . . 99
4.4.2 Pure Random selection (PR) 100
4.4.3 Estimation of Distribution Algorithm (EDA) 103
4.4.4 “Exhaustive” Greedy Search (EGS) 106
4.4.5 Randomized Greedy Algorithm (RGS) 109
4.5 Models for learning comparators 111
4.6 Hard comparators vs soft comparators 114
4.7 Summary 115

5 Results 117
5.1 Datasets used for experimental validation 119
5.1.1 OMIB database (synthetic, complete, $N = 10$) 120
5.1.2 Sushi database (real life, partial, $N = 100$) 121
5.1.3 MovieLens database (real life, partial, $N = 1682$) 122
5.2 Effect of the model mode in a RPC scheme 123
5.2.1 Using optimized tree parameters 124
5.2.2 Using sub-optimal tree parameters 125
5.3 Evaluation of the PR selection method 133
5.3.1 Effect of imposing a full class coverage 134
5.4 Evaluation of EDA, EGS and RGS 136
5.4.1 Relevance of optimizing with respect to the **LS** outputs . 137
5.4.2 Ranking score on OMIB 139
5.4.3 Ranking score on Sushi 140
5.4.4 Ranking score on MovieLens 142
5.5 Effect of (meta-)parameters in a trimmed RPC scheme 145
5.5.1 Effect of the model mode 145
5.5.2 Effect of sparsity control 148
5.5.3 Influence of the tree parameters (on RGS) 154
5.5.4 Influence of the default order used in tie breaking 158
5.5.5 Removing the object ID from the model construction . . 161
5.5.6 Using simulated perfect models 163
5.5.7 Setting the j_1 and j_3 RGS parameters to huge values . . . 167
5.6 Complexity of our set selection algorithms 167
5.6.1 Time Complexity (computational speed) 169
5.6.2 Space Complexity (memory usage) 170
5.7 Concluding remarks 171

6 Applications 175
6.1 Decision support systems 175
6.2 Recommender systems 176
6.3 Personalized devices 178
6.4 Ranking search results 179
6.5 Text categorization 180

7 Future work 181
7.1 Limits of this work 181
7.2 Possible improvements 182
7.2.1 Using prior knowledge 182

7.2.2 Other trimming algorithms 183
7.2.3 Method parameters 185
7.2.4 Controlling partial ranking and noise 186
7.2.5 Extremely Randomized Ranking 186
7.2.6 Optimizing with a given budget 187
7.2.7 Using the mean ordering to select Q 187
7.2.8 Weighted distribution based on comparator **LS** 188

8 Conclusion 189

9 Published work 191
9.1 Bioinformatics publications 191
9.2 Preference learning publications 193

10 Acknowledgments 195

A Efficient implementations 197
A.1 Weighted distribution 198
A.1.1 Simple array 199
A.1.2 Linked List 199
A.1.3 Binary tree 200
A.2 Spearman's ρ correlation coefficient 201
A.2.1 Naive method 201
A.2.2 Optimized method 203
A.2.3 Computational gain 204
A.2.4 Efficient computation of ρ in an EGS/RGS context 205
A.3 Pre-computing 206
A.3.1 Tree building 207
A.3.2 Rejected comparisons 207

B Result figures 209
B.1 Accuracy 210
B.1.1 Reverse alphanumerical order 210
B.1.2 Reverse alphanum / Mean order 219
B.1.3 Mean order 224
B.2 Correlation between LS and TS ranking scores 226
B.3 Complexity 227
B.4 Robustness of RGS 229
B.4.1 Omib 229

B.4.2 Sushi_0, Dropping empty comparisons 231
B.4.3 Sushi_0, Using a dummy model 232
B.4.4 Sushi_3, Dropping empty comparisons 233
B.4.5 Sushi_3, Using a dummy model 235
B.4.6 MovieLens, Dropping empty comparisons 236
B.4.7 MovieLens, Using a dummy model 237

C Alternative approach 239
C.1 Introduction 239
C.2 LRANN loss function 240
C.3 Experimental setup 240
C.4 Results 241
C.5 Conclusion 242

Bibliography 243

Chapter 1

Introduction

Contents

1.1 **Context and History** 15
1.2 **Definition of AI** 17
1.3 **Problem solving** 18
1.4 **Knowledge representation** 19
1.5 **Machine Learning** 20
1.6 **Objective** 21
1.7 **Thesis outline** 22

This chapter provides an overview, in a quite broad manner, of artificial intelligence. The task of summarizing AI is very challenging due to the vast diversity of domains which emerged from it, so we will limit this introduction to (i) a discussion about AI over the ages, (ii) the definition of an intelligent system, (iii) an explanation on how an AI can find the solution to a problem, (iv) how AI stores knowledge and (v) how to provide AI with learning abilities. This chapter ends with the thesis outline.

1.1 Context and History

Since many decades, mankind tried to manufacture robots and computers that could think and reason or, at least, pretend that they do. But where are we now?

Nello Cristianini, in his plenary talk named “Are we there yet?” [Cri09], given during ECML 2009 at Bled, Slovenia, gives us a hint. No, we are not there yet, but we have done quite a long journey.

The real origins of the first machine gifted with “intelligence” remain quite vague, as many examples of artificial beings can be found in mythologies over the ages, but without any discovered associated physical devices. However, even if one can be, from a scientific point of view, legitimately suspicious about the development of AI in ancient times, the will to “forge the gods” is present in several texts, according to Pamela McCordruck in her book “Machines Who Think” [McC79].

Research in AI began during a conference on the campus of Dartmouth College in the summer of 1956, where participants were asked to share ideas about artificial intelligence in general, all fields included. From this conference emerged several applications which were considered astonishing at that time, such as theorem proving or English speaking. AI founders were very optimistic and stated, in the middle of the 1960’s that within twenty years, AI would be so advanced that a computer could replace a human in most of the tasks.

This prediction was a bit too optimistic, as they lacked the ability to correctly estimate the difficulty of some of the problems they were addressing. In 1974, due to the lack of productive projects, US and British government cut off all undirected exploratory research in AI. This period of time, where funding on this topic was hard to acquire, was called the “AI winter”.

Fortunately, research on AI in the early 1980’s was revived thanks to expert systems, which could simulate knowledge and skill of one or several experts. In 1985, the market of AI reached billions of dollars, but in 1987, a second AI winter occurred and lead to the collapse of the Lisp machine market.

Finally, in the 1990’s, AI knew a second blow and was used in several fields like logistics, data mining and medical diagnosis.

At present time, we can make computers learn, decide, and even reason on some subjects. Yet the question “Can a machine act intelligently?” is still an open problem, and the simulation of intelligence has been separated into several fields, depending on traits or skills that the developer wants his machine to acquire.

We will use, as main references, the books “Machine Learning” [Mit97] by

Tom Mitchell and the “Encyclopedia of Artificial Intelligence” [SE87] by Shapiro and Eckroth. Although we will try to cover as many aspects of artificial intelligence and machine learning as possible, these two books provide a much more complete overview of the knowledge in these domains.

1.2 Definition of AI

Defining formally an intelligent system is not an easy task for two reasons. The first one is that the notion of intelligence is relative. From a fly point of view, a chicken is very smart, but it is no longer the case from a human point of view. Hence a human being would consider a system to be intelligent if its response is equivalent or better than a classical human response. The second one is that we expect more and more cognitive behavior from an intelligent system. In 1956, optical character recognition (OCR) was considered as AI but, at the present time, most scanners' drivers provide an OCR service and no one considers this kind of software as AI anymore.

The first attempt to decide whether a system is intelligent or not was the Turing test. In this experiment, the system **A** interacts with a human **B** through written messages. This human **B** simultaneously converses with another human **C** in the same manner. The system passes the test if the human **B** is unable to differentiate the AI system **A** from the human being **C**. This test has been criticized for mainly two reasons: the subjectivity of the judge (human **B**) and the restriction of the machine to behave like a human, but not smarter (otherwise, the judge would notice that the machine is “too intelligent” and is hence not a human).

In the Encyclopedia of Artificial Intelligence [SE87], a system is considered to be intelligent if it possesses at least one of the following two features:

1. It produces an “intelligent” response to a given task. The internal functioning is not considered in this statement, as long as the output corresponds to an intelligent response to a given input.

2. It simulates the behaviour of a human brain. In this dual case, the response itself does not need to be optimal, as human behavior is sometimes suboptimal. The most important criterion is that the process of the system is similar to the process of the human brain. Estimation of the performance of such systems are performed by comparing in very small intervals

(say one second) the output of the computer and the human behavior.

To these two features is often added a third one: adaptivity. The system should be able to learn from its experiences and adapt itself to a changing environment or goal, by updating its decisions accordingly. This could be achieved by encoding all possible situations that the system could possibly meet or by using a more flexible approach where the system uses a feedback loop (it receives information on the accuracy of its previous decisions) to adapt its behavior.

Finally, Marvin Lee Minsky, one of the AI founders, defined the field as *"the building of computer programs which perform tasks which are, for the moment performed in a more satisfactory way by humans because they require high level mental processes such as: perception learning, memory organization and critical reasoning"*.

1.3 Problem solving

Problem solving plays an important role in AI. A solver should be able, in a domain specification D, to find a solution h, where h belongs to the set of possible solutions H such that h satisfies a problem condition C. C and H should be expressed in elements of D. So, the representation of a problem is

$$h \in H,\, C(h) = true,\, D \vdash \{C, H\}. \tag{1.1}$$

One possible manner of algorithmically solving a problem is to use a state-space search. Having a starting hypothesis containing 5 elements $\{\mathbf{S}, s_0, \mathcal{T}, G, \mathcal{H}\}$ where $\mathbf{S}$ is the state space, s_0 is the initial state, $\mathcal{T}$ is the transition graph, G is the set of goal states and $\mathcal{H}$ is the heuristic knowledge, the system should find a sequence of actions (i.e. transitions) U such that G is reachable from s_0 when sequentially applying actions in U.

In order to achieve this, one can build a tree structure $\mathcal{M}$ rooted at s_0 and expand it by adding j branches to every state s_i such that the j moves satisfy both $\{s_i, s_{k_j}\} \in \mathcal{T}$ and $s_{k_j} \notin \mathcal{M}$. The tree can be built in at least three ways [Pea84]:

- **Breadth first**: Each possible move is taken into account in the branch construction for a given state before analyzing the next one, and all states of depth d are evaluated before considering states at depth $d + 1$.

- **Depth first**: Each state is evaluated as soon as it is added in the tree, and other possible moves from the parent state are considered only after all the moves from the child states have been explored.
- **Best first**: The next evaluated state depends on a heuristic (an estimate of the distance from the current state to the goal state), and the state which appears as the closest to G is expanded.

The accuracy of the heuristic is crucial for the best first search, otherwise states which are actually farther from the solution could be evaluated before a closer one, leading to a loss of time or, even worse, to a failure in finding a solution. Heuristics are typically used when the state space is too large to be searched entirely in a reasonable amount of time (as in chess, for instance). In this latter case, the tree search will be arbitrarily limited to a certain depth (e.g. limited lookahead) and the heuristic is applied to all leaf states to evaluate the head subset of U.

1.4 Knowledge representation

Efficiency of many solutions in AI depends more on the availability of a large amount of knowledge rather than on complex algorithms. For example, in medical care, the accuracy of the diagnosis of a patient and the list of available cures depend on medical research. Moreover, human beings do not need to learn several times how to perform a given task; one does not need to learn again how to drive before using one's car. So, AI systems should be able to store and use some kind of knowledge in order to perform their tasks.

In practice, a Knowledge based systems (KBS) should at least contain three functionalities:

1. **Storing**: The system is able to keep the information as well as to check that the fact to be added is well-formed and does not conflict with previously stored facts.
2. **Retrieval**: The system can use his knowledge to answer direct questions about its stored facts (e.g. "When did the first man walk on the moon?" or "Provide the list of all past U.S.A. presidents')
3. **Inference**: From facts and rules in its knowledge base, the system is able to infer new knowledge using, for instance, syllogisms (e.g. if x is a car and if all cars have wheels then x has wheels)

Several structures have been proposed to allow a computer to possess these functionalities, but they are all included in three classes.

1. **Slot-and-filler structures**: Objects or classes of objects are represented by a slot (a box, a frame, a node, ...) and relations by arcs between them (is_a, has, produces, ...). The two main structures using this representation are semantic networks[Sow91] and frames[Min74].

2. **Production rules**[BFKM85]: The knowledge is stored by the means of IF THEN statements. It can be used to produce new facts (e.g. IF person is a smoker THEN person has a higher risk of lung cancer. John is a smoker. ⇒ John has a higher risk of lung cancer.) or to perform actions (e.g. IF temperature is under 10°C AND people are in the building THEN turn on the heat.).

3. **Logical formulas**[Men87]: Facts are represented by formulas which consists of atomic terms (e.g. p, q, ...), functions (e.g. $Solid(p)$), connectors ($\neg$, $\wedge$, $\vee$, ...) and quantifiers ($\forall$, $\exists$). For instance, a logical formula could be :
   $\forall x\ (Smoker(x) \Rightarrow HigherLungCancerRisk(x))$.

1.5 Machine Learning

Although the field of machine learning will be more thoroughly explored in the next chapter, we will hereby provide a general overview of this domain.

The goal of machine learning is to provide a machine or software with the ability to learn or to adapt itself. Historically, the field arose in the mid-1950's and became really active in the 1980's when expert systems, despite their successes, showed three major limitations:

1. They require man-years to construct them and to maintain them.

2. They are problem-specific and can not easily be transposed to other functions neither be adapted if the working environment changes.

3. They are unable to model human learning mechanisms.

Developing machine learning algorithms can be hard due to the lack of interpretability from human behaviors compared to other algorithms. For instance, if one considers sorting algorithms, one can easily describe a step by step methodology used by human beings and conceive the bubble sort algorithm [Knu98], which would be non optimal in terms of computational time, but will be intuitive. On the other hand, would one be asked to describe one's methodology for learning, this would be very difficult, as the mental process used in learning is somehow performed at the subconscious level.

The most widely studied problem in machine learning is to learn a concept from examples. This is also well correlated to what one would expect human beings to use for learning. Indeed, in this learning framework, the computer is provided with (positive and/or negative, labeled or unlabeled) examples, or samples. The goal is to find a generalization of these examples in the form of a concept, such that the concept is consistent with the given examples[1]. The generated concept can then be used to perform a prediction on unseen data.

The "concept" to learn in machine learning can take various forms. For instance, it could be a function, a decision strategy, a probability distribution, etc.

Without the philosophical aspect that this implies, we can consider that human beings behave in a similar fashion in order to learn the difference between "good" and "bad". They are confronted to situations or events, which correspond to the "examples" in machine learning. From these examples, they can generate a concept of "good vs. bad". This concept can evolve with the appearance of new examples. In the same manner that two human beings can have a different representation of the "good vs. bad" concept, two concepts produced by the same machine learning algorithm can differ according to a difference in the learning examples.

1.6 Objective

This thesis will focus on a particular problem in machine learning, namely preference learning[2]. In this context, the examples are characterized by a (possibly partial) ordering of classes (a label ranking instance) and are also described by an attribute vector. The objective of preference learning algorithms is to

[1]Without anticipating, we would like to note that perfectly fitting the learning sample is generally not a good idea (or is sometimes impossible). Nevertheless, we expect the produced concept to have a reasonable correlation with the learning sample.

[2]and, more specifically, label ranking, which will be further defined.

find a concept which correlates the attribute vector to the corresponding class-label ordering. The predicted ordering obtained with this concept should be as close as possible to the true labeled ordering. For instance, the problem of designing a recommender system in a media selling website (e.g. Amazon) can be addressed using a preference learning protocol: the output labels would correspond to the set of items to be recommended, while the attributes would correspond to information about the potential consumer. We will especially focus on the Ranking by Pairwise Comparison algorithm (RPC) [HFCB08], whose learned concepts are generally reliable but are obtained at a prohibitive computational price when the number of labels is large. Our contribution aims at drastically reducing this computational complexity (for both the learning and the prediction stages) while generating concepts which remain of competitive accuracy with respect to the state-of-the-art.[3]

1.7 Thesis outline

Chapter 2 provides the reader with background knowledge in machine learning required to fully understand the notions that we will be using further, e.g. models and supervised learning.

Chapter 3 depicts published preference learning algorithms for label ranking, including the Ranking by Pairwise Comparison algorithm, which we will try to improve. We will discuss different representations of preference relations, different classes of learning to rank problems, as well as the main problem of combing several rankings into a single one.

Chapter 4 will start by formally describing the problem which we are trying to solve, then present the algorithm used to compute the correlation between two rankings. Furthermore, it will detail methods and algorithms which we will be using to solve our problem, whether low-level as, for instance, the model used in pairwise comparators, or high-level, as the comparator selection method. Some solutions to the data sparsity due to partial ranking will be presented, and this chapter concludes by considering the mode (classification or regression) of base models.

[3]This informally describes the subject of the thesis. A more formal definition of the problem and of our contribution will be given in chapter 4.

Chapter 5 collects the empirical validations we carried out, and provides a thorough analysis of these. We first consider accuracy for our main concern, where each trimming methodology is evaluated. Then we investigate the robustness of our most efficient comparator selection method.

Chapter 6 is dedicated to a discussion of the large-scale applications that could not be directly addressed by the original RPC but that our algorithms of reduced complexity could tackle.

Chapter 7 sets the limits of our research and discusses about future work, and how the proposed methods could still be improved, and which fields of research can emerge from it.

Chapter 8 gives a conclusion on this thesis.

Chapter 9 lists the publications that were produced during the thesis as first author, whether related to preference learning or not.

Chapter 10 ends this thesis with acknowledgments.

Appendix A provides technical information about some implementation issues and how they were addressed.

Appendix B contains all detailed result figures, which were not included in the corresponding chapter (5) for the sake of readability.

Appendix C presents an alternative approach line that we have investigated during our work. It is based on using a single artificial neural network to solve preference learning problems. We provide here some preliminary test results about this idea.

Chapter 2

A gentle introduction to Machine Learning

Contents

2.1 Machine Learning 26
2.2 Types of ML protocols 27
2.2.1 Supervised learning 27
2.2.2 Unsupervised learning 28
2.2.3 Semi-supervised learning 31
2.2.4 Reinforcement learning 31
2.2.4.1 Active learning 33
2.2.5 Interoperability of ML techniques 33
2.2.5.1 Solving SL with a NSL framework 33
2.2.5.2 Solving RL with a SL framework 35
2.3 Supervised learning algorithms 36
2.3.1 Definition of "model" 37
2.3.2 Problem settings 37
2.3.2.1 Attribute selection 37
2.3.2.2 Object selection 38
2.3.2.3 Different types of goals 38
2.3.3 Definition of "good model" 39
2.3.4 Predictive accuracy 39
2.3.4.1 Estimation on the **LS** 41
2.3.4.2 Using a test set 41

2.3.4.3 M-fold cross validation 42
2.3.4.4 Avoiding over-fitting 42
2.3.5 Examples of SL algorithms 45
2.3.5.1 Multi layer perceptrons (MLPs) 45
2.3.5.2 K-nearest neighbors 46
2.3.5.3 Decision Trees 47
2.3.5.4 Support vector machines 50

This chapter provides the user with an intuitive introduction to the sub-domain of artificial intelligence we are working in, namely Machine Learning. The first section will formally define this domain. Differences between machine learning protocols will be explained in Section 2.2. Section 2.3 will define the term "model" in the context of supervised learning, its purpose and how to estimate its predictive accuracy, and present some state-of-the-art supervised learning algorithms.

2.1 Machine Learning

Machine Learning [Mit97] (also quoted to as "ML") algorithms are intended to give computer systems the ability to learn. But before formally defining ML, we need to define what is "learning".

All human beings learn every day, without even being aware of it. When we think about the term "learning", we think about school, and the vast amount of information that we had to somehow put into our memory. This is one aspect of learning, that is: store information. But this type of learning does not allow you to infer information from your knowledge. For instance, if I tell you that $f(2) = 8$, it does not help you to infer the value of $f(3)$. However, if you can learn a more general rule, you can then apply this rule to produce new knowledge. In our example, if I tell you also that $f(0) = 0$, that $f(1) = 1$, and that $f(4) = 64$, you might infer that for any v, $f(v) = v^3$, and you could then use this general rule to compute $f(v)$ for any given value of the variable v, and in particular find that $f(3) = 27$.

The machine learning field focuses on the second aspect of learning (i.e. learn a general rule). Indeed, any developer can easily write a software which only stores information, but inferring new knowledge from these informations is more challenging.

The process used to provide a computer with the ability to learn is inspired from the process in which humans do learn. They acquire new information from experiments and use this information to update their policy of actions so that their general behaviour is better according to a performance measure. This is consistent with the definition which Tom M. Mitchell provided in [Mit97]: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E".*

The use of ML algorithms in AI problems has two benefits. First, the application of this method does not require any prior knowledge of the problem. They can thus be applied to numerous problems without any further development (apart from formatting the data into the expected input). Secondly, if the software is to be used in a changing environment, it will be able to adapt its policy of actions accordingly.

2.2 Types of ML protocols

Several types of ML protocols have emerged. The diversity comes from the type of practical needs. We will further define supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning.

2.2.1 Supervised learning

In supervised learning (also quoted to as SL), input data are objects which have a attributes and a (potentially multi-dimensional) output of size p. Every example is labeled with the observed output value. Formally, every object is a pair $[x, y]$ with $x \in \mathbf{X}$, the input space, being an attribute vector and $y \in \mathbf{Y}$, the output space, is the desired output. For example, if all input and output variables are numerical, we have that

$$[x, y] \quad \in \quad \mathbb{R}^a \times \mathbb{R}^p. \tag{2.1}$$

These objects are drawn i.i.d. (independently and identically distributed) from an unknown probability distribution $P(x, y)$, which we are trying to learn, and are grouped into a learning set (quoted to as "**LS**") . Thus, if we are provided with $\#\mathbf{LS}$ objects, then $\mathbf{LS} \in (\mathbf{X} \times \mathbf{Y})^{\#\mathbf{LS}}$.

We are also given a loss function $\ell : \mathbf{Y} \times \mathbf{Y} \to \mathbb{R}^+$ which, $\forall y, y' \in \mathbf{Y}$, usually represents the distance between y and y'. Finally, we are provided with a hypothesis space H which contains functions $h : \mathbf{X} \to \mathbf{Y}$.

We can evaluate a function $h \in H$ by defining

$$Loss(h) \quad = \quad E_{P(x,y)}\{l(y, h(x))\}.$$

The aim is to find a function $h^* \in H$ such that

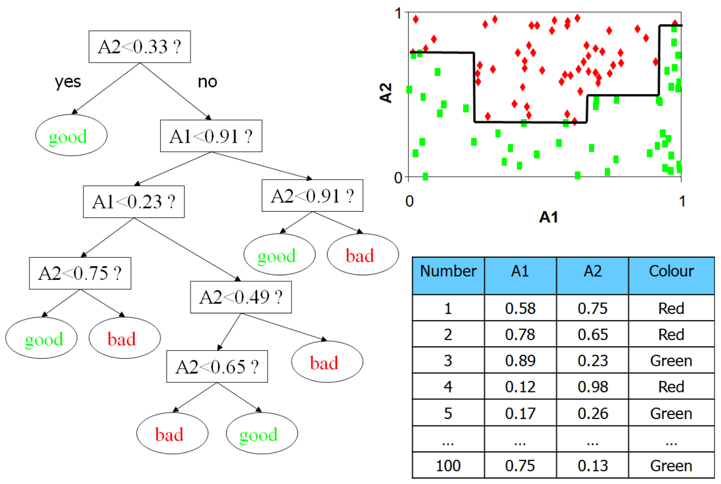
$$h^* \quad \in \quad \arg\min_{h \in H} Loss(h).$$

We can also differentiate two variants of supervised learning algorithms: batch and online. Batch mode SL algorithms simultaneously use all the objects of the database in order to choose the function h whereas, in online mode, every new observation helps to sequentially improve this function. Note that these modes are interchangeable as one could sequentially use each object even if the observation set will not further change or reversely, one could gather information from an online flow and only consider the learning step when a sufficiently large batch of observations has been collected.

An example of a supervised learning algorithm is the decision tree induction method [BFSO84]. Figure 2.1, illustrates a decision tree for a simple toy problem. Each internal node (including the root-node, generally depicted at the top of the tree) contains a test whose outcome determines which branch will be considered next in order to progress downwards in the tree. The traversal stops when a leaf node is reached at the bottom of the tree, and the prediction value $h(x)$ corresponds to the label of this leaf. In section 2.3.5.3 we will explain how decision trees are learnt from a learning sample.

2.2.2 Unsupervised learning

With unsupervised learning (or non-supervised learning; NSL), the input data (objects) are unlabeled. Instead of a $P(x, y)$ distribution, the data is drawn i.i.d. from a distribution $P(x)$. The goal is thus to extract information about $P(x)$ from the data.



Number	A1	A2	Colour
1	0.58	0.75	Red
2	0.78	0.65	Red
3	0.89	0.23	Green
4	0.12	0.98	Red
5	0.17	0.26	Green
...	...	...	...
100	0.75	0.13	Green

Figure 2.1: A decision tree is used in a top-down fashion to separate the input space with elementary attribute-wide tests.
(Figure reproduced from [WGIA]).

One technique for unsupervised learning is clustering, that is: separate data points into several clusters such that a new entry could be attributed to one of these clusters (ex: Figure 2.2). The clusters correspond to modes of the density $P(x)$. Several types of algorithms can be used, namely connectivity models (e.g. hierarchical clustering [HTF03]), centroid models (e.g. k-means [Mac67]), distribution models (e.g. expectation-maximization algorithm [DLR77]), density models (e.g. dbscan [EKJX96]) or subspace models (e.g. biclustering [CC00]), this list not being exhaustive.

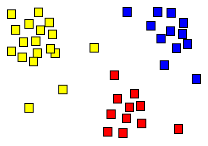
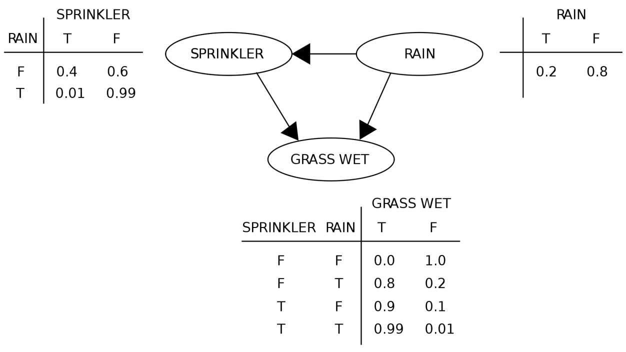


Figure 2.2: Clustering partitions the unlabeled data

Other unsupervised learning techniques include, but are not limited to:

- **Density estimation algorithms [Ros56].** These algorithms construct an estimate of the probability density function $P(x)$. The aim is to be able to compute a good estimate of the value of $P(x)$ for any (and possibly unseen) x.
- **Graphical models and relational learning.** In order to infer probabilistic and/or causal relations between features, one can use graphical models, such as Bayesian networks [Pea85] or Markov chains [Mar71]. Figure 2.3 shows an example of the former where the grass can be wet (G) if it is raining (R) or if the sprinkler is turned on (S). The state of the sprinkler also depends on the rain, that is, the probability to be on during rain is very low. The joint probability function is $P(G,S,R) = P(G|S,R)P(S|R)P(R)$. With this model, it is possible to answer questions such as "what is the probability that it is raining, given that the grass is wet", by computing $P(R = T|G = T)$.
- **Association rules.** From an unlabeled database, one can extract association rules [AIS93]. For instance, in a supermarket database, the rule $\{Sugar \Rightarrow Eggs\}$ may apply if most customers buy eggs every time that they buy Sugar.



	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99

RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Figure 2.3: Example of a bayesian network
Source : Wikipedia

As there is no associated label and no supervision in unsupervised learning, evaluating the reliability of the method can be very challenging, yet possible. In clustering, for instance, one would expect a high similarity between objects in the same cluster, and a low similarity between clusters, so it could be possible to evaluate the clustering by computing these two similarity levels.

2.2.3 Semi-supervised learning

Semi-supervised learning [BM98] is at the junction between supervised and unsupervised learning. Only a fraction of the objects available at the learning stage are labeled. A simplistic manner of dealing with such databases would be to remove unlabeled objects and apply supervised learning to the remaining objects. However, this would generally lead to a loss of accuracy, as unlabeled data still provide clustering information, and thus could help in labeling missing data. An illustration of this remark can be found on figure 2.4 which represents a semi-supervised learning problem and figure 2.5 which shows the difference between the application of supervised and semi-supervised learning to this problem.

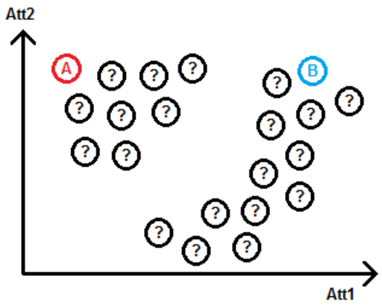


Figure 2.4: Two data points are labeled, the rest is unlabeled. This is a semi-supervised learning problem.

This illustration shows that semi-supervised learning can improve the modeled function.

2.2.4 Reinforcement learning

Reinforcement learning (RL) [Sut88] differs from (semi-)supervised learning in the sense that the correct label is not explicitly given to the learning system. Instead, it has to interact with its environment to collect information.

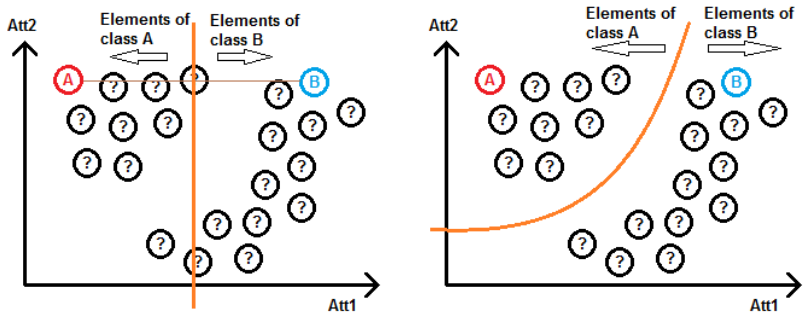


Figure 2.5: Left graph: the model (orange) is built with a supervised learning method. Right graph : the model is built with a semi-supervised learning method.

The system starts in the state s_0 belonging to the state space $\mathbf{S}$. If it is aware of its current environmental state, we speak about “full observability”, otherwise the system only receives an observation from its environment and we then speak about “partial observability”. In both cases, it works in discrete iterations. At any time i, it first receives an observation o_i which contains information about the current state s_i and it then it has to choose an action u from the set of possible actions $\mathbf{U}$. This action is then transmitted to the environment which changes the state of the system into s_{i+1} and also leads to the observation of a numerical reward $r_i \in \mathbb{R}$.

The aim of reinforcement learning is to develop algorithms which maximize the cumulative reward. For instance, putting money into a savings account will immediately reduce the total amount of money that you have, but possibly this investment will in the long term be more rewarded than not sparing money. A common RL technique to solve this optimization problem is Q-learning. This technique consists in learning a function $Q(s, u) \to \mathbb{R}$ which represents the expected utility of action u when the system is in the state s.

Another interesting aspect of this type of algorithms is the exploration-exploitation dilemma. To illustrate this concept, imagine that you are in a room with five dispensers in front of you and that you possess ten coins of one euro. You put one euro into a randomly chosen dispenser and receive two coins of one euro from that dispenser. You repeat this process three times and, each time, you collect two coins of one euro. At this point, you never tried the other dispensers, so you have to choose between using the same dispenser again, which seems to have a good reward or try another dispenser which may give you a better reward (or a worse one). So, the exploration-exploitation dilemma is the need to choose between exploiting the rewards using the optimal actions from the set of actions that were already performed or to explore the action space by choosing another action (for instance at random). In practice, this choice is performed by a random draw of a uniform variable $v \in [0, 1]$ and applying the exploitation if $v > \epsilon$ and exploring otherwise. The value of ϵ can be chosen arbitrarily or be adaptive using a heuristic (see [Tok10]).

2.2.4.1 Active learning

Although not directly related to reinforcement learning, active learning [FSST97] shares the same conceptual issue, namely that the label on a particular object must be queried explicitly. The aim of such algorithms is to request a particular observation in order to refine the model. If we are in the context of SL and trying to find a separation between two or more classes, the more relevant points are often located in the boundary between the classes.
Active learning can be used in two situations:

- The determination of a label is expensive (e.g. it requires human intervention or some costly experimentation protocol). In this case, particular attention should be given to the observations required to refine the model, and avoid performing useless experiments.
- The database contains a large number of labeled observations. Thus, asking for the value of particular points may help in “compacting” the database and more rapidly extracting the relevant information.

2.2.5 Interoperability of ML techniques

Due to the structure used to describe ML techniques, the reader might think that SL, NSL and RL are completely different and mutually exclusive. However, we will show, using two examples, that a problem in a ML domain can be solved using algorithms from another ML domain.

2.2.5.1 Solving SL with a NSL framework

Assume that we are provided with a SL database where each object corresponds to a person and possesses a single attribute named *glu* representing the level of plasma glucose concentration for this person. Each object is labeled with a binary variable *diabetes* representing the fact that this patient is suffering from diabetes or not. Typically, we want to model the correlation between the glucose level and the disease, which is effectively a SL problem.

We can compute the conditional density of glu given $diabetes$, which is shown on Figure 2.6. The black curve represents the density of $P(glu)$, which is the only information that we could retrieve if we were working in a NSL context. However, as objects are labeled, we can also compute the density estimation of $P(glu|diabetes = 1)$ and $P(glu|diabetes = 0)$.

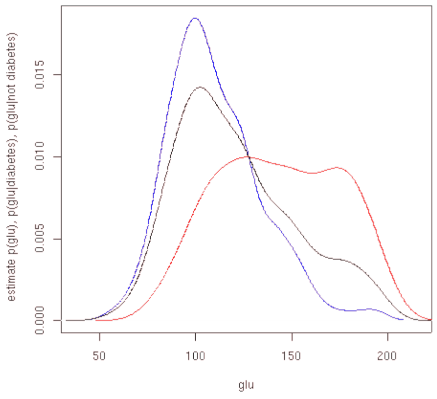


Figure 2.6: Estimated density of $P(glu|diabetes = 1)$ (red), $P(glu|diabetes = 0)$ (blue), and $P(glu)$ (black). Source : Wikipedia

The Bayes' law [Bay63] states that:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{2.2}$$

We can now compute the probability that the patient suffers from diabetes according to its plasma glucose concentration by:

$$\begin{aligned}
&P(diabetes = 1|glu)\\
&= \frac{P(glu|diabetes = 1)P(diabetes = 1)}{P(glu)},\\
&= \frac{P(glu|diabetes = 1)P(diabetes = 1)}{P(glu|diabetes = 1)P(diabetes = 1) + P(glu|diabetes = 0)P(diabetes = 0)}.
\end{aligned}$$

The corresponding probability curve is shown on Figure 2.7 and can be used for prediction, e.g. predict $diabetes = 1$ if $P(diabetes = 1|glu) > 50\%$.

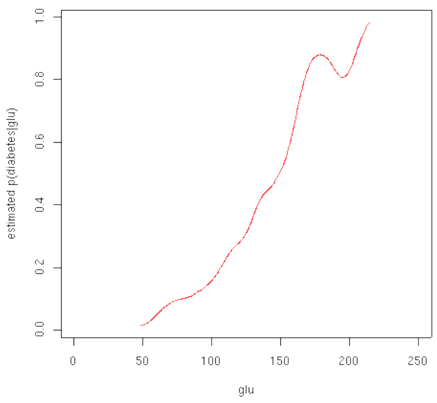


Figure 2.7: Estimated probability of $P(diabetes = 1|glu)$.
Source : Wikipedia

2.2.5.2 Solving RL with a SL framework

Fitted Q iteration [EGWL05] (or FQI) is a perfect example of the application of SL framework to a RL problem.

In order to compute the optimal Q function, it proceeds by iterations. As a reminder, the Q function is provided with two arguments: a state s and an action u, and outputs the corresponding cumulative reward r. FQI induces a $\hat{Q}$ function, which is guaranteed to converge under certain conditions, by iteratively solving consecutive SL problems (see algorithm 1).

Conceptually, the procedure is to move the horizon at each step, starting by taking the action which maximizes the immediate reward, then considering the impact from farther states. If the problem has a fixed horizon, the advantage is that the corresponding $\hat{Q}_i$ function will be specifically designed to maximize the reward over $i+1$ steps, which would not necessarily be true with a generic Q function which considers infinite horizons.

Algorithm 1 Fitted Q Iteration [EGWL05]

```
Input: a set of four-tuples $\mathcal{F} = \{(s_t^l, u_t^l, r_t^l, s_{t+1}^l), l = 1, \ldots, \#\mathcal{F}\}$ and a supervised learning (regression) algorithm
Output: a matrix $\mathbf{S} \times \mathbf{U}$ representing a $Q(s,u)$ function which provides, for a given state $s$, the action $u$ which maximizes the cumulated reward

$i \leftarrow 0$
$\hat{Q}_i \leftarrow 0$ everywhere on $\mathbf{S} \times \mathbf{U}$
repeat
  $i \leftarrow i + 1$
  Build the learning set $\mathbf{LS} = \{(x^l, y^l), l = 1, \ldots, \#\mathcal{F}\}$ based on $\hat{Q}_{i-1}$ and on $\mathcal{F}$
      $x^l = (s_t^l, u_t^l)$
      $y^l = r_t^l + \gamma \max_{u \in U} \hat{Q}_{i-1}(s_{t+1}^l, u)$
  Use the regression algorithm to induce from $\mathbf{LS}$ the function $\hat{Q}_i(s,u)$
until $i$ is sufficiently large
```

2.3 Supervised learning algorithms

As previously explained, Supervised Learning needs to build and store a representation of a function. One can use models, which are data structures (graphs, trees, kernels, ...) in order to represent a specific function. This section will be separated in five parts. Firstly, we will define the term “model”. Secondly, we will discuss about the parameterization of models. Then we will define what we will call a “good model”. Next, we will see how to estimate the efficiency of models and, finally, we will provide some examples of SL algorithms.

2.3.1 Definition of "model"

From the introductory sentence at the beginning of this section, we have now an intuition of what a model is. But we will now formally define it.

A model is typically a data structure which is built according to an algorithm $\mathcal{A}$ based on a set of observations (also called "learning set") **LS** and which produces, from an input vector x, an output y by using a prediction procedure $\mathcal{P}$.

Let us illustrate this definition with an example. A decision tree is a very well known model. As its name indicates, it is a tree and often a binary tree. The algorithm $\mathcal{A}$ used to build this structure could be the c4.5 algorithm [Qui92] and will be described in Algorithm 2, page 48. The prediction procedure $\mathcal{P}$ consists in taking the input vector x and, starting from the root node of the tree, to iteratively verify the conditions at the test node (using the input vector x) and to continue to the child node whose condition matches the values in x. The procedure $\mathcal{P}$ stops when a leaf node has been reached, and the value of the output y corresponds to the label of the leaf. An example of such tree was given in Figure 2.1.

2.3.2 Problem settings

Models can be parameterized. Some of them have more specific parameters, but common parameters can be applicable to all models.

2.3.2.1 Attribute selection

The model is not forced to be built according to each element of the attribute vector. From the previous decision tree example, we can see that if the input vector x is two slots wide (thus, the observation o_i has two associated attributes) and the tree only has one test node, then one of the attributes will not be used.

The aim of using only a part of the attributes is to improve the models by dropping attributes that do not seem to be correlated to the goal attribute. For instance, the resting metabolic rate (the amount of daily energy expended by humans at rest) depends notably on age and gender, but is not correlated to the identification number of the person. This information should thus not be taken into consideration when building a predictive model of the resting metabolic rate. Of course, in this example, the knowledge that we have about the problem helps us to filter irrelevant attributes, but this is not always the case. To determine which attributes are to be dropped, one can compute the variable importance (which can be done in several ways [SBZH07]) and build the model using only the j most important variables.

2.3.2.2 Object selection

In the same way that one can select some of the attributes from the attribute vector, one can also select some of the objects from the database in order to build the model. This can be done either in a passive mode (just take a subset of the objects arbitrarily) or in an active mode (ask for particular objects in the database in order to refine the model; we explained already that active learning algorithms could be used for this purpose).

One of the reasons why one would want to select only a part of the database for learning a model is to be able to perform the evaluation of this model on the not used part (see Section 2.3.4).

2.3.2.3 Different types of goals

There are basically two representations of the attribute we want to predict (also known as "goal") plus their derived representations.

The first one is for real values, we call it "regression". For instance, the speed of a car depends on the brand of the car, the RPM and the gear. In this case, models can predict values that were not explicitly given in the input data and will be as close as possible to the true value.

The second one is classification. Every object (or observation) belongs to one or several classes. For instance, you can divide your e-mails in two groups: Spam and the rest. Every e-mail is then being attributed a class based on its content or based on an expert knowledge, but cannot belong to both. You can also imagine that you have instead of two (making it a binary classification problem), three or more classes (we then talk about multi-class problems). For instance: Spam, Work, Other. But the model will never be able to predict a class that was not originally planned.

A generalization of this last representation is what we call multi-label classification. This is very similar to the regular classification, except that one object can be labeled with more than one class. Think of movies, they can often be marked with a single class, but there are also many of them which can belong to two styles, for instance drama and horror, or comedy and sci-fi. So you need multi-label classification if you want to handle this.

2.3.3 Definition of “good model”

When building a model, one wants it to be as good as possible. But this notion of “goodness” depends on what is the most important to achieve. Basically, we can consider three criteria: accuracy, complexity, interpretability.

In general, the accuracy is the most important criterion. It is a measure, usually given in the $[0, 1]$ interval, providing an estimation on the distance between $h(x)$ (the prediction) and y (the true value associated to x) over pairs (x, y) drawn i.i.d. from $P(x, y)$. The complexity is the effort needed to build and use the model. We can divide this effort into two parts: time complexity and space complexity. The space complexity is the amount of memory needed to build and store the model. The time complexity is the time needed to build the model and to perform a prediction. Finally, interpretability is described as the ease for human beings to understand the function represented by the model.

Currently, no machine learning algorithm can simultaneously and universally be champion on those three criteria, as more complex algorithms are generally more accurate but in the same time less interpretable.

2.3.4 Predictive accuracy

Accuracy can be formally defined in terms of average loss, which actually depends on the SL problem type. In classification, we can use a 0-1 loss while, in regression, the mean squared error would be more appropriate.

$$\begin{aligned} Loss &= \frac{\text{Number of well classified objects}}{\text{Total number of predictions}} && \text{In classification.} \\ Loss &= \sum_{i=1}^{I} (h(x_i) - y_i)^2 && \text{In regression.} \end{aligned}$$

The more a model reduces the loss, the more it is accurate.

In the particular case of binary classification, we can also speak in terms of precision and recall. Table 2.1 summarizes the statistical information which we can gain after a batch of predictions, where we can distinguish four kinds of predictions: (i) the object is actually labeled as "Positive" and is predicted likewise, (ii) the object is actually labeled as "Positive" and is predicted "Negative", (iii) the object is actually labeled as "Negative" and is predicted likewise, (iv) the object is actually labeled as "Negative" and is predicted "Positive".

		True label		
		Positive	Negative	
Prediction from the model	Positive	True Positive	False Positive	$\rightarrow$ Positive predicted value or Precision
	Negative	False Negative	True Negative	$\rightarrow$ Negative predicted value
		$\downarrow$ Sensitivity or recall	$\downarrow$ Specificity	$\searrow$ Accuracy

Table 2.1: Definition of accuracy, precision, recall, sensitivity and negative predictive value

For instance, precision is the probability that a predicted positive object will be effectively positive. On the dual, sensitivity is the probability that a positive object is effectively predicted as positive.

Let us use the following notation : TP for True Positive, FP for False Positive, TN for True Negative and FN for False Negative. We can then define each measure as

$$\begin{aligned}
\text{Precision} &= TP/(TP+FP) \\
\text{Negative predicted value} &= TN/(FP+TN) \\
\text{Sensitivity} &= TP/(TP+FN) \\
\text{Specificity} &= TN/(FP+TN) \\
\text{Accuracy} &= (TP+TN)/(TP+FP+TN+FN)
\end{aligned}$$

Accuracy of the model, even if not the priority in some cases, should at least be reasonably high. Indeed, a fast or readable model would be quite useless if the prediction is not reliable. Thus we need tools and techniques to estimate the reliability of the model and to ensure that the prediction is good enough. We can think of three ways of estimating the predictive accuracy of a model: estimation on the **LS**, using a test set and the m-fold cross validation procedure. We will discuss these three methods in the further subsections, then will conclude this section about a remark on the efficiency/complexity dilemma.

2.3.4.1 Estimation on the LS

A simple procedure for computing the accuracy of a model consists in using the produced model to perform a prediction on all the labeled objects in the database, even if these objects were used for model construction.

This is in general a very poor estimation on the model accuracy, since a model which would simply store the training information without inferring a general rule would be 100% accurate using this estimation procedure. However, without inference, this dummy model would be unable to perform a prediction on unseen data.

2.3.4.2 Using a test set

Estimating the predictive accuracy can be done by keeping apart some observations as we previously noticed the possibility. If the database contains many objects, then removing a small part of these objects from the **LS** will not drastically affect the model accuracy. Hence, after the model creation this part of this database can be used through the model in order to determine how well the model performs on unseen data (Figure 2.8).

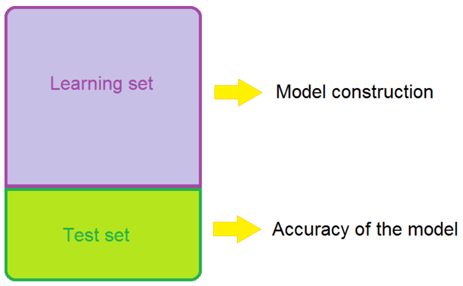


Figure 2.8: A part of the database is kept apart to compute an estimate of the accuracy of the model

2.3.4.3 M-fold cross validation

The problem of a simple test is that it is highly sensitive to the split (which can lead to optimistic or pessimistic evaluations). This effect is even worse when only a few data are available.

A good way of solving this problem is to use an m-fold cross validation procedure (where m is typically equal to 10). In this scenario, m models are built independently, each model using an independent partition[1] of size $\#\mathbf{LS}/m$ as a test set (as shown on figure 2.9) and being built using the remaining data. The error output is then averaged over the m models, which reduces the test variance.

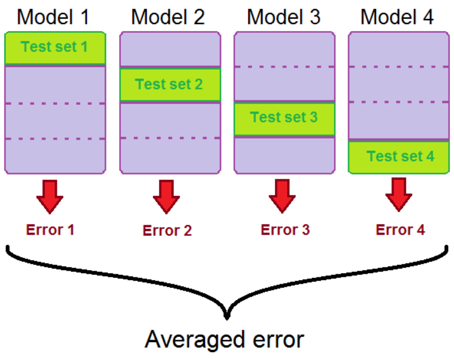


Figure 2.9: Building m models with separate test samples reduces the test variance. Here $m = 4$.

2.3.4.4 Avoiding over-fitting

One of the major issues in machine learning is the efficiency/complexity dilemma. One might think that the more complex your model is, the more accurate it will be, but that is not completely true. At some point, adding more complexity to the model will cause what we call "over-fitting", because the model will be too specific to the input data but will be more erroneous on predictions because it misses the general rule. This is even more true if the training sample is noisy.

[1]Each object in **LS** will be used in the test set of only one model.

Figure 2.10 shows an example where an extremely complex model would perfectly fit the noisy sample (in green) while a simpler model (in black) will be closer to the true function and will thus perform more accurate predictions than the complex model when considering unseen inputs.

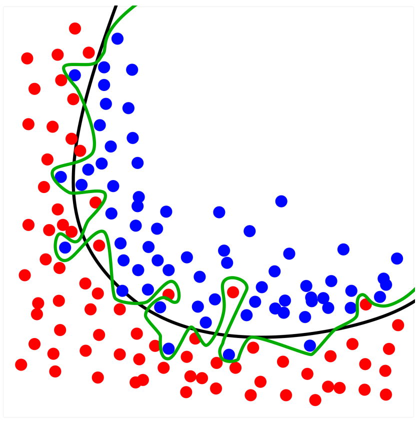


Figure 2.10: The green curve is perfectly separating the noisy sample, but the black curve is a simpler function and is closer to the true function, hence its predictions will be better for unseen points

As shown in figure 2.11, at some point, the red curve, representing the error on the test set (the part of the database used to test the model, also quoted **TS**), goes up, meaning that the error increases. This is because the model fits the learning set (**LS**) too closely, as we can see that the blue curve representing the error on the **LS** keeps going down.

This could also be expressed using the terms of "bias" and "variance". Using a SL algorithm, we could train several models using different training samples drawn from the same $P(x, y)$ distribution. The bias is the distance between an average prediction over all models and the true output. The variance is the average distance between each individual prediction. Reducing the variance generally induces an additional bias. Variance reduction techniques should have a significant impact on the variance while tempering the increase of the bias.

This concept is often illustrated using the dartboard example. Imagine that

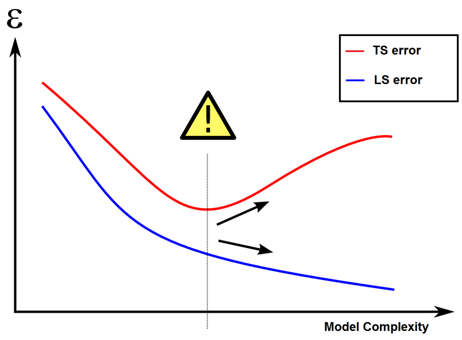


Figure 2.11: Fitting the learning sample too close will lead to over-fitting and produce a less accurate model

you are given the opportunity to launch four darts on a board. The bias would be the distance between the center of the board and the centroid of your launches. The variance would be the average distance between your individual launches. Figure 2.12 illustrates this concept.

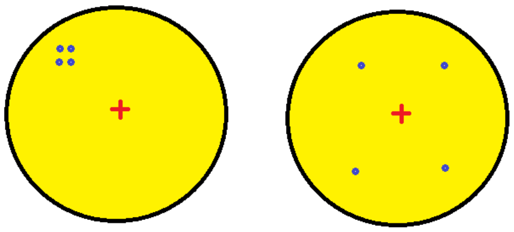


Figure 2.12: Left : High bias, low variance. Right : Low bias, high variance.

Some variance reduction techniques have been proposed. For instance, in the context of decision tree induction, it is possible to prune the tree either during the model construction (pre-pruning; e.g. by increasing the n_{min} parameter which controls the number of objects required to perform a node split) or after the model construction (post-pruning), hence artificially limiting its complexity so as to reduce the variance. Another example is regularization which, in linear regression models (which searches for a linear combination of features minimizing the quadratic error) adds an extra term in the minimization process in such a way to penalize models based on the norm of the weight of their parameters (features).

2.3.5 Examples of SL algorithms

In this section, we will present some examples of SL algorithms. They will be used as a reference in the following chapter.

2.3.5.1 Multi layer perceptrons (MLPs)

Multi layer perceptrons (quoted to as MLPs) are SL models mainly used when the desired approximated function is hard to guess, but when many observations can be obtained at low cost. These networks are built in perceptron layers, each node in a layer being connected to every node in the next and previous layer, and the inner layers are hidden as shown in Figure 2.13. To compute the output of a neuron, the weighted sum of the output of nodes from the previous layer is calculated, then applied to a (non linear for hidden layers) function, such as a sigmoid or an hyperbolic tangent.

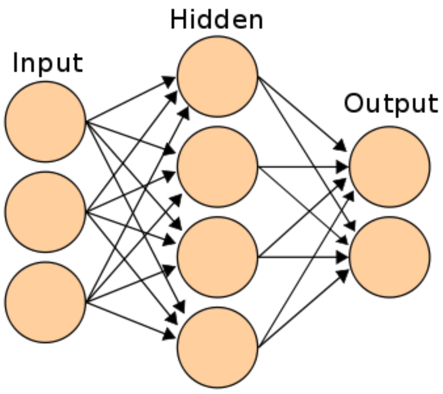


Figure 2.13: Each neuron in a layer feeds every neuron in the next layer

The accuracy of such a model strongly depends on the weights of the vertices and the training is usually performed in an iterative manner. Every weight is first set to a random value between $[-1, 1]$ and the model is used to predict the output of the objects in the learning sample. The predicted value is then compared to the true value and this gives an error δ which will be used to update the weights. Because neurons are arranged in layers, the error can be propagated iteratively on each layer using the back-propagation algorithm [Roj96] to compute the gradient of the error with respect to the weights and therefore update the weights with gradient descent. The prediction process will be referred to as the "forward pass" while the error propagation will be referred to as the "backward pass".

An iteration (or "epoch") is composed of a pass of forward-backward propagations over the whole **LS**, combined with a weight update reducing the cumulated error over the learning sample. Particular attention should be taken to choose a satisfactory end point as this method has a propensity to over-fit. At each iteration, the whole learning set must be considered, but this can be done either in an online or in a batch mode. However, D. Randall Wilson and T. R. Martinez empirically showed in [WM03] that the use of batch mode is counter-productive as, to quote the conclusion of their work, online mode is a better option *"due to the ability of on-line training to follow curves in the error surface throughout each epoch, which allows it to safely use a larger learning rate and thus converge with less iterations through the training data"*.

2.3.5.2 K-nearest neighbors

The k-nearest neighbor algorithm (also quoted to as kNN) is a SL method which takes as an hypothesis the fact that the class of an object is generally identical to the class of its neighbor(s) (see figure 2.14). To predict a label for a given object, the algorithm looks at the k nearest objects in the feature space. The label with a majority of objects in this set of k objects will be used as the prediction. If ties occur between the top labels, then one of them is randomly chosen.

The kNN algorithm does not build any model. Instead, the k nearest objects are scanned at each prediction, which can be very time-consuming.

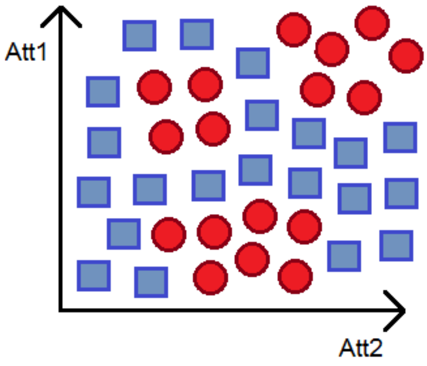


Figure 2.14: An object which is close or surrounded by identically labeled objects will be labeled according to the label of its neighbors

If the data labels are mixed across the feature space, then the quality of the prediction will depend on the noise of the learning sample. Choosing a small value of k will then generally lead to noise sensitive predictions, of lower accuracy than if a higher value of k is used. Overall, the optimal value of k is problem specific, and also depends on the size of the learning set. The value of k can be chosen by minimizing the cross-validation error.

In terms of computational complexity, the total number of objects of the learning sample has generally a stronger influence than the particular value of k that is used.

2.3.5.3 Decision Trees

A decision tree is a model which recursively separates the learning set into subsets according to a collection of comparative tests on an attribute value (see an example on figure 2.15). During the learning procedure, the recursion stops when a split criterion has been met (e.g. a node becomes a leaf if all his associated objects are labeled identically, if the number of associated objects is smaller than a given parameter n_{min} or if all features (attributes) are constant). The c4.5 algorithm[Qui92], which can be used to create a decision tree is given in Algorithm 2.

Algorithm 2 C4.5 [Qui92]

```
Input: A set of labeled objects LS, $n_{\min} \in \mathbb{N}$
Output: A tree model

Create a new node $\mathcal{M}$
if All objects in LS have the same label $y$ or the size of LS is smaller than $n_{\min}$ or all attributes are constant in LS then
  Turn the node $\mathcal{M}$ into a leaf, labeled by $y$ or by the most representative class in LS
else
  for All attributes $a$ do
    Compute the normalized information gain $IG$ from splitting on $a$
  end for
  Let $a\_best$ be the attribute maximizing $IG$
  Add a test in $\mathcal{M}$ with $a\_best$
  for All sublists $\mathbf{LS}_i$ obtained from the split do
    Recurse by computing $\mathcal{M}_i = C4.5(LS_i)$
    Attach $\mathcal{M}_i$ as children of $\mathcal{M}$
  end for
end if
return $\mathcal{M}$
```

To choose the best couple attribute/threshold in a test node, each of the possible couples is considered and the information gain is computed. This computation is based on Shannon's entropy [Sha48]. Class entropy in a subset of objects X is computed as:

$$H_C(X) = -\sum_{i=1}^{m} P(c_i|X) \log_2 P(c_i|X), \tag{2.3}$$

where X is a set of objects, c_i denotes one of the output classes of these objects and $P(c_i|X)$ is the relative frequency of objects of class c_i observed in the set X. The information gain is then computed as:

$$\text{Information gain} = H_C(X) - \left(\frac{\#X_1 H_C(X_1) + \#X_2 H_C(X_2)}{\#X}\right), \tag{2.4}$$

where X_1 and X_2 are the mutually exclusive subsets of X obtained after the

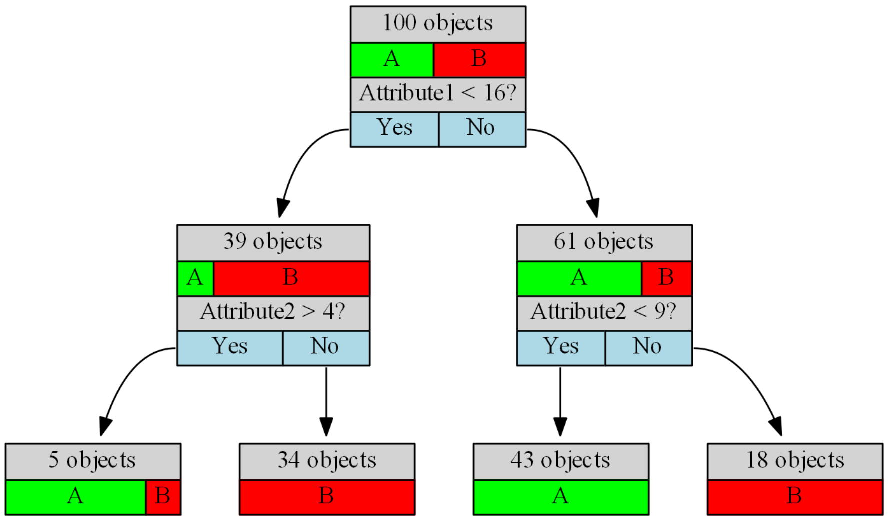


Figure 2.15: A decision tree (here with $n_{\text{min}} = 10$) separates the feature space in a linear fashion to purify each cluster as much as possible

node split, and where $\#X$ is the number of objects in X. Let us illustrate this concept with an example. Consider that X contains 8 objects. 4 of these objects are labeled with class A and the others with class B. Then

$$\begin{aligned} H_C(X) &= -(0.5\log_2 0.5 + 0.5\log_2 0.5), \\ &= -\log_2 0.5, \\ &= \log_2 2, \\ &= 1. \end{aligned}$$

The entropy is maximum. If we randomly picked an element from X, it would be impossible to predict (or at least, give a statistical preference to) the associated class.

Let us now consider two extreme node splits. The first one separates perfectly X into two sets containing 4 objects labeled A and 4 objects labeled B respectively. The second one creates two sets, each one containing two objects labeled A and two objects labeled B. The information gain for those splits are then:

$$\begin{aligned}
\text{IG for split1} &= 1-(4(-(1\log_2 1+0\log_2 0)) \\
&\quad +4(-(0\log_2 0+1\log_2 1)))/8, \\
&= 1-(0+0)/8, \\
&= 1;
\end{aligned}$$

$$\begin{aligned}
\text{IG for split2} &= 1-(4(-(0.5\log_2 0.5+0.5\log_2 0.5)), \\
&\quad +4(-(0.5\log_2 0.5+0.5\log_2 0.5)))/8, \\
&= 1-(4+4)/8, \\
&= 0.
\end{aligned}$$

Split1 has a gain of 1, meaning that it is now possible to predict at 100% which label will be associated to any object randomly picked in one of the subsets (given that we know in which set the selection occurred). Conversely, Split2 has a gain of 0, meaning that even if we knew in which subset the random selection would occur, we would still be unable to predict the associated label. In this case, Split1 is better than Split2 as the information gain for the former is greater than the information gain for the latter.

2.3.5.4 Support vector machines

The Support vector machine (SVM) algorithm[CV95] was designed by Cortes and Vapnik. This algorithm aim at finding a linear hyperplane which separates the input data into two classes.

The whole learning set is not necessarily relevant to build the model, and only a few data points (also called “support vectors”) are being used. These support vectors are determined by selecting the points which maximize the separating margin (i.e. the distance of the hyperplane to the closest points). Within this margin, an infinity of hyperplanes can be defined (see Figure 2.16, (a)). The SVM algorithm then selects the hyperplane which minimizes the distance to each support vector (the closest points to the plane) as depicted on Figure 2.16, (b).

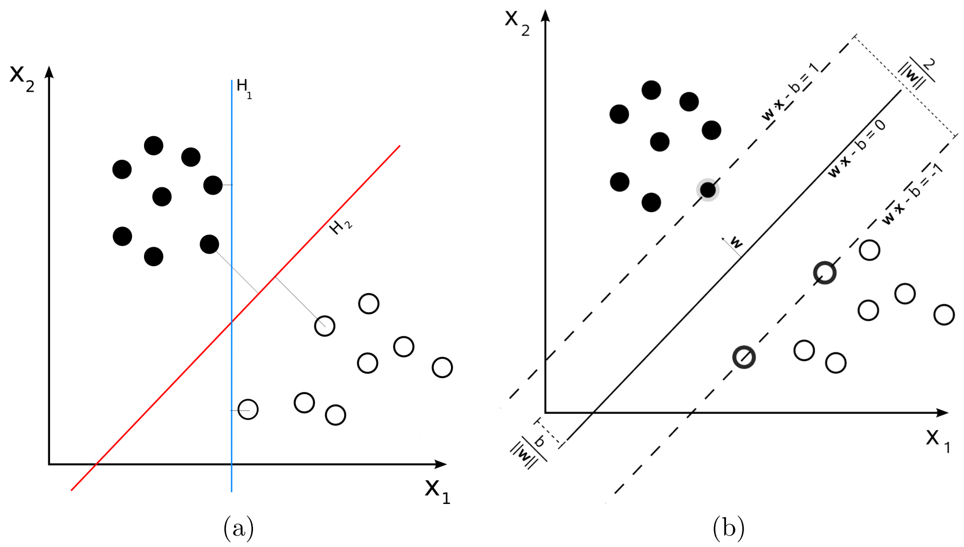


Figure 2.16: The is an infinity of separating hyperplanes (a), but SVM algorithms seek the hyperplane which maximizes the margin (b). Source : Wikipedia

Formally, the SVM algorithm solves an optimization problem which can be expressed by the following equations:

$$h(x) = w.x + b \tag{2.5}$$

$$w.x + b \geq 1 \qquad \text{(for each positive example)} \tag{2.6}$$

$$w.x + b = 0 \qquad \text{(for the separating hyperplane)} \tag{2.7}$$

$$w.x + b < 1 \qquad \text{(for each negative example)} \tag{2.8}$$

$$d(x) = \frac{|w.x + b|}{||w||} \qquad (d(x) \text{ is the distance of } x \text{ to the hyperplane}) \tag{2.9}$$

$$\tag{2.10}$$

We thus need to find w and b such that each example is correctly classified using the $h(x)$ function and such that the distance of each point to the hyperplane is maximal (which is equivalent to minimizing $||w||$).

Solving this problem directly is quite challenging. Using Lagrangian multipliers, we can transpose this problem into its dual form, i.e. maximizing

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i \alpha_i y_j \alpha_j x_i^T x_j \tag{2.11}$$

subject to

$$\forall i \alpha_i \geq 0, \tag{2.12}$$

$$\sum_{i=1}^{n} y_i \alpha_i = 0 \tag{2.13}$$

where y_i (y_j) is the label of object o^i (respectively o^j) which equals 1 for a positive example and -1 for a negative example. One of the most common SVM solvers is the Sequential Minimal Optimization (SMO) designed by John Platt [Pla98].

Using this dual form, the separating hyperplane w which maximizes the margin can we rewritten as a linear combination of the support vectors, i.e. $w = \sum_{i=0}^{\#SV}(\alpha_i x_i)$, where w is the separating hyperplane maximizing the margin (represented by a vector), x_i is the i^{th} support vector, $\#SV$ is the number of support vectors and α_i is the weight associated to x_i.

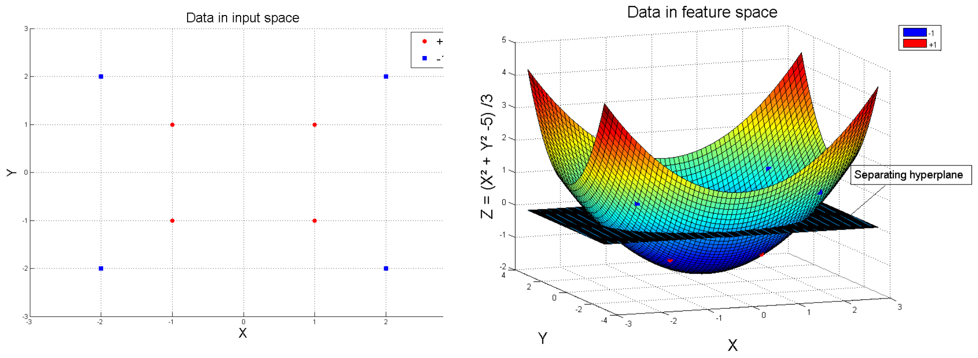
The prediction can be computed as $f(x) = \sigma(\sum_{i=0}^{\#SV}(\alpha_i x_i^T x))$, where σ is the sign function and x is a point in the space.

Many supervised learning algorithms make the assumption that a classification problem can be easily solved by using the correct set of features. Of course, finding this set is not always trivial as, for instance, the discriminant feature may not have been directly measured. One can assess this problem by adding dimensions into the input space such that the problem becomes linearly separable in the new feature space. To transform the data in input space to data in feature space, one has to use a ϕ function which is able to change the dimensionality of the problem. Note that ϕ can also be extremely simple (e.g. the identity function for problems which are already linearly separable in the input space).

This dimensionality extension can be applied to SVMs. Figure 2.17 illustrates an example where data in the input space are not linearly separable, while a linear hyperplane can be found in the new feature space, adding the dimension z and defining ϕ as

$$\phi(x,y) \quad = \quad (x, y, x^2 + y^2 - 5)/3).$$

Note that, in the prediction phase, ϕ must be applied both on the support vectors and on the data point for which we want a prediction.



(a) (b)

Figure 2.17: Data is not linearly separable in the original input space (a) but well in the extended feature space (b)

The problem of using the ϕ function is that the computations have to be performed in an higher dimensional space, which drastically increases the time complexity and prevents the use of an infinite number of dimensions. Moreover, the appropriate definition of ϕ is crucial to maintain an acceptable accuracy. To solve this issue, one can use the so-called "kernel trick" which consists in using a kernel $K(x_i, x)$ instead of $\phi(x_i)^T\phi(x)$ during the prediction and the learning phase. A kernel can potentially represent any ϕ function while the computations are still performed in the original input space. In addition, some kernels have been studied in the literature (e.g. polynomial, gaussian, sigmoid, etc) and provide an acceptable reliability for many problems.

Chapter 3

Preference Learning

Contents

3.1 Preference representations 57

3.1.1 Utility function 57

3.1.2 Pairwise information 58

3.1.3 Orderings 58

3.1.4 Interchangeability 58

3.1.4.1 Utility to pairwise 58

3.1.4.2 Utility to orderings 59

3.1.4.3 Pairwise to utility 59

3.1.4.4 Pairwise to orderings 59

3.1.4.5 Orderings to utility 60

3.1.4.6 Orderings to pairwise 60

3.2 Learning to rank 60

3.2.1 Label ranking 60

3.2.1.1 Ranking by Pairwise Comparison 61

3.2.1.2 Constraint classification 64

3.2.1.3 Solving label ranking with MLPs 67

3.2.1.4 Solving label ranking with decision trees . 67

3.2.2 Object ranking 68

3.2.2.1 Solving object ranking using SVM 68

3.2.3 Instance ranking 69

3.2.3.1 Ordinal Regression by Extended Binary Classification 70

3.3 Combining (partial) orderings 71
3.3.1 Difficulty of combining orderings 72
3.3.2 Minimizing pairwise disagreements 72
3.3.3 FAS-PIVOT 73
3.3.4 PTAS 74
3.3.5 Mallows model 75
3.3.5.1 Mallows model over complete rankings . . 76
3.3.5.2 Probability to observe an incomplete ranking 77
3.3.5.3 Maximum likelihood estimation of the model parameters 77
3.3.5.4 Approximate solution of the optimal centroid problem 78
3.3.5.5 Handling partial rankings 79
3.4 Similar domains 80
3.4.1 Multi-label classification (MLC) 80
3.4.1.1 Solving MLC with MLP's 80
3.4.1.2 Solving MLC with SVM 81
3.4.1.3 (Probabilistic) Chained Classification . . . 82
3.4.2 Multi-label ranking (MLR) 86
3.4.2.1 Solving MLR 86
3.5 Existing reduction techniques 86
3.5.1 QWeighted algorithm 87
3.5.2 QWeighted for multi-label classification 87
3.6 Summary and outlook 88

This chapter is dedicated to preference learning and the methods from this domain which are published in the scientific literature.

It is divided into five sections. In the first section, we will discuss about the possible representations of preference data. Then, in the second section, we will describe the three types of ranking problems, namely label, object and instance ranking. For each of these problem types, we will provide at least one algorithmic protocol to solve it. The third section will concern a fundamental problem in preference learning, which concerns the measurement of distances among orderings, the determination of a “centroid” ordering which is on the average closest to a set of given orderings, and measuring the quality of this centroid approximation. In section four, we will talk about other fields which can not be included *stricto sensu* in the field of **preference** learning, but which are very closely related and whose problems could be addressed using a preference learning methodology, or conversely. In section five, we describe published algorithms which tackle the problem of reducing the number of comparisons in a RPC framework by predicting only the top elements of the ranking. In the last section, we will state in this light our contribution to the domain.

Algorithms and methods depicted in this chapter were mainly found in [FH10] and [GV10].

3.1 Preference representations

In ranking problems, one has to represent the preference relation, either over a collection of objects (in object ranking problems) or over a collection of class labels (in label ranking problems).

There are several ways of representing preference information: utility function, pairwise information and orderings. Each of these can be complete or partial (some information is missing). These representations may be used in order to represent information in training datasets, as well as to represent output information of prediction models.

We will briefly discuss each of these representations as well as how one representation could be transformed into another.

We will use the term "item" to denote either a label or an object.

3.1.1 Utility function

In this representation, each item receives a score, either numerical (1,2,3,...) or symbolic (++,+,-,–), The higher the score, the most preferred this item is. Using this representation, it might be possible to have items having the same preference. A completely specified utility function thus induces a total, but not necessarily strict, order relation over items.

Preference information using a utility function is, for instance, expressed by $\{A = 3, B = 2, C = 1, D = 0\}$.

3.1.2 Pairwise information

Preference data can also be represented in the form of pairwise preferences. One can obtain this kind of information by asking what is the preference between two items. Assuming that a pairwise comparison indicates for two items whether one is preferred over the other (and vice-versa), then $N(N-1)/2$ such pairwise comparisons may be performed, where N is the total number of items.

In general, a set of pairwise comparisons does not necessarily satisfy the transitivity property of an order relation. If the relation is transitive, then all pairs can be reconstructed from $O(N)$ comparison in the best case, and $O(N \log N)$ comparisons on average (and worst case). Cycles can occur with intransitive relations. For instance, if we consider the binary preference relation $\succ$, we could have $A \succ B$, $B \succ C$ and yet $C \succ A$. In this case, we would be unable to determine which one is preferred over A, B and C.

3.1.3 Orderings

Finally, the preference information can also be represented by a (total or partial) ordering of items. For instance, considering the preference relation $\succ$, we could have $A \succ B \succ C$.

The main difference between this latter representation and utility function based representation, is that in the case of a utility function one also represents a quantitative strength of the preference relation.

In the sequel, we will use indifferently the terms "permutation" and "ordering". A total ordering of set is a permutation of this set, while a partial ordering of a set is a permutation of a subset of this set.

3.1.4 Interchangeability

Some representations cannot be transformed into others without loss of information under certain circumstances. We will consider all six possibilities.

3.1.4.1 Utility to pairwise

This transformation can be performed by considering, for each pair, the score of each element then attributing the preference to the element with the greater score. However, since ties can occur using utility functions, some comparisons can not be constructed using a 0-1 preference. Moreover, a pairwise comparison will only assess that an element is preferred over another, without quantitative information. For instance, the utility function $\{A = 5, B = 0\}$ will be transformed into the same pairwise information than $\{A = 3, B = 2\}$. The reverse transform is thus not guaranteed to provide the same utility function.

Consider, for instance, a utility function of $\{A = 3, B = 2, C = 2, D = 0\}$. From this data, we can induce that $A \succ B$, $A \succ C$, $A \succ D$, $B \succ D$ and $C \succ D$. However, if we want to perform the reverse transformation, by attributing a vote for each preferred element in the pairwise comparisons, we will obtain $\{A = 3, B = 1, C = 1, D = 0\}$ which, although similar, is different from the original utility values.

3.1.4.2 Utility to orderings

Obtaining an ordering from a utility function can be performed by sorting the elements according to their utility value. Again, ties have to be broken, and in this case the inverse transform will not provide the same utility values. For instance, $\{A = 3, B = 3, C = 1\}$ can be transformed into either $A \succ B \succ C$ or $B \succ A \succ C$, but the reverse transform of the former will provide $\{A = 3, B = 2, C = 1\}$ while the latter will provide $\{A = 2, B = 3, C = 1\}$. On the other hand, preference intensity will be discarded, as $\{A = 5, B = 1, C = 0\}$ will be transformed into the same ordering as $\{A = 3, B = 2, C = 1\}$.

3.1.4.3 Pairwise to utility

We can transform pairwise information into a utility function by attributing a vote to the preferred element for each comparison. If the preference relation is not transitive, we will loose information in this process. For instance, if we have $A \succ B$, $B \succ C$ and $C \succ A$, then the utility values will be $\{A = 1, B = 1, C = 1\}$ and no pairwise comparison can be extracted from the reverse transform.

3.1.4.4 Pairwise to orderings

Typically, this transform is performed by changing pairwise information into utility values, then changing this representation to an ordering. We have previously shown that both transforms can lead to an information loss. Moreover, it might be impossible to express pairwise information in the form of an ordering. For instance, no ordering could express only $A \succ B$, $C \succ D$ without imposing an implicit order between A and C.

3.1.4.5 Orderings to utility

This transform can be performed by attributing a score to each element between 0 and $N-1$ according to its position in the ordering. A full ordering can be transformed into utility scores without loss and recovered in the reverse transform. In the case of partial rankings, we would have ties for unrepresented elements (the utility value would be equal to 0).

3.1.4.6 Orderings to pairwise

One can obtain a pairwise comparison by constructing it according to the given ordering. These comparisons can be used to reconstruct the original ordering, even if this ordering is partial, assuming that the reverse transform is performed directly and not by using a utility function.

3.2 Learning to rank

We will now define the three types of ranking problems, namely "label", "object" and "instance" ranking problems, and we will provide at least one solution protocol for each type.

3.2.1 Label ranking

In label ranking, one is provided with a dataset where each object is labeled by a possibly partial ordering of a set of labels and a vector of attribute values. The goal is to build a model which is able to provide a total ordering of the set of labels for any new object as a function of its attribute values and in such a way that each predicted ordering is close to the corresponding labeled ordering in the training sample.

Formally, given a training set $\{(x^i, \pi^i) | i \in \{1, 2, \dots, n\}\}$ of input-output pairs, where each input x^i is a vector (of attributes) belonging to the input space $\mathbf{X}$, and where each output π^i is a permutation of a subset of a given set of labels $\mathbf{Y} = \{y_j | j \in \{1, 2, \dots, N\}\}$, the aim is to build a model $\hat{\pi}(x)$ representing a ranking function, i.e. a function which maps any $x \in \mathbf{X}$ to a permutation $\hat{\pi}(x)$ of the full set of labels $\mathbf{Y}$, in such a way that $\hat{\pi}(x^i)$ is close to π^i, for any $i \in \{1, 2, \dots, n\}$, and so that the model works well for any unseen $x \in \mathbf{X}$. We postpone to chapter 4 the discussion of loss functions used in the context of label ranking.

In the sequel we will use the following notations:

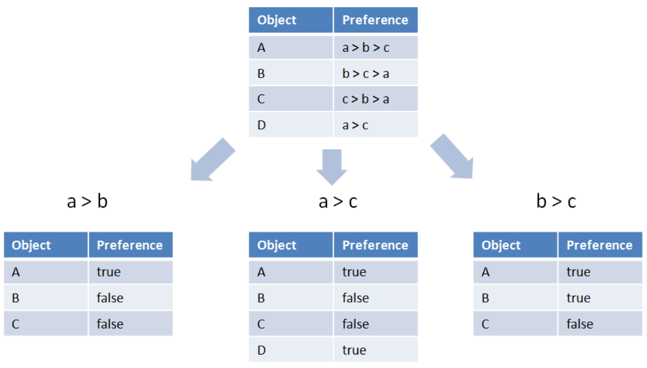
- $\Pi(\mathbf{Y})$ is the set of all possible permutations of $\mathbf{Y}$.
  Hence $\hat{\pi}(x) \in \Pi(\mathbf{Y}), \forall x \in \mathbf{X}$.
- $\overline{\Pi}(\mathbf{Y})$ is the set of all possible permutations of all *subsets* of $\mathbf{Y}$.
  Hence $\pi^i \in \overline{\Pi}(\mathbf{Y}), \forall i = 1, \ldots, n$, and $\overline{\Pi}(\mathbf{Y}) \supset \Pi(\mathbf{Y})$.
- An element of $\overline{\Pi}(\mathbf{Y})$ is said to be trivial if it is empty or contains only one label. We always suppose that there are non-trivial rankings (i.e. $N \geq 2$).
- For some $\pi \in \overline{\Pi}(\mathbf{Y})$, we will say that $(y_k, y_l) \in \pi$, if label y_k appears before y_l in π, and we will say that $y_k \in \pi$ if label y_k appears in π.
- More generally, we will say that $\pi' \subset \pi$ if the labels of π' all appear in π in a compatible ordering.
  Hence, in the non-trivial case, $[\pi' \in \pi] \equiv [(y_k, y_l) \in \pi' \Rightarrow (y_k, y_l) \in \pi]$.

3.2.1.1 Ranking by Pairwise Comparison

Ranking by Pairwise Comparison [HF04], also quoted to as RPC, is a method developed by Johannes Fürnkranz and Eyke Hüllermeier in 2004. RPC works in three steps: decomposition, modeling and aggregation.

The decomposition phase consists in transforming the original training set into several auxiliary training sets, one for each possible pairwise label comparison: for a given pair of labels (y_k, y_l) $(k = 1, \ldots, n; l = k+1, \ldots, n)$, the corresponding auxiliary training set is composed of those observations $i \in \{1, \ldots n\}$ for which both y_k and y_l appear in the permutation π^i, and those objects are built as a pair $(x^i, y^i_{k,l})$, where $y^i_{k,l}$ is a boolean variable, true if $(y_k, y_l) \in \pi^i$, and false otherwise (see Figure 3.1 and Algorithm 3).

The second phase is model building. For each auxiliary training sample, a pairwise comparator is trained with a supervised learning algorithm, for instance: neural networks, support vector machines, ensemble of trees or single decision trees (this list is not exhaustive) as depicted in Figure 3.2. At this stage, the learning process is completed.



Object	Preference
A	a > b > c
B	b > c > a
C	c > b > a
D	a > c

Object	Preference
A	true
B	false
C	false

Object	Preference
A	true
B	false
C	false
D	true

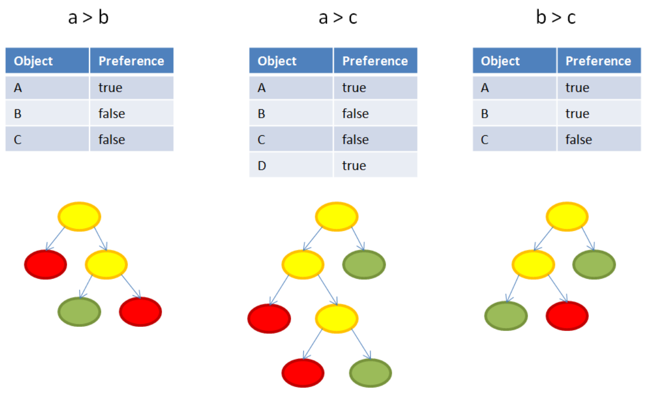
Object	Preference
A	true
B	true
C	false

Figure 3.1: RPC first decomposes the given partial label orderings into pairwise comparisons and creates the corresponding auxiliary training sets

Algorithm 3 Decomposition phase

```
Input: a training set LS
Output: $N(N-1)/2$ auxiliary training sets
for all Possible pairs of labels $(y_k, y_l) | k = \{1, \dots, N\}, l = \{k+1, \dots, N\}$ do
    $AuxLS[(y_k, y_l)] \leftarrow \emptyset$
    for all Objects $o^i = (x^i, \pi^i) \in$ LS do
        if $(y_k, y_l) \in \pi^i$ then
            Add $< x^i, true >$ in $AuxLS[(y_k, y_l)]$
        else if $(y_l, y_k) \in \pi^i$ then
            Add $< x^i, false >$ in $AuxLS[(y_k, y_l)]$
        end if
    end for
end for
```

The third step occurs during prediction. When a new object needs to be evaluated, the prediction of all pairwise comparisons are computed by using the corresponding comparators. In this process, each comparator gives one vote to the class label that it prefers among those two that it was built for. Sorting the class labels according to their cumulated votes (or utility) will produce the predicted ranking as output (Figure 3.3 and Algorithm 4). Note that, in this output, every class label will be represented, even if ties might have to be broken arbitrarily.

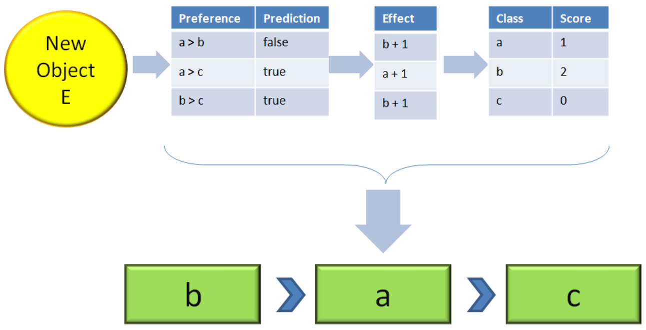


Object	Preference
A	true
B	false
C	false

Object	Preference
A	true
B	false
C	false
D	true

Object	Preference
A	true
B	true
C	false

Figure 3.2: Pairwise comparators are learned using supervised learning on the corresponding auxiliary training sets



Preference	Prediction
a > b	false
a > c	true
b > c	true

Effect
b + 1
a + 1
b + 1

Class	Score
a	1
b	2
c	0

Figure 3.3: The set of binary comparators is used to produce a predicted ordering.

In the case where the predicted comparisons are transitive, each label receives a distinct number of votes between 0 and $N - 1$ (N being the total number of labels) in such a way that the corresponding ordering is a strict total order.

Even if this method has become state-of-the art in the field of label ranking, it has its bottleneck, as $\frac{N(N-1)}{2}$ different pairs need to be evaluated and merged. For a small problem, this can be easily handled, but when dealing with a huge

Algorithm 4 Prediction phase

```
**Input:** an input attribute vector $x$, and $N(N-1)/2$ pairwise comparators
**Output:** $\hat{\pi}(x)$
$PairwiseOutputs[N(N-1)/2] \leftarrow$ Apply the object's input attributes $x$ to every pairwise comparator
$Scores[N] \leftarrow$ null vector of size $N$
**for all** Pairwise outputs $q_{kl}(x) \in PairwiseOutputs[N(N-1)/2]$ corresponding to the label pair $(y_k, y_l)$ **do**
    $Scores[y_k] \leftarrow Scores[y_k] + q_{kl}(x)$
    $Scores[y_l] \leftarrow Scores[y_l] + 1 - q_{kl}(x)$
**end for**
$Sorted[N] \leftarrow$ The set of $N$ labels
Sort $Sorted[N]$ according to $Scores[N]$ in descending order
**return** $Sorted[N]$
```

number of labels (say $N = 10,000$ or more), the method collapses caused by the computational complexity and/or memory usage, preventing the researchers to use it in many problems in active research fields.

3.2.1.2 Constraint classification

Sariel Har-Peled et al. modified the SVM protocol to develop the Constraint Classification [HpRZ02] algorithm. We describe its application to label ranking.

Let us assume that the inputs x^i of our n training examples are vectors of $\mathbb{R}^d$, and suppose that for each possible label $k \in \{1, \ldots, N\}$ we dispose of a vector $v_k \in \mathbb{R}^d$. Using these vectors, we can infer a (possibly non-strict) ordering of the N labels based on an input vector x by the following operation

$$\hat{\pi}(x) \quad = \quad \text{argsort}\{v_k^T x\}_{k=1}^N \tag{3.1}$$

where "argsort" is a function which ranks the indexes of an array according to the cell content of this array (in descending order). In other words, we compute N scores as scalar products between the vectors v_k and the input x, and then sort the resulting numbers, in order to compute a permutation over our label set.

Given a training set $\{(x^i, \pi^i)\}_{i=1}^n$ where the outputs are permutations of subsets of $\mathbf{Y}$, the goal is hence to determine a set of vectors $\{v_k : k \in \{1, \ldots, N\}\}$

such that the resulting predictions $\hat{\pi}(x^i)$ are compatible with the corresponding targets π^i. The authors of [HpRZ02] show how this problem may be solved by constructing a separating hyperplane (e.g. by using a support vector machine approach) in $\mathbb{R}^{Nd}$, based on a single auxiliary training set of binary labelled examples derived from $\{(x^i, \pi^i)\}_{i=1}^n$.

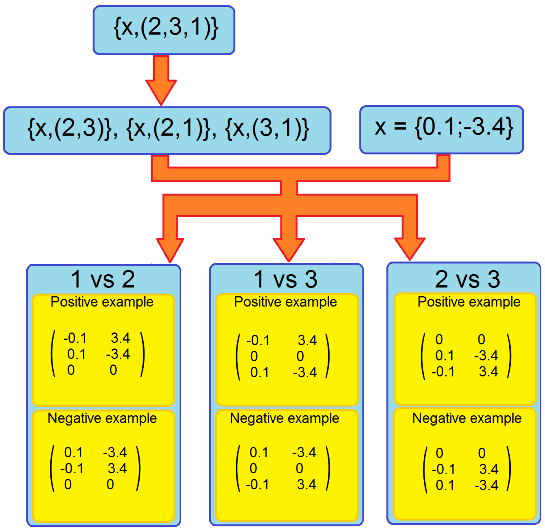


Figure 3.4: Ranking information π^i is separated into $|\pi^i|(|\pi^i| - 1)/2$ constraints and two examples are constructed per constraint and per object

Figure 3.4 illustrates the procedure used to derive from a given input-output pair (x^i, π^i), a set of positive and negative auxiliary examples, a positive and a negative one for each pairwise comparison that may be extracted from π^i: $\forall (y_k, y_l) \in \pi^i$ we first construct a vector

$$z^i = (\mathbf{0}^{(k-1)d}, x^i, \mathbf{0}^{(N-k)d}) - (\mathbf{0}^{(l-1)d}, x^i, \mathbf{0}^{(N-l)d}) \in \mathbb{R}^{Nd},$$

composed of zeroes except for the chunks of length d corresponding to labels k and l, which respectively receive a positive and a negative copy of x^i, and from which we derive a positive example $(z^i, 1)$ and a negative example $(-z^i, -1)$

(notice that in Figure 3.4, the subvectors of dimension d have been stacked vertically, for the sake of legibility). The main idea behind this representation is that, for each $(y_k, y_l) \in \pi^i$, we wish that

$$\begin{aligned} v_k^T x^i > v_l^T x^i \quad &\Rightarrow \quad v_k^T x^i + v_l^T(-x^i) > 0 \text{ (positive example)}, \qquad (3.2)\\ &\Rightarrow \quad v_k^T(-x^i) + v_l^T x^i < 0 \text{ (negative example)}. \qquad (3.3) \end{aligned}$$

The complete training set is obtained by merging all the subsets obtained $\forall i \in \{1, \ldots, n\}$.

Assuming that the n original examples are all completely ranked over $\mathbf{Y}$, the auxiliary dataset constructed will be of cardinality $n' = 2nN(N-1)/2$, and each example will be of dimensionality $d' = Nd$. The z vectors are however very sparse, since they will contain at most $2d$ non-zero entries. These observations suggest to use an on-line algorithm rather than a batch-mode approach in order to determine a separating hyperplane, such as Algorithm 5, adapted from [HpRZ02] to our notations.

Algorithm 5 Constraint classification

```
Input: A training sample $S = \{(x^i, \pi^i)\}_{i=1}^n$, where $S \in (\mathbb{R}^d \times \overline{\Pi}(\mathbf{Y}))^n$
Output: A ranking function $\hat{\pi}(x) : \mathbf{X} \mapsto \Pi(\mathbf{Y})$
for $k = 1, \ldots, N$ do
  Initialise $v_k \in \mathbb{R}^d$
end for
repeat
  for $i = 1, \ldots, n$ do
    for all $(y_k, y_l) \in \pi^i$ do
      if $v_k^T x^i < v_l^T x^i$ then
        promote($v_k$) (e.g. $v_k = v_k + x^i$)
        demote($v_l$) (e.g. $v_l = v_l - x^i$)
      end if
    end for
  end for
until convergence
return $\hat{\pi}(x) = \text{argsort}\{v_k^T x\}_{k=1}^N$.
```

3.2.1.3 Solving label ranking with MLPs

Chris Burges et al. proposed in [BSR+05] a method of label ranking using multilayer perceptrons. In their model, the input layer corresponds to the input vector x and the output layer has one neuron per possible label.

In order to handle ranking problems, he first extracts training examples from the original training set in the form $(x^i, (y_k, y_l))$, such that $(y_k, y_l) \in \pi^i$. Then, for each such training example, a forward pass is performed for label y_k and a second one for label y_l. The internal state of the network (input and output values for each neuron) is stored for both passes. The cost function becomes a function f of the difference of the outputs $(o_k - o_l)$, and the associated gradient $\frac{\partial f}{\partial \alpha} = (\frac{\partial o_k}{\partial \alpha} - \frac{\partial o_l}{\partial \alpha})f'$, where α represents the model parameters and ∂ is the partial derivative.

3.2.1.4 Solving label ranking with decision trees

Weiwei Cheng et al. proposed in [CHH09] an adaptation of the decision tree induction algorithm to solve label ranking problems in a direct fashion. Since in this scenario, the training information is labeled by orderings π^i, this requires to make two adaptations to the standard tree induction method:

- Adaptation of the information theoretic class purity measure to evaluate the diversity of output orderings in a sample of observations of a node, so as to determine a split that would lead to successor nodes as pure as possible in terms of the observed partial rankings.
- Adaptation of the procedure that determines an optimal prediction at a terminal node, in the form of a ranking that is as similar as possible on the average to all the partial rankings observed in the corresponding training subset.

As a matter of fact, these two problems are intimately linked, since to compute the "information gain", they need to be able to find the center ordering for the subset of training examples corresponding to each node of the decision tree.

The methodology for solving this latter problem will be discussed in section 3.3, but let us assume for the moment that this center ordering can be found. They then compute a θ, which measures the average similarity of a set of orderings to the center ordering of this set. Roughly, when θ is equal to 0, the set contains all permutations of the center ordering while $\theta \to \infty$ indicates that all orderings in the set are identical to the center ordering.

This parameter θ can then be used to evaluate the "purity" of the outputs in a node, and a split will be optimal if it maximizes:

$$|X|^{-1}(|X_1| * \theta_1 + |X_2| * \theta_2), \tag{3.4}$$

where θ_1 and θ_2 are estimated in the subsets of examples X_1 and X_2 induced by the split.

3.2.2 Object ranking

In object ranking, instead of ranking the labels, one wants to rank the objects. For instance, assume that you are a potato chips seller and have (at least partial) information about the general preference of clients for your products. Each of your products is described by a set of features (size, weight, salt level, etc...). If new products are designed, you might want to predict the market response for your new products and obtain a ranking in the form of an ordering which contains your new products and possibly (a part of) your existing products.

Thus, given a (possibly large) set of observations $X = \{x^k | k \in \{1, 2, \ldots, N\}\}$, and set of pairwise comparisons $Z = \{z^i | z^i = (x^{i_1} \succ x^{i_2})\}_{i=1}^n$ over the set X, the aim is to find a ranking function which ranks a set of (possibly unseen) objects X' by generating a permutation of this set. In this protocol, all objects will have to be ranked by using only their feature vector.

3.2.2.1 Solving object ranking using SVM

Ralph Herbrich et al. [HGBSO98] proposed the use of the SVM protocol to solve object ranking problems. To prevent the non transitivity of pairwise preference aggregation, they try to model a linear utility function $\hat{u}(x) = w^T x + b$, compatible with the available information about pairwise object preferences.

Considering m comparisons in a training sample in the form $\{((x_1^i, x_2^i), y^i)\}_{i=1}^n$, where y^i is equal to 1 if $x_1^i \succ x_2^i$ and -1 otherwise, the function $\hat{u}(x)$ is inferred by considering two support vectors at once and is then defined as (actually the value of parameter b is irrelevant in this problem and can be fixed arbitrarily, e.g. $b = 0$):

$$\begin{aligned}
\hat{u}(x) &= w^T x + b && (3.5)\\
w &= \sum_{i=1}^{n} \alpha^i y^i (x_1^i - x_2^i) && (3.6)\\
\alpha &= \arg\max_{(\alpha^1,\ldots,\alpha^n)} \left(\sum_{i=1}^{n} \alpha^i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha^i \alpha^j y^i y^j (x_1^i - x_2^i)^T (x_1^j - x_2^j) \right) && (3.7)
\end{aligned}$$

$$\Rightarrow \hat{u}(x) = \sum_{i=1}^{n} \alpha^i y^i (x_1^i - x_2^i)^T x. \qquad (3.8)$$

These formulas can be extended to the nonlinear case by replacing x by $\phi(x)$ in the above equations, and if we have a kernel $K(x, x')$ to compute $\phi(x)^T \phi(x')$, both the optimization and prediction steps may be expressed, by using the "kernel trick", only in terms of the function $K(x, x')$. This is relevant for problems where it is easier to define a good kernel than a good feature space, such as for instance document ranking, and when the problem is non-linear.

3.2.3 Instance ranking

The third and last type of ranking problem is instance ranking. In this scenario, you want to sort objects, which are labeled by only one label and the labels have a natural order between them. The most common example is the ranking of papers in the categories *rejected*, *accepted with modifications* and *accepted*.

Hence, given a set of observations $X = \{x^i | i \in \{1, 2, ..., n\}\}$, a set of labels $\mathbf{Y} = \{y_j | j \in \{1, 2, ..., N\}\}$ with an order $y_1 \prec y_2 \prec ... \prec y_N$ and for each x^i an associated label y^i, find a ranking function allowing to order a new set of instances X'.

Note that a standard classification algorithm could be used to solve this problem, as each object is labeled with only one label. But these algorithms would minimize the classification error and not the ranking error, so more specific algorithms have to be developed to take this into account.

3.2.3.1 Ordinal Regression by Extended Binary Classification

Ling Li and Hsuan-Tien Lin proposed in [LHT07] an algorithm to solve instance ranking problems. They also consider the fact that standard classifiers do not take into consideration the ordering between labels. For instance, consider a problem with four labels, which are ordered as $\{1, 2, 3, 4\}$. For the example $(x, 4)$, two classifiers r_1 and r_2 will be penalized in the same way if $r_1(x) = 3$ and $r_2(x) = 1$ whereas r_1 is a better predictor than r_2 in this instance ranking problem.

For $k = 1, \ldots, N-1$, they define a binary classifier $f_b(x, k)$ which predicts 1 if the rank of the label associated to x is strictly greater than k, and 0 otherwise. They can thus define their ranker $r(x)$ as

$$1 + \sum_{k=1}^{N-1} f_b(x, k). \tag{3.9}$$

The ordinal information can help to model the relative confidence in the binary outputs. That is, when k is very distant from the rank of x, the prediction from $f_b(x, k)$ should be very confident. The confidence can be modeled by a real-valued function $f : X \times \{1, 2, ..., N-1\} \to \mathbb{R}$, with $f_b(x, k) = [[f(x, k) > 0]]$, where $[[f(x, k) > 0]]$ is equal to 1 if $f(x, k) > 0$ and is equal to 0 otherwise, and the confidence encoded in the magnitude of f. Thus $r(x)$ can now be defined as

$$1 + \sum_{k=1}^{N-1} [[f(x, k) > 0]]. \tag{3.10}$$

Learning an instance ranking scheme is thus reduced to the problem of learning a set of $N-1$ binary classifiers or class-probability estimators, based on $N-1$ auxiliary training sets. Any available base-learner could be applied to these subproblems.

Note that we described an ordinal regression protocol in order to solve an instance ranking problem. Although these problem types are closely related, their goal are different. Ordinal regression tries to predict an ordered set of labels, whereas the goal of instance ranking is to sort objects (whether each object has a predicted label is irrelevant). This can be seen as another example of interoperability between different techniques[1].

[1]See Section 2.2.5 for the first example of interoperability.

3.3 Combining (partial) orderings

Combining (partial) orderings is another challenging field of preference learning. In this field, the goal is not to learn a predictor but to find a center ordering from a set of orderings in such a way that the mean distance between the center and the orderings in the set is minimal. Formally, given a set of (partial rankings) $\{\pi^i \in \overline{\Pi}(\mathbf{Y})\}_{i=1}^{n}$ and a loss function $\ell(\pi, \pi') \to \mathbb{R}^+$, the problem may be stated as finding an optimal total ranking $\pi \in \Pi(\mathbf{Y})$, minimizing the average loss

$$\sum_{i=1}^{n} \ell(\pi, \pi^i). \tag{3.11}$$

Finding a center ordering can be necessary in the decision tree split process, as seen earlier, or when using KNN's in order to make a prediction. The trivial application of KNN algorithms to ranking problems would be to consider each permutation as a single element and to perform the prediction using the majority class as previously explained. However, this technique would probably provide poor results as, unless the value of K is set to 1, the amount of training data is huge or the number N of labels is very small, the aggregation phase for the majority vote will be similar to randomly selecting one ordering in the K nearest training samples. Reducing a ranking to a single variable does not take into account the complex information given in this structure. Moreover, this simple technique supposes that the training data is composed of complete rankings, which is not always the case.

We would like to emphasize on the fact that some of the algorithms designed to combine several orderings require a full ranking information from the learning sample. As this is not always possible, Mark Huber proposed a method [Hub06] for sampling exactly uniformly from the set of linear extensions of a partial ranking[2]. Using his method, it then becomes possible to transform a partial ranking into one of its computed linear extensions (which are fully ordered) for the labeled examples in the learning set.

We will first consider the difficulty of finding such a center, then show a method to estimate the distance between the center and the orderings in the set, and finally describe four different methods to achieve the goal of combining orderings.

[2]Linear extension : Full ranking which is consistent with a given partial ranking

3.3.1 Difficulty of combining orderings

J. Bartholdi et al. demonstrated in [BTT89] that finding the exact center ranking using rank aggregation techniques is NP-Hard, either with a Dodgson or a Kemeny voting scheme. The former is based on the Condorcet criterion which stands that the class label which is ranked in the first position should defeat any other class label in a pairwise election with simple vote counts. Of course, in practice, it is generally impossible to find such label, due to cyclic preference relations. But Dodgson proposed to attribute a score to each label based on the smallest number of adjacent switches required in the voter's preference order such that this label becomes a Condorcet winner. The label with the lowest score (or randomly selected along the smallest scores) is then top ranked. The latter (i.e. the Kemeny voting scheme) defines a distance between two rankings according to the number of discordant pairs, and the goal of Kemeny ranking is to find a consensus which minimizes this distance. As both of these optimization problems have been proven to be NP-Hard, we should only expect approximations for rank aggregation in a polynomial computational time.

3.3.2 Minimizing pairwise disagreements

Klaus Brinker and Eyke Hüllermeier proposed the use of a *consensus label ranking* which is a generated permutation of labels which "minimizes the sum of pairwise disagreement indices with respect to all K rankings" [BH06]. Formally, they seek to solve the problem of finding permutation $\pi \in \Pi(\mathbf{Y})$ such that the total loss over a set $\{\pi\}_{i=1}^{n}$

$$L(\pi) = \sum_{i=1}^{n} \ell(\pi, \pi^i) \tag{3.12}$$

is minimal. The method used to solve this optimization problem depends on the loss function ℓ.

In their paper, they consider various loss functions, and various techniques for finding approximate solutions to the optimization problem

$$\pi^* = \arg \min_{\pi \in \Pi(\mathbf{Y})} \{L(\pi)\}. \tag{3.13}$$

They also discuss the case where the given rankings may contain partial ones.

3.3.3 FAS-PIVOT

Nir Ailon et al. consider in [ACN05] the problem of efficiently computing an approximation of the so-called *Kemeny optimal aggregation* of a set $\{\pi^1, \dots, \pi^n\}$ of total rankings of a set $\mathbf{Y}$ of labels, which amounts to solving the optimization problem of eqn. (3.12) when using the *Kendall-tau* loss function ℓ_K, which in the case of total rankings is defined by

$$\begin{aligned} \ell_K(\pi, \pi') &= |\{(y_k, y_l) \in \pi : (y_k, y_l) \notin \pi'\}| \\ &= |\{(y_k, y_l) \in \pi' : (y_k, y_l) \notin \pi\}|, \end{aligned} \tag{3.14}$$

and amounts to counting the number of different pairs $\{y_k, y_l\}$, among the $N(N-1)/2$ possible ones, for which the two rankings π and π' disagree.

Notice that a simple approximation $\hat{\pi}^*_e$ of the solution of (3.12), would consist of selecting the best one among the given set of π^i's, which can be computed efficiently by enumeration:

$$\hat{\pi}^*_e = \arg \min_{\pi \in \{\pi^1, \dots, \pi^n\}} \{L_K(\pi)\}. \tag{3.15}$$

where L_K denotes the total loss obtained by eqn. (3.12) when using ℓ_K as the loss function.

One can show that $\hat{\pi}^*_e$ is a 2-approximation of π^*, namely that

$$L_K(\hat{\pi}^*_e) \leq 2L_K(\pi^*). \tag{3.16}$$

As a matter of fact, for any π^i we have $L_K(\pi^i) \leq 2L_K(\pi^*)$.

The idea of the authors is to combine two efficient but complementary algorithms, both only having a guarantee to find a 2-approximation. The first one, (Algorithm 6), merely picks an element in $\{\pi^1, \dots, \pi^n\}$ at random.

Algorithm 6 PICK-A-PERM

```
Input: $\{\pi^1, \dots, \pi^n\}$
Output: a 2-approximation $\hat{\pi}^*_r$ of $\pi^*$
return A permutation $\pi^i$ chosen at random from the input
```

The second one, is inspired by the QuickSort algorithm, and is called FAS-PIVOT (Algorithm 7). It is applied to a directed *tournament* graph $G = (V, A)$ derived from the set of permutations in the following way: the set of vertices $V = \mathbf{Y}$, and the set of arcs A contains all pairs of labels (y_k, y_l) such that $w_{k,l} \geq w_{l,k}$, where $w_{k,l} = |\{\pi^j : (y_k, y_l) \in \pi^j\}|$, i.e. the number of input permutations which rank y_k before y_l.

Algorithm 7 FAS-PIVOT($G = (V, A)$)

```
Input: a directed graph $G = (V, A)$
$V_L \leftarrow \emptyset$
$V_R \leftarrow \emptyset$
$v_i \leftarrow$ random pivot $\in V$
for all $v_j \in V \backslash \{v_i\}$ do
  if $(v_j, v_i) \in \mathbf{A}$ then
    add $v_j$ in $\mathbf{V}_L$
  else
    add $v_j$ in $\mathbf{V}_R$
  end if
end for
$G_L = (V_L, A_L)$, the subgraph induced by $V_L$ in $G$
$G_R = (V_R, A_R)$, the subgraph induced by $V_R$ in $G$
return FAS-PIVOT($G_L$), $i$, FAS-PIVOT($G_R$)
```

The authors then show theoretically that taking the best result among the two produced by these two algorithms yields a permutation that is a 11/7-approximation of the optimal one, due to the fact that when one of two heuristics works badly, the other one works well.

Anke van Zuylen proposed a method for selecting the best pivot [VZPW07]. His idea is to *"choose a pivot so that the cost $w_{i,j}$ of the backward arcs created by pivoting on this vertex is at most twice the budget for these arcs"*.

3.3.4 PTAS

The algorithms that we described in the previous sections are constant factors approximation (e.g. a 2-approximation for PICK-A-PERM and FAS-PIVOT, a 11/7-approximation for the combination of the two). Claire Kenyon-Mathieu and Warren Schudy designed a polynomial time approximation scheme (PTAS) [KMS07] which is able, for any given λ, to find an approximation $\hat{\pi}^*$ such that

$$L(\hat{\pi}^*) \quad \leq \quad L(\pi^*) + \lambda \tag{3.17}$$

in polynomial time.

Their algorithm supposes the existence of a *AddApprox* algorithm as, according to [FK99], *"there is a randomized polynomial-time approximation scheme for maximum acyclic subgraph on weighted tournaments. Given $\beta, \eta > 0$, the algorithm outputs, in polynomial time (...) an ordering whose cost is, with probability at least $1-\eta$, less than $OPT + \beta N^2$. The algorithm can be derandomized into an algorithm which we denote by AddApprox, which increases the runtime to $N^{O(1/\eta)}$"*. They also define the β parameter as

$$\beta(\varepsilon) \quad = \quad 2^{-O(1/\varepsilon \log(1/\varepsilon))}. \tag{3.18}$$

where $\varepsilon = \lambda b$ and b defines the transitivity of the weights by

$$\forall i, j \in \{1, \ldots, N\} : w_{ij} + w_{ji} \in [b, 1]. \tag{3.19}$$

Their algorithm exists in a randomized and a deterministic version, but we will only describe the latter even if the former is, according to the authors, faster than the latter, the latter is more intuitive. Firstly, they round the weights to integers so as to reduce the computational complexity and ensure that this complexity is bounded. Secondly, they apply a constant factor approximation algorithm to determine the initial prediction $\hat{\pi}$. Finally, they refine the prediction by the use of single vertex moves (i.e. changing the rank of a single label in the ordering) and the *AddApprox* function on a subset of the ordering.

Their deterministic PTAS is given in Algorithm 8.

3.3.5 Mallows model

Eyke Hüllermeier and his co-authors proposed another method [CH08, CHH09] based on a probabilistic model (so-called Mallows model) over the set of permutations $\Pi(\mathbf{Y})$, whose parameters are derived by the maximum likelihood principle from a sample $\{\pi^1, \ldots, \pi^n\}$ of possibly partial rankings (i.e. $\pi^i \in \overline{\Pi}(\mathbf{Y})$). We will explain in details the ideas introduced by these authors while using our notations, since several of these ideas are exploited in later chapters.

Algorithm 8 Deterministic PTAS [KMS07]

```
**Input :** Fixed parameters $\varepsilon > 0$ and $b \in ]0, 1]$, A weighted tournament.
**Output :** An additive approximation $\hat{\pi}$ whose cost is at most OPT + $\varepsilon/b$
Round weights to integer multiples of $\varepsilon/N^2$.
$\hat{\pi} \leftarrow$ constant factor approximation [ACN05].
**while** Some moves decrease the cost **do**
  Apply these moves. Types of moves:
  **Single vertex move** =
  Choose vertex (label) $y$ and rank $j$, take $y$ out of the ordering $\hat{\pi}$ and put it back in so that its rank is $j$.
  **Additive approximation** =
  Choose intergers $i < j$.
  Let $U = \{$ vertices (labels) with rank in $[i, j]\}$.
  $\pi_U^{'} \leftarrow AddApprox(U)$ with parameter $\beta(\varepsilon)$.
  Replace the restriction $\pi_U$ of $\hat{\pi}$ to $U$ by $\pi_U^{'}$.
**end while**
```

3.3.5.1 Mallows model over complete rankings

The Mallows model is based on a loss-function $\ell(\cdot, \cdot)$ defined over $\Pi(\mathbf{Y})$ and is defined by two "parameters", a centroid permutation $\overline{\pi} \in \Pi(\mathbf{Y})$, and a measure of concentration around the center $\theta \in [0, \infty[$ (the authors of [CH08, CHH09] use the slightly counter-intuitive term of "spread" to denote θ).

Given these parameters and the loss function, the probability of observing a permutation $\pi \in \Pi(\mathbf{Y})$ is defined by

$$P(\pi|\overline{\pi}, \theta, \ell) = \frac{\exp\left(-\theta\ell(\pi, \overline{\pi})\right)}{Z(\theta, \overline{\pi}, \ell)}, \tag{3.20}$$

where Z is a normalization constant defined by

$$Z(\theta, \overline{\pi}, \ell) = \sum_{\pi \in \Pi(\mathbf{Y})} \exp\left(-\theta\ell(\pi, \overline{\pi})\right). \tag{3.21}$$

Thus, if $\theta = 0$, this model actually defines a uniform distribution over $\Pi(\mathbf{Y})$ whereas, if $\theta \to \infty$, the probability mass is concentrated on the centroid $\overline{\pi}$ (assuming that the loss function $\ell(\pi, \pi')$ is non-negative and equal to zero only if $\pi = \pi'$, as it is the case for all loss-functions used in practice).

The authors of [CHH09] use the Kendall-tau loss-function ℓ_K defined by eqn. (3.14), and show that in this case the normalization constant is actually independent of $\overline{\pi}$ and may be computed efficiently by

$$Z(\theta, \overline{\pi}, \ell_K) = \prod_{k=1}^{N} \frac{1 - \exp(-k\theta)}{1 - \exp(\theta)}. \tag{3.22}$$

3.3.5.2 Probability to observe an incomplete ranking

Given a partial information about the true permutation π', in the form of a partial ranking $\pi \in \overline{\Pi}(\mathbf{Y})$, we can infer that the probability to observe this partial ranking is simply the cumulated probability mass over the set $E(\pi) = \{\pi' \in \Pi(\mathbf{Y}) : \pi \in \pi'\}$ of complete rankings compatible with π

$$P(E(\pi)|\overline{\pi}, \theta, \ell) = \sum_{\pi' \in E(\pi)} P(\pi'|\overline{\pi}, \theta, \ell). \tag{3.23}$$

Obviously, if $\pi \in \Pi(\mathbf{Y})$, then $P(E(\pi)|\overline{\pi}, \theta, \ell)$ reduces to $P(\pi|\overline{\pi}, \theta, \ell)$.

3.3.5.3 Maximum likelihood estimation of the model parameters

In some context, if we are given a set of observations $\{\pi^1, \ldots, \pi^n\}$, where each $\pi^i \in \overline{\Pi}(\mathbf{Y})$, i.e. is a possibly partial ranking of $\mathbf{Y}$, and if we assume that these observations were obtained by independently drawing them according to a Mallows model of parameters $(\overline{\pi}, \theta, \ell)$, we may use the maximum likelihood principle in order to infer suitable estimates of $\overline{\pi}$ and θ. Namely

$$(\hat{\overline{\pi}}, \hat{\theta}) = \arg \max_{\overline{\pi} \in \Pi(\mathbf{Y}), \theta \in [0, \infty[} \prod_{i=1}^{n} P(E(\pi^i)|\overline{\pi}, \theta, \ell). \tag{3.24}$$

Notice that when $\ell = \ell_K$ and all the observations are complete rankings, this maximization problem can be rewritten as follows

$$\begin{aligned} \arg\max_{\overline{\pi},\theta} \prod_{i=1}^{n} P(E(\pi^i)|\overline{\pi}, \theta, \ell_K) &= \arg\max_{\overline{\pi},\theta} \prod_{i=1}^{n} P(\pi^i|\overline{\pi}, \theta, \ell_K), & (3.25) \\ &= \arg\max_{\overline{\pi},\theta} \prod_{i=1}^{n} \frac{\exp\left(-\theta \ell_K(\pi^i, \overline{\pi})\right)}{Z(\theta, \overline{\pi}, \ell_K)}, & (3.26) \\ &= \arg\max_{\overline{\pi},\theta} \frac{\prod_{i=1}^{n} \exp\left(-\theta \ell_K(\pi^i, \overline{\pi})\right)}{\prod_{i=1}^{n} \prod_{k=1}^{N} \frac{1-\exp(-k\theta)}{1-\exp(\theta)}}, & (3.27) \end{aligned}$$

$$= \arg\max_{\overline{\pi},\theta} \frac{\exp\left(-\theta \sum_{i=1}^{n} \ell_K(\pi^i, \overline{\pi})\right)}{\left(\prod_{k=1}^{N} \frac{1-\exp(-k\theta)}{1-\exp(\theta)}\right)^n}. \quad (3.28)$$

The last maximization problem may be decomposed into two successive optimization problems, namely

$$\hat{\overline{\pi}} = \arg\min_{\overline{\pi}\in\Pi(\mathbf{Y})} \sum_{i=1}^{n} \ell_K(\pi^i, \overline{\pi}), \quad (3.29)$$

$$\hat{\theta} = \arg\max_{\theta\in[0,\infty[} \frac{\exp\left(-\theta \sum_{i=1}^{n} \ell_K(\pi^i, \hat{\overline{\pi}})\right)}{\left(\prod_{k=1}^{N} \frac{1-\exp(-k\theta)}{1-\exp(\theta)}\right)^n}. \quad (3.30)$$

Notice that the first (minimization) problem is the one that we have already discussed in the preceding sections, and we know that this problem is difficult to solve exactly.

On the other hand, once the solution of the first problem is given, the second maximization problem is easy to solve. Indeed taking the logarithm of the right-hand side of eqn. (3.30), we need to maximize

$$-\theta \sum_{i=1}^{n} \ell_K(\pi^i, \hat{\overline{\pi}}) - n \sum_{k=1}^{N} \frac{1-\exp(-k\theta)}{1-\exp(\theta)}, \quad (3.31)$$

which can be shown to be equivalent to solving the following equation w.r.t. θ

$$0 = \frac{1}{n}\sum_{i=1}^{n} \ell_K(\pi^i, \hat{\overline{\pi}}) - \frac{N\exp(-\theta)}{1-\exp(-\theta)} + \sum_{k=1}^{N} \frac{k\exp(-k\theta)}{1-\exp(-k\theta)} \quad (3.32)$$

where the right-hand side is a monotone increasing function of θ [CHH09].

3.3.5.4 Approximate solution of the optimal centroid problem

The authors of [CHH09] recognize the difficulty to solve the problem

$$\hat{\overline{\pi}} = \arg\min_{\overline{\pi}\in\Pi(\mathbf{Y})} \sum_{i=1}^{n} \ell_K(\pi^i, \overline{\pi}) \quad (3.33)$$

exactly and hence propose another heuristic to solve it approximately, based on so-called "Borda counts". Because we will use this idea in later chapters, we describe here this method together with the intuitions behind it.

The main idea is to use as proxy for the Kendall-tau loss-function the Spearman rank correlation, which is often strongly correlated with ℓ_K and which leads to a tractable optimization problem. The Spearman rank correlation between two complete rankings in $\Pi(\mathbf{Y})$ is defined as a sum of squared rank differences

$$\ell_S(\pi, \pi') = \sum_{k=1}^{N} (\pi(y_k) - \pi'(y_k))^2, \tag{3.34}$$

where we denote by $\pi(y_k) \in \{1, \ldots, N\}$ the position at which label y_k appears in the ranking π. Given a set of full rankings $\{\pi^1, \ldots, \pi^n\}$, one can find a permutation π_S^* minimizing the total loss

$$L_S(\pi) = \sum_{i=1}^{n} \ell_S(\pi^i, \pi) \tag{3.35}$$

by a very simple and efficient procedure. Indeed, an optimal π_S^* may be obtained by sequentially voting over the label set, and then sorting the labels according to their votes.

In this so-called Borda counting scheme, each π^i gives a vote of $N+1-\pi^i(y_k)$ to each label $y_k \in \mathbf{Y}$. The n votes obtained by each label are cumulated, and the labels are then sorted according to these cumulated votes in decreasing order, yielding an ordering which is optimal with respect to ℓ_S.

3.3.5.5 Handling partial rankings

In order to solve the general problem of finding a centroid $\overline{\pi}$ and a measure of concentration θ well representing a sample $\{\pi^1, \ldots, \pi^n\}$ of possibly partial rankings (i.e. $\pi^i \in \overline{\Pi}(\mathbf{Y})$), the authors of [CHH09] propose an Expectation-Maximization like algorithm, of which we briefly sketch here the main ideas (see [CHH09] for a more accurate description of the corresponding algorithms).

- At the first iteration, they choose an initial $\hat{\overline{\pi}}$ and $\hat{\theta}$ by assuming for each partial ranking that its possible extensions (all full rankings compatible with the partial one) are equally likely, and by exploiting slightly adapted versions of equations (3.35) and (3.32) to this assumption.
- At any subsequent iteration, they proceed in two steps:

  - use of the current probabilistic model parameters $\hat{\bar{\pi}}$ and $\hat{\theta}$ to complete the partial rankings π^i by their most probable extension π_e^i, namely the one minimizing $\ell_K(\pi^1, \pi_e^i)$ (it can be computed exactly, in an efficient way);
  - use of the completed dataset $\{\pi_c^1, \dots, \pi_c^n\}$, and equations (3.35) and (3.32) to compute updated values of $\hat{\bar{\pi}}$ and $\hat{\theta}$

- until $\hat{\bar{\pi}}$ (and hence $\hat{\theta}$) has reached a fixed point.

3.4 Similar domains

Some learning problems cannot be considered *stricto sensu* as a problem of learning to rank, but are so closely related that they can be still addressed by using ranking ideas. These problems include multi-label classification and multi-label ranking. Each of these problems will be described in its own section and at least one resolution protocol will be given for each problem.

3.4.1 Multi-label classification (MLC)

In multi-label classification (or MLC), each object $x^i \in X$ is labeled with a subset of relevant labels $Y^i \subset \mathbf{Y}$, inducing that unused labels are not relevant to this specific object. For instance, we could label movies according to their types considering that a movie can have different types (Comedy and Sci-Fi, or Drama and Horror, ...). The goal is to learn a model able to determine which subset of labels $\hat{Y}(x) \subset \mathbf{Y}$ is relevant and which are irrelevant for a given (possibly unseen) object x.

3.4.1.1 Solving MLC with MLP's

A modified version of MLP suitable for multi-label classification was proposed by Koby Crammer et al. in [CDK$^+$06]. Their idea is to output a score for each label which is enforced to be such that scores of relevant labels are higher than scores of irrelevant ones. As they are working on multi-label classification rather than ranking, they want to separate relevant labels from irrelevant ones. The margin is the difference between the score of the lowest ranked relevant label and the score of the highest ranked irrelevant label. A positive margin means that every relevant label is ranked above irrelevant ones, but they also impose the margin to be higher than 1 by applying an error loss according to the following hinge-loss function:

$$loss = \begin{cases} 0 & margin \geq 1, \\ 1 - margin & otherwise. \end{cases} \tag{3.36}$$

Koby Crammer et al. also proposed an algorithm that considers that the obtained feedback takes the form of a partial ranking. They can thus separate relevant labels from irrelevant ones. This information is represented using a bipartite graph where relevant labels are on top of irrelevant ones, and each top label is connected to each bottom label. The vertex $[A, B]$ will thus be consistent with the predicted total order if $A > B$ in the order, and non consistent otherwise. The loss function is then derived from this graph in four possible manners:

$$loss = \begin{cases} \dfrac{\text{Number of inconsistent vertices}}{\text{Total number of vertices}} & \text{(all-pair cover)}, \\[2ex] \begin{cases} 0 & \text{All consistent vertices} \\ 1 & \text{otherwise} \end{cases} & \text{(0-1 cover)}, \\[2ex] \dfrac{\text{Number of relevant labels which do not dominate all the irrelevant labels}}{\text{Number of relevant labels}} & \text{(domination cover)}, \\[2ex] \dfrac{\text{Number of irrelevant labels which are not dominated by all the relevant labels}}{\text{Number of irrelevant labels}} & \text{(dominated cover)}. \end{cases} \tag{3.37}$$

and applied in a MLP algorithm.

3.4.1.2 Solving MLC with SVM

André Elisseeff and Jason Weston proposed in [EW02] the algorithm "RankSVM" to perform multi-label classification by maximizing a separation margin, an approach which is in some sense similar to SVMs. This algorithm first outputs a ranking and an additional procedure is required to set a separation between relevant and irrelevant labels. However, we could skip this procedure if we solely want to obtain a ranking as output.

Elisseeff and Weston work with m training samples of the form (x^i, Y^i) where Y^i is a subset of the complete set $\mathbf{Y}$ of labels associated with the input data x^i.

He denotes by $\overline{Y^i}$ the subset $\mathbf{Y}\backslash Y^i$, that is, the subset of labels not associated with object x^i. Considering that the objective is to find $N = |\mathbf{Y}|$ vectors w_k which are used by sorting the values of $\langle w_k, x\rangle + b_k$ to obtain a predicted ranking, the following optimization problem needs to be solved :

$$\min_{i,j=1,\ldots,N} \sum_{k=1}^{N} ||w_k||^2 + C \sum_{i=1}^{n} \frac{1}{|Y^i||\bar{Y^i}|} \sum_{(k,l)\in Y^i \times \overline{Y^i}} \xi_{ikl} \tag{3.38}$$

$$\text{s.t.: } \langle w_k - w_l, x^i\rangle + b_k - b_l \geq 1 - \xi_{ikl}, \forall (k,l) \in Y^i \times \bar{Y^i}, \tag{3.39}$$

$$\xi_{ikl} \geq 0. \tag{3.40}$$

However, this problem is quadratic in terms of the learning set size and is generally solved in $O(n^3)$ steps, which is not acceptable, so the authors used a linearization method combined with a predictor-corrector logarithmic barrier procedure to be able to test their method on large datasets.

3.4.1.3 (Probabilistic) Chained Classification

The concept of Chained Classification first appeared in [RPHF09] by Read et al. In opposition to **Binary Relevance**, or BR, which consists in training N independent classifiers, each one predicting the relevance of one label, Chained Classification takes the correlation between class labels into account. It also builds N classifiers, but the data used to build the k^{th} classifier contains the input vector x plus the $k-1$ outputs of classifiers $\{1, \ldots, k-1\}$. The first classifier is build in a traditional manner, i.e. using only the feature vector x. In the learning phase, the true outputs $y_j, j \in \{1, \ldots, k\}$ given as supervision are used for training the classifier $k+1$ but, in the prediction phase, only the predictions of classifiers $\{1, \ldots, k\}$ (plus the feature vector x) can be used as input for classifier $k+1$ since the true outputs are not available, which can add some bias. In the case where probabilistic classifiers are used (i.e. the output of classifier k represents the probability that the corresponding class label y_k is relevant), the final prediction can be computed by thresholding the classifiers outputs such that a set of binary values are obtained. Algorithm 9 describes this process.

Since we are trying to maximize

$$P(\mathbf{y}|x) \quad = \quad \prod_{k=1}^{N} P(y_k|x, y_1, \ldots, y_{k-1}), \tag{3.41}$$

Algorithm 9 Inference by Greedy Search [RPHF09]

```
Input : $N$ binary classifiers $h_1, \ldots, h_N$, feature vector $x$
Output : A relevance vector representing a subset of labels
for $k = 1 \ldots N$ do
    $y_k \leftarrow h_k(x, y_1, \ldots, y_{k-1})$
    if $y_k \geq 0.5$ then
        $y_k \leftarrow 1$
    else
        $y_k \leftarrow 0$
    end if
end for
return $\mathbf{y} = \{y_1, \ldots, y_N\}$
```

Dembvzynski et al. proved in [DWH12] that this greedy approach can lead to a prediction which can be quite far from the true optimum. Consider for instance the example given in Figure 3.5 representing a probability tree which is built using the conditional outputs of each classifiers for a given x, and where the leafs represents $P(\mathbf{y}|x)$ for each possible combinations $\mathbf{y}$ of the set of binary variables. In this example, we are working with 3 classes. The first classifier outputs the value of 0.6, meaning that there's a 60% chance that label y_1 is relevant to the current feature vector x. According to the greedy policy, this label is thus considered relevant and the output of the second classifier is calculated based on the fact that $y_1 = 1$. Again, $P(y_2 = 1|y_1 = 1) = 0.6$, thus we will consider y_2 as relevant and continue this process until we reach the leaf labeled by $(y_1 = 1, y_2 = 1, y_3 = 1)$ which has a probability $P(\mathbf{y}|x)$ of $0.6 * 0.6 * 0.6 = 0.216$ to be the prediction which minimizes the loss. However, if we could have explored the entire probability tree, we would have seen that a better solution was $(y_1 = 0, y_2 = 1, y_3 = 1)$ since it has a probability of $0.4 * 0.9 * 0.9 = 0.324$ to be the prediction which minimizes the loss.

Exploring the entire probability tree would require the analysis of 2^N label combinations, which is extremely costly. The authors of [DWH12] provide two algorithms in order to smartly search the tree. The first algorithm is based on monte-carlo search. Starting from the root node, the search is performed by flipping a biased coin at each node, where the probability to explore the left branch is given by the output of the classifier (thus, the probability to explore the right branch is equal to 1 - the output). This search is repeated s times

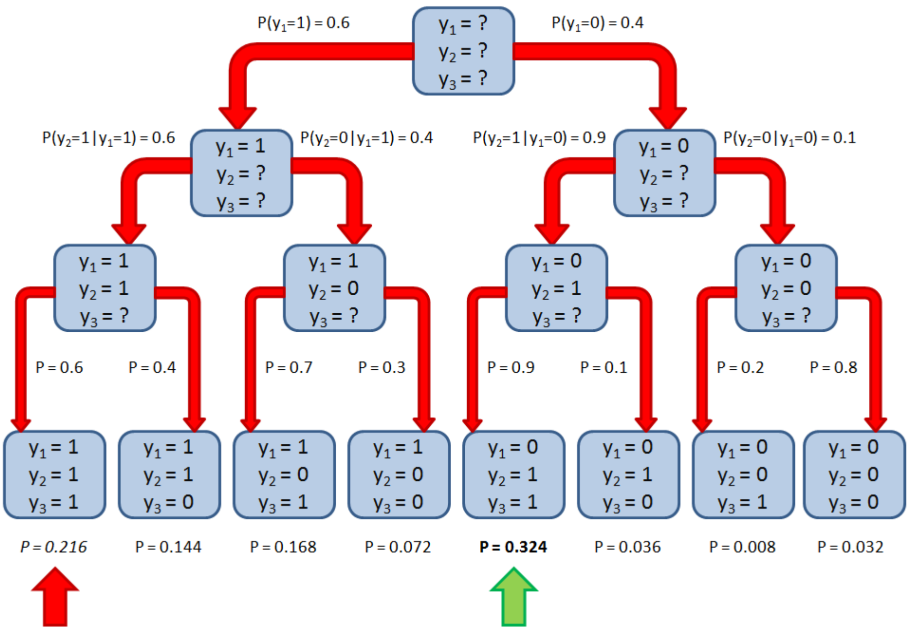


Figure 3.5: A probability tree. The probability of each leaf is computed as the product of the classification outputs from the direct hierarchy and represents the chance that the labeled solution minimizes the loss.

and the leaf with the maximum probability, across all discovered leaves, is the prediction. The second algorithm is called ε-approximate inference and is shown in Algorithm 10. In a nutshell, this algorithm explores the probability tree in a best first manner. As long as $P(\mathbf{y}_v|x) \geq \varepsilon$, where $\mathbf{y}_v$ is the (partial) solution represented by node v, the left and right child of v are expanded, otherwise v is stored in another array K and the exploration on v is stopped. If v is a leaf, then the solution $\mathbf{y}_v$ is output. If no leaf can be reached, the greedy search is applied to each node in K and the solution maximizing the probability is kept.

Algorithm 10 ε-approximation inference [DWH12]

```
Input : A probability tree $\mathcal{M}$ representing $N$ binary classifiers outputs based
on feature vector $x$, parameter $0 < k \leq N$
Output : A relevance vector representing a subset of labels
ordered list $Q \leftarrow \{v_{\mathcal{M}}\}$ (contains root node initially)
ordered list $K \leftarrow \{\}$ (non-survived parents)
define $\prod(v_{\mathcal{M}}) = P(\mathbf{y}_{\mathcal{M}}|x) = 1$ ($\mathbf{y}_{\mathcal{M}}$ is the solution represented by node $v_{\mathcal{M}}$)
$\varepsilon \leftarrow 2^{-k}$
while $Q \neq \emptyset$ do
    $v \leftarrow$ pop first element in $Q$
    if $v$ is a leaf then
        delete all elements in $K$ and break the while loop
    end if
    $lc(v), rc(v) \leftarrow$ left and right child of $v$
    compute $\prod(lc(v))$ and $\prod(rc(v))$
    if $\prod(lc(v)) \geq \varepsilon$ then
        insert $lc(v)$ in list $Q$ sorted according to $\prod(lc(v))$
    end if
    if $\prod(rc(v)) \geq \varepsilon$ then
        insert $rc(v)$ in list $Q$ sorted according to $\prod(rc(v))$
    end if
    if $lc(v)$ and $rc(v)$ are not inserted in $Q$ then
        insert $v$ in $K$ sorted according to $\prod(v)$
    end if
end while
$\varepsilon \leftarrow 0$
while $K \neq \emptyset$ do
    $v' \leftarrow$ pop first element in $K$ and apply Alg. 9 on the classifiers not handled
at this node
    if $\prod(v') \geq \varepsilon$ then
        $v \leftarrow v'$ and $\varepsilon \leftarrow \prod(v')$
    end if
end while
return The solution represented by $v$
```

The authors of [DWH12] conclude by showing, through an empirical validation, that their probability tree approach is more appropriate when the subset 0/1 loss, i.e. the loss is equal to 0 if the prediction perfectly matches the supervision and is equal to 1 otherwise, needs to be minimized whereas the BR algorithm should be used when the Hamming loss, i.e. the loss is the number of independent label mistakes normalized in $[0, 1]$, needs to be minimized.

3.4.2 Multi-label ranking (MLR)

Multi-label ranking is a combination of label ranking and multi-label classification. The output of such a model is composed of a ranking of the set of labels and a partition of it into two parts (relevant and irrelevant).

3.4.2.1 Solving MLR

The aggregation technique described in Section 3.3.2[3] was later applied to MLR by the same authors (Brinker and Hüllermeier) [BH07]. To update this technique into MLR, a virtual label λ_0 is added into the training sample between the two partitions. As MLR is also susceptible to be applied to MLC data, the loss function should be able to consider ties between labels (i.e. relevant labels are preferred over irrelevant ones, but two labels from the same partition have the same preference) and the optimization method is modified accordingly.

3.5 Existing reduction techniques

The aim of this thesis, which will be more extensively and formally described at the beginning of Chapter 4, is to reduce the complexity of the Ranking by Pairwise Comparison algorithm (RPC; described in Section 3.2.1.1). In a nutshell, for a problem which has N labels, RPC requires the training of $N(N-1)/2$ pairwise comparators. We develop methods and algorithms which aim at finding a subset of $T << N(N-1)/2$ comparisons, prior to the building of the corresponding T comparators, which, if used in a RPC scheme, would perform predictions which are nearly as accurate as the prediction of a RPC scheme using all $N(N-1)/2$ comparisons. Note that the subset of T comparisons is selected offline, hence the same T comparators will be used to perform the predictions of each object in the test sample.

[3]which is used in a kNN algorithm

A similar idea has already been proposed in the literature, in order to reduce the complexity of the prediction procedure in an on-line fashion, which is different from our own objective although related. We will therefore describe the QWeighted algorithm, which addresses the reduction problem at the prediction time by trying only to predict the top ranked element, and one of its variants, which predicts the top k elements in order to solve a multi-label classification problem.

3.5.1 QWeighted algorithm

The QWeighted algorithm [PF07], designed by Sang-Hyeun Park and Johannes Fürnkranz, proposes to reduce the complexity of RPC by trying to predict only the top element in the ranking. Their algorithm 11 iteratively selects the comparisons which provide the most information in order to determine the best label. They argue that *"if class a has already received more than $N - j$ votes, and label b lost j votings, then it is impossible for b to achieve a higher total voting mass than a"*.

In the best case, only $N - 1$ comparisons are required to discover the top ranked element (for instance, if the comparators are trained in classification mode, y_{top} is selected as the first label in the first step and each comparator which compares y_{top} to the other labels gives their preference to y_{top}), but the worst case still requires the whole set of $N(N-1)/2$ comparisons (when each comparator provides a vote of 0.5). In average, the prediction of the top class requires $N \log N$ comparisons. Note that, since the comparisons are chosen at runtime, this supposes the availability of the whole set of $N(N-1)/2$ comparators. The computational gain thus only occurs *at prediction time*.

3.5.2 QWeighted for multi-label classification

A variant of the QWeighted algorithm has been proposed by Loza Mencía et al. [LMPF10]. The authors notice that the QWeighted algorithm could be applied k times in order to obtain the top-k elements of the ranking (given that the loss of the discovered top element is set to the infinite before the next call to the QWeighted algorithm).

They can then solve multi-label classification problems, by inserting a factious label y_0, which separates the set of relevant labels and the set of irrelevant

Algorithm 11 QWeighted algorithm [PF07]

```
Input : A set $Y$ of $N$ labels, a set of $N(N-1)/2$ comparators $q_{k,l}$.
Output : The top ranked label $y_{top}$
$l_i \leftarrow 0$ for all $i \in \{1, \ldots, N\}$
while $y_{top}$ not determined do
  $y_a \leftarrow$ label $y_i \in Y$ with minimal $l_i$
  $y_b \leftarrow$ label $y_j \in Y \setminus \{y_a\}$ with minimal $l_j$ and comparator $q_{a,b}$ has not yet been evaluated
  if $\neg\exists y_b$ then
    $y_{top} \leftarrow y_a$
  else
    $v_{ab} \leftarrow$ Evaluate($q_{a,b}$)
    $l_a \leftarrow l_a + (1 - v_{ab})$
    $l_b \leftarrow l_b + v_{ab}$
  end if
end while
return The top ranked element $y_{top}$
```

labels, then iteratively calling QWeighted until the predicted label is y_0. Using this technique, each multi-label classification problem can be solved using $N + dN \log N$ comparators, where d is the average number of labels per instance.

3.6 Summary and outlook

In this chapter we have sought to give an in depth analysis of the main stakes in the field of preference learning, while reporting about the main sub-problems in this field and the diversity of algorithmic approaches that have been proposed in the literature.

In practice, the applications of preference learning are multitudinous, as suggested by some of our examples used to illustrate ideas. But many of these interesting applications lead to large scale problems, with large sample sizes n, high input-space dimensions d, and large numbers N of items to rank.

Several preference learning approaches are based on reductions of the preference learning problem to a set of classical supervised learning problems (e.g. RPC, most notably) or to an auxiliary problem obtained by restating the original problem in a higher-dimensional space (e.g. the constraint based approach).

While very elegant, in principle, these approaches lead to algorithmic schemes which can not be applied to very large scale preference problems, because of computational complexity reasons.

Other approaches are based on *embedded* adaptations of existing supervised learning techniques, so as to handle the particular structure of ranking problems (e.g. the adaptation of decision tree induction, or of support vector machines, proposed in the literature to cope with the goals of preference learning). While very useful, per se, these latter approaches are however limited to the particular base-learning algorithm to which they were tailored, and they also inherit its intrinsic weaknesses. This makes it more difficult to take advantage, in the scope of these approaches, of the many ongoing developments in the field of supervised learning.

In this context, our investigations, reported in the subsequent chapters, were originally motivated by the intuition that in the RPC approach it might not be necessary, in order to learn a good ranking scheme, to exhaustively consider the whole quadratic number of pair-wise comparisons as auxiliary subproblems. Indeed, if it were possible to strongly trim the number of used comparators, say to order N or $N \log N$, without sacrificing accuracy, then this “trimmed” version of RPC would become a very attractive approach to label-ranking problems, since it can exploit any of the very effective supervised learning algorithms available from the shelf.

We will analyze, in the two subsequent chapters, whether indeed this approach is interesting. In Chapter 4, we will precisely describe the algorithms that we have designed and implemented to carry out this research, ranging from very simple randomized schemes subsampling the set of possible comparisons, to more sophisticated approaches aiming at finding an optimal subset of fixed size of comparisons for a given problem by exploiting a training set for this problem. In Chapter 5, we will provide an empirical evaluation of these approaches on several datasets, of variable size, dimensionality and annotation quality, showing that indeed it is possible to exploit these ideas in practice without loss of accuracy, while keeping the algorithms intrinsically scalable. In the conclusion chapter of this thesis, we will discuss whether these ideas could be carried over to other types of algorithmic frameworks discussed in the present chapter, such as the constraint based approach or the embedded techniques.

Chapter 4

Methods and algorithms

Contents

4.1 Formal description of the problem 93

4.1.1 Complexity reduction 94

4.1.2 Sparsity in partially ranked datasets 95

4.2 Scoring a ranking scheme on a sample 96

4.3 Dealing with sparsely ranked datasets 98

4.3.1 Modeling empty comparisons by dummy models . . 98

4.3.2 Dropping empty comparisons 98

4.4 Comparison selection algorithms 99

4.4.1 Evaluating a set of comparisons during optimization 99

4.4.2 Pure Random selection (PR) 100

4.4.2.1 Imposing a full class coverage 101

4.4.3 Estimation of Distribution Algorithm (EDA) 103

4.4.3.1 General principle of EDA 103

4.4.3.2 Application of EDA to comparator subset selection 104

4.4.4 "Exhaustive" Greedy Search (EGS) 106

4.4.4.1 General principle of greedy algorithms . . . 106

4.4.4.2 Application of greedy algorithms to comparison subset selection : EGS 108

4.4.4.3 Alternative approach for EGS 109

4.4.5 Randomized Greedy Algorithm (RGS) 109

4.5 Models for learning comparators 111

4.6 Hard comparators vs soft comparators 114

4.7 Summary **115**

This chapter describes the algorithms, methods and protocols that we propose to use in order to efficiently solve the problem which we want to address. In the first place, we will define some notions and notations which are required to understand the sequel of this chapter.

For a given dataset, a fixed number of class labels are being considered, and we denote them as $\mathbf{Y} = \{y_1, \dots, y_N\}$, y_k being a class label, $\mathbf{Y}$ is the total number N of considered labels in the dataset. In each dataset, we are provided with a learning sample **LS** which contains n tuples $\{(x^i, \pi^i)\}_{i=1}^n$, where x^i is a feature vector describing the object o^i and π^i is an ordered subset of $\mathbf{Y}$ corresponding to x^i given as a supervision. We are also provided a test sample **TS** which also contains tuples (x^i, π^i) for (possibly) different objects than those on the **LS**. **LS** will be used for training and **TS** will be used for evaluation.

For each pair of labels (y_k, y_l) where both labels are ranked in π^i, we can deduce a binary preference relation $\succ_{kl}^i$ (also denoted as q_{kl}^i) which is true if y_k is ranked before y_l in π^i, and false otherwise. We denote this relation by the term "comparison". A comparison q_{kl} can thus only be verified for some objects in the **LS** (i.e. those whose π indeed ranks y_k with respect to y_l). If we want to predict the preference relation between y_k and y_l for any (and possible unseen) object, we need to infer a model, based on objects in the **LS** providing preference information for q_{kl}, such that we can obtain a binary preference prediction for each object. We define such a model as a "comparator" and denote it as $f_{kl}(x)$.

When considering a set $\mathbf{Y}$ of N labels, a maximum of $N(N-1)/2$ comparisons can be considered. We denote this set of comparisons by Q^{Full}.

In order to produce a prediction $\hat{\pi}$ for a given vector of attributes x, we build a comparator for each comparison in Q^{Full}. Each comparator $f_{kl}(x)$ will give a vote v^k to label y_k and $v_l = 1 - v_k$ to label y_l. These votes are kept in a table of votes of size N, then each label in $\mathbf{Y}$ is ranked according to this table (i.e. the label with most votes is ranked first, then the label with most votes except the first ranked label is ranked second, and so on).

We evaluate the accuracy of one prediction by comparing $\hat{\pi}^i$, the predicted ordering for x^i (i.e. $\hat{\pi}^i = \hat{\pi}(x^i)$) to its supervision π^i. The metric used to compare two orderings will be discussed later. In order to evaluate the overall accuracy of our predictions, we perform a prediction for each object in the **TS** and compute the correlation (using the given metric) between this prediction and its corresponding supervision label ordering given for this object. The mean value of these correlation measures provides the ranking score $S_{\mathbf{TS}}(Q^{Full})$ for the set of comparisons.

We propose to use a subset $Q \subset Q^{Full}$ of comparisons of size $T << \#Q^{Full}$ so as to reduce the complexity of RPC (which uses Q^{Full}) without degrading $S_{\mathbf{TS}}(Q)$ and we aim to design algorithms for finding the subset Q which maximizes $S_{\mathbf{TS}}(Q)$ for a given T.

In some cases, building a comparator for each comparison is not a trivial task. This problem, as well as the complexity reduction issue, will be formally defined in the next section.

We will then define the selected protocol used to compare the predictions to the labeled orderings in Section 4.2. Section 4.3 will explain how we addressed the issue of "hard-to-build" comparators. Section 4.4 describes the algorithms that we will use to select a subset Q of comparisons. The SL algorithm used to train the comparators is discussed in 4.5. Section 4.6 will discuss the possible effect of the mode (regression vs classification) of the model used in the comparator training. Finally, Section 4.7 concludes by describing the way in which we will validate our methods within the next chapter.

4.1 Formal description of the problem

The problem that we want to address can be divided into two subproblems. Firstly, we want to reduce the complexity of the RPC algorithm by trimming the set of considered comparisons while not drastically affecting its reliability. Secondly, partially ranked datasets (where at least one object is not labeled by a permutation of all possible labels) can be so sparse that some comparators become hard to train. We will define these two subproblems in the following two sections.

4.1.1 Complexity reduction

Let $x \in \mathbf{X}$ denote a vector of attributes of the object of concern, and let $\mathbf{Y} = \{y_1, \ldots, y_N\}$ be a set of labels (finite but often very large). The goal of learning-to-rank is to infer a function $f(x)$ computing complete orderings of $\mathbf{Y}$, based on a learning-sample **LS** composed of n pairs (x^i, π^i), where each $\pi^i = (y_{i_1}, \ldots, y_{i_{k_i}})$ is an ordering of a subset of $\mathbf{Y}$.

For each $y_k, y_l \in \pi^i, k \neq l$, we can induce a preference relation $y_k \succ^i_{kl} y_l$ indicating that y_k is preferred over y_l in the (possibly partial) ordering π^i. We denote this preference relation based on the **LS** samples by the term "comparison" which can be true or false for a given object o^i in the training sample **LS** and for a pair of labels (y_k, y_l) where $y_k, y_l \in \pi^i$. When trying to infer this preference relation with SL algorithms, we are building a "comparator", which predicts the preference over two labels for a given (and possibly unseen) vector of attributes.

In the RPC method (see [HFCB08]) one needs to train one comparator for each comparison $q_{kl}, k = \{1, ..., N-1\}, l = \{k+1, ..., N\}$. At prediction time, each one of these $N(N-1)/2$ comparators gives a vote v_k to the label y_k and $v_l = 1 - v_k$ to the other label (y_l), then the resulting votes are used to sort the label set. The label with most votes is ranked first, then the label with most votes apart from the first ranked label is ranked second, and so on. Ties are broken arbitrarily (e.g. using a default order or at random). This method is often accurate, but at a prohibitive computational price both for training and prediction, because it uses $\mathcal{O}(N^2)$ comparators. Based on the intuition that many of the corresponding comparisons are redundant and/or unreliable, we propose to pre-select, from a given dataset, a reduced subset of T comparisons to be used by RPC. Once these are selected, a base learner is applied to the **LS** to train a comparator for each one of them, and then used to rank unseen objects.

Let Q^{Full} be the set of all $N(N-1)/2$ comparisons and $Q \subset Q^{Full}$ be a subset of comparisons of size T. We can define a scoring function $S(Q, x, \pi)$ where x represents a vector of attributes for a given object (e.g. one from the training sample) and π stands for the ordering labeled on x. This scoring function S returns a real value which represents the correlation between the predicted ranking $\hat{\pi}$; obtained by combining the predictions of the T comparators corresponding to the comparisons in Q based on the feature vector x; and the labeled rankings π. Obtaining the maximum score means that the prediction $\hat{\pi}$ is compatible with the labeled example π (also quoted to as "control ranking"), i.e. $\forall y_k, y_l \in \pi^i, k \neq l : y_k \succ_{kl}^{\pi} y_l \Rightarrow y_k \succ_{kl}^{\hat{\pi}} y_l$, where $\succ_{kl}^{\pi}$ (and $\succ_{kl}^{\hat{\pi}}$) is the preference relation between y_k and y_l as given in π (respectively in $\hat{\pi}$) while a minimum score means that the reverse sequence of the prediction is compatible with the control ranking. An average score (i.e. the mean between the maximum and the minimum score) means that the prediction has no correlation with the control ranking.

To compute the performance of the predictions on a given sample, we average the scores $S(Q, x^i, \pi^i)$ for all objects o^i in the sample. We denote by $S_{\mathbf{LS}}(Q)$ (and $S_{\mathbf{TS}}(Q)$) the performance of the predition, when using the subset of comparisons Q, on the objects in the **LS** (respectively **TS**).

The aim of the algorithms developed in this chapter is to find an integer value $T << \frac{N(N-1)}{2}$ and a corresponding subset of comparisons Q such that :

$$\begin{aligned} S_{\mathbf{TS}}(Q) &\geq S_{\mathbf{TS}}(Q^{Full}) - \varepsilon, && (4.1)\\ T &<< \frac{N(N-1)}{2}. && (4.2) \end{aligned}$$

Although we do not define ε explicitly, we could reformulate the goal as: "Find a value T and a subset $Q \subset Q^{Full}$ of size T such that T is significantly smaller than $N(N-1)/2$ and that the accuracy of the predictions, based on a given scoring function $S_{\mathbf{TS}}(Q)$, when using this set Q is not significantly lower than the accuracy of the predictions when using all pairwise comparators."

4.1.2 Sparsity in partially ranked datasets

In a given dataset, if each π^i is a permutation of the whole set of labels, then this dataset is said to be completely ranked, otherwise it is only partially ranked. Typically, when N is large, the dataset is partially ranked.

In partially ranked datasets, most comparisons are only defined for a subset of objects which is smaller than the set of all objects in the dataset. In general, these objects are not uniformly distributed across all comparisons, hence some comparisons might not be verified by any object (we will define them as "empty comparisons"), thus some comparators cannot be trained (quoted "empty comparators").

We did not find a solution to the problem of empty comparisons in the literature. We need to find a procedure to deal with these empty comparisons in such a way that the predicted orderings, when aggregating the comparators outputs, remain as accurate as possible. We will discuss our proposed solutions in Section 4.3.

4.2 Scoring a ranking scheme on a sample

Let us first describe how we compute a score to assess the quality of a given ranking scheme based on a subset of comparisons Q and a sample $\mathbf{LS} = \{(x^i, \pi^i)\}_{i=1}^n$. We assume that for each comparison $q_{kl} \in Q$ we can compute a vote $v_k^i \in [0,1]$ for y_k (and $v_l^i = 1 - v_k^i$ for y_l) for any $(x^i, \pi^i) \in \mathbf{LS}$. We can compute v_k^i in three ways. Firstly, we can build a comparator $f_{kl}(x)$ based on the **LS** and use this model to perform a prediction on the preference of y_k with respect to y_l for the feature vector x^i in the form of a binary decision, thus $v_k^i \in \{0,1\}$. Secondly, we can build a model, in a similar fashion, which provides a class-probability estimation, thus v_k^i would be a decimal value in [0,1] which represents the probability that y_k is preferred over y_l. Finally, we can construct our votes by scanning π^i and, for instance, give the value of 1 to v_k^i if y_k is preferred over y_l in π^i, and 0 otherwise.

For any $(x^i, \pi^i) \in \mathbf{LS}$, we start by initializing the votes of all labels in $\mathbf{Y}$ to 0. We then scan Q, and for each of its comparisons we add the vote v_k^i (respectively v_l^i) given by the method to that of y_k (respectively y_l). Then, we consider π^i, and compute its rank correlation measure $S(Q, x^i, \pi^i)$ with the ranking of its subset of labels computed by the voting scheme. Finally, these correlation measures are averaged over the sample **LS** to yield the overall **ranking score** $S_{\mathbf{LS}}(Q)$ of the ranking scheme Q. Notice that a value of 1 of this ranking score would mean that the ranking scheme Q is able to produce orderings which are compatible with all the partial rankings $\{\pi^i\}_{i=1}^n$ given in our sample.

In the literature, two major correlation estimation functions arise, namely, Spearman Rank Correlation [Spe04] (also quoted to as SRC, ρ, or rho) and Kendall Tau [Ken38]. These two functions are defined by

$$(\pi, \pi') \mapsto 1 - \frac{6 \sum_{i=1}^{\#\pi} (\pi(i) - \pi'(i))^2}{\#\pi((\#\pi)^2 - 1)},$$

for Spearman rank correlation and

$$(\pi, \pi') \mapsto 1 - \frac{4|\{(i,j)|(i < j) \bigwedge (\pi(i) < \pi(j)) \bigwedge (\pi'(i) > \pi'(j))\}|}{\#\pi(\#\pi - 1)},$$

for Kendall tau rank correlation, where π and π' are two rankings over the same subset of labels, $\pi(i)$ represents the position of label y_i in ranking π and $\#\pi$ denotes the number of labels in π. These two criteria differ only a little. They both consider each label in the two rankings and compare their rank but, as opposed to Kendall Tau which counts the number of pairwise disagreements between π and π', SRC considers the rank distance between identical labels in π and π'.

We did not find any paper proving that the former is better than the latter or conversely, but we will further want to compare our algorithms with published results on the same datasets, and those results are expressed in terms of SRC. We will thus use SRC to compute the scoring function $S(Q, x^i, \pi^i)$ between the control rankings and our predictions. Note that this implementation of $S(Q, x^i, \pi^i)$ is consistent with its definition in Section 4.1.1.

In a RPC scheme, the prediction is a permutation of the whole set of labels, but the labeled examples may only contain a subset of labels (i.e. the examples are partially ranked). The Spearman Rank Correlation and the Kendall Tau Rank Correlation can only be applied on two rankings over the same subset of labels. Before computing the correlation measure, we will therefore remove, from the complete ranking predicted, those labels which are not present in the control ranking.

We would like to emphasize the fact that our algorithms used to select Q (which will be presented in Section 4.4) are completely independent of the correlation estimation procedure, and we could also have performed the optimization process on Kendall Tau coefficients or any other function taking two orderings as argument and returning a single numerical value. They are as well independent of the precise way partial rankings given in the **LS** are handled.

4.3 Dealing with sparsely ranked datasets

In most datasets, each π is only an ordered subset of $\mathbf{Y}$, thus each object will not be able to provide any preference information for some, and often for many, comparisons. To cope with this, the standard protocol in RPC (as advocated in [HFCB08]) is to drop, for the training of a particular comparator $f_{kl}(x)$ comparing y_k to y_l, all observations $o^i, i \in \{1, ...n\}$ which do not provide explicitly the information about the preference of label y_k over y_l in π^i. However, in many cases, the datasets are so sparse that several pairwise comparisons are not provided with any information from the learning sample **LS** (called "empty comparisons") thus making the training of the corresponding comparators (quoted "empty comparators") an impossible task. We therefore consider two approaches so as to counter this issue, and describe them in the two following subsections.

Note that we address the problem of dealing with empty comparisons **before** running our algorithms aiming at selecting the subset Q of comparisons.

4.3.1 Modeling empty comparisons by dummy models

If we really want to model all $N(N-1)/2$ possible comparisons, we could represent an empty comparison by a constant model which would always give a vote of 0.5, independently of the input vector of attributes, providing half a point to each label that it compares.

The use of such dummy models is supposed to set unseen labels near the center of the predicted ordering. Since we evaluate the rank correlation by computing the mean square distance between labels, we hope to reduce this distance on most cases.

4.3.2 Dropping empty comparisons

The solution of dropping an empty comparison (and its corresponding comparator) is close to the original RPC algorithm in the sense that RPC removes uninformative objects, and this solution also removes uninformative comparisons. It states that, in the event where it is impossible to find any object o^i $\forall i \in \{1, ..., n\}$ which could provide preference information about y_k vs y_l during base model construction, the corresponding pairwise comparison q_{kl} would not be considered at all, i.e. Q^{Full} is replaced by one of its subset Q'^{Full}. Note that we will further use "Q^{Full}" to represent both Q^{Full} and Q'^{Full} in the description of our comparison selection and comparator training algorithms since they can work on both.

Notice that the idea of dropping a priori from Q^{Full} all comparisons for which no explicit supervision information is provided in the **LS** can be slightly extended. Indeed, we could replace it by requiring that for a comparison to be kept as a candidate member of the ranking scheme, it has to correspond to a minimum number of objects in the **LS** for which explicit information is given about this comparison. This idea is explored in the next chapter as a way of pre-trimming the set of candidate comparisons.

4.4 Comparison selection algorithms

In this section, we will describe the algorithms that we propose to use in order to select a subset of comparisons Q of a priori given size T. We formulate an optimization problem and solve it using different heuristics[1]. Most of these heuristics search for the optimal solution by considering one (or several) set(s) of comparisons, compute the ranking score(s) $S_{\mathbf{LS}}(Q)$ for this (these) scheme(s), then use this (these) ranking score(s) in order to compute the new set(s) which will be evaluated in the next iteration. Since the evaluation process of a given scheme will be repeated frequently, particular attention should be given to the implementation of this evaluation process. In the next subsection, we will define how we chose to evaluate a set of comparisons during the optimization phase. Sections 4.4.2 to 4.4.5 then describe four algorithms aiming to find near-optimal solutions to the problem of selecting Q.

4.4.1 Evaluating a set of comparisons during optimization

In order to identify a subset Q of T pairwise comparisons we formulate an optimization problem which uses only the output information $\{\pi^i\}_{i=1}^n$ from the learning sample **LS**. For a given scheme Q, and for a given π^i, this method computes for each label pair (y_k, y_l) corresponding to $q_{kl} \in Q$ a vote v_k^i which value is 1 if y_k is ranked before y_l in π^i, 0 if y_l is ranked before y_k in π^i and 0.5

[1]A heuristic is an algorithm aiming at finding in a computationally efficient way a (nearly) optimal solution to a given problem. Typically, a heuristic does not guarantee that it will find the optimal solution. The use of a heuristic is justified when no efficient exact solution of the optimization problem is available.

if at least one of the two labels (y_k, y_l) does not appear in π^i .

Thus, instead of training T comparators on the **LS** in order to compute the votes v_j^i, $\forall j \in \{1, \ldots, N\}$ and use these votes to construct a predicted ranking $\hat{\pi}^i$ on the **LS** objects, we use (as a proxy) the labeled ranking π^i given as a supervision in the **LS** to extract our votes, and compute the ranking ${\pi^*}^i$ using the $2T$ votes. Indeed, if we used comparators trained on the **LS** instead of these given comparisons to construct the predicted ordering, we would expect the base-learners to predict the votes v_j^i on the **LS** in a very accurate manner. Therefore, these trained models would with high-probability provide a predicted ranking $\hat{\pi}^i$ which would be very strongly correlated with the ${\pi^*}^i$ proxy derived directly from the supervision information. Hence, optimizing Q according to the ranking score with respect to ${\pi^*}^i$ rather than to $\hat{\pi}^i$ should not drastically change the final outcome.

Using this proxy, allows to compute the **ranking score** $S_{\mathbf{LS}}(Q)$ of the ranking scheme Q on the **LS** independently of the base-learners used subsequently and without training these comparators, which greatly reduces the computing times required for selecting the comparisons.

Next we describe the comparison subset selection algorithms that we have investigated. They will aim at finding a subset $Q \subset Q^{Full}$ maximizing $S_{\mathbf{TS}}(Q)$, by optimizing Q according to $S_{\mathbf{LS}}(Q)$ computed by our proxy.

4.4.2 Pure Random selection (PR)

In this approach we actually do not take advantage of the **LS** outputs. We merely pick at random T comparisons without replacement from the uniform distribution. We use a binary tree in order to save this distribution into memory, such that both drawing a comparison and updating the distribution to remove this comparison (in order to ensure that a comparison is not drawn twice in the same set Q) is performed in $\mathcal{O}(\log_2(N^2))$ operations (see Section A.1 for more details about this implementation). This very simple comparison selection technique is called PR in the sequel.

This trivial approach seems useless, but it has in fact three advantages.

1. This is the fastest possible method, as no hard computations must be performed;

2. The **LS** outputs will not be stored. These outputs will be required during the model construction phase (for prediction), but only the T corresponding outputs will have to be considered in that phase, rather than $N(N-1)/2$. It will thus save memory;
3. We will be able to see how redundant the data are, and what we can globally expect from the method. It thus provides a baseline which can be compared to further improvements in order to estimate the gain of accuracy.

We will express the complexity of the PR algorithm (and subsequent ones) with respect to the number of times that the Spearman's ρ function is called during the set selection phase. This method has a time complexity of 0, in terms of the number of Spearman's ρ evaluations, and is described by algorithm 12.

Algorithm 12 Pure Random Selection (PR)

```
Given: A set $Q^{Full}$ of binary comparisons, an integer value $T \leq \#Q^{Full}$

Initialize $P$ to the uniform distribution over $Q^{Full}$
$Q \leftarrow \emptyset$
for $i = 0 \rightarrow T-1$ do
    $q \leftarrow$ randomly selected comparison in $Q^{Full}$ via $P$, without replacement
    $Q(i) \leftarrow q$
    $Q^{Full} \leftarrow Q^{Full} \setminus \{q\}$
end for
return $Q$
```

4.4.2.1 Imposing a full class coverage

Let us denote by C_Q the coverage of the ranking scheme Q over the set of labels $\mathbf{Y}$. This coverage is defined as the number of labels which are compared at least once to another label by a comparison in Q. We want C_Q to be equal to N, i.e.we want that for each label $y_k \in \mathbf{Y}, k \in \{1, ..., N\}$ there exists a comparison $q_{kl} \in Q$ such that y_k is compared to another label $y_l, l \neq k$. If this condition is not respected, some class labels are never given a chance to be well ranked, as they are never compared to other labels in the subset Q.

During the random selection process, we will impose that each label y_k is compared at least once to another label y_l by a comparison in Q. We start by setting the count of occurrences of each label to the value of 0. Then, each time a comparison q_{kl} is randomly drawn, we scan the table of occurrences. If y_k or y_l has an occurrence count of 0, then q_{kl} is added in Q and the occurrences of both y_k and y_l are set to 1. Otherwise, the comparison q_{kl} is rejected.

In the case where we are dropping empty comparisons (see Section 4.3.2) and, if for any set Q of size T, finding a set of comparisons $Q \subset Q^{'Full}$ such that every label in $\mathbf{Y}$ is compared at least once to another label is impossible, then this full coverage condition falls.

The feasibility of imposing the full coverage condition is determined at first offline (i.e. independently of the comparison selection algorithm and in an asynchronous manner) by counting the occurrences of each label when using all available comparisons in $Q^{'Full}$. This test is performed for each **LS** that we will consider in the future (and for each dataset). If the condition is impossible to satisfy when using all available comparisons in $Q^{'Full}$, it will also be impossible to satisfy when using a smaller subset of comparisons. The feasibility is then determined online (i.e. during the set selection) by dropping the condition if all available comparisons were randomly drawn once, and yet $\#Q < T$.

The pseudo-code description of this algorithm is given in Algorithm 13.

Algorithm 13 PR by imposing full coverage

```
Given: A set $Q^{Full}$ of binary comparisons, an integer value $T \leq \#Q^{Full}$

Initialize $P$ to the uniform distribution over $Q^{Full}$
$Q \leftarrow \emptyset$
$R \leftarrow \emptyset$
$possible \leftarrow true$
for $i = 0 \rightarrow T - 1$ do
  $q_{kl} \leftarrow$ randomly selected comparison over $y_k$ and $y_l$ in $Q^{Full}$ via $P$, without replacement
  if $possible = false$ or $C_Q = N$ or $C_{Q \cup \{q_{kl}\}} > C_Q$ then
    $Q \leftarrow Q \cup \{q_{kl}\}$
  else
    $i = i - 1$
    $R \leftarrow R \cup \{q_{kl}\}$
  end if
  $Q^{Full} \leftarrow Q^{Full} \setminus \{q_{kl}\}$
  if $Q^{Full}$ is empty then
    $possible \leftarrow false$
    $Q^{Full} \leftarrow Q^{Full} \cup R$
  end if
end for
return $Q$
```

4.4.3 Estimation of Distribution Algorithm (EDA)

4.4.3.1 General principle of EDA

The aim of the estimation of distribution algorithm [LL01], also quoted to as EDA, is to compute a distribution P over the solutions to a given problem, taking into consideration that, in some cases, solutions are built by combining several objects. The distribution P is iteratively updated. At each step i, the distribution P_i is used to generate s solutions (each solution might consist of several objects). The distribution P_{i+1} is derived by attributing more weight to the (objects which were comprised in the) top b solutions out of the s solutions, according to a scoring scheme S. The distribution update process is repeated j times.

The outcome of the EDA algorithm is strongly linked to the value of s, b and

j. Parameter s controls the sampling level and should be chosen high enough such that the drawn sample is representative of the current distribution P_i. However, the higher this value, the more computations have to be performed. Parameter b controls the sensitivity/specificity level. Low values of b will make the process very selective, with a high chance of missing less good but still interesting solutions, but the retained solutions will be excellent. Conversely, high values of b will make EDA consider less good solutions and make them more likely to be drawn in the next iteration, but will lower the chance of missing an interesting solution. In our implementation, the computational complexity of the update of P_{i+1} after obtaining the s evaluations also depends on b (if b is low, the update will be faster than if b is high). Finally, parameter j sets the computational effort. Too high values will make the computational time unnecessarily high when the probability distribution has converged with less than j iterations, while too low values will stop the algorithm before convergence.

Figure 4.1 gives a good example of what the method can produce. At each step i, a set of 12 objects X_s are drawn according to distribution P_i. The 6 objects X_b minimizing $f(x)\ \forall x \in X_b$ are used to estimate P_{i+1}. Each iteration will make the set of sampled objects X_s get iteratively closer to the optimum O.

4.4.3.2 Application of EDA to comparator subset selection

In our context, this algorithm first starts by assigning a uniform distribution P_0 over the comparisons in Q^{Full}, then iteratively updates this distribution by deriving P_{i+1} from the top b out of s comparison sets of size T drawn from P_i, and stops after j iterations. The EDA method is sketched in Algorithm 14. Note that we will use EDA to find sets Q which **maximize** the ranking score while we showed an example of a minimization problem. Conceptually, this changes only a little. Instead of considering items with the lowest scores (according to the scoring scheme S) to compute the new distribution P_{i+1}, we simply consider items with the higher scores.

At each iteration, P_{i+1} is computed as follows: the number of occurrences of each comparison in the b out of s comparison sets are kept. For each comparison $q_j \in Q^{Full}$, let us denote by $\mathcal{N}_j$ a value which is comprised between 0 and b, representing the number of sets in which q_j appears in the top b sets. The probability to draw a comparison q_j at iteration $i+1$ is given by

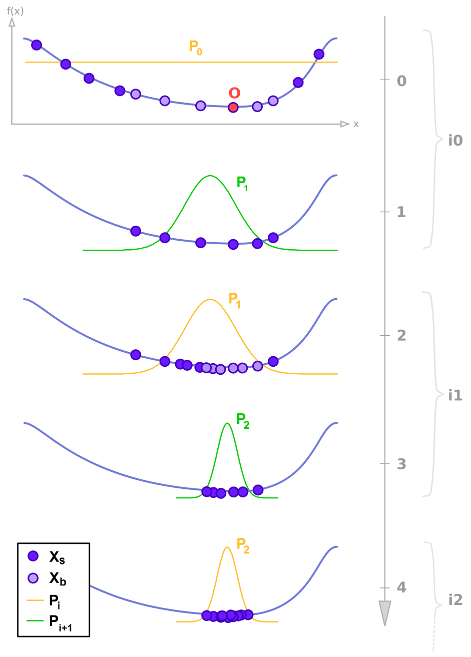


Figure 4.1: Source : Wikipedia. Image : Johann Drého.
The EDA algorithm iteratively reduces the state space to keep interesting elements. Here a minimization problem.

$$P_{i+1}(q_j) \quad = \quad \frac{\mathcal{N}_j}{\sum_{k=1}^{\#Q^{Full}} \mathcal{N}_k}. \tag{4.3}$$

EDA should normally allow candidates which were not yet drawn in previous iterations. But, using our algorithm, it would be impossible to draw a comparison which would not have been considered in previous iterations. To counter this issue, we set $\mathcal{N}_j$ to 1 when q_j was not in any of the s drawn sets, then use these new $\mathcal{N}_j$ to compute P_{i+1}. We thus prevent a comparison to be discarded because it was not drawn at the first iteration.

Since EDA uses PR to draw its s sets, the full class coverage property is conserved (if possible).

The time complexity of the EDA method is nsj Spearman evaluations and the corresponding pseudo-code is given in algorithm 14.

Algorithm 14 Estimation of Distribution Algorithm (EDA)

```
Input: dataset LS, $s$, $b$, $j$, $T$, $Q^{Full}$
Output: set $Q$ of $T$ comparisons
Initialize $P$ to the uniform distribution over $Q^{Full}$
for $i = 1$ to $j$ do
  Draw $s$ sets of $T$ comparisons via $P$, without replacement, using PR
  Keep the best $b$ sets according to their ranking score $S_{\mathbf{LS}}(Q)$
  Set $P$ based on the counts of occurrence of kept comparisons
end for
$Q$ = best set of $T$ comparisons found along all iterations
```

4.4.4 “Exhaustive” Greedy Search (EGS)

One manner of finding the best subset $Q \subset Q^{Full}$ according to the ranking score $S_{\mathbf{LS}}(Q)$ is to consider all possible subsets of size T, rank them according to their ranking score, then select the best subset. The problem of this approach is that the search space contains $C^T_{\#Q^{Full}}$ possible subsets[2], and large values of N, in combination with medium values of T (with respect to $N(N-1)/2$), will make this solution infeasible in practice. A very well known method to deal in a heuristic way with this issue is the greedy search approach.

4.4.4.1 General principle of greedy algorithms

The greedy approach consists in (e.g. randomly) selecting an already considered object in the search space, and then navigating in this space by selecting (e.g. randomly) a neighbor among those which locally improve the performance as much as possible. The size of the neighborhood has an influence on the performances (both speed and accuracy). One of the extreme cases would be to consider the whole search space as neighborhood, which would maximize the accuracy but is equivalent to an exhaustive search. The other extreme is to consider only the randomly selected object in the neighborhood, maximizing

[2] $C^k_n = \frac{n!}{k!(n-k)!}$

the speed and equivalent to a pure random selection.

In general, with a greedy approach, the true optimum is not guaranteed to be reached because the union of all neighborhoods will typically not cover the whole search space and, depending on the heuristic, the optimum might not be present in this union of neighborhoods. Greedy algorithms iteratively perform local optimizations, but the choices which are locally optimal are not necessarily globally optimal. For instance, Figure 4.2 shows an example of a search space with the global minimum on the right and a local minimum on the left. The reached minimum depends on the starting point and, in the example, dots connected to red arrows will find the true optimum while dots with black arrows will be stuck on the local minimum. Moreover, if we consider that the neighborhood only contains the closest neighbors, the two dots at the extreme left or right will also be stuck in their respective positions since their closest neighbors do not improve the score measure. One way to bypass this local minimum problem is to repeat the process a certain number of times and randomize the starting point, so that the true optimum has a higher chance to be reached in at least one of these runs.

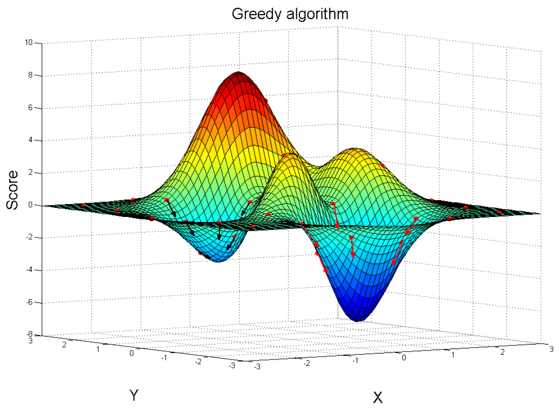


Figure 4.2: Dots connected to red arrows will find the true optimum while dots with black arrows will be stuck in a local minimum

4.4.4.2 Application of greedy algorithms to comparison subset selection : EGS

The greedy approach can be applied to our trimming problem. We first consider the "exhaustive" greedy search (EGS) described by Algorithm 15. At each step, this method screens all possible substitutions of a comparison q in Q by another one (q', not in Q), and keeps the best alternative in terms of ranking score improvement. We are thus performing a maximization problem rather than a minimization problem, but the principle remains the same. The advantage of EGS is to be faster than evaluating all possible combinations while approaching the best possible set of comparisons. However, this approach is still quite slow and this method sometimes leads to a local maximum which can be quite far from the global maximum (or, at least, from the best observed local maximum).

When validating this algorithm, we will run it several times and report averages and standard deviations of the scores $S_{\mathbf{TS}}$ obtained over these runs.

The process of replacing each comparison q in Q is repeated j times. This j value is not directly controlled but usually varies between 10 and 20 at each run in our experimental setup. The iteration process stops when the ranking score of Q_i at iteration i is equal to the ranking score of Q_{i+1} at iteration $i+1$.

We chose to discard the full class coverage property (except for the first PR selection) because EGS computes the ranking score S at each comparison swap (in opposition to EDA which redraws T comparisons before computing $S_{\mathbf{LS}}(Q)$), and we want to select the swap which maximizes the score, even if this swap will disable the full class coverage property.

The EGS method has a time complexity of $njT(N(N-1)/2-T)$ Spearman evaluations ($\mathcal{O}(njTN^2)$), and is described in algorithm 15. Note that in this case, the computational time of the Spearman evaluations is independent of T, which is not the case for EDA. Indeed, we can store the table of label votes for a given set Q, then update this table by removing the effect of comparison q and add the effect of comparison q', which can be done in constant time. More details about this implementation can be found in Section A.2.4.

Algorithm 15 Exhaustive Greedy Approach (EGS)

```
Input: dataset $\mathbf{LS}$, $T$, $Q^{Full}$
Output: set $Q$ of $T$ comparisons
Initialize $P$ to the uniform distribution over $Q^{Full}$
Set $Q$ by uniformly drawing $T$ comparisons from $P$ without replacement using PR
Set $S$ = ranking score of $Q$ computed from $\mathbf{LS}$
while the score $S$ improves do
  for each $q \in Q$ do
    for each $q' \notin Q$ do
      Compute the ranking score $S^*$ of $Q^* = (Q \setminus \{q\}) \cup \{q'\}$ from $\mathbf{LS}$
      if $S^* > S$ then
        set $Q$ to $Q^*$ and set $S$ to $S^*$
      end if
    end for
  end for
end while
```

4.4.4.3 Alternative approach for EGS

As an attempt to reduce the local maxima impact, we tried to improve the EGS method by iteratively replacing two comparisons simultaneously per iteration. From a logical point of view, this is equivalent to extending the research radius (neighborhood) such that some local minima would be ignored. This extension gave very good accuracy results, as most of the runs provided the same set Q whose ranking score was the highest across all runs, but was catastrophic in terms of complexity. We could only test this approach on a very small database where $N = 10$, thus $N(N-1)/2 = 45$. Yet the computational time was approximately 10 minutes for $T = 10$ while it was in the order of the second with the basic EGS for the same value of T. Indeed, the time complexity of this method is $njT(T-1)(N(N-1)/2-T)(N(N-1)/2-T-1))$ Spearman evaluations $(\mathcal{O}(njT^2N^4))$. We thus rejected this approach.

4.4.5 Randomized Greedy Algorithm (RGS)

Unfortunately, each iteration of the EGS algorithm needs to evaluate $\mathcal{O}(njTN^2)$ rank comparisons, which is unfeasible in practice for large label sets. A classical strategy to reduce the computational burden of exhaustive search consists in combining it with randomization [CLR90]. We thus modified the EGS algorithm as follows: instead of considering all possible swaps, Algorithm 16 randomly determines a candidate swap at each iteration and accepts it if the score is improved. Otherwise, the previous set is kept for the next iteration. In both cases, the sampling probability of the worst comparison is reduced so as to reduce the probability of re-considering it in subsequent iterations. Let S_q denote the ranking score on the **LS** when using the comparison q in Q, instead of q'. In the same manner, we denote $S_{q'}$ the ranking score when using q' in Q instead of q. Considering that q is a worse comparison[3] than q', we compute $P_{i+1}(q)$, the probability to draw comparison q at iteration $i+1$, by

$$P_i(q) * \frac{\frac{S_q+1}{2}}{\frac{S_{q'}+1}{2}}.$$

Each other probabilities remain unchanged ($P_{i+1}(q')$ included). We then normalize P_{i+1} in such a way that $\forall q_j \in Q^{Full}, \sum_{j=1}^{\#Q^{Full}} P_{i+1}(q_j) = 1$. We repeat the process of swapping comparisons until stopping conditions are met. In this study, we chose to perform at least j_1 iterations and to stop when either the last j_2 iterations did not improve the score or when j_3 iterations have been performed. We call this method RGS (standing for "randomized greedy search").

The analogy in our geometrical point of view corresponds to a search with the same radius around the starting point, but only one random neighbor is considered at a time. Hence the neighbor minimizing the error is not guaranteed to be selected. We only require the neighbor to perform a local improvement in order to move to that point. In some sense, this can be seen as a two-level randomization process, where the starting point is chosen within a uniform distribution and the neighbor object is selected using a weighted distribution.

We chose to drop the full class coverage property (except for the first PR selection) for the same reason that we dropped it from EGS, namely because if a comparison swap disables the full class coverage property but improves the ranking score, we wish to keep this swap.

The complexity of the RGS method is $n(j_1 + j_3)/2$ Spearman evaluations and the corresponding pseudo code is given in Algorithm 16. We chose j_1

[3]If q' is worse than q, replace "q" by "q'"' in the sequel of the section, and conversely.

and j_3 in such a way that the number of Spearman evaluations is close to the number of Spearman evaluations when using the EDA selection algorithm (i.e. $(j_1 + j_3)/2 \approx sj$) in such a way that the comparison between RGS and EDA is equitable.

Algorithm 16 Randomized Greedy Approach (RGS)

```
Input: dataset LS, $T$, $Q^{Full}$, $j_1, j_2, j_3$
Output: set $Q$ of $T$ comparisons
Set the sampling distribution $P$ to the uniform distribution over $Q^{Full}$
Set $Q$ by drawing $T$ comparisons via $P$ without replacement
Set $S$ = ranking score of $Q$ computed from LS
while stopping conditions($j_1, j_2, j_3$) are not met do
    Draw $q \in Q$ uniformly
    Draw $q' \notin Q$ according to $P$
    Compute the ranking score $S^*$ of $Q^* = (Q \setminus \{q\}) \cup \{q'\}$ from LS
    if $S^* > S$ then
        set $Q$ to $Q^*$ and set $S$ to $S^*$
        decrease the weight of $q$ in $P$ and re-normalize $P$
    else
        decrease the weight of $q'$ in $P$ and re-normalize $P$
    end if
end while
```

4.5 Models for learning comparators

Once the set Q is determined, the RPC approach consists in using the available set **LS** (both inputs and outputs), for constructing one comparator for each comparison in Q. This can be done by using a classification or a regression method, and by exploiting relevant information from the **LS**. To infer a comparator $f_{kl}(x)$ corresponding to a comparison $q_{kl} \in Q$ for a certain label pair (y_k, y_l) we will use the available $\mathbf{LS} = \{(x^i, \pi^i)\}_{i=1}^{n}$ in order to generate a *comparator training set* $\mathbf{LS}_{k,l} = \{(x^j, y^j_{k,l})\}_{j=1}^{n'}$, where $y^j_{k,l}$ is derived from π^j by attributing 1 to $y^j_{k,l}$ if $y_k \succ_{\pi^j} y_l$ and 0 if $y_k \succ_{\pi^j} y_l$. If none of these preference relations can be extracted from π^j, the corresponding x^j is dropped from $\mathbf{LS}_{k,l}$. In the present section we motivate our choice of the base learner.

The previously depicted selection algorithms are completely independent from the choice of the base learner used to compute the pairwise comparators from the **LS**. However, the more accurate the comparators are, the more accurate our predicted rankings will be and, since this research is based on numerous empirical tests, we also want the base learner to be both fast at the training time and at the prediction time, and as far as possible free of meta-parameters.

The Extra-Trees algorithm [GEW06] meets these requirements. It constructs M randomized trees. In opposition to classical decision or regression trees, the best feature and the best cut-point are not selected at each node. Instead, the method picks K features (attributes) in a random fashion among the locally non constant ones and, for each such feature, a random cut-point in its interval of variation. Then, it selects the couple feature/cut-point which minimizes the impurity of the output variable in the successor nodes (measured either by the conditional entropy, in classification mode, or by the conditional variance, in regression mode [GEW06]) and splits the node, as long as at least $n_{\min}$ objects reached this node and not all features are constant. The pseudo-code for the Extra-trees algorithm is depicted in Algorithm 17.

The algorithm has three meta-parameters (namely K, $n_{\min}$ and M) which have an influence on the models that it computes. A discussion of default settings of these meta-parameters can be found in Sections 3.1, 3.2 and 3.3 of [GEW06]. Parameter K controls the dependence of the tree structure on the output value of the learning sample. In the particular case where $K = 1$, feature selection is completely independent on the output value, and one thus calls such tree ensembles "Totally Randomized Trees". The default setting for classification is $K = \sqrt{\#A}$, and it is $K = \#A$ for regression, where $\#A$ is the number of attributes. The parameter $n_{\min}$ controls the pruning of the trees. With $n_{\min}$ $= 2$, trees are fully grown, which is the default setting for classification. The default value of $n_{\min}$ for regression is 5. Finally, parameter M allows to settle the compromise between accuracy and complexity. The larger M is, the better the model is in principle, but also the slower the model construction and their use for prediction is and the higher the memory requirements for storing the models is. There is thus no default setting for parameter M. The default settings for the meta-parameters K and $n_{\min}$, as cited from [GEW06], *"appear to be robust choices in a broad range of typical conditions"*.

Algorithm 17 Extra-Trees algorithm [GEW06]

```
  **Input:** dataset $\mathbf{LS}$, $M$, $n_{\min}$, $K$
  **Output:** a set $\mathbf{M}$ of $M$ trees
  $\mathbf{M} \leftarrow \emptyset$
  **repeat**
    $\mathcal{M} \leftarrow BuildOneTree(\mathbf{LS})$
    Add $\mathcal{M}$ in $\mathbf{M}$
  **until** $M$ trees have been constructed

  $BuildOneTree(O)$
  **Input:** Input: the local learning subset $O$ corresponding to the node which we want to split
  **Output:** a tree structure $\mathcal{M}$
  $s \leftarrow GetASplit(O)$
  **if** $s =$ nothing **then**
    return a leaf $\mathcal{M}$ labeled with $O$
  **else**
    Separate $O$ in $O_1$ and $O_2$ using the split $s$ such that $O_1$ contains the objects of $O$ which satisfy the split condition of $s$ and $O_2$ containing the objects of $O$ which are not in $O_1$
    $\mathcal{M}_1 \leftarrow BuildOneTree(O_1)$
    $\mathcal{M}_2 \leftarrow BuildOneTree(O_2)$
    Label $\mathcal{M}$ with $s$
    Attach $\mathcal{M}_1$ as left branch of $\mathcal{M}$ and $\mathcal{M}_2$ as right branch of $\mathcal{M}$
    return $\mathcal{M}$
  **end if**

  $GetASplit(O)$
  **Input:** the local learning subset $O$ corresponding to the node which we want to split
  **Output:** a split $[a < ac]$ or nothing
  **if** $StopSplit(O)$ is TRUE **then**
    return nothing
  **else**
    select $K$ attributes $\{a_1, ..., a_K\}$ among all non constant (in $O$) candidate attributes
    Draw $K$ splits $\{s_1, ..., s_K\}$, where $s_i = PickARandomSplit(O, a_i), \forall i = 1, ..., K$
    Return a split $s^*$ such that $Score(s^*, O) = max_{i=1,...,K} Score(s_i, O)$
  **end if**

  $PickARandomSplit(O, a)$
  **Input:** a subset $O$ and an attribute $a$
  **Output:** a split
  Let $a^O_{\max}$ and $a^O_{\min}$ denote the maximal and minimal value of $a$ in $O$;
  Draw a random cut-point $ac$ uniformly in $]a^O_{\min}, a^O_{\max}]$
  Return the split $[a < ac]$
```

Algorithm 17 (continuation) Extra-Trees algorithm [GEW06]

```
Stopsplit(O)
Input: a subset O
Output: a boolean
if $\#O < n_{\min}$ or all attributes are constant in O or the output is constant in O then
  return TRUE
else
  return FALSE
end if
```

When using an Extra-Trees model for prediction, each tree in the ensemble of M trees provides an output value according to the values of the input features of the test object and the terminal node induced by these values, by either returning the majority class among training objects that went to this node (if the tree is used in a classification mode), or the average output value (if it is used in a regression mode). In order to compute a prediction over the ensemble of M trees, the predictions of the individual trees are aggregated in the following way: for regression, the predictions are computed as the average prediction of the individual trees, while for classification the predicted class is computed as the class that receives a majority of votes.

4.6 Hard comparators vs soft comparators

Using all pairwise comparators is indeed time-consuming, but it has the advantage of giving a distinct score to each label between $[0, N-1]$, with no ties (as long as the predicted pairwise relations are transitive and the comparators are built in classification mode). Aggregating this information into a ranking is thus deterministic.

Unfortunately, once the comparator set is being trimmed, even by removing only a single comparator, one eventually has to deal with ties on some labels. And the more comparators are removed, the more ties there are. So, we need to define how to break these ties. The first and trivial approach is to arbitrarily break them (e.g. at random, or using an alphanumerical order on the labels, etc), but another approach could be to use soft comparators instead of hard ones.

In the pairwise comparators construction, we only considered classification mode so far, as we are only provided with a 0/1 vote in the **LS**. But there is no restriction to use the regression mode in the model construction instead of the classification mode. Having a decimal prediction rather than a 0/1 prediction provides class-probability scores, rather than hard binary decisions. Hence, variations in the label score table are more likely to occur, and would probably make the breaking of ties a less arbitrary task. We will thus consider both classification mode and regression mode when constructing our base models.

4.7 Summary

In this chapter, we have motivated and described four algorithms for selecting a subset of comparisons to be used by RPC, namely PR, EDA, EGS and RGS. We have also discussed the ways partial ranking information can be handled and we motivated our choice of the Extra-Trees algorithm, which we will use subsequently as base learner in the RPC framework. The next chapter will analyze the preformances of these methods, in terms of ranking score, by means of an empirical study over three different datasets.

Chapter 5

Results

Contents

5.1 Datasets used for experimental validation 119
5.1.1 OMIB database (synthetic, complete, $N = 10$) . . . 120
5.1.2 Sushi database (real life, partial, $N = 100$) 121
5.1.3 MovieLens database (real life, partial, $N = 1682$) . . 122
5.2 Effect of the model mode in a RPC scheme . . . 123
5.2.1 Using optimized tree parameters 124
5.2.2 Using sub-optimal tree parameters 125
5.2.2.1 Influence of parameter n_{min} 127
5.2.2.2 Influence of parameter K 128
5.2.2.3 Influence of parameter M 129
5.2.2.4 Influence of #**LS** 132
5.3 Evaluation of the PR selection method 133
5.3.1 Effect of imposing a full class coverage 134
5.4 Evaluation of EDA, EGS and RGS 136
5.4.1 Relevance of optimizing with respect to the **LS** outputs137
5.4.2 Ranking score on OMIB 139
5.4.3 Ranking score on Sushi 140
5.4.4 Ranking score on MovieLens 142
5.5 Effect of (meta-)parameters in a trimmed RPC scheme 145
5.5.1 Effect of the model mode 145
5.5.2 Effect of sparsity control 148
5.5.3 Influence of the tree parameters (on RGS) 154

5.5.4 Influence of the default order used in tie breaking . . 158
5.5.5 Removing the object ID from the model construction 161
5.5.5.1 Using a 10-fold cross validation on both Sushi databases 162
5.5.6 Using simulated perfect models 163
5.5.7 Setting the j_1 and j_3 RGS parameters to huge values 167
5.6 Complexity of our set selection algorithms 167
5.6.1 Time Complexity (computational speed) 169
5.6.2 Space Complexity (memory usage) 170
5.7 Concluding remarks 171

In order to validate the comparison set selection algorithms presented in Chapter 4, we performed an empirical survey on three datasets. The first dataset, called OMIB, is a synthetic dataset which can be used to train classification or regression models, and where the numerical output variable is a function of the input variables. A perfect model would thus predict the output variable with 0% error, hence experiments on this set would be performed in a controlled environment. The OMIB dataset contains 8 features (attributes) and 20.000 objects. The second database, called Sushi, is a partially ranked database consisting of 500 objects, where each object is labeled by an ordering on 10 classes out of 100. Sushi has 11 features. The third dataset, called MovieLens, is composed of 100.000 observations, each of these being a triplet (x, y, S) where x is the feature vector of one of the 943 objects, y is one of the 1682 labels and S is an integer value in $\{1, ..., 5\}$ representing the rating of label y by object x. MovieLens has 5 features. More details about the OMIB, Sushi and MovieLens databases and, in particular, about the way we transformed OMIB and MovieLens into label ranking databases will be given in Section 5.1.

The remaining sections are organized as follows : in Section 5.2, we will study the effect of the mode of the base model (classification vs regression) on a RPC-like ranking scheme, where $Q = Q^{Full}$, in terms of accuracy and robustness; we compare the ranking score of the ranking schemes when varying the value of T[1] and using the PR algorithm to select the subset Q in Section 5.3. We evaluate the EDA, EGS and RGS algorithms as well as the relevance of optimizing the subset based on the **LS** ranking scores in Section 5.4. In section 5.5, we investigate the effect of the meta-parameters, as well as the j_1 and j_3

[1]The case where $T = \#Q^{Full}$ being considered as well.

parameters. We evaluate the complexity of each comparison set selection algorithm with respect to the original RPC scheme in terms of space complexity and time complexity in Section 5.6. Finally, Section 5.7 will end this chapter with some concluding remarks.

Across all experiments, each dataset is partitioned into two parts: a learning sample **LS** and a test sample **TS**. The test sample did not vary across all tests (the last 10,000; 50; 143 objects for OMIB; Sushi; MovieLens respectively) and the **LS** contained the remaining objects. When we vary the size of the considered learning sample, we select the first objects. These sizes were selected arbitrarily for OMIB and MovieLens, but we used the same percentage for Sushi as in its related publication ([KKA05])

We evaluate the accuracy of the ranking schemes by their **overall ranking score** $S_{\mathbf{TS}}$ computed on the **TS** using the Spearman's ρ correlation evaluations to compare the labeled ranking π^i and the prediction $\hat{\pi}^i$ for each object $o^i \in \mathbf{TS}$ and calculate the mean value of these correlation measures as $S_{\mathbf{TS}}$. In the case of partial rankings in the test set, we removed, from the prediction, labels which were not present in the control, for a given object, before performing the correlation measure.

The base learners were always modeled using the Extra-Trees algorithm [GEW06].

Because of the huge number of performed tests, we chose to select the set of figures which are required to confirm our assessments and we discarded the others, for readability purpose only. However, the complete set of figures can be found in Appendix B.

5.1 Datasets used for experimental validation

In this section, we will present the three datasets used in our empirical validation, namely OMIB, Sushi and MovieLens. Table 5.1 provides an overview of each dataset. This table contains, for each set, the number of objects (n), the number of attributes ($\#A$), the number of considered labels (N), the minimum (min) and maximum (max) length of the provided rankings π^i, and the ranking score $\overline{S_{\mathbf{TS}}}$ obtained by using the mean ordering $\hat{\pi}$ (learned on the **LS**) for each prediction on the **TS**. The mean ordering $\hat{\pi}$ is obtained by using the generalized borda count discussed in Sections 3.3.5.4 and 3.3.5.5. In this scenario, each π^i in the **LS** is used to generate the mean ordering, and the label on rank $j \in \{1, \ldots, \#\pi^i\}$ receives $(\#\pi^i - j + 1)(N+1)/(\#\pi^i + 1)$ votes. Each missing label receives $(N+1)/2$ votes. According to the Theorem 1 of [CH09], this procedure will produce a mean ordering π^i whose sum of squared rank distances is minimal.

Database	n	$\#A$	N	min	max	$\overline{S_{\mathbf{TS}}}$
OMIB	20.000	9	10	10	10	0.609.
Sushi_0	500	11	100	10	10	0.319.
Sushi_3	500	11	100	10	10	0.679.
MovieLens	943	5	1682	20	736	0.467.

Table 5.1: General informations about each considered dataset.

5.1.1 OMIB database (synthetic, complete, $N = 10$)

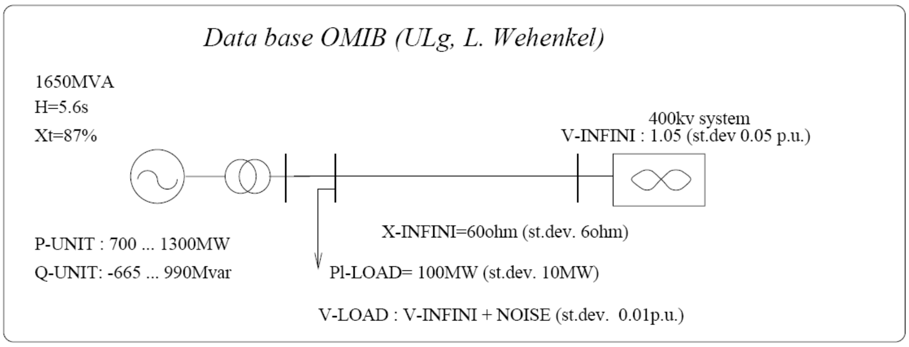

Data base OMIB (ULg, L. Wehenkel)
1650MVA
H=5.6s
Xt=87%
400kv system
V-INFINI : 1.05 (st.dev 0.05 p.u.)
X-INFINI=60ohm (st.dev. 6ohm)
P-UNIT : 700 ... 1300MW
Q-UNIT: -665 ... 990Mvar
Pl-LOAD= 100MW (st.dev. 10MW)
V-LOAD : V-INFINI + NOISE (st.dev. 0.01p.u.)

Figure 5.1: OMIB represents the stability of an electrical system

The OMIB database is related to a power system, depicted on Figure 5.1, which can be secure or not with respect to its transient stability [Weh98].

It contains 7 input features (all numerical; PU, QU, XINF, VINF, PL, VL, ObjectID), two output features (one numerical; CCT-SBS; and one symbolic;

SECURITY) and 20,000 objects. Using some features describing these objects, one can try to predict the degree of stability of the power system by estimating the Critical Clearing Time (quoted to as CCT-SBS, in the sequel). Larger values of CCT-SBS correspond to more stable situations (the system is secure if CCT-SBS > 0.155). 30.915% of the objects are labeled with the INSECURE class, the rest being labeled with the SECURE class.

To transform the OMIB dataset into a ranking dataset, we created 10 equal-size bins according to the CCT-SBS value whose interval of variation is $[0.005, 0.495]$. We set the size of each bin to 0.05, the first bin starting at 0.0 and the last bin ending at 0.5. Each object always prefers the bin in which its corresponding CCT-SBS value falls into. Then the preference goes to other bins, starting with the closest one and ending with the farthest one. This yields a completely ranked dataset, as depicted in Figure 5.2.

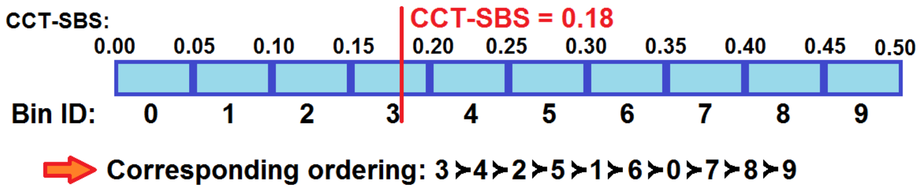


Figure 5.2: Example of the transform of CCT-SBS $= 0.18$ to a ranking of size 10

Note that, considering the artificial aspect of this ranking, the experimental results could be biased, as there exists a relation between bins. An object preferring a bin $\mathcal{B}_i$ over $\mathcal{B}_j$ would always prefer $\mathcal{B}_i$ over all the bins that are farther from it than $\mathcal{B}_j$. Nevertheless, this relation was not considered and, as far as the set selection algorithm could tell, the bins (and thus their associated class labels) are independent.

5.1.2 Sushi database (real life, partial, $N = 100$)

The Sushi real life database [KKA05] contains information on the preferences of Japanese people for various kinds of sushis according to their location. It was built by asking some people to select, among 100 presented sushis, the 10 ones that they liked the most, and then to rank these latter according to their preferences.

The original dataset contained 5000 objects (people) which was reduced to 500 objects by using the k-o's mean clustering algorithm [Kam03] and taking the cluster whose mean square distance to the order mean was the smallest. It uses 11 numerical features to describe the resulting objects, namely the User ID, gender (0 or 1), age bracket ($0 \to 5$), time to fill the questionnaire (in seconds), prefecture ID, region ID and east/west ID of the place where they spent their childhood, prefecture ID, region ID and east/west ID of the place where they currently live, and 0 or 1 for the last attribute whether their currently live in their childhood location or not.

Another version was also extracted from the original dataset. The features remain identical but the 500 objects were selected at random from the 5000-object database. To distinguish those sets, the authors quotes the version of this database with 500 random objects as "Sushi_0" and the one resulting from clustering as "Sushi_3". Both datasets are only partially ranked.

Since the ordering which is provided as a supervision is an ordered subset of the 10 most preferred sushis, each sushi in this ordering should be ranked higher than any other unselected sushi. However, we will discard this information, and reserve this problem for future work (see Section 7.2.1).

Some additional variants of these databases were also created from the two 500 object sets by truncating the labeled ranking and keeping the 2, 5 and 7 first sushis. Another variant where the number of labeled sushis varied across objects (but was still between 5 and 10 for each object) was also created. However, these variants were not taken into account in our experimental validation process and we only considered the two databases (Sushi_0 and Sushi_3) containing 10 ordered sushis per object.

5.1.3 MovieLens database (real life, partial, $N = 1682$)

The MovieLens real life database contains user ratings of movies [GRP73] and exists in 3 versions; for memory usage and computational time reasons, we used the smallest one. It contains 100,000 evaluations, in which a user gives a number of stars (from 1 to 5) to a movie. Each one of the users has evaluated at least 20 and at most 736 movies of the 1682 ones present in the database. In the dataset that we have used, there are 943 different objects (users), and 5 features are used to describe them (4 numerical ; User ID, age, zip code and gender; and 1 symbolic; occupation). Ranks are derived from the stars by breaking ties arbitrarily based on the alphanumerical order (see Figure 5.3 for an example of the transform of ratings to an ordering). This yields a rather noisy and only very partially ranked dataset.

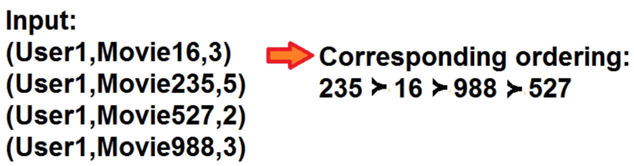


Figure 5.3: Example of the transform of 4 ratings into a ranking of size 4 (MovieLens)

5.2 Effect of the model mode in a RPC scheme

A very simple modification of the RPC setting consists in changing the mode of the base learners. We will investigate the effect of the use of class-probability base learners (i.e. models trained in regression mode) instead of hard classifiers (i.e. models trained in classification mode) in the RPC framework in the particular case where $T = N(N-1)/2$ and when using Extra-trees. A similar analysis has been performed in [HFCB08], when the authors compared the ranking score of RPC when used in classification mode or in regression mode. They obtain probability estimations by applying a sigmoid function to the unthresholded classification outputs [Pla99].

In this experiment, we will consider several values for the tree parameters $n_{\min}$, K and M as well as the number of objects in the **LS**, quoted #**LS**. Each of these values is depicted in Table 5.2. We will use the symbol A to represent the set of candidate attributes, and denote the size of the set A by $\#A$. These tree parameters were explored in an exhaustive manner: each possible combination was taken into account.

Note that, for this test, we only considered the Sushi_3 version of the Sushi database (i.e., the one with the highest published ρ) because Sushi_0 is a noisy and partially ranked dataset, and we believe that testing on this dataset would

Parameter	Values	
n_{min}	2, 5, 10, 15 and 20.	
K	1, $\sqrt{\#A}$ and $\#A$.	
M	25, 50, 100, 200 and 500.	
$\#\mathbf{LS}$	OMIB	5, 10, 20, 40, 80, 150, 300, 600, 1200, 2500, 5000 and 10000.
	Sushi	5, 10, 20, 40, 80, 150, 300 and 450.
	MovieLens	5, 10, 20, 40, 80, 150, 300, 500 and 800.

Table 5.2: Considered values for n_{min}, K, M and $\#\mathbf{LS}$

not provide significantly different informations than testing on the MovieLens database, which is also noisy and partially ranked.

We chose to drop empty comparison at this stage, i.e. we dropped a comparison q_{kl} if no object providing preference information about y_k vs. y_l were available in the $\mathbf{LS}_{k,l}$.

We will first examine the effect of the model mode, in terms of ranking score $S_{\mathbf{TS}}$, when optimizing the tree parameters, then discuss the influence of suboptimal tree parameters.

5.2.1 Using optimized tree parameters

To estimate the accuracy of the models, we performed a simple test, i.e. we kept a part of the database as a test set, which was never used in model construction.

Figure 5.4 shows the maximum values of the Spearman's ρ (on the Y axis) when optimizing the tree parameters (which are given in Table 5.3) across all data sets for the two different methods (hard classification vs. aggregation of soft class-probability estimators). We observe that the two methods are nearly equivalent in terms of ranking accuracy when each method uses its optimal tree parameter values, including the size of the training sample.

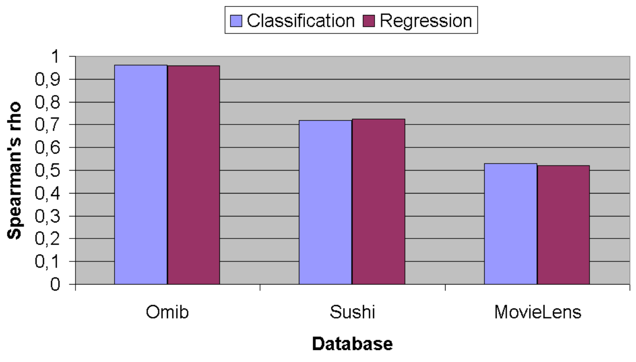


Figure 5.4: Classification and regression have the same performance when using optimal parameters

Database	$n_{\min}$	K	M	#**LS**	Mode
OMIB	10	3	500	10,000	Regr.
Sushi_3	15	3	500	450	Regr.
MovieLens	20	2	500	800	Regr.
OMIB	15	3	500	10,000	Clas.
Sushi_3	15	3	500	450	Clas.
MovieLens	20	2	500	800	Clas.

Table 5.3: Optimal tree parameters for each dataset and for each mode (classification or regression).

5.2.2 Using sub-optimal tree parameters

As we wanted to have an idea of the relative performances of both approaches, regardless of the values of the tree parameters, we thus counted the number of occurrences where the regression method provided a ranking score which was higher than the ranking score obtained using the classification method with the same parameters, and we did this for each point in the parameter state space.

We then divided this number of occurrences by the number of possible parameter combinations, providing a percentage of "wins", which is displayed on

Figure 5.5, where the Y axis is the percentage of tree parameter combinations which makes the regression mode provide better ranking score than the classification mode.

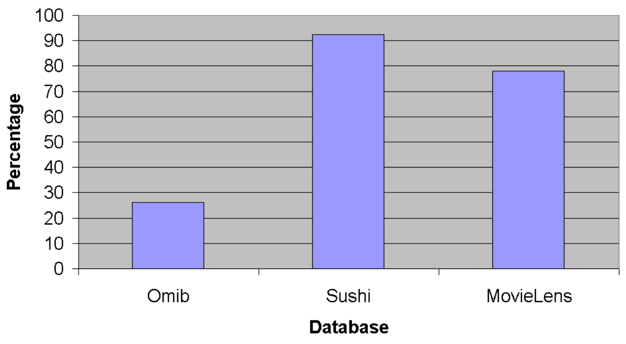


Figure 5.5: When considering sub-optimal tree parameters, regression mode is globally better than classification mode on partially ranked datasets. The Y axis represents the percentage of tree parameters combination which favors regression with respect to classification in terms of ranking score.

We clearly see that the regression method has better results on the datasets with partial rankings than on the one with full rankings. Indeed, we observe even extremely good results as, in the case of the Sushi database, regression tree ensembles beat decision tree ensembles on 92% of the cases.

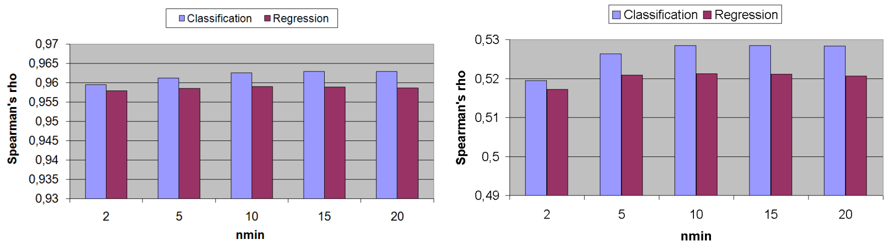
As we expected, the base learners trained in regression mode give on average better results on datasets which contain partial rankings (Sushi and MovieLens), but can also sometimes outperform models trained in classification mode on complete datasets (OMIB), with particular values of the parameters, and especially when the parameters are set so that the built models are quite bad or over-fit the data (small **LS**, large K, small n_{min}, small M).

In order to evaluate the robustness of both modes with respect to each of the tree parameters, we compared the ρ's obtained by a model build in classification mode versus regression mode when varying those tree parameters. For each parameter (namely K, n_{min}, M and #**LS**), we fixed the value of one of them while considering optimal selection for the remaining ones. We then compared how the mode of the model affected the prediction.

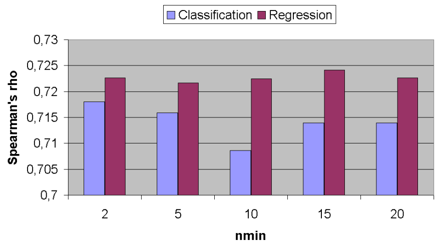
Clearly, the regression mode is more robust (i.e. parameters has less influence) than classification mode on small and partially ranked datasets. Below, we analyze this robustness more in detail.

5.2.2.1 Influence of parameter n_{min}

Our first results showed that the influence of the n_{min} value is marginal on OMIB, has a similar effect on MovieLens when considering both modes, but seemed to have a bigger impact on Sushi when building in classification mode than in regression mode (Figure 5.6). The value of Spearman's ρ is represented on the Y axis, while n_{min} values are on the X axis.



(a) OMIB (b) MovieLens



(c) Sushi

Figure 5.6: The impact of parameter n_{min} can particularly be seen on Sushi.

We will define "the n_{min} effect" as the fact that the n_{min} parameter has more influence on models built in classification mode than on models built in regression mode.

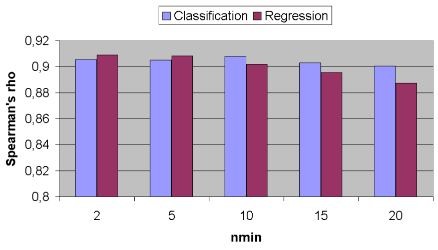
We then considered the differences between each dataset so as to determine a particularity of Sushi which would explain why the n_{min} parameter has more influence with respect to the mode on Sushi than on OMIB or MovieLens. We considered the fact that Sushi is partially ranked, but MovieLens is also partially ranked and the n_{min} effect was not observed on MovieLens. We then considered the fact that Sushi is not very noisy, but OMIB is noise-free and we did not notice any n_{min} effect on OMIB. Finally, we considered the fact that Sushi contains few observations, and this particularity is not shared by OMIB and MovieLens. It is hence possible that the n_{min} effect is more marked on datasets with few observations.

In order to confirm (or infirm) our hypothesis implying that the n_{min} parameter would affect more models built in classification mode than models built in regression mode when built using a small **LS**, we repeated the same experiment, but limited the value of #**LS** to 80 for all datasets, i.e. we took the first 80 objects in each dataset to build the **LS**. This value was determined arbitrarily. The results of this experiment are shown in Figure 5.7. Considering the two partially ranked datasets (Sushi and MovieLens) that we tested, the n_{min} value had much more influence on the Spearman's ρ using classification mode than regression mode. In regression mode, the Spearman's ρ is more stable, with a peak at $n_{\text{min}} = 5$, than in classification mode, where it starts decreasing from $n_{\text{min}} > 2$. Such a difference was not observed using the completely ranked OMIB dataset.

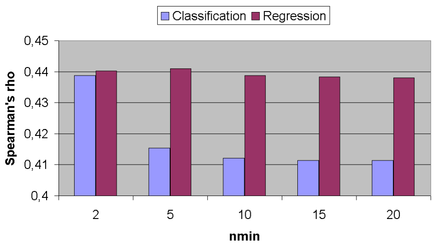
5.2.2.2 Influence of parameter K

We performed similar experiments but varied the K parameter instead of the n_{min} parameter. We also performed two experiments: in one experiment we limited the #**LS** to 80 and in the other one we did not impose this constraint. The observed results are depicted in Figure 5.8, whose axis follows the same structure as in Figures 5.6 and 5.7, but with a different parameter (Spearman's ρ on the Y axis, value of parameter K on the X axis).

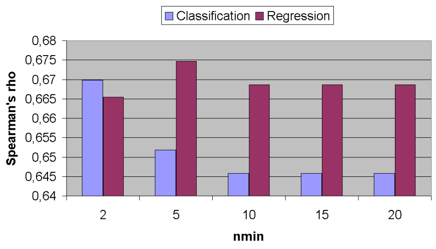
The K parameter seems to act similarly on both methods. The only significant observable event is that using $K = 1$ (i.e. totally randomized trees) gives worse results on the completely ranked OMIB dataset, trimming the learning



(a) OMIB, imposing #**LS** ≤ 80



(b) MovieLens, imposing #**LS** ≤ 80



(c) Sushi, imposing #**LS** ≤ 80

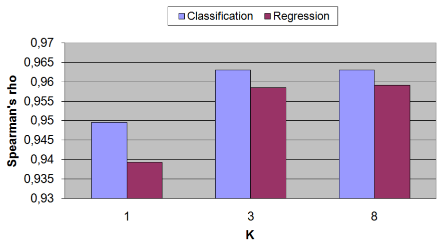
Figure 5.7: The influence of parameter n_{min} on partially ranked datasets is stronger when limiting #**LS** to 80 than without this limitation.

set sample or not. This can be explained by the fact that OMIB contains an attribute "Object ID", whose value is not correlated to the output variable CCT-SBS, output variable that we used to construct our orderings. In the case of $K = 1$, this attribute has a chance to be selected for node split, while with greater values of K, this attribute would be less likely to be selected, hence reducing the number of irrelevant splits. For partially ranked datasets Sushi and MovieLens, parameter K has little influence.

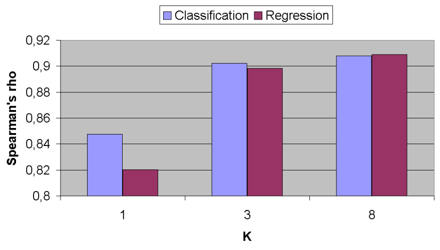
5.2.2.3 Influence of parameter M

We performed two similar experiments (limiting #**LS** to 80 or not) with respect to the M parameter. Figure 5.9 shows the observed results.

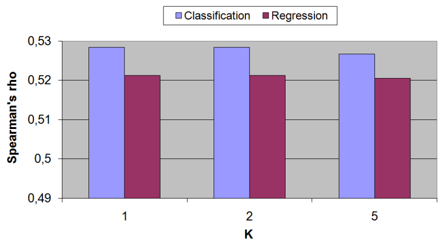
The number of trees does not have a significant effect, for every considered combination of full ranking, partial ranking, small data set and big data set,



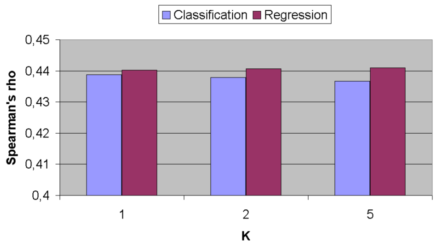
(a) OMIB, no constraint on #**LS**



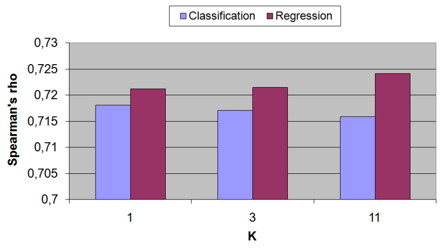
(b) OMIB, imposing #**LS** ≤ 80



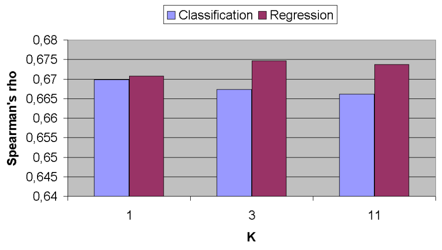
(c) MovieLens, no constraint on #**LS**



(d) MovieLens, imposing #**LS** ≤ 80



(e) Sushi, no constraint on #**LS**

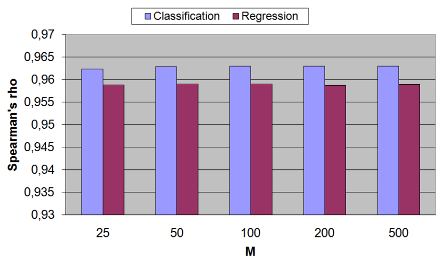


(f) Sushi, imposing #**LS** ≤ 80

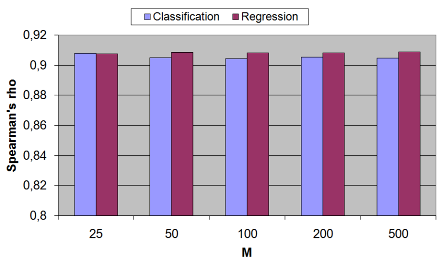
Figure 5.8: The influence of parameter K is equivalent for both modes.

which only means that nearly optimal values of ρ are reached using a very small number of trees for these particular data sets, which are thus not very demanding in terms of the number of trees.

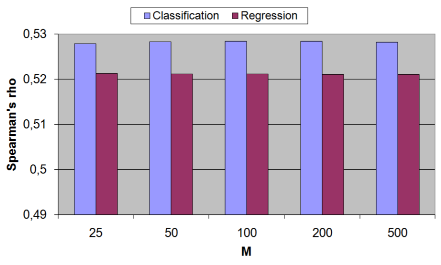
In order to quantify the magnitude of this effect, we divided the ranking score $S_{\mathbf{TS}}$ when using $M = 25$ trees by the ranking score when using $M = 500$



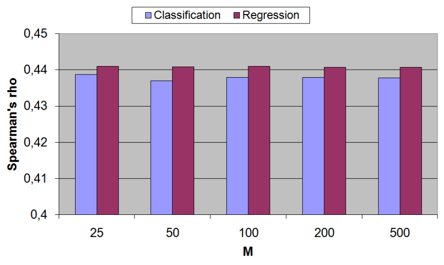
(a) OMIB, no constraint on #**LS**



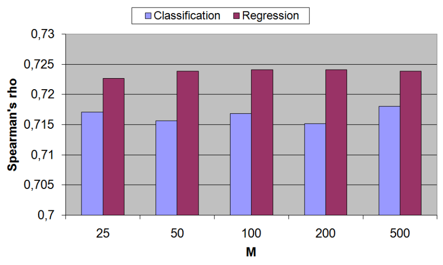
(b) OMIB, imposing #**LS** ≤ 80



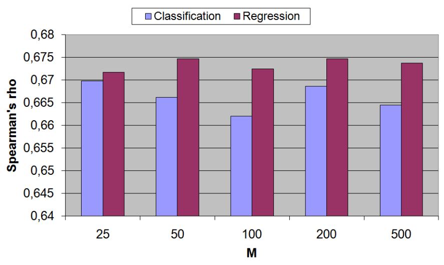
(c) MovieLens, no constraint on #**LS**



(d) MovieLens, imposing #**LS** ≤ 80



(e) Sushi, no constraint on #**LS**



(f) Sushi, imposing #**LS** ≤ 80

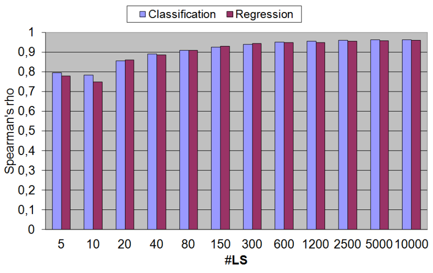
Figure 5.9: The influence of parameter M is marginal on a full RPC scheme.

trees. For both modes and across each datasets, this quotient was superior to 99.8% and was even slightly over 100% for OMIB in regression mode.

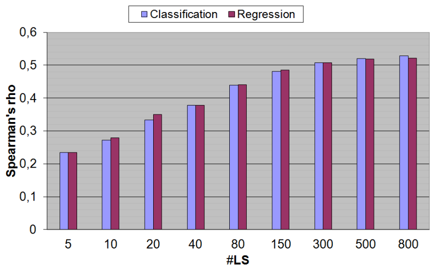
We could also explain this phenomenon by considering that the aggregation phase (i.e. the transform of pairwise comparators into an ordering) induces a level of "smoothness" with respect to the predictions. For instance, if we consider the predictions of q_{ij}, q_{ik} and q_{jk} to be equal to 0.8;0.7;0.2 respectively, the produced ordering $i \succ k \succ j$ would be similar to the ordering produced by $q_{ij} = 0.7$, $q_{ik} = 0.55, q_{jk} = 0.3$. Thus, the accuracy of the base learners have a moderate influence on a full RPC scheme, which would explain why the ranking score $S_{\mathbf{TS}}$ would be acceptable, even with $M = 25$.

5.2.2.4 Influence of #LS

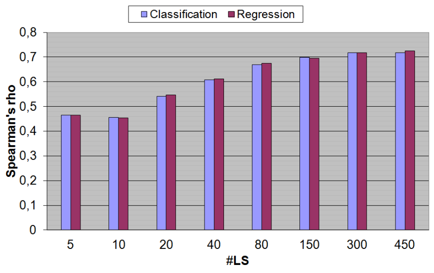
Finally, we performed the same experiment with respect to #**LS**. Of course, the variant with the constraint #**LS** ≤ 80 was not taken into account. Figure 5.10 shows the observed results.



(a) OMIB



(b) MovieLens



(c) Sushi

Figure 5.10: The ranking scores of the predictions start to degrade when #**LS** ≤ 300 on all datasets.

The size of the training sample does not seem to have much effect on the prediction when comparing classification mode and regression mode. As expected, we see that the prediction with respect to the ranking score is improving with #**LS** and reaches convergence at about #**LS** = 300. For instance, on the OMIB database, the ranking scores are very similar for each **LS** such that #**LS** $\geq$ 1200.

5.3 Evaluation of the PR selection method

We will set the baseline of our trimming methodology by computing the ranking score $S_{\mathbf{TS}}$ when using the PR (pure random) algorithm to select the set $Q \subset Q^{Full}$ of comparisons. We grow the number of comparisons according to the values in Table 5.4 where these values represent $\sqrt{N}$, $\sqrt{N}\log_{10}(N)$, $N/2$, N, $2N$, $N\log_{10}(N)$, $N(\log_{10}(N))^2$ and $N\sqrt{N}$. Some values of T account for several of these representations depending on the dataset.

Database	N	$N(N-1)/2$	$\#Q^{'Full}$	Values of T
OMIB	10	45	45	3, 5, 10, 20 and 30.
Sushi_0	100	4950	3408	10, 20, 50, 100, 200, 400 and 1000.
Sushi_3	100	4950	3485	10, 20, 50, 100, 200, 400 and 1000.
MovieLens	1682	1,413,721	955,623	41, 132, 841, 1682, 3364, 5436, 17503 and 68983.

Table 5.4: Considered values for T and $\#Q^{'Full}$. Recall of values of N and $N(N-1)/2$.

We will only use regression trees as base learners, as we showed in Section 5.2 that both modes are approximately equivalent, with the advantage of regression trees to be more robust with respect to the tree parameters. We will assess the effect of the model mode in Section 5.5.1.

The parameters of the Extra-trees method are set in the following way : $K = \sqrt{\#A}$, $n_{\min} = 2$, $M = 500$. We have seen in Section 5.2 that using $K = \sqrt{\#A}$ or $K = \#A$ gave similar results, and using only the square root of the number of attributes will speed up the split selection. We use $n_{\min} = 2$ to construct fully grown trees, which will perfectly perform on the **LS**, because it works well in average on the 3 datasets (see Section 5.2.2.1). We use $M = 500$ in order to ensure that the variations in the ranking score will be inherent to the value of T, and not due to a lack of accuracy in our base models.

We will use both Sushi_0 and Sushi_3 because we have seen that the size of the training set could have some collateral effect (e.g. the "$n_{\min}$ effect" in Section 5.2.2.1) and we might observe different behaviors between MovieLens and Sushi_0, which are both noisy and partially ranked while MovieLens has more observations and more labels than Sushi_0. Both Sushi_0 and Sushi_3 consider the same T values.

We will drop empty comparisons at this stage (i.e. use $Q^{'Full}$ instead of Q^{Full}) and we will repeat the selection process 100 times (we will thus evaluate 100 ranking scores) per value of T.

In this section, each graph represents, on the vertical axis, the ranking score $S_{\mathbf{TS}}$ as a function of the value of T. The green line represents the ranking score of RPC when using all available comparisons (in $Q^{'Full}$). The blue line, if present, represents the higher published ρ for the dataset. Results are depicted in the form of a box plot on Figure 5.11.

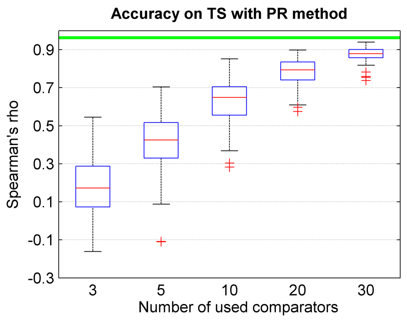
We can see that a large value of T is required to obtain a set whose prediction is interesting, although significantly lower than RPC. If we consider, for instance, the accuracy on Sushi_3 database, we see that even with 1000 comparisons (thus $T = N\sqrt{N}$), the ranking score $S_{\mathbf{TS}}$ is lower than 90% of the ranking score when using all comparisons. The predictions are rapidly degrading with smaller values of T.

5.3.1 Effect of imposing a full class coverage

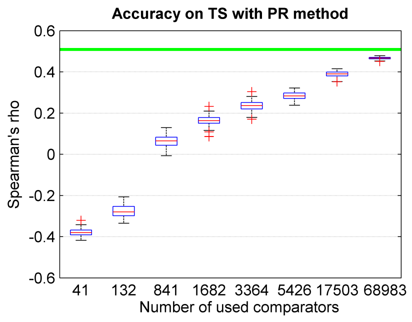
We formulated, in Section 4.4.2.1, the hypothesis that imposing (as much as possible) a full class coverage over the set of labels by the comparisons in Q would improve our predictions because, without a full class coverage, some labels would never be given the chance to be well ranked.

We will now verify this hypothesis by repeating the previous test[2] but without imposing any class coverage. That is, the comparisons are drawn from

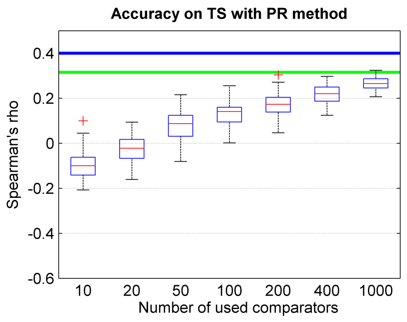
[2]Performing 100 runs where each run draws T comparisons using the PR algorithm.



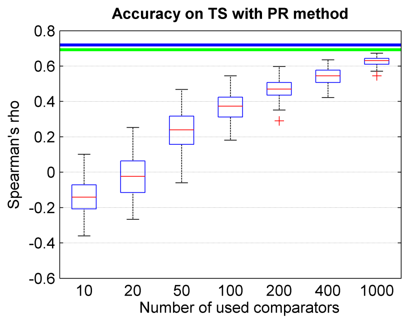
(a) OMIB $N = 10$, $\#Q^{Full} = 45$.



(b) MovieLens, $N = 1682$, $\#Q^{Full} = 955,623$.



(c) Sushi_0, $N = 100$, $\#Q^{Full} = 3408$.

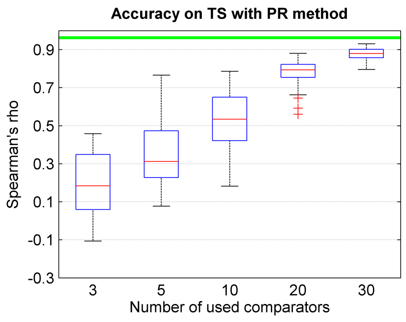


(d) Sushi_3, $N = 100$, $\#Q^{Full} = 3485$.

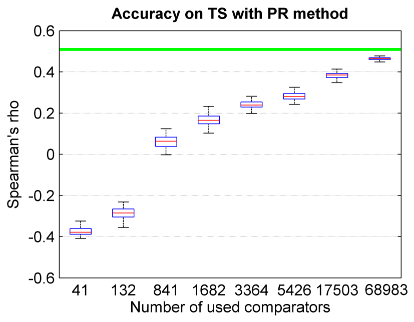
Figure 5.11: The ranking scores of the predictions are rapidly degrading when removing comparisons in a random fashion.

the uniform distribution without replacement and without any additional constraint. The results are depicted in Figure 5.12.

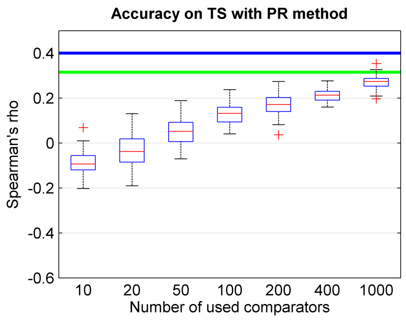
Globally, imposing a full class coverage does not significantly improve the predictions. In some cases, the constraint improves the prediction (e.g. OMIB, $T = 10$; Sushi_0, $T = 50$; Sushi_3, $T \leq 200$; MovieLens, $T = 132$), but in other cases, the opposite holds true (e.g. OMIB, $T = 3$; Sushi_0, $T = 1000$; Sushi_3, $T \geq 400$; MovieLens, $T = 3364$). In most cases, however, and in particular with MovieLens, the variations are very slight, thus hardly observable, and probably only due to the random initialization variance. We will not perform a similar



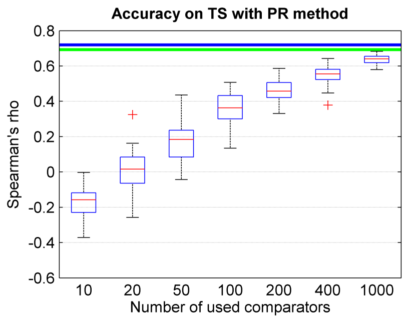
(a) OMIB $N = 10$, $\#Q^{Full} = 45$.



(b) MovieLens, $N = 1682$, $\#Q^{Full} = 955, 623$.



(c) Sushi_0, $N = 100$, $\#Q^{Full} = 3408$.



(d) Sushi_3, $N = 100$, $\#Q^{Full} = 3485$.

Figure 5.12: The class coverage constraint does not seem to have a significant effect on the predictions.

comparison when using the EDA algorithm, because we think that the observed results would be similar. Although we will still impose the full class coverage in EDA, even if we showed that the constraint has a marginal influence on the accuracy, because it is a more fair approach with respect to the labels.

5.4 Evaluation of EDA, EGS and RGS

In this section, we will discuss the accuracy, in terms of Spearman's ρ, of the EDA, EGS and RGS set selection methods.

We carried out tests, for growing numbers of comparisons, with our 3 more elaborated set selection algorithms. Each setting was tested 100 times for EDA (while each execution of the algorithm is performed by selecting the top 50 sets of comparators among 100), 100 times for EGS (which was only performed on the OMIB database for computational time reasons) and 100 times for RGS. All of these tests were performed in regression mode and we used the same tree parameters as for the PR algorithm[3].

For EDA, we selected $j = 20$, $s = 100$ and $b = 50$ arbitrarily. For RGS, we chose $j_1 = 2000, j_2 = 100$ and $j_3 = 3000$ in such a way that the number of Spearman evaluations are at least equivalent to the number of Spearman evaluations performed by EDA. With respect to EDA however, we set a small margin during which RGS is allowed to continue its process as long as it improves the ranking score at least once in 100 iterations.

Each graph will follow the same representation as in section 5.3 and show a boxplot diagram per couple dataset/method. The Y-axis is the Spearman's ρ observed on the test set and the X-axis is the number of comparators being used. The control is the original RPC method and is depicted with a green horizontal line. The blue line, if present, shows the best published Spearman's ρ for this dataset.

We will first justify the relevance of optimizing on the **LS** outputs, then we will compare the results of our trimming algorithms in terms of ranking score.

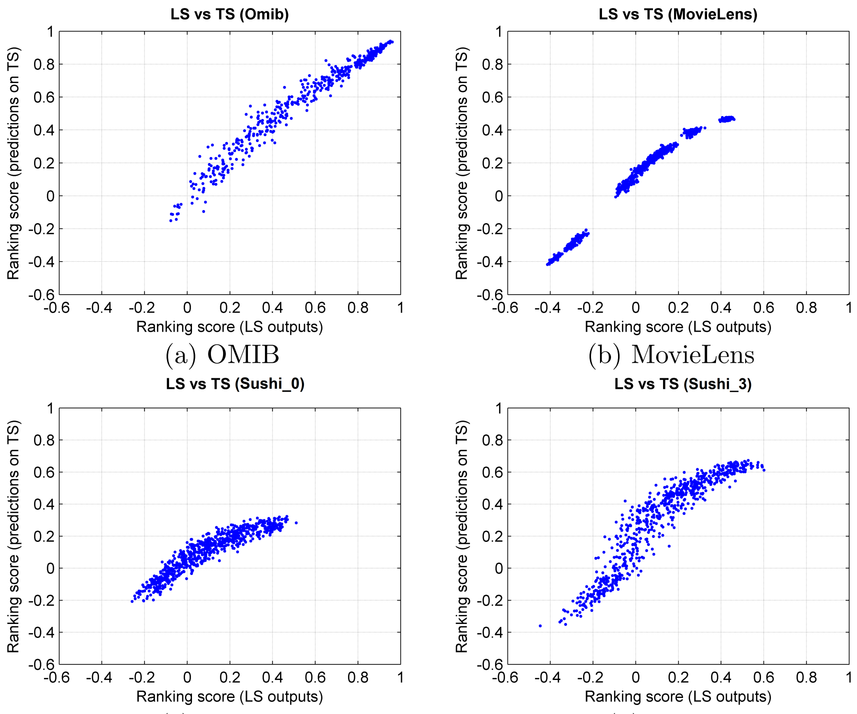
5.4.1 Relevance of optimizing with respect to the LS outputs

Each trimming algorithm that we will further consider (namely EDA, EGS and RGS) have this in common that the optimization process (i.e. the search for a Q maximizing the ranking score $S_{\mathbf{TS}}$) is based on the maximization of the ranking score $S_{\mathbf{LS}}$. That is, in each of these algorithms, we consider that the higher the ranking score $S_{\mathbf{LS}}$ on the **LS** outputs for a given Q is, the higher the ranking score $S_{\mathbf{TS}}$ on the model predictions will be for the same Q. We thus iteratively select the set Q which maximizes $S_{\mathbf{LS}}$. However, this only makes sense if there

[3] $K = \sqrt{\#A}$, $n_{\text{min}} = 2$ and $M = 500$.

effectively is a correlation between $S_{\mathbf{LS}}$ and $S_{\mathbf{TS}}$.

We define by "architecture" a combination of tree parameters and the parameter T. In Section 5.3, we obtained 100 sets Q per architecture when using the PR algorithm. We now use each of these sets Q to compute $S_{\mathbf{LS}}$ and $S_{\mathbf{TS}}$ for each Q and compare these two values in a two dimensional graph. Figure 5.13 shows the observed results, where the vertical axis represents the ranking score on the **TS** while the same measure on the **LS** is represented on the horizontal axis. Each dot represents a set Q in an architecture. The corresponding figures when drawing the set Q from the set Q^{Full} instead of Q'^{Full} can be found in Section B.2.



(a) OMIB (b) MovieLens (c) Sushi_0 (d) Sushi_3

Figure 5.13: Correlation between $S_{\mathbf{LS}}$ and $S_{\mathbf{TS}}$.

In all four datasets, there effectively is a (approximately linear) correlation between $S_{\mathbf{LS}}$ and $S_{\mathbf{TS}}$, this correlation being strongly marked. We can thus assess that, in general, the higher the ranking score on the training sample is, the higher the ranking score on the test sample is, thus we can assess that performing the optimization process by using the ranking scores $S_{\mathbf{LS}}$ is relevant.

5.4.2 Ranking score on OMIB

This section will discuss the observed ranking scores $S_{\mathbf{TS}}$ on OMIB when using EDA, EGS and RGS. Note that EGS was only tested on the OMIB database, as the computational time of each iteration of this method is $\mathcal{O}(N^2)Tn$ [4] in the number of Spearman evaluations and hence was not feasible on the other datasets having much larger numbers of labels. The observed results are depicted in Figure 5.14.

Without much surprises, the EGS selection method is the most efficient one as it can reach an average[5] ranking score of nearly 0.9 (i.e. 0.877), in terms of Spearman's ρ, with only 5 comparisons. However, since this algorithm was only tested on the OMIB dataset, this assessment can only be verified once. The RGS method, while slightly better than PR, is rather disappointing in the sense that a high value of T is still required to obtain good accuracy results, i.e. RGS requires 10 comparisons to reach an average ranking score of 0.832 and requires as high as 20 comparisons to reach an average ranking score of 0.88. EDA actually performs quite well and is able to reach an average ranking score of 0.864 with 10 comparisons.

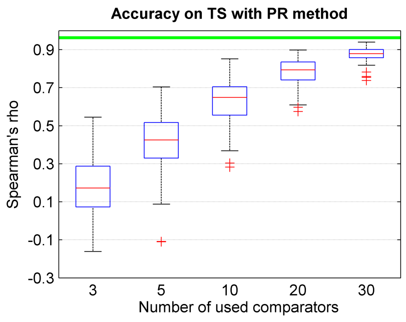
We will define a normalized version of $S_{\mathbf{TS}}$ by $\overline{S_{\mathbf{TS}}}$ which is defined in $[0, 1]$ by

$$\overline{S_{\mathbf{TS}}} = \frac{S_{\mathbf{TS}} + 1}{2}. \tag{5.1}$$

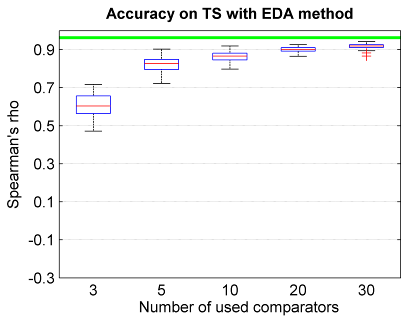
We will express the improvement in accuracy by a percentage $p_{\mathbf{TS}}$, which is computed by dividing $\overline{S_{\mathbf{TS}}}$ when using the method to compare (EGS, RGS,

[4]Note that this square order also directly depends on T. Indeed, with a very large value of T, the number of possible candidates for replacement is strongly reduced, up to the extreme case where $T = N(N-1)/2$ and where no candidates for replacement can be found.

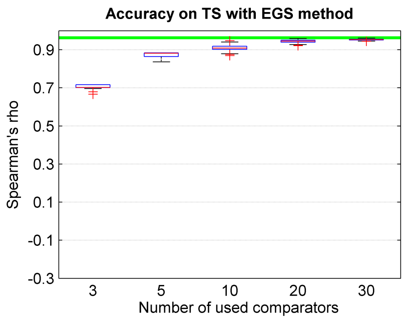
[5]computed as the mean of the ranking scores obtained over 100 runs



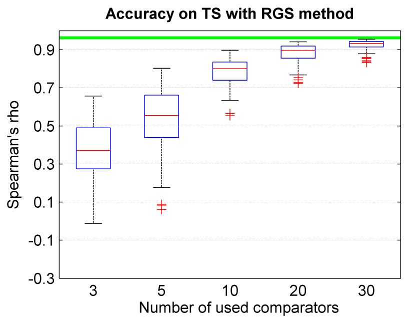
(a) OMIB, PR



(b) OMIB, EDA



(c) OMIB, EGS



(d) OMIB, RGS

Figure 5.14: Comparison between all set selection algorithms on OMIB.

EDA) by $\overline{S_{\mathbf{TS}}}$ when using PR, for a given Q.

With respect to the PR algorithm, each method performs much better than PR when T is small, while the gain is marginal when T is large (e.g. $p_{\mathbf{TS}}$, when using EGS, is 145.16% for $T = 3$ and is only 104.17% for $T = 30$). The complete percentage results can be found in Table 5.5.

5.4.3 Ranking score on Sushi

We performed the same experiments on the Sushi databases using the same experimental setup as in OMIB. Figure 5.15 shows the observed results for Sushi_0 and Sushi_3.

T	EDA vs PR	RGS vs PR	EGS vs PR
3	137.07%	115.85%	145.16%.
5	129.36%	109.41%	133.41%.
10	114.61%	109.83%	117.48%.
20	106.57%	105.40%	109.04%.
30	102.33%	102.68%	104.17%.

Table 5.5: Comparison of the relative performance of EDA, EGS and RGS w.r.t. PR on OMIB. The percentage represents $(S_{\mathbf{TS}}+1)/2$ when using either EDA, EGS or RGS divided by $(S_{\mathbf{TS}}+1)/2$ when using PR for the same value of T.

EDA and RGS still perform better than PR. RGS is better than EDA when $T \leq 200$ but is beaten by EDA by a small margin when $T \geq 400$. Note that, again, RGS and EDA improve the ranking score with respect to PR in a stronger way when T is small than when T is large. The comparison with PR for each considered value of T, for the two methods (EDA and RGS) and for the two datasets (Sushi_0 and Sushi_3) is given in Table 5.6.

	Sushi_0		Sushi_3	
T	EDA vs PR	RGS vs PR	EDA vs PR	RGS vs PR
10	111.27%	115.73%	125.79%	137.12%.
20	111.62%	114.19%	127.54%	133.23%.
50	108.02%	111.96%	115.39%	120.44%.
100	106.29%	109.85%	109.76%	113.63%.
200	105.76%	106.65%	106.87%	107.88%.
400	104,15%	103.45%	105.47%	104.84%.
1000	102.35%	102.13%	102.39%	101.48%.

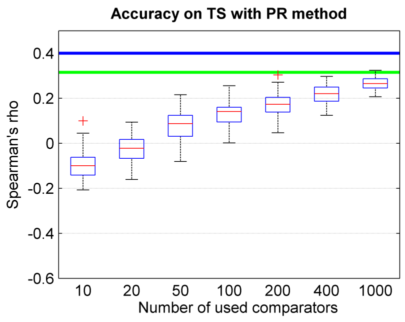
Table 5.6: Comparison of the relative performance of EDA and RGS w.r.t. PR on Sushi_0 and Sushi_3. The percentage represents $(S_{\mathbf{TS}}+1)/2$ when using either EDA or RGS divided by $(S_{\mathbf{TS}}+1)/2$ when using PR for the same value of T.

5.4.4 Ranking score on MovieLens

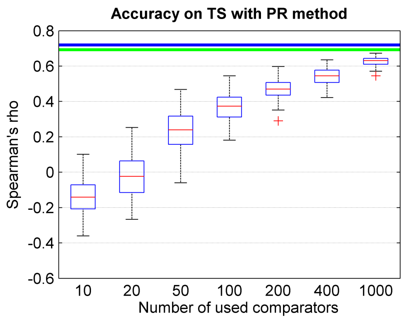
When performing the survey on MovieLens, we can conclude in the same way as for Sushi that RGS is the most accurate method. This time, RGS clearly converges more rapidly than EDA (and the former is better than the latter for any considered value of T) and can produce a set Q whose prediction is interesting starting at $T = 841$ (thus $T = N/2$). Figure 5.16 shows the observed results and Table 5.7 quantifies the accuracy improvement for EDA and RGS.

T	EDA vs PR	RGS vs PR
41	107.68%	146.88% .
132	108.72%	159.16%.
841	105.80%	129.29%.
1682	104.28%	120.23%.
3364	103.51%	113.54%.
5426	102.82%	109.65%.
17503	101.36%	103.64%.
68983	100.64%	100.74%.

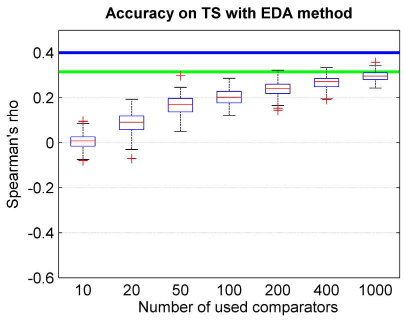
Table 5.7: Comparison of the relative performance of EDA and RGS w.r.t. PR on MovieLens. The percentage represents $(S_{\mathbf{TS}} + 1)/2$ when using either EDA or RGS divided by $(S_{\mathbf{TS}} + 1)/2$ when using PR for the same value of T.



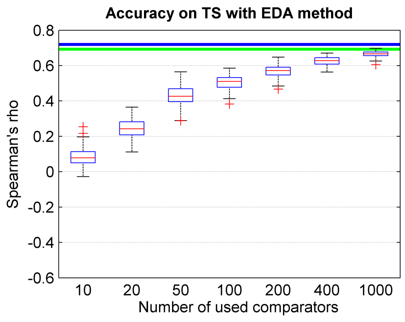
(a) Sushi_0, PR



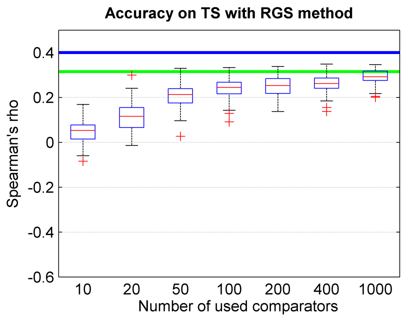
(b) Sushi_3, PR



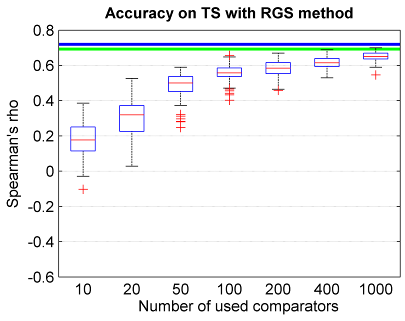
(c) Sushi_0, EDA



(d) Sushi_3, EDA

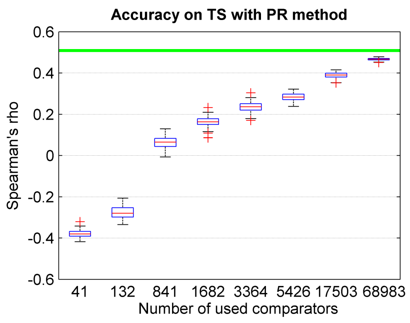


(e) Sushi_0, RGS

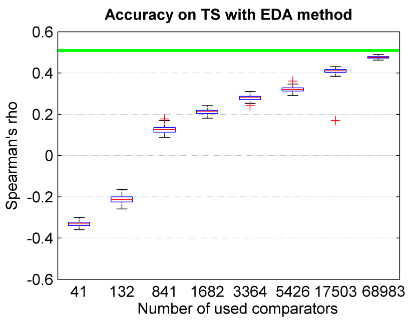


(f) Sushi_3, RGS

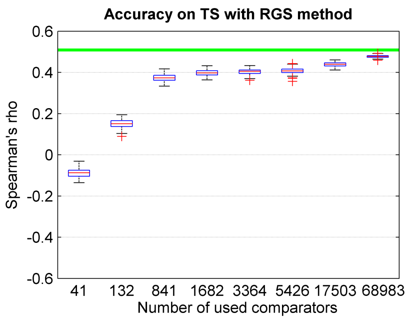
Figure 5.15: Comparison between all set selection algorithms on Sushi_0 and Sushi_3.



(a) MovieLens, PR



(b) MovieLens, EDA



(c) MovieLens, RGS

Figure 5.16: Comparison between all set selection algorithms on MovieLens.

5.5 Effect of (meta-)parameters in a trimmed RPC scheme

We will differentiate the regular parameters, which are directly related to the set selection algorithms and can change its output (e.g. s, b, j, j_1, j_2, j_3, ...) from the *meta-parameters*, which also have an impact on the ranking score, but not on the set selection algorithms output (e.g. the tree parameters, the mode of the model (classification or regression), ...).

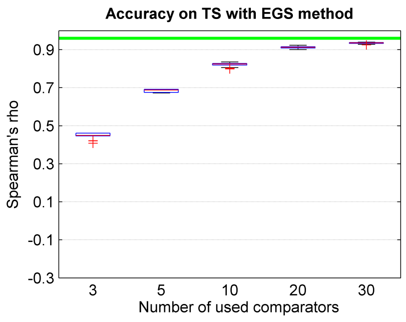
In this section, we perform an analysis of the latter, as well as the j_1 and j_3 parameters, with respect to the impact that they have on the accuracy. In Section 5.5.1, we analyze the effect of the model mode. Then, we investigate the effect on the sparsity control in Section 5.5.2. The effect of the tree parameters on RGS will be discussed in Section 5.5.3. In Section 5.5.4, we analyze the effect of the default order used to break the ties. In section 5.5.5, we remove the "ObjectID" attribute and evaluate its impact on the predictions. In Section 5.5.6, we evaluate the RGS set selection algorithm when simulated perfect models are used as comparators. Finally, in Section 5.5.7, we perform a small experiment to validate our arbitrary choice of the j_1 and j_3 parameters.

5.5.1 Effect of the model mode

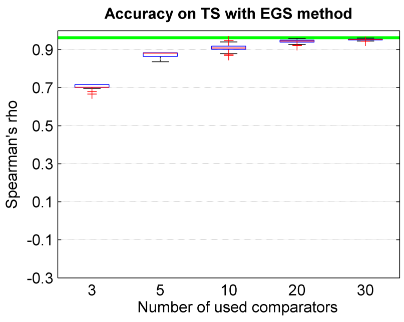
So far, we only considered the regression mode in our experiments, as we have shown that on a full RPC scheme, this choice was generally beneficial. We will now consider the effect of classification vs regression when trimming the set of comparators. We will consider the effect on the most accurate trimming methods (EGS for OMIB, RGS for Sushi and MovieLens) and display this effect on Figure 5.17 for OMIB and Figure 5.18 for Sushi and MovieLens. Note that we performed this experiment for all set selection methods. The corresponding figures can be seen in Section B.1.

Regression mode improves the accuracy with small values of T, while this effect is less pronounced with higher values of T, meaning that using class-probability estimators (models in regression mode) rather than binary classifiers allows to use less comparators for the same desired ranking score $S_{\mathbf{TS}}$. This effect is more pronounced on the OMIB database, but can be seen across all sets.

The general trend of regression outperforming classification can be seen



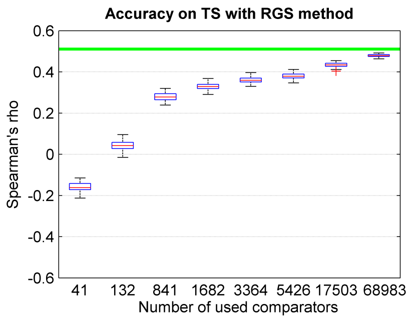
(a) OMIB, EGS, Classification



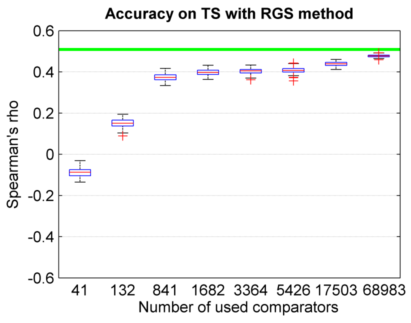
(b) OMIB, EGS, Regression

Figure 5.17: Effect of the model mode on the predictions when using optimal selection algorithm on OMIB.

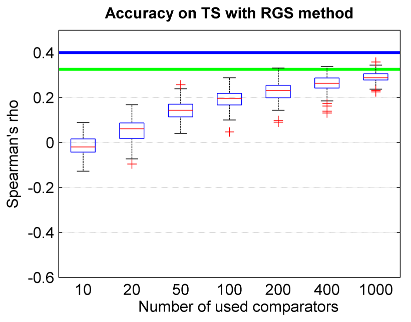
across all datasets and for each set selection method. We will thus not detail each figure separately. Note that regression mode outperforms classification mode for any considered value of $T < N(N-1)/2$.



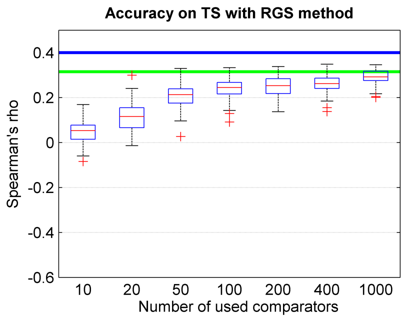
(a) MovieLens, RGS, Classification



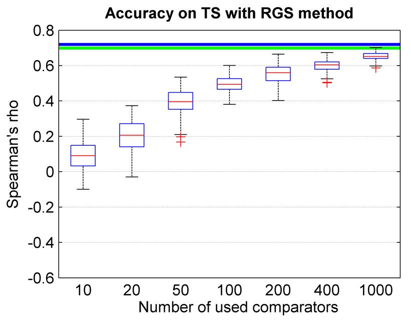
(b) MovieLens, RGS, Regression



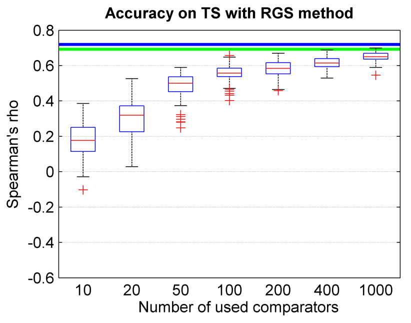
(c) Sushi_0, RGS, Classification



(d) Sushi_0, RGS, Regression



(e) Sushi_3, RGS, Classification



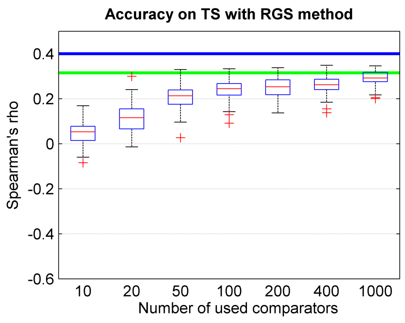
(f) Sushi_3, RGS, Regression

Figure 5.18: Effect of the model mode on the predictions when using optimal selection algorithm for Sushi and MovieLens.

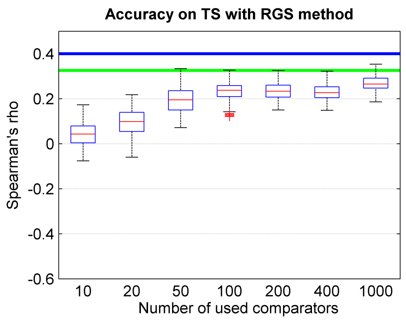
5.5.2 Effect of sparsity control

Until now, we have addressed the problem of sparsity in partial rankings by dropping empty comparisons. In this section, we will now consider the effect of our alternative manner of dealing with sparsity in partial rankings, namely modeling the absence of information by using a dummy model (or "UDM", which stands for "Using a Dummy Model", see Section 4.3.1). We will refer to the fact of dropping empty comparisons as "DEC". We only show the performances of models in regression mode and use the same architectures as in Sections 5.3 and 5.4. We will compare accuracy in terms of ranking score $S_{\mathbf{TS}}$ on all datasets and for the RGS set selection algorithms, these results being depicted in Figure 5.19 for Sushi_0, Sushi_3 and MovieLens. OMIB being a fully ranked dataset, we did not have to deal with the sparsity issue. The performances of models used in classification mode as well as the performances for the other comparison selection methods can be seen in Section B.1.

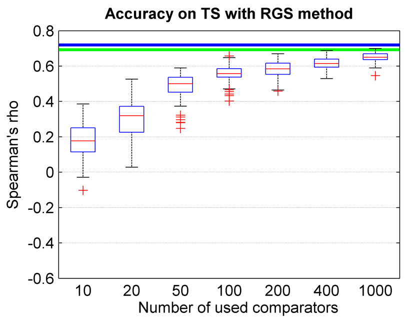
Dropping empty comparisons is detrimental to RPC as it slightly reduces the ranking score (as you can see by comparing the green lines). However, when using a trimmed set, dropping empty comparisons improves the ranking score when compared to the ranking score obtained by using constant models to represent empty comparisons, for any value of T. Both effects can be seen across all sets and for all set selection method.



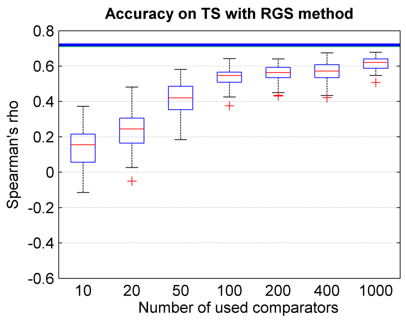
(a) Sushi_0, RGS, DEC



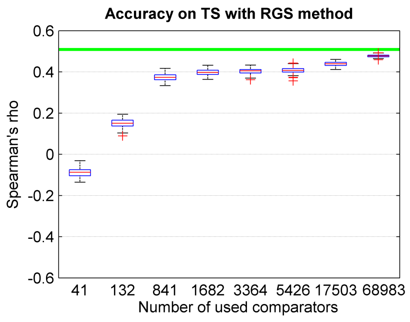
(b) Sushi_0, RGS, UDM



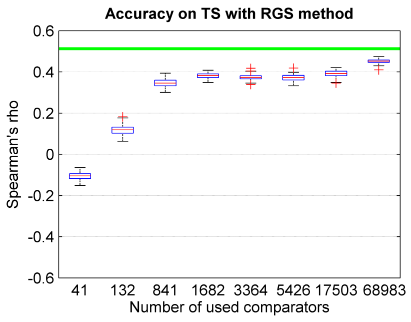
(c) Sushi_3, RGS, DEC



(d) Sushi_3, RGS, UDM



(e) MovieLens, RGS, DEC



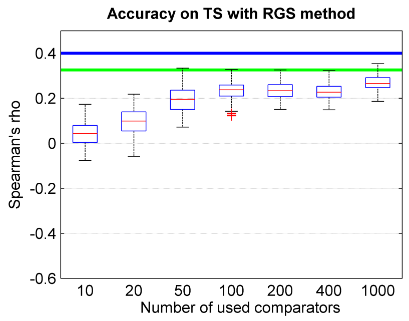
(f) MovieLens, RGS, UDM

Figure 5.19: Effect of the manner of dealing with sparsity in Sushi_0, Sushi_3 and MovieLens.

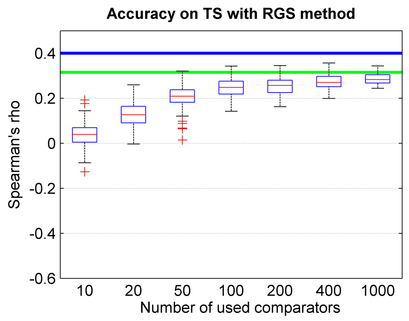
Since we have seen that removing empty comparisons was beneficial when using a trimmed set, we will investigate how the ranking score evolves (both when $Q = Q^{Full}$ and when using trimmed sets) when removing even more comparisons *a priori* from Q^{Full}. Indeed, we assume that using a comparison which only has one object in its own **LS** would also be detrimental as the corresponding comparator would be a constant model.

We will thus investigate the effect of removing comparisons containing only few objects in their own **LS**. We will perform two additional experiments. In the first one, a comparison must have at least two objects in its **LS** in order to have a chance to be selected. In the second one, this criterion is raised to 5 objects. We only tested the most accurate selection method, RGS, in regression mode. Results are depicted in Figure 5.20 for Sushi_0, Figure 5.21 for Sushi_3 and Figure 5.22 for MovieLens.

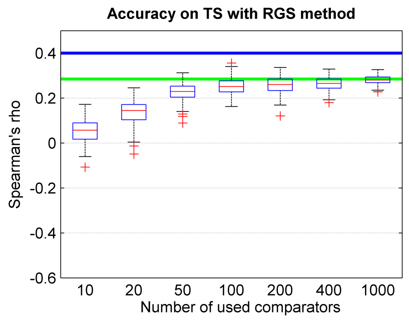
What we see, by comparing the green lines, is that the performance of RPC when using all available comparisons decreases when the constraint is stronger. However, RGS will converge faster to the performance of RPC. Therefore, for a small T, the performance of RGS is improved when $\#Q'^{Full}$ is strongly reduced. Once again this can be seen across all figures, which we will not detail. In the case when we only wish to use $T = N$, we should thus impose that each comparison is represented by at least 5 objects in order to be modeled by a comparator. We wanted to push this idea to the extreme case and compute the ranking score of a scheme that ranks the comparisons according to their #**LS** and use the first T of these comparisons. Results are shown in Table 5.8.



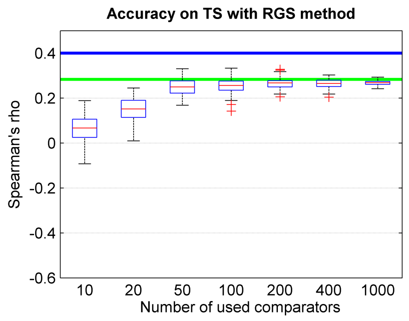
(a) Sushi_0, UDM, $\#Q^{Full} = 4950$



(b) Sushi_0, DEC, $\#Q^{Full} = 3408$



(c) Sushi_0, at least 2 objects, $\#Q^{Full} = 2487$

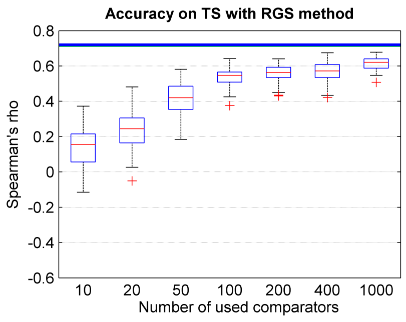


(d) Sushi_0, at least 5 objects, $\#Q^{Full} = 1278$

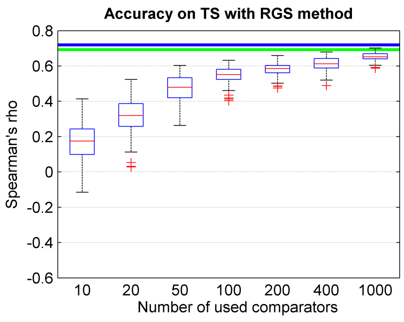
Figure 5.20: Effect of alternative manners of dealing with sparsity in Sushi_0.

T	Sushi_0	Sushi_3	MovieLens
$\sqrt{N}$	-0.104 (0.048 ± 0.053)	0.120 (0.178 ± 0.101)	-0.213 (-0.089 ± 0.021)
$\sqrt{N}\log_{10}(N)$	0.079 (0.113 ± 0.062)	0.240 (0.296 ± 0.109)	-0.140 (0.151 ± 0.021)
$N/2$	0.143 (0.209 ± 0.052)	0.392 (0.486 ± 0.073)	0.083 (0.374 ± 0.017)
N	0.174 (0.241 ± 0.042)	0.403 (0.556 ± 0.047)	0.145 (0.398 ± 0.015)
$2N$	0.190 (0.249 ± 0.043)	0.500 (0.583 ± 0.042)	0.217 (0.403 ± 0.013)
$N\log_{10}(N)$	0.190 (0.249 ± 0.043)	0.500 (0.583 ± 0.042)	0.254 (0.408 ± 0.015)
$N\log_{10}(N)^2$	0.218 (0.262 ± 0.035)	0.544 (0.616 ± 0.033)	0.369 (0.439 ± 0.010)
$N\sqrt{N}$	0.287 (0.294 ± 0.029)	0.595 (0.650 ± 0.026)	0.421 (0.477 ± 0.006)

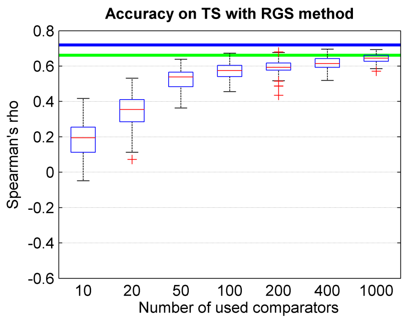
Table 5.8: Ranking score obtained by selecting the comparators with the most objects in their own **LS**. The numbers in brackets represents the average (left) and standard deviation (right) of $S_{\mathbf{TS}}$ over 100 runs of the RGS algorithm when using dummy models.



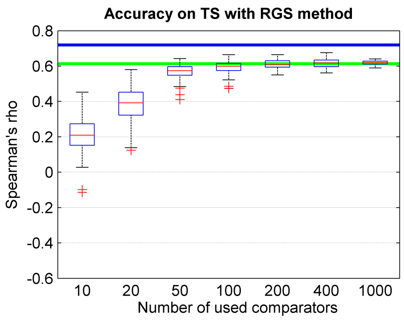
(a) Sushi_3, UDM, $\#Q^{Full} = 4950$



(b) Sushi_3, DEC, $\#Q^{Full} = 3485$



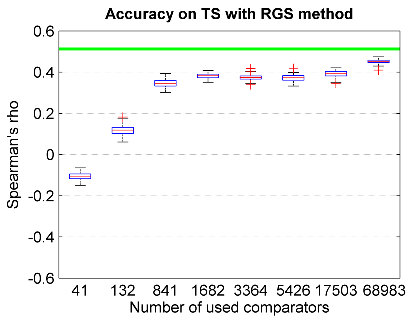
(c) Sushi_3, at least 2 objects, $\#Q^{Full} = 2544$



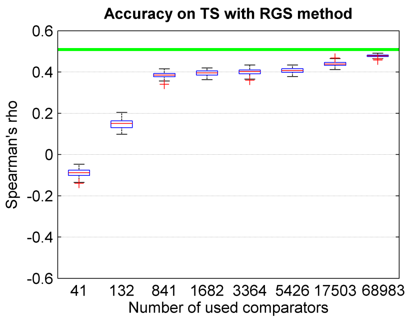
(d) Sushi_3, at least 5 objects, $\#Q^{Full} = 1290$

Figure 5.21: Effect of alternative manners of dealing with sparsity in Sushi_3.

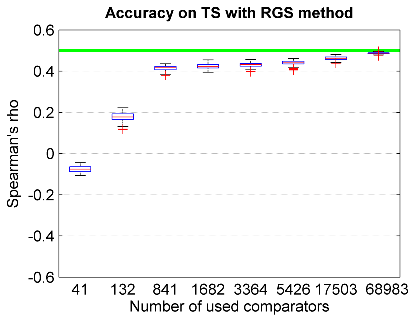
When selecting the comparisons based only on the number of objects in their **LS**, we see that the performance is strongly degrading and, for instance, when $T = N$, letting RGS select the set of size T of comparisons which have 5 objects or more is better than selecting the T comparisons which have the most objects.



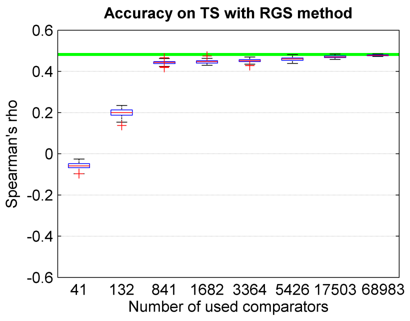
(a) MovieLens, UDM, $\#Q^{Full} = 1,413,721$



(b) MovieLens, DEC, $\#Q^{Full} = 955,623$



(c) MovieLens, at least 2 objects, $\#Q^{Full} = 665,936$

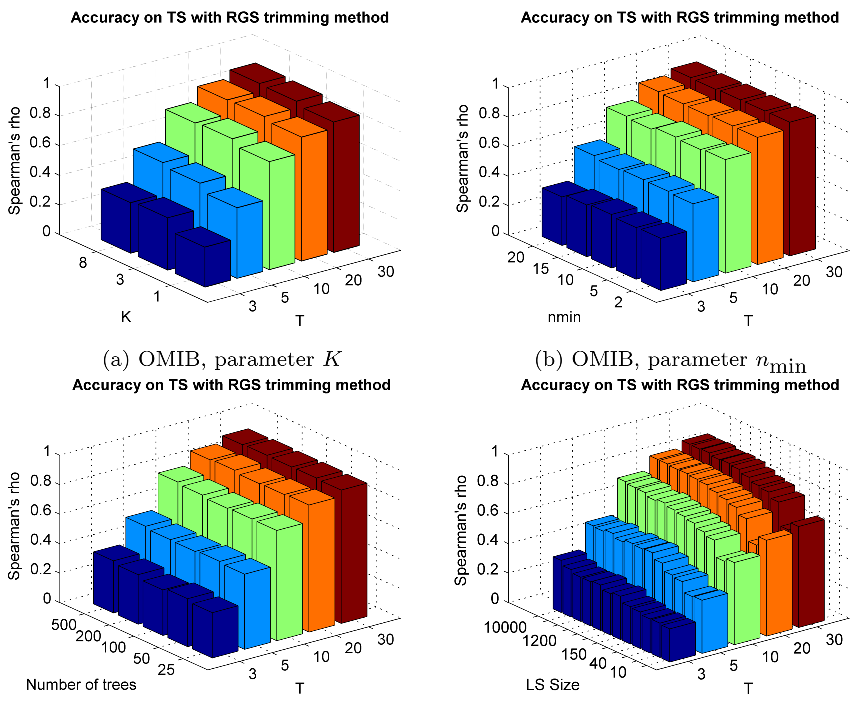


(d) MovieLens, at least 5 objects, $\#Q^{Full} = 393,685$

Figure 5.22: Effect of alternative manners of dealing with sparsity in MovieLens.

5.5.3 Influence of the tree parameters (on RGS)

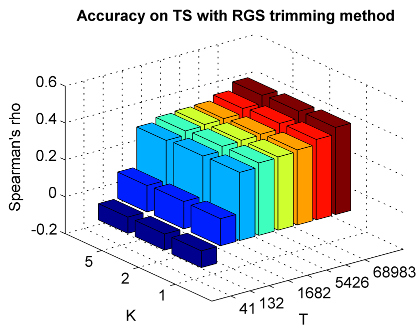
In this section, we perform a similar experiment as in Section 5.2.2, i.e. we analyze the impact of each tree parameter on the ranking score $S_{\mathbf{TS}}$, when using the RGS set selection algorithm, since we have proven that this algorithm was the most efficient. We consider the same architectures as in Section 5.6 and show the results when using regression mode and dropping empty comparisons. The corresponding figures when building the models in classification mode and/or using constant models to represent an empty comparison can be found in Section B.4. Figures 5.23, 5.24, 5.25 and 5.26 show the observed results for OMIB, MovieLens, Sushi_0 and Sushi_3 respectively.



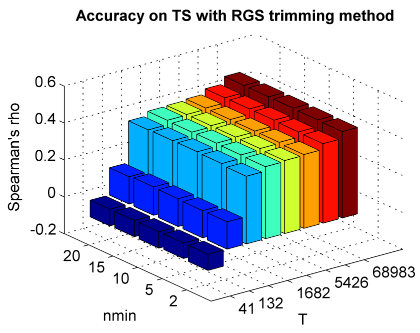
(a) OMIB, parameter K (b) OMIB, parameter n_{min}

(c) OMIB, parameter M (d) OMIB, #**LS**

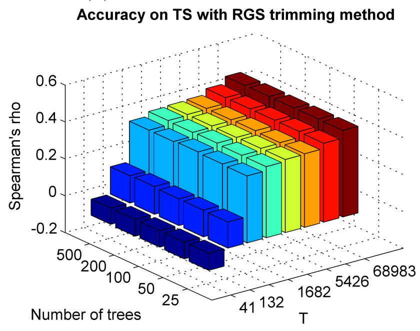
Figure 5.23: Influence of the tree parameters on OMIB when using RGS.



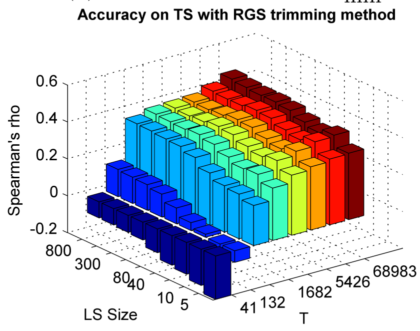
(a) MovieLens, parameter K



(b) MovieLens, parameter $n_{\min}$



(c) MovieLens, parameter M

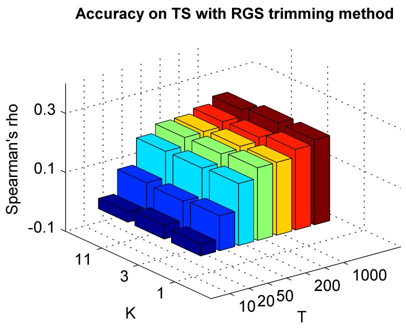


(d) MovieLens, #**LS**

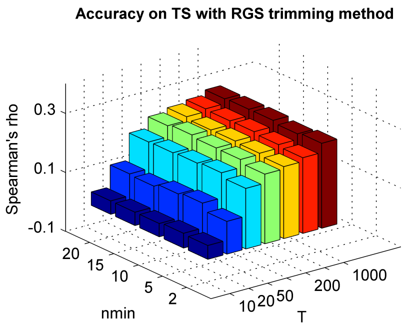
Figure 5.24: Influence of the tree parameters on MovieLens when using RGS.

For each dataset, the observed results are similar. The tree parameters does not seem to have a significant effect on the accuracy of the predictions. What is more interesting is that the parameter M follows the same trend. This means that using as few as 25 trees in the model is sufficient, hence the smoothing effect that we noticed in Section 5.2.2.3 is also noticed when using trimmed sets. A similar effect was observed in classification mode (see Appendix B).

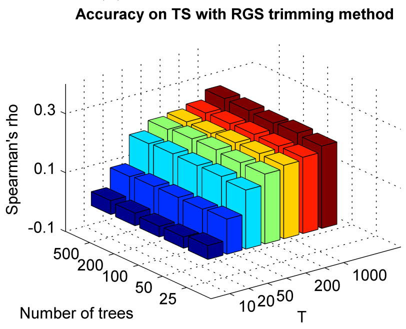
Interestingly, a similar number of observations (with respect to the full RPC) are required to produce accurate models, meaning that the trimming methodology does not increase the “learning effort”.



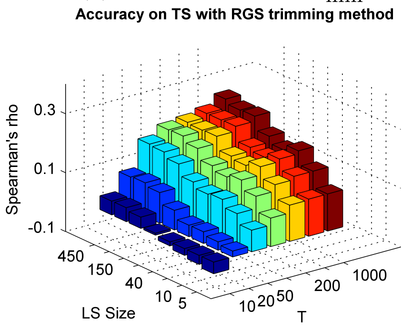
(a) Sushi_0, parameter K



(b) Sushi_0, parameter $n_{\min}$

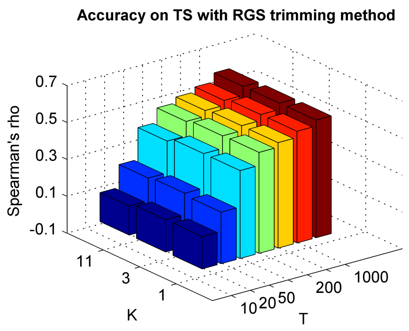


(c) Sushi_0, parameter M

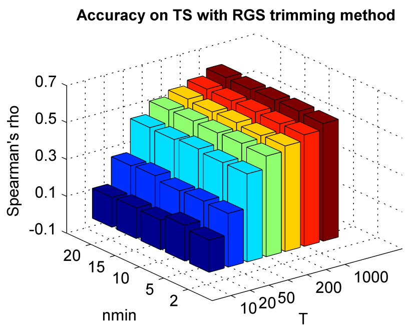


(d) Sushi_0, #**LS**

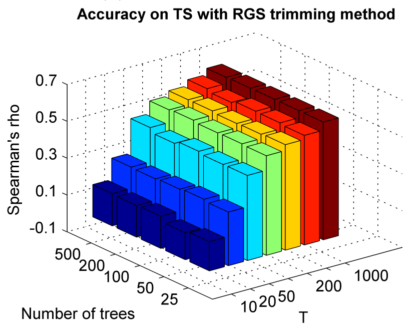
Figure 5.25: Influence of the tree parameters on Sushi_0 when using RGS.



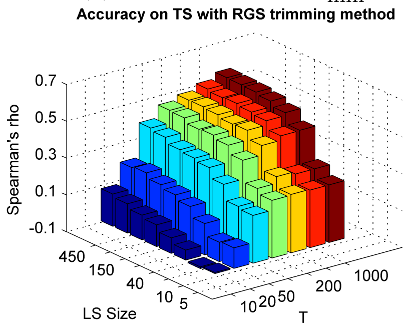
(a) Sushi_3, parameter K



(b) Sushi_3, parameter n_{min}



(c) Sushi_3, parameter M



(d) Sushi_3, #**LS**

Figure 5.26: Influence of the tree parameters on Sushi_3 when using RGS.

5.5.4 Influence of the default order used in tie breaking

Until now, we used the **reverse alphanumerical order** as default order to break our ties. However, this choice was completely arbitrary and we wish to estimate how relevant this choice is and, if not, what other alternatives could be interesting.

To evaluate the reverse alphanumerical order in tie breaking, we computed the ranking score, obtained on the **TS**, when using this order as unique prediction for each object. This score is given in Table 5.9. We can see that, on each dataset, the ranking score is negative. By definition of the Spearman's rank correlation coefficient, we can obtain the opposite score by reversing the sequence and obtain better predictions. However, we wanted to achieve state-of-the-art and applied Theorem 1 of [CH09]. This theorem states that, for any set of partial orderings of size p and under certain assumptions, it is possible to find an ordering which minimizes the average Spearman's rho by attributing, for each partial ordering in the set:

$$\begin{array}{cl} (p-i+1)(N+1)/(p+1) & \text{votes to each label on rank } i \in \{1 \dots p\} \text{ , and} \\ (N+1)/2 & \text{votes to each missing label.} \end{array}$$

These votes are aggregated and used to form the mean ordering, by ranking the label with most votes at the first place, then the label with most votes except the first one in the second place and so on. The ranking score obtained by using this mean order as unique prediction is also depicted in Table 5.9. We now see that this order outperforms the two other ones and, according to [CH09], minimizes the average Spearman distance.

Database	Reverse Alphanum	Alphanum	Mean
OMIB	-0.255	0.255	0.609
Sushi_0	-0.175	0.175	0.319
Sushi_3	-0.322	0.322	0.679
MovieLens	-0.429	0.429	0.467

Table 5.9: Ranking score obtained by using the default order (i.e. alphanumerical, reverse alphanumerical, or mean) as unique prediction for each object of the **TS**.

We now perform an experiment where we use the mean order to break our ties. We performed this experiment using EGS on OMIB and RGS on MovieLens and on both Sushi databases. We trained our comparator in both classification and regression mode. We dropped empty comparisons. Results are depicted in Figure 5.27.

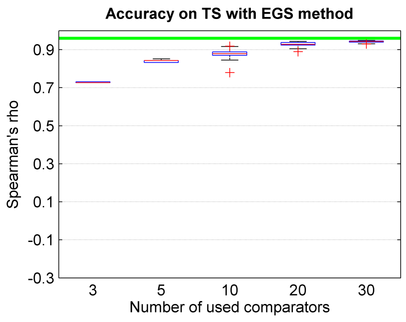
We can clearly see that, for small values of T, the effect is very significant: using the mean order to break ties is far more efficient than using the reverse alphanumerical order. When T gets bigger, this effect fades.

We also see that the mode of the comparators have less effect on the accuracy. Since the ties are broken in a more efficient manner, we assume that the fact that classification mode implies more of these ties is less detrimental.

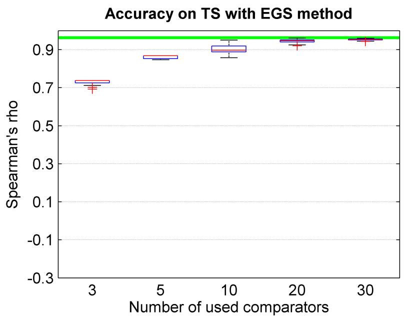
Please note, however, that we still use the reverse alphanumerical order during the evaluation process of our set selection algorithm. Indeed, we noticed that using the mean order in the learning phase causes the algorithm to produce worse predictions than when using the reverse alphanumerical order. The possible reason for this behaviour is that using the mean order to break ties makes the predictions very accurate, hence improving these predictions is more difficult than with the reverse alphanumerical order, where there is more room for improvement. The figures corresponding to this alternative experiment (i.e. when using the mean order to break ties in the learning phase) are shown in Appendix B.1.3.

We also want to repeat the experiment where we select the comparisons based on the size of their auxiliary **LS**, but using the mean order to break the ties. Results are shown in Table 5.10. We see that we obtain similar results as when using RGS. This time, however, models trained in classification mode clearly outperform models trained in regression mode.

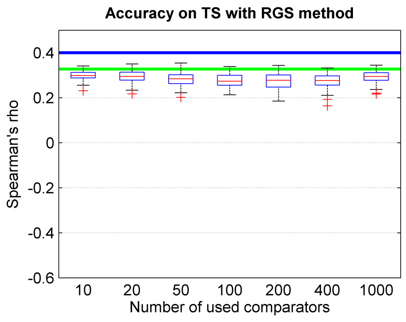
Our major concern w.r.t. the results (both for RGS and when sorting the comparisons based on their $\#\mathbf{LS}_{aux}$) is that we observe, for the two partially ranked databases, that adding some comparisons can degrade the prediction quite significantly and, as a matter of fact, the most efficient method consists in using the mean order as unique prediction, which is quite disappointing. Of course, this assessment is only valid on the databases that we used and given the values of T which we chose to analyze. We assume that our comparators are not built in the most efficient manner and we will discuss this issue in the next section.



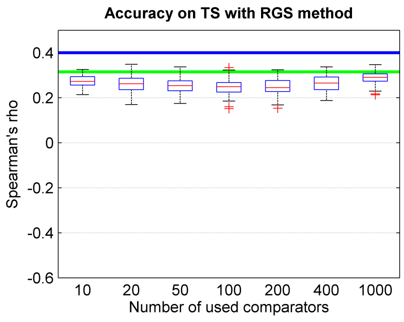
(a) OMIB, EGS, Classification



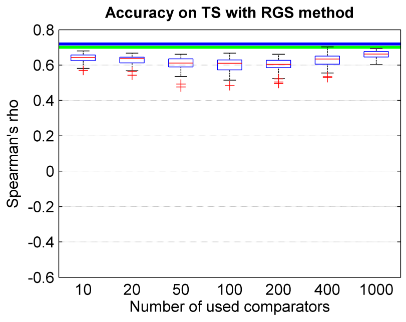
(b) OMIB, EGS, Regression



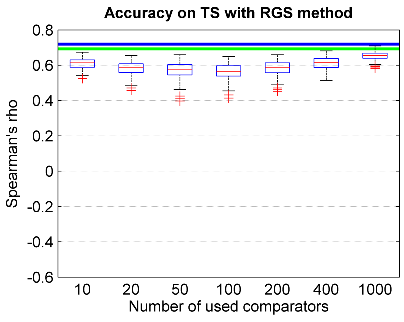
(c) Sushi_0, RGS, Classification



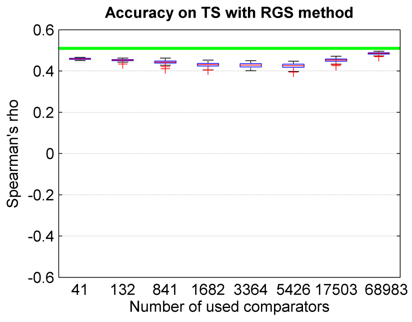
(d) Sushi_0, RGS, Regression



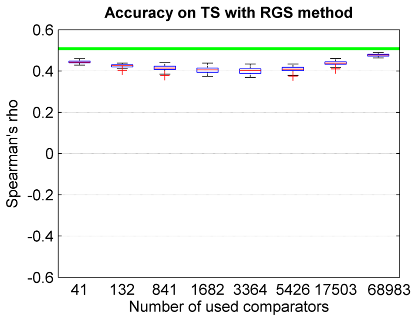
(e) Sushi_3, RGS, Classification



(f) Sushi_3, RGS, Regression



(g) MovieLens, RGS, Classification



(h) MovieLens, RGS, Regression

Figure 5.27: Ranking score when using the mean order to break ties.

T	Clas	Regr
10	0.272	0.234
20	0.292	0.275
50	0.285	0.261
100	0.255	0.229
200	0.269	0.233
400	0.263	0.242
1000	0.297	0.288

(a) Sushi_0

T	Clas	Regr
10	0.618	0.597
20	0.633	0.588
50	0.634	0.577
100	0.594	0.506
200	0.592	0.543
400	0.578	0.561
1000	0.621	0.596

(b) Sushi_3

T	Clas	Regr
41	0.454	0.449
132	0.454	0.446
841	0.438	0.416
1682	0.429	0.409
3364	0.414	0.394
5426	0.409	0.394
17503	0.423	0.409
68983	0.442	0.428

(c) MovieLens

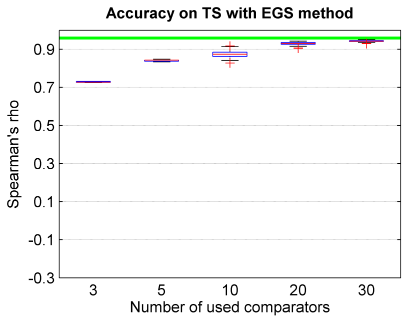
Table 5.10: Ranking score obtained by sorting the comparisons based on their number of examples and using the mean order to break ties.

5.5.5 Removing the object ID from the model construction

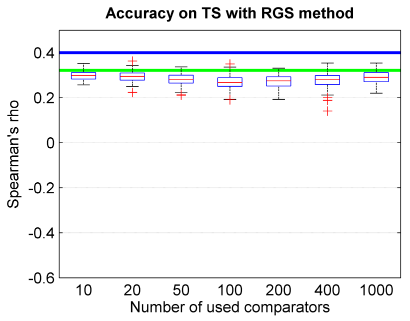
In order to construct our comparators, we used all available information (i.e. we used each attribute in the feature vector). This seems to have sense, but can actually be counter-productive. Indeed, we know, from general machine learning knowledge, that using an attribute which is not correlated to the output adds bias to the prediction because, in that case, performing a class separation based on this attribute is equivalent to performing a random class separation. The "Object ID" is such an attribute, and appears in all of our databases. In OMIB and MovieLens, this attribute is sorted. Since we chose to use the first part of our databases as learning set, and the last part as test set, we have an even worse problem, which is that the object in the test set will always follow the left branch of a split based on that attribute.

We wanted to know how this attribute degrades our predictions, and therefore we performed an experiment, where the object ID is removed from the list of candidate attributes in the comparator construction. We run the EGS algorithm on OMIB and RGS on both Sushi and MovieLens. We drop empty comparisons. We perform this experiment in classification mode and in regression mode, but only show the results in classification mode in Figure 5.28. The figures corresponding to the regression mode can be found in the Appendix B.

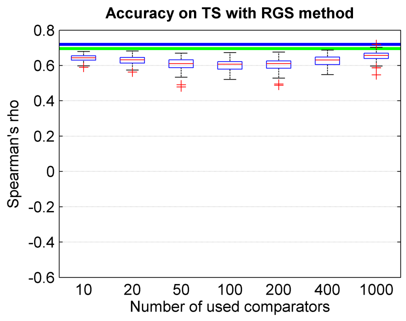
Actually, the ranking score does not seem to be much affected by the "ObjectID" attribute, as observed by comparing Figure 5.28 to Figure 5.27. This can be explained by the fact that the extra-trees algorithm is very robust to the presence of irrelevant features, especially when we built 500 of these trees.



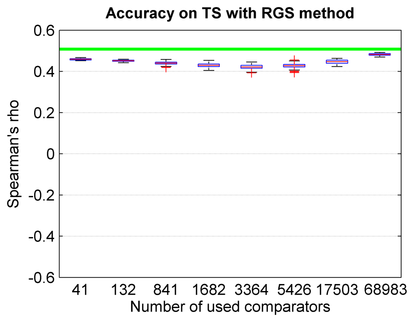
(a) OMIB, EGS



(b) Sushi_0, RGS



(c) Sushi_3, RGS



(d) MovieLens, RGS

Figure 5.28: Ranking score when using the mean order to break ties and removing the object ID from the model construction (Classification mode).

5.5.5.1 Using a 10-fold cross validation on both Sushi databases

So far, we compared ourself to the state-of-the-art provided in [KKA05]. However, the authors performed a 10-fold cross validation and we only perform a single test. It is hence possible that our predictions are too optimistic (or conversely too pessimistic) depending on our selected fold. We thus repeated the previous test on both Sushi databases, but performed a 10-fold validation. Figure 5.29 shows the results where we display the mean value of the 1000 ranking scores per value of T.

As we see no significant difference from the general trend which we observed until now, we can conclude that our accuracy estimates were acceptable.

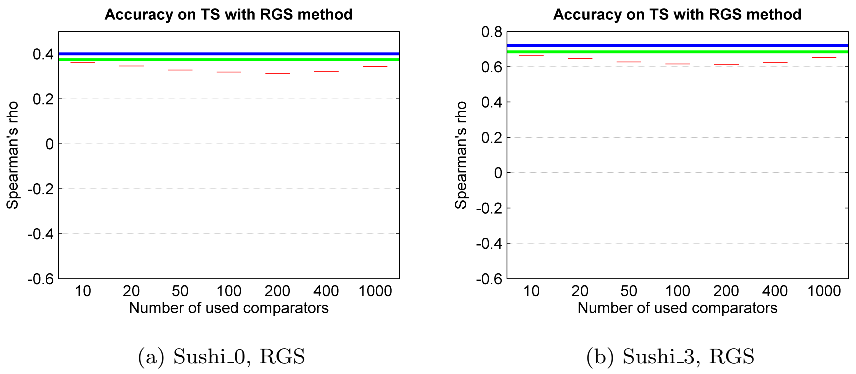


Figure 5.29: Ranking score when using the mean order to break ties and removing the object ID from the model construction (Classification mode). The mean value of the 100 runs of RGS in the 10-Fold cross validation is shown.

5.5.6 Using simulated perfect models

The last results are quite disappointing, because we discovered that using the mean order as unique prediction was equivalent or better on all of our partial ranking datasets than any other trimming technique. We assumed that the use of an attribute which is not related to the output in the model construction was one of the reasons, but we showed in the previous section that it is not the case.

What is reassuring though, is that the performance of RPC is similar to the performance of RGS, although slightly better than the mean order prediction. If we could use perfect models, the RPC would provide a ranking score of 1, which is not the case, and implies that supervised learning algorithms somehow fail at predicting the correct output from the input vector (at least for the partially ordered datasets). We can guess that, effectively, predicting the sushi preference of Japanese people based on their age, gender and location is a quite hard task, as well as predicting the movie preference of users based on their age, gender, occupation and zip code. On the other hand, since the CCT-SBS variable (in the OMIB database) is a function of attributes in the input vector, we should be able to accurately predict this variable.

To evaluate the accuracy of our set selection algorithms without the influence from the supervised learning algorithm, we considered perfect models, i.e. we scanned the ordering given as a supervision to extract our votes during the prediction phase. Note that, in opposition to what we do during the learning phase, when a class label is missing for a particular object, all comparisons related to this label give a vote of 0 to each label (as opposed to 0.5 in the learning phase). We used EGS for OMIB and RGS for MovieLens and both Sushi databases. With the partially ordered datasets, we perform two variants of the experiment: in the first variant, we dropped empty comparisons, in the other one we impose a comparison to be provided with at least 5 examples in the **LS** in order to be a candidate. Figure 5.30 shows the results for the OMIB database. Figure 5.31 shows the results for the MovieLens and both Sushi databases.

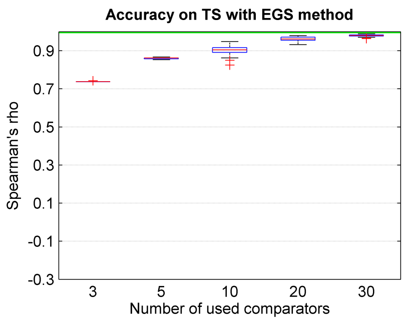
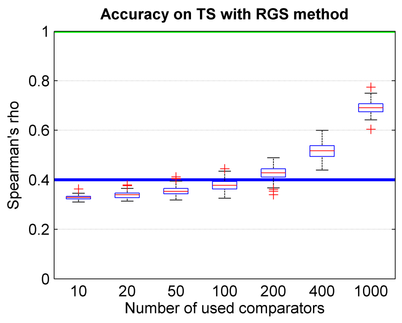
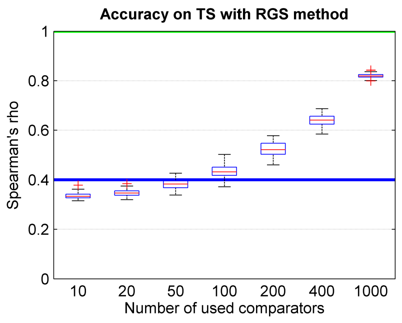


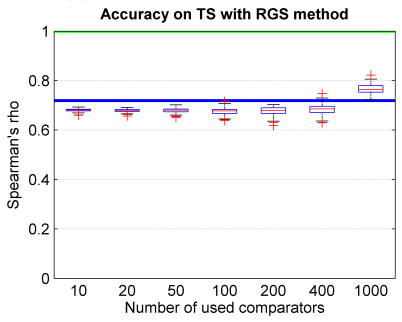
Figure 5.30: Ranking score when using the mean order to break ties and using perfect models (OMIB).



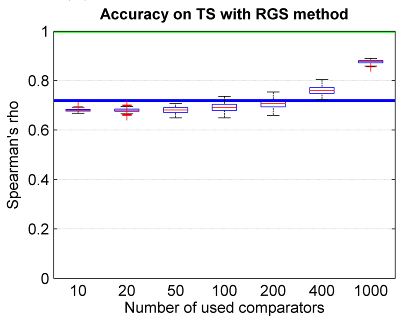
(a) Sushi_0, RGS, #$\mathbf{LS}_{aux} \geq 1$



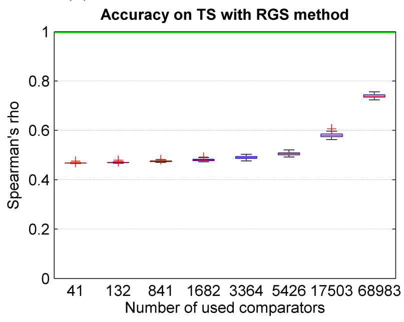
(b) Sushi_0, RGS, #$\mathbf{LS}_{aux} \geq 5$



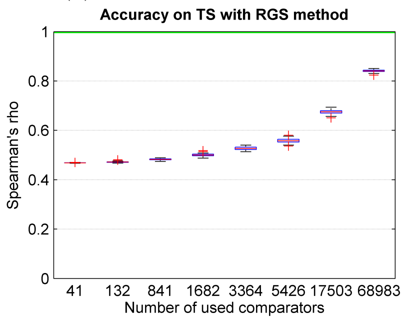
(c) Sushi_3, RGS, #$\mathbf{LS}_{aux} \geq 1$



(d) Sushi_3, RGS, #$\mathbf{LS}_{aux} \geq 5$



(e) MovieLens, RGS, #$\mathbf{LS}_{aux} \geq 1$



(f) MovieLens, RGS, #$\mathbf{LS}_{aux} \geq 5$

Figure 5.31: Ranking score when using the mean order to break ties and using perfect models (Sushi_0, Sushi_3 and MovieLens).

On OMIB, the use of perfect models do not change much, which was predictable since the extra-trees were able to predict, in a very efficient manner, the output. On MovieLens and both Sushi databases, however, we now see that adding more comparisons become beneficial, up to the point where all comparisons are used, hence producing a prediction which has a ranking score of 1. As already observed, imposing $\#\mathbf{LS}_{aux} \geq 5$ to a comparison in order to be a candidate for selection is also beneficial.

We have seen that $\#\mathbf{LS}_{aux}$ has an impact on the accuracy, which might seems quite intriguing at first sight as we are using perfect models (hence, the size of the **LS** should not have any influence), but since RGS updates the set of comparisons based on the ranking score observed on the **LS** and since a comparison which is highly represented in the **LS** would be likely to be also highly represented in the **TS**, then focusing on $\#\mathbf{LS}_{aux}$ to improve the performance makes sense. We thus perform an experiment where we sort the comparisons based on the number of examples in the **LS**. Results are shown in Table 5.11. The ranking scores in this table are more or less equivalent to the ranking score observed when using RGS, which means that sorting the comparisons based on the $\#\mathbf{LS}_{aux}$ is a good alternative.

T	Score
10	0.343
20	0.354
50	0.417
100	0.456
200	0.559
400	0.680
1000	0.828

(a) Sushi_0

T	Score
10	0.684
20	0.687
50	0.704
100	0.705
200	0.741
400	0.798
1000	0.884

(b) Sushi_3

T	Score
41	0.481
132	0.489
841	0.512
1682	0.530
3364	0.561
5426	0.587
17503	0.683
68983	0.827

(c) MovieLens

Table 5.11: Ranking score obtained by sorting the comparisons based on their number of examples, using the mean order to break ties and simulated perfect models.

5.5.7 Setting the j_1 and j_3 RGS parameters to huge values

In our experiments, we arbitrarily fixed the j_1, j_2 and j_3 parameters of RGS to 2000, 100 and 3000 respectively[6]. We wonder how sound this choice was.

In this section, we will not thoroughly evaluate the effect of the variation of each of these parameters on the ranking score; this study is left open for future work; but we will compare the ranking score of the RGS method with our arbitrarily chosen parameters to the ranking score of RGS when the j_1 parameter is set to 100,000 and the j_3 parameter is set to 150,000. We expect the RGS method to be very slow with such high parameters, but at least we will have an idea about the performance of RGS with a very large number of iterations, and we will be able to ensure that setting the j_1 and j_3 parameters to 2000 and 3000 respectively is sufficient.

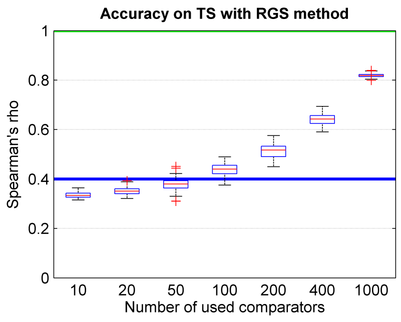
In this experiment, we use simulated perfect models, break the ties with the mean order and impose a comparison to have $\#\mathbf{LS}_{aux} \geq 5$ in order to be a candidate. We perform this experiment on Sushi_0, Sushi_3 and MovieLens and set the j_1 parameter to 100,000 and the j_3 parameter to 150,000. The j_2 parameter remains at 100. Figure 5.32 shows the results. We see no significant differences w.r.t. to the smaller arbitrary parameters which we used in the previous sections (see Figure 5.31 for comparison). We can thus deduce that setting j_1 and j_3 to 2000 and 3000 respectively is wise enough.

5.6 Complexity of our set selection algorithms

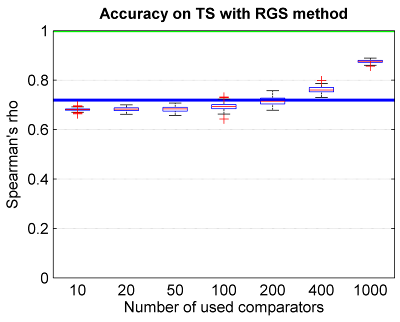
From the accuracy analysis, we were able to assess that the RGS selection method was the most appropriate for large values of N. However, we now want to consider the complexity of our methods when we impose a given $S_{\mathbf{TS}}(Q)$ to be reached. This complexity will be expressed in terms of computational time for time complexity in section 5.6.1 and in terms of number of comparisons for space complexity in section 5.6.2

We define a "meta-architecture" as a combination of an architecture and a mode (classification or regression) We consider several architectures, starting from the architecture used in PR, EDA, EGS and RGS and create variants by

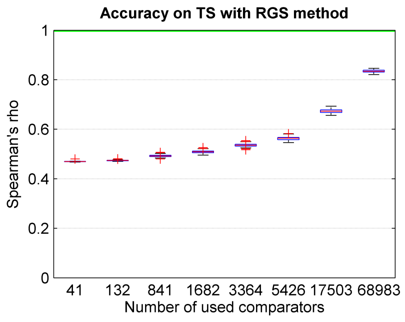
[6]i.e. The RGS algorithm stops when either 3000 iterations have been performed or when 2000 iterations have been performed and the last 100 iterations did not improve the score.



(a) Sushi_0



(b) Sushi_3



(c) Movielens

Figure 5.32: Ranking score obtained by using the mean order to break ties and using perfect models. The j_1 parameter is set to 100,000 and the j_3 parameter is set to 150,000.

varying one meta-parameter at a time according to the values in table 5.2. Each possible meta-architecture based on those architectures was taken into account. We dropped empty comparisons in each architecture and used the reverse alphanumerical order to break our ties.

For each set, we first consider the Spearman's ρ observed with the full RPC (with optimal meta-architecture among the considered ones, fixing T to $\#Q^{Full}$) and considered it as the reference. We then calculated 0.7, 0.8 and 0.9 of this value, which would represent the fact that a meta-architecture reaching this level would be "barely acceptable", "satisfying" and "good" respectively.

5.6.1 Time Complexity (computational speed)

Time complexity results are shown in Figure 5.33. The graphs are composed of an histogram and a green curve. The Y-axis represents the computational time in seconds for the histogram, and the ranking score $S_{\mathbf{TS}}$ in terms of ρ for the green line. Each histogram bar is composed of two series: the red series represents the computational time required to build the base learner while the blue series is the time required to select the subset of size T and to generate and store the **LS** outputs. The X-axis represents different combinations of Method/Expected rho on a logarithmic scale. In addition to the three expected rho values, we also show the most accurate meta-architecture for each selection method and for RPC. Each selected meta-architecture has been chosen so as to minimize the computational time while the ranking score on the **TS** reaches at least the required percentage (0.7, 0.8 or 0.9). When a histogram bar is missing, it means that none of our considered meta-architecture can be used to reach the required ranking score for a given set selection algorithms, and that larger values of T should have been considered.

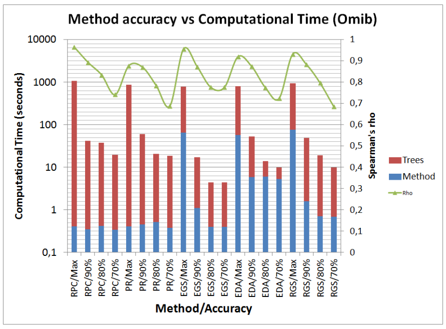
.

For OMIB and both Sushi databases, which are very small in terms of number of labels N, none of our selection methods can beat the computational time of RPC, unless the expectation is low (i.e. 0.70-0.80 of RPC's rho), In that case, the random selection is generally the best option.

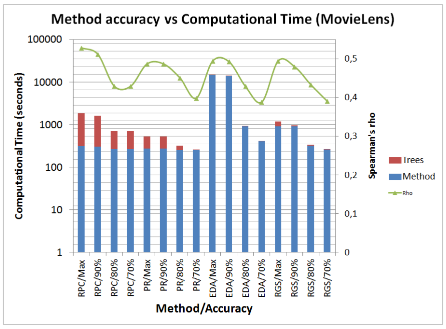
On MovieLens, however, the RGS and PR selection method are faster than RPC, even when 0.9 of RPC's rho is expected.

The computational complexity of comparator subset selection of RGS is linear in the size T of the target comparator subset and in the parameters j_1 and j_3. Indeed, each swap of one comparator can be carried out in constant time, independently of the value of T. To fix ideas, in the most demanding of our trials (MovieLens), we evaluated about 72,000 comparisons out of 1,413,721. Notice also that this elementary operation (evaluation of the ranking score on the **LS**) is quite fast compared to training a base learner, even with the very efficient Extra-Trees method.

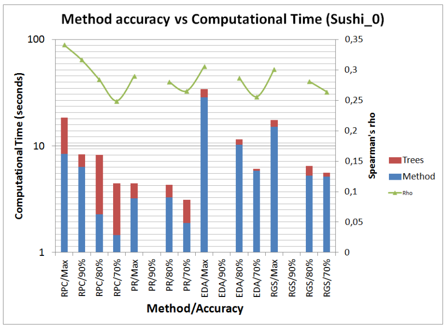
Finally, the EDA approach is very time consuming and can only beat RPC in terms of time complexity with the lowest rho expectation.



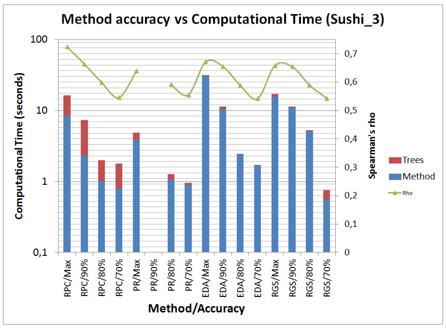
(a) OMIB



(b) MovieLens



(c) Sushi_0



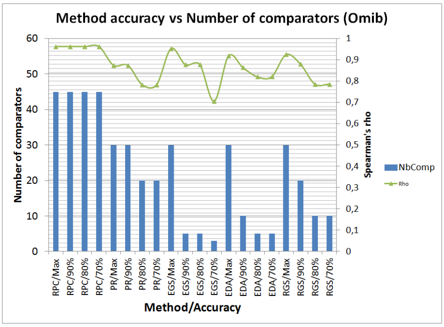
(d) Sushi_3

Figure 5.33: Comparison of the computational times. The blue histogram represents the time required to store the **LS** outputs in memory, select the set Q and compute its ranking score on **LS**. The red histogram represents the time required to build the corresponding models and compute its ranking score on **TS**. The green line represents $S_{\mathbf{TS}}$.

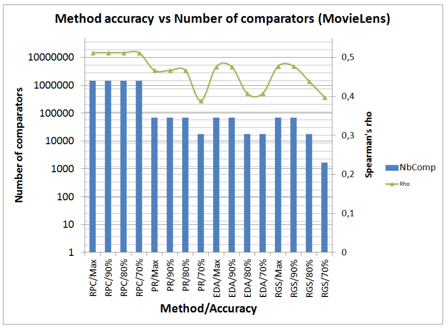
5.6.2 Space Complexity (memory usage)

The space complexity (and, as a corollary, the time complexity of the prediction phase) of our approach is determined essentially by T, the number of selected comparisons. This section will consider the quality of our selection methods (i.e. how many comparisons are required to form a "good" set) with respect to our requirements (rho value). Results are depicted in Figure 5.34. Note that, in this case, the green dots represents the ranking score of the "standard architecture" ($n_{\min} = 2$, $K = \sqrt{\#A}$, $M = 500$, Regression mode) for the corresponding T

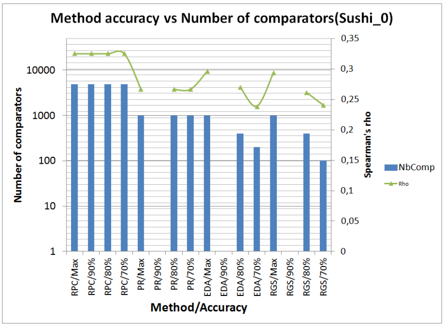
value.



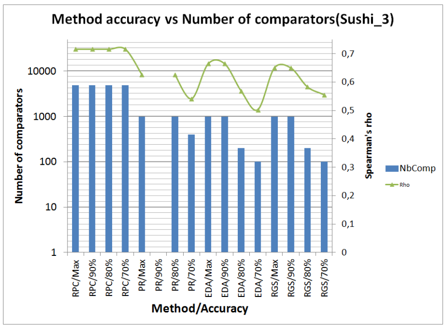
(a) OMIB



(b) MovieLens



(c) Sushi_0



(d) Sushi_3

Figure 5.34: Comparison of the required T for a given ρ. The blue histogram represents the required T. The green line represents $S_{\mathbf{TS}}$.

In many cases, the RGS method requires smaller subsets to cope with our requirements than any other considered set selection method. For instance, on MovieLens, 1682 comparisons are sufficient to reach 0.7 of RPC's rho. In general however, $N\sqrt{N}$ comparisons are required to reach 0.9 of RPC's rho, no matter what method is being used.

5.7 Concluding remarks

From all performed experiments, we extrapolate several important facts.

- When using the RPC scheme, the use of regression mode in the base learners construction is beneficial for any value of T, including $T = \#Q^{Full}$.
- Selecting the set Q which maximizes the ranking score on the **LS** outputs is relevant.
- When trimming the set of comparators, the RGS algorithm is the best method (compared to the other considered algorithms in this study) in most cases, as the number of required comparators is significantly reduced while not drastically affecting the ranking score $S_{\mathbf{TS}}$.
- Considering only the comparators which have at least a certain number of objects in their own **LS** is beneficial when trimming the set of available comparators. The extreme case of taking the T most represented comparators is not.
- The default order used to break ties can play an important role w.r.t the predictive accuracy, and especially when T is small. We have shown that, on the two partially ranked databases which we tested (Sushi and MovieLens), the problem is so hard to learn, from the given features, that the use of the mean order as unique prediction is actually the best alternative.
- The computational time of RGS does not depend on the number of labels N (since it is essentially determined by the parameters j_1 and j_3) and can be adjusted easily, which can be seen as another advantage of RGS.
- The RGS method is robust with respect to its tree parameters and does not require more observations than the full RPC scheme.

To conclude, we have proven that finding a nearly optimal subset $Q \subset Q^{Full}$ is not an easy task and we proposed several algorithms in order to achieve this goal, RGS, in combination with the drop of comparisons having less than 5 objects in their **LS**, being so far the best alternative. We thus have proven that one can efficiently trim the set of comparators when the number N of labels makes a problem impossible to be considered in a full RPC scheme.

In fine, we performed 120,000 tests on the OMIB database, 180,600 tests on the two Sushi databases and 107,400 tests on the MovieLens database which makes a total of 408,000 tests. Performing this huge amount of computations was made possible thanks to the SEGI (SErvice Général d'Informatique) which placed the NIC3 computing array at our disposal. This super-computer is composed of 1500+ cores, each of which has a maximum available memory of 32 gigabytes. Each user can access to a maximum of 240 cores simultaneously.

Chapter 6

Applications

Contents

6.1 Decision support systems 175
6.2 Recommender systems 176
6.3 Personalized devices 178
6.4 Ranking search results 179
6.5 Text categorization 180

Preference learning has many applications. Birlutiu et al., in their article [BGH10] divide them in three groups: Decision support systems (DSS), recommender systems and personalized devices.

We will also consider text categorization and search results clustering as applications which can be addressed using preference learning protocols.

This chapter will describe the role of preference learning (and especially label ranking) in those five fields of application and show how our trimming methodology could improve (or simply be used instead of) existing techniques.

6.1 Decision support systems

A DSS agent is a computer software which uses observations (and sometimes a knowledge base) to help a human user take an optimal decision. Mostly used in business companies, current implementation are satisfactory since the decision can in general be taken asynchronously from the observations. That is, if a software needs several days to compute the annual policy, it could be considered as acceptable.

However, in the medical field, a clinical DSS (or CDSS) should choose a valid action in a limited period of time. We could for instance think of a medical device which monitors a patient and could automatically provide medicine according to the patient state. The workflow of such a system in depicted on Figure 6.1. If the machine fails to compute the correct action or if the response is delayed, death can be the final result. Some CDSS systems currently do exist, but either (i) the relevant data is too huge to perform a decision process in real time or (ii) only a few symptoms are monitored and the system can thus only work for a specific disease. Using our methodology, an accurate and quick response could be obtained and would help current systems in monitoring more symptoms in real time.

An example of asynchronous CDSS is given in [Rul99]. This study investigates the relation between nurse care planning based on patient preferences and the patient outcome. The patient preferences were collected using preference elicitation techniques and displayed on the patient's chart. The study concludes that *"decision support for eliciting patient preferences and including them in nursing care planning is an effective and feasible strategy for improving nursing care and patient outcomes"*.

Rather than collecting the patients' preferences, we could train a preference learning model based on patients' features and apply the nursing care protocol according to the model predictions.

6.2 Recommender systems

In recommender systems, the objective is to design an agent which is able to propose some references to items which the user should find interesting according to his profile or his past preferences. An online book store would typically use a recommender system in order to boost its sales.

Algorithmically speaking, we can differentiate collaborative filtering, which builds a model according to the user's past preference and/or similar preferences of other users, and content-based filtering, which uses the features (character-

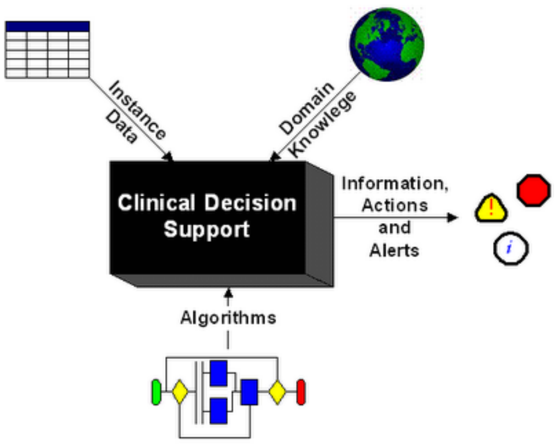


Figure 6.1: CDSS workflow
(Source:
http://motorcycleguy.blogspot.com/2008/06/clinical-decision-support.html)

istics) of selected items in order to define the profile of the user, then use this profile as feature vector of a model in order to predict items which are relevant for the given profile [dGIL$^+$]. *Amazon.com* and *Facebook.com* are websites using the former ([LSY03]) while *Internet Movie Database (IMDb)* and *Pandora Radio* are website using the latter [Raf10] (Figure 6.2).

Collaborative filtering can be addressed using label ranking algorithms (as quoted in [HFCB08] and in [WK10]). However, in website recommendations, a small number of items are generally provided, due to computational time issue. Indeed, if the website requires more than a few seconds to predict the user preferences, the customer will probably leave the website and the sale will be missed. But, using our trimming method, a lot more items could be provided, giving more chance that at least one of them would attract a customer, or the system response would be significantly lower for the same number of offered items, which should theoretically increase the sale rate.

Figure 6.2: Very famous websites use recommender systems

6.3 Personalized devices

Personalized devices are designed to be kept by a person at all times.

A hearing aid (Figure 6.3) is such a device. It amplifies and modulates sound for the wearer. As the required added intensity and modulation differs from a person to another, adjustments have to be performed before installing the device. Birlutiu et al. [BGH10] proposed the use of Multi-Task Preference Learning in order to model user preferences about different sound qualities. Our models could also be used in this context.

Figure 6.3: A hearing aid is a personalized device

6.4 Ranking search results

Although similar, the task of ranking search results differs from the task of a recommender system in the sense that the ordering depends both on the user preferences and on the query. Typically, a recommender system in a book store context will not consider the genre of the book it suggests. However, a user will not expect his search results to contain references to items which are not related to his query even if, on a general scale, he would be interested by those items.

One of the major issues when using search engines is that the number of relevant results can be huge (e.g. if the query is broad, like "power" or "tree"). The *cluster hypothesis* [vR79] suggests that clustering the search results would be beneficial, assuming that relevant documents are close to each other in the document space.

The task of clustering search results by using label ranking was proposed by Zhang et al. in [ZLTC07] where the authors extract the cluster labels from the documents content, from words appearing in similar search queries and from snippets (a short description of the web page provided by the search engine). They then build a utility function based on these three features and use this function to rank cluster labels. The top ranked cluster labels are then used to perform clustering and the corresponding clusters are ordered according to this ranking. Although the task of ranking cluster labels is performed without supervision, we could, instead of computing a utility function, compute pairwise comparisons according to the probability that the cluster label y_k is preferred over label y_l, and apply our trimming methodology on those comparisons.

6.5 Text categorization

The task of assigning one or more category to a text is called "Text categorization". Usually, a multi-label classifier is trained on a set of texts labeled with one or more classes. The prediction of such models will be strict, i.e. either a text belongs to a category or it does not belong into it.

The use of preference learning protocols to solve text categorization problems was already addressed in [ASS07] where the prediction is a ranking over the set of categories for a given object, representing the degree of suitability of this category with respect to the object. Clearly, this label ranking problem could also be solved using our methodology.

Chapter 7

Future work

Contents

7.1 Limits of this work 181

7.2 Possible improvements 182

7.2.1 Using prior knowledge 182

7.2.2 Other trimming algorithms 183

7.2.2.1 Best first 183

7.2.2.2 Greedy backward elimination 184

7.2.2.3 Conditional frequencies in EDA 185

7.2.3 Method parameters 185

7.2.4 Controlling partial ranking and noise 186

7.2.5 Extremely Randomized Ranking 186

7.2.6 Optimizing with a given budget 187

7.2.7 Using the mean ordering to select Q 187

7.2.8 Weighted distribution based on comparator **LS** . . . 188

Although this research was consequent, it did not explore all possible improvements and let some parts of this work open. We will first set the limits of the current work, then give hints and thoughts about possible future work.

7.1 Limits of this work

Firstly, our method was only tested on three dataset (four if one considers that Sushi actually contains two separate datasets). For an empirical research, this could be considered as low. However, one should take into account the fact that the preference learning field emerged only a few years ago. Hence, finding preference learning datasets is not a trivial task.

Secondly, only a few comparison selection methods were tested, but other trimming methods might exist and might potentially be even more accurate and/or robust. Concerning the EDA and RGS trimming methods, the number of iterations was chosen arbitrarily. We think that it would be interesting to test the evolution of the accuracy curve according to the number of iterations. And, to conclude with RGS, the weight update function was also chosen arbitrarily. It is possible that more constraining functions will make the model converge to a good solution more rapidly.

Finally, in this thesis, the hypothesis were only confirmed (or infirmed) by an empirical survey. A mathematical proof would probably be more convincing.

7.2 Possible improvements

In this section, we will present some thoughts and ideas which could be interesting to implement and test. Even if we cannot guarantee that these will help improving the model, they should at least be considered in future work.

7.2.1 Using prior knowledge

A generic approach consists in considering that the labeled partial orderings are strict information (i.e. elements in this ordering are not necessarily preferred over the unlabeled ones).

But sometimes, we know that the partial ranking is an ordering of selected objects and that unselected objects are less important. This is the case of the Sushi database, where we know that selected Sushis are preferred over unselected ones. We could then add this information into the model to improve the method.

Formally, on a q_{kl} comparison, the value 1 means that y_k is preferred over y_l, 0 that y_l is preferred over y_k, and 0.5 means that this information is not provided.

So there are three way of transforming an ordering into a set of pairwise comparisons. When considering strict information on partial rankings, we attribute the value of 0.5 to any comparison where neither y_k nor y_l is in the control ranking. This is the usual procedure and we only considered this transform in this thesis. If we know that the partial ordering is a first selection, we can give a score of 1 to a q_{kl} comparison if y_k is in the control ranking and y_l is not. Finally, if we want to avoid decimal values, we can also consider that every label in the control ranking is preferred over the unlabeled ones and, moreover, that these unlabeled labels are sorted alphanumerically. So, with this last method, we are ensured that each comparison will always predict a preference of 0 or 1, even if this value is somewhat factious.

Of course, these transforms would not impact the control ordering, and the scoring scheme would be identical, even with the last proposed transform.

7.2.2 Other trimming algorithms

As previously explained, this study only accounted four trimming algorithms, namely PR, EGS, RGS and EDA.

Finding inspiration from existing variable selection techniques, we could transpose some of them in ensemble trimming methods. These methods will be explained in the following sections.

7.2.2.1 Best first

Best first algorithms work by constructing and evaluating partial solutions, then improving those which provide the best scores. The search is performed along a solution tree and, at each step, each solution improvement from the best node is considered and represented as a branch. The node which has a locally optimal score is fully expanded.

This could easily be applied to our preference learning problem, by iteratively expanding the set of comparators which provides so far the best ranking score.

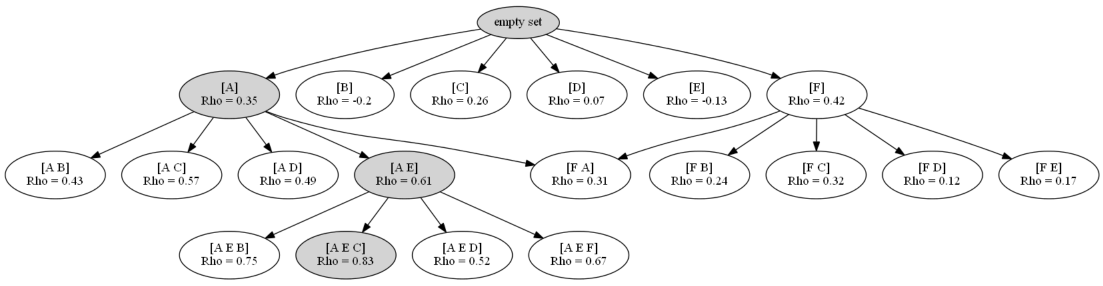


Figure 7.1: Set selection using best first search

Figure 7.1 provides an example of this solution building. In this example, we are provided with 6 binary comparisons (A to F) and wish to find the set of three comparisons which maximizes the ranking score. Starting from the empty set, the first iteration considers every comparison. The use of comparison F provides the best score (0.42) and its branch is thus expanded. However, after expanding the branch, every possible combination of F and another comparison provides a score which is worse than the score obtained by using only comparison A, which is now the best solution and will thus be expanded at the next iteration. Continuing this process, we find that the set $[AEC]$ is the best solution.

However, this technique has the same drawback as the greedy approach, namely that the solver can be stuck in a local maximum. Indeed, if we consider the previous example, it is possible that a solution expanded from the node containing the set $[D]$ would be eventually better than the proposed solution, but this path is never considered in the solution building due to a very low score on the first node of this branch.

So, as well as the greedy approach, this solution can potentially provide a good solution, but not necessarily the optimal one.

7.2.2.2 Greedy backward elimination

The previously depicted method builds a solution by growing partial ones.

However, the dual approach can be considered. Instead of iteratively adding comparisons to progressively improve the solution, we could start with the complete set Q^{Full} of comparisons and iteratively compute the ranking score of the method when using the sets $Q^{Full} \backslash \{q_i\} \forall q_i \in Q^{Full}$, then updating Q^{Full} such that

$$Q^{Full} = Q^{Full} \backslash \{\arg\max_{q_i} S_{\mathbf{LS}}(Q^{Full} \backslash \{q_i\})\}. \tag{7.1}$$

Of course, starting from a huge set Q^{Full}, the computational time could be prohibitive (i.e. each update requires $\mathcal{O}(N^2)$ Spearman evaluations). But we could, instead of considering one comparison at a time, remove a certain number j of comparisons. Again, the optimum is not guaranteed to be reached.

7.2.2.3 Conditional frequencies in EDA

The Estimation of Distribution Algorithm generally provided poor results (i.e. close to results observed with random selection). One possible explanation would be that the combination of some comparisons makes the prediction very accurate while the effect of those comparisons taken independently is marginal. If this hypothesis is true, producing sets of comparisons based on individual frequencies observed on top ranked sets would effectively provide poor results.

To take the correlation between comparisons into account, we could count conditional frequencies instead of independent frequencies. That is, how many times comparison q_{kl} was observed on top ranked sets given that comparison q_{ij} was observed in such sets. These conditional frequencies could then be used to build a conditional distribution from which comparisons would be drawn.

Another approach could be the use of Bayesian Inference to compute, at each iteration, the posterior distribution over the model parameters and use this distribution to derive the posterior predictive distribution, rather than maximizing the likelihood of the parameters [GWKS08].

7.2.3 Method parameters

As emphasized in the first section of this chapter, several parameters were fixed arbitrarily. For instance, the randomized greedy algorithm will perform at least 2,000 iterations, and at most 3,000 iterations. Also, the EDA approach counts the frequencies of comparisons in the top 50 sets out of 100. These parameters were chosen by following our intuition, but we do not have any guarantee that these are optimal. We think that fine-tuning these parameters could be interesting, as we could estimate the impact of these parameters on the accuracy of the set selection methods.

7.2.4 Controlling partial ranking and noise

The use of the fully ranked database OMIB and the partially ranked databases Sushi and MovieLens helped us to infer a general hypothesis about the effect of partial ranking and noise on our trimming methods.

However, a more controlled experiment can be performed by creating variants of the synthetic OMIB database. We could add noise (either by modifying the feature vectors or by performing swaps in the labeled orderings) or simulate partial rankings by removing labels from the labeled ordering. Several levels of noise/missing information could be considered (e.g. 10%, 30%, 50%, 70% and 90%) and the effect on the accuracy be measured.

7.2.5 Extremely Randomized Ranking

In this thesis, we constructed one ranking model, based on T comparisons, each corresponding comparator model consisting of M trees.

But, getting inspiration from the Extra-Tree ensemble algorithm, we could construct R ranking models, still based on T comparisons, each corresponding comparator model consisting of M/R trees. The predictions on the R models would be aggregated so as to produce the final ordering.

In the same manner that building M identical trees using a deterministic approach does not have any sense (in that case, building one of these trees would produce the same result), building R models with the same sets of T comparisons would not be useful. The PR selection method could then be used in that purpose. Hence, the computational time required to build R models based on M/R trees would be similar to the computational time required to build one model based on M trees. This would not be necessarily true when using more complex selection methods (EGS, RGS or EDA) as their computational complexity cannot be neglected.

We hope that this technique would, as in Extra-trees, reduce the variance while not significantly increase the bias.

The rank aggregation could be performed in two manners. Either the majority ordering is selected (as in classification mode), but this would only have sense if some orderings are significantly represented (e.g. their number of occurrences in the R outputs are strictly bigger than $2 * \lceil \frac{R}{N!} \rceil$), or we could compute a mean ordering (as in regression mode) by using one of the rank aggregation techniques seen in Section 3.3.

7.2.6 Optimizing with a given budget

In this study, we trimmed the comparison set by training only a subset of size T, each corresponding comparator being modeled using M trees, M being identical for all comparators.

We could add more flexibility and consider the total number of trained trees (TM) as a budget. Each tree in this budget would be allocated to the modelization of one comparison, but two distinct comparisons could be modeled by a different number of trees. For instance, instead of training 100 comparators with 500 trees each, we could train 1000 comparators with 50 trees each, or we could allocate 25,000 trees to build 500 comparators and then the remaining 25,000 trees on 1000 other comparators,... Any combination would be acceptable as long as the budget is respected.

The problem of allocating the trees for a given budget could be solved using an optimization technique, based on the ranking score of a given tree allocation.
Once a nearly optimal protocol for tree allocation is discovered, we could monitor the accuracy curve when progressively decreasing the allocated budget, in the same manner that we monitored the accuracy curved for various values of parameter T.

7.2.7 Using the mean ordering to select Q

In order to select our set Q of comparisons, we could first compute the mean ordering of Π, then use this ordering to select N comparisons by choosing every q_{kl} where y_k is ranked directly before y_l in the mean ordering.

We think that such a set Q would have a ranking score which is in average greater than the ranking score of a randomly drawn set, with the advantage of satisfying the surjectivity condition described in Section 4.4.2.1.

7.2.8 Weighted distribution based on comparator LS

We have shown in Section 5.5.2 that pre-selecting the set of available comparisons based on the number of objects in their own **LS** is a good idea.

We could compute a weighted distribution over the set of all comparisons where the probability to draw a comparison would be proportional to the number of objects in its own **LS**. Then, we could use apply our PR algorithm to draw a comparison set from this distribution (with no rematch).

Chapter 8

Conclusion

In this thesis, we first provided an introduction to artificial intelligence, by describing the historical context and by defining the numerous domains in that field.

Then we provided the reader with some gentle background information and definitions about machine learning, model construction, preference learning, and we presented the Ranking by Pairwise Comparison algorithm.

Our main contribution was to study the interest of pre-selecting in a dataset dependent way a small subset of size T, preferably with $T = \mathcal{O}(N)$ of binary label comparisons in order to scale the Ranking by Pairwise Comparison framework to problems with a large number of labels. The motivations of this idea are similar to those of feature selection largely studied in the context of supervised learning, in the sense that selecting a subset of comparisons may both improve computational complexity and accuracy, specially on partially ranked and/or noisy datasets.

Among several randomization based comparison filtering methods that we have studied, we found that the **Randomized Greedy Search** was the most effective one, both in terms of computational complexity and accuracy. Our empirical evaluations on both synthetic and real-world datasets from the literature, show that this method may yield nearly **state-of-the-art accuracy** with a computational complexity essentially **linear** in the number N of labels.

We also showed that using class-probability estimators (regression mode) rather that hard classifiers further helps the method to reach a sufficient level of accuracy by using a smaller number of label comparators and adds some robustness to RPC with respect to its n_{min} meta-parameter. Finally, we also discussed about the robustness of the method, and showed that RGS is quite robust with respect to meta-parameter choices of the base learner.

We have shown that pre-trimming the set of available comparisons and keeping the comparisons which have at least 5 objects in their own **LS** is beneficial when selecting $T = \mathcal{O}(N)$ comparisons. However, the extreme case of ranking the comparisons based on the number of objects in their own **LS** and using the top T comparisons is too restrictive.

Our proposed methodology reduces the computational and space complexity of RPC while producing accurate models. Thus applying our methods on label ranking applications would be beneficial.

Chapter 9

Published work

Contents

9.1 **Bioinformatics publications** 191
9.2 **Preference learning publications** 193

During my PhD studies, I had the opportunity to work on various topics and to publish my work in conferences and journals. You will find below the list of my "first author" publications. The first section contains publications that are not directly related to my work on preference learning but which were produced during the earlier stages of my PhD studies when I focused on Bioinformatics, and the second section is more specific to my thesis on preference learning.

9.1 Bioinformatics publications

Hiard, S., Marée, R., Colson, S., Hoskisson, P. A., Titgemeyer, F., van Wezel, G. P., Joris, B., Wehenkel, L. & Rigali, S. (2007). PREDetector: A new tool to identify regulatory elements in bacterial genomes. *Biochemical and Biophysical Research Communications*, 357(4), 861-864.

Abstract
In the post-genomic area, the prediction of transcription factor regulons by position weight matrix-based programmes is a powerful approach to decipher biological pathways and to modelize regulatory networks in bacteria. The main difficulty once a regulon prediction is available is to estimate its reliability prior to start expensive experimental validations and therefore

trying to find a way how to identify true positive hits from an endless list of potential target genes of a regulatory protein. Here we introduce PREDetector (Prokaryotic Regulatory Elements Detector), a tool developed for predicting regulons of DNA-binding proteins in bacterial genomes that, beside the automatic prediction, scoring and positioning of potential binding sites and their respective target genes in annotated bacterial genomes, it also provides an easy way to estimate the thresholds where to find reliable possible new target genes. PREDetector can be downloaded freely at http://www.montefiore.ulg.ac.be/-hiard/PreDetector (c) 2007 Published by Elsevier Inc.

Hiard, S., Charlier, C., Coppieters, W., Georges, M. & Baurain, D. (2010). Patrocles: a database of polymorphic miRNA-mediated gene regulation in vertebrates. *Nucleic Acids Research*, 38(Database), 640-D651.

Abstract
The Patrocles database (http://www.patrocles.org/) compiles DNA sequence polymorphisms (DSPs) that are predicted to perturb miRNA-mediated gene regulation. Distinctive features include: (i) the coverage of seven vertebrate species in its present release, aiming for more when information becomes available, (ii) the coverage of the three compartments involved in the silencing process (i.e. targets, miRNA precursors and silencing machinery), (iii) contextual information that enables users to prioritize candidate 'Patrocles DSPs, including graphical information on miRNA-target coexpression and eQTL effect of genotype on target expression levels, (iv) the inclusion of Copy Number Variants and eQTL information that affect miRNA precursors as well as genes encoding components of the silencing machinery and (v) a tool (Patrocles finder) that allows the user to determine whether her favorite DSP may perturb miRNA-mediated gene regulation of custom target sequences. To support the biological relevance of Patrocles' content, we searched for signatures of selection acting on 'Patrocles single nucleotide polymorphisms (pSNPs) in human and mice. As expected, we found a strong signature of purifying selection against not only SNPs that destroy conserved target sites but also against SNPs that create novel, illegitimate target sites, which is reminiscent of the Texel mutation in sheep.

9.2 Preference learning publications

Hiard, S. & Wehenkel, L. (2011). Using Class-probability Models instead of Hard Classifiers as Base Learners in the Ranking by Pairwise Comparison Algorithm. In S., Thatcher (Ed.), *ICMLC 2011 3rd International Conference on Machine Learning and Computing* Volume 1 (pp. 218-222). Chengdu, China: IEEE.

Abstract
In the field of Preference Learning, the Ranking by Pairwise Comparison algorithm (RPC) consists of using the learning sample to derive pairwise comparators for each possible pair of class labels, and then aggregating the predictions of the whole set of pairwise comparators for a given object in order to produce a global ranking of the class labels. In its standard form, RPC uses hard binary classifiers assigning an integer (0/1) score to each class concerned by a pairwise comparison. In the present work, we compare this setting with a modified version of RPC, where soft binary class-probability models replace the binary classifiers. To this end, we compare ensembles of extremely randomized classprobability estimation trees with ensembles of extremely randomized classification trees. We empirically show that both approaches lead to equivalent results in terms of Spearman's rho value when using the optimal settings of their metaparameters. However, we also show that in the context of small and noisy datasets (e.g. with partial ranking information) the use of class-probability models is more robust with respect to variations of its meta-parameter values than the hard classifier ensembles. This suggests that using (soft) class-probability comparators is a sensible option in the context of RPC approaches.

Hiard, S., Geurts, P. & Wehenkel, L. (2012). Comparator selection for RPC with many labels, *To appear in ECAI 2012 20th European Conference on Artificial Intelligence*. Montpellier, France.

Abstract
The Ranking by Pairwise Comparison algorithm (RPC) is a well established label ranking method. However, its complexity is of $O(N^2)$ in the number N of labels. We present algorithms for selecting, before model construction, a subset of comparators of size $O(N)$, to reduce computational complexity without loss in accuracy.

Chapter 10

Acknowledgments

Firstly, I would like to thank my promoter, Louis Wehenkel, who dedicated a lot of his time to my supervision and whose advices were always very relevant. I also enjoyed our more informal discussions, and he proved many times that I could count on him, even on darker days during my thesis.

I also want to thank the University of Liège for its funding as well as the ARC Biomod and PAI BioMAGNet, giving me the opportunity to work on this research, which was challenging but also thrilling.

Johannes Fürnkranz and Eyke Hüllermeier helped me to understand the RPC algorithm during conferences or by e-mail, and I am very grateful for this.

The extra-tree implementation was provided by Pierre Geurts and it helped me focus on the trimming part of the job without having to develop this supervised learning algorithm from scratch. Many thanks for this significant saving in time. I also would like to thank Pierre for being the co-reviewer of the thesis and for his feed-back.

I thank Toshihiro Kamishima and the GroupLens project for the database usage allowance, and again Dr. Kamishima for his complete explanation on the way that the Sushi database was constructed.

During my PhD thesis, I had many informal discussion with some other PhD students or postdocs, and great ideas often emerged from that. These people are, in alphabetical order, Florence Belmudes, Vincent Botta, Jing Dai, Damien Ernst, Raphaël Fonteneau, Pierre Geurts, Fabien Heuze, Vân Anh Huynh-Thu, Francis Maes, Raphaël Marée, Jonathan Pisane, Olivier Stern, François Van-Lishout and Da Wang.

I thank all my colleagues in the research unit or at the GIGA, which I did not explicitly quote but who contributed to a very nice working environment, with also a part of fun and laugh.

Finally, I would like to finish with a special thanks to my friends and relatives for their support, and in particular to my companion, Frédérique, who spent several evenings alone and who did the household chores far more often than me without complaining.

Appendix A

Efficient implementations

Contents

A.1 Weighted distribution 198
A.1.1 Simple array 199
A.1.2 Linked List 199
A.1.3 Binary tree 200
A.2 Spearman's ρ correlation coefficient 201
A.2.1 Naive method 201
A.2.2 Optimized method 203
A.2.3 Computational gain 204
A.2.4 Efficient computation of ρ in an EGS/RGS context . 205
A.3 Pre-computing 206
A.3.1 Tree building 207
A.3.2 Rejected comparisons 207

During the implementation phase, we had to face some technical difficulties which do not directly appear in the abstract concept, but which arises when trying to put the formalism into shape. We though that mentioning these issues and the manner that we chose to solve them is important and will be helpful to anyone who would want to implement our methods.

Indeed, from a mathematical point of view, any implementation of a function would be acceptable as long as it produces the desired output. From a computer science point of view though, computational time is a critical aspect of function implementation as inappropriate coding could lead a function to produce its correct output several months or years later than an optimally implemented function. One of the most common example is the implementation of the Fibonacci series. The computation of one element in this series is performed by

$$\begin{aligned} Fib(0) &= 0, \\ Fib(1) &= 1, \\ Fib(N) &= Fib(N-1) + Fib(N-2). \end{aligned}$$

The most direct (and naive) implementation of this function is given in Algorithm 18 where the computation of $Fib(N-1)$ and $Fib(N-2)$ are performed recursively. The main issue is that $Fib(N-1)$ needs to compute $Fib(N-2)$ and $Fib(N-3)$, thus $Fib(N)$ will be computed by $Fib(N-2) + Fib(N-2) + Fib(N-3)$, and so on, leading to unnecessary computations. Computing the value in a loop while keeping the values of $Fib(N-1)$ and $Fib(N-2)$ in two variables (as depicted in Algorithm 19) would be a more optimized implementation of this function.

Algorithm 18 Fibonacci naive implementation (FibNaive)

```
Input: An integer value $v \geq 0$
Output: The value of element $v$ in the Fibonacci series

if $v < 2$ then
  return $v$
else
  return FibNaive($v - 1$)+FibNaive($v - 2$)
end if
```

In this section, we will emphasize on our implementation of some time-consuming methods.

A.1 Weighted distribution

When drawing a comparison q from the distribution P, we wish to ensure that $q \notin Q$. Moreover, the Estimation of Distribution Algorithm (see section 4.4.3 page 103) iteratively updates a (possibly) non uniform distribution. We thus

Algorithm 19 Fibonacci optimized implementation (FibOptim)

```
Input: An integer value $v \geq 0$
Output: The value of element $v$ in the Fibonacci series

if $v = 0$ then
  return 0
else
  $v_1 \leftarrow 0$
  $v_2 \leftarrow 1$
  for $i = \{2, ..., v\}$ do
    $v_3 \leftarrow v_1 + v_2$
    $v_1 \leftarrow v_2$
    $v_2 \leftarrow v_3$
  end for
  return $v_2$
end if
```

need to efficiently handle this kind of distribution such that picking with no rematch but also updating should be done quickly.

Several options were taken into account.

A.1.1 Simple array

Storing the elements in an array and giving a variable number of slots to each of them is a good way to represent a weighted distribution. To give a real world representation, this technique is similar to dividing a slice of paper into areas of different sizes, and throwing a dart at it. Objects with a greater area are more likely to be hit. (Figure A.1).

Unfortunately, if the picking is done in an order of $\mathcal{O}(1)$, the updating process is $\mathcal{O}(N)$ in the number N of elements, since the entire array needs to be considered.

A.1.2 Linked List

To counter the order $\mathcal{O}(N)$ in the previous method, one could instead use a linked list (Figure A.2).

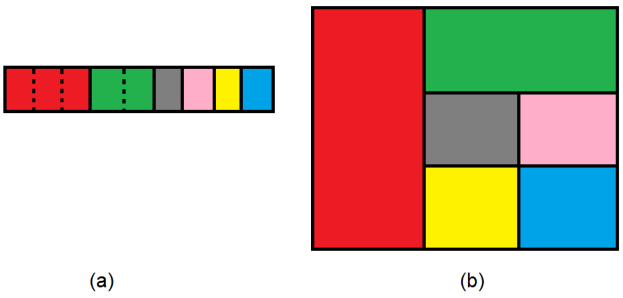


Figure A.1: Objects with more weight are more likely to be picked if the area is larger (b) or, informatically, if the number of slots is bigger(a)

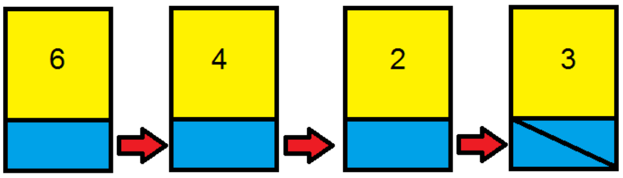


Figure A.2: A linked-list improves the updating process, but the gain is canceled in the picking phase

Advantages are two-fold. In the first place, memory usage is smaller, as one slot is sufficient to represent an element and, secondly, the updating process is performed in the order of $\mathcal{O}(1)$, considering that pointers to the current and previous element are kept after picking. However, the picking phase needs in average $N/2$ iterations.

A.1.3 Binary tree

Using a tree structure (Figure A.3) is actually a very good compromise.

The node structure contains a maximum value, a threshold, the value of the element, two pointers to sons and a pointer to the parent. The structure is built is the following way : given a set of objects O, each $o^i \in O$ having its own weight w^i and its value v^i, we select a pivot at the center of the set. This pivot sets the last element of the subset $O_1 \subset O$, the remaining items form the subset O_2. A node is created, with the threshold set to $\sum_{i=1}^{\#O_1} w^i$ and the maximum value of $\sum_{i=1}^{\#O} w^i$. We recurse on O_1 and O_2 and attach the corresponding structures to the left and right pointer (respectively) of the node. In the case where $\#O = 1$, the created node is a leaf, and the value v^i is labeled on this node.

At each pick, a random value is drawn from 0 to the maximum value of the root node. Then, as long as we do not reach a leaf, we continue to the left son if this value is smaller than the threshold or to the right otherwise (in that case, the random value is decreased by the threshold). This is done in $\mathcal{O}(logN)$.

To update the tree structure, one only has to modify to nodes in the direct hierarchy of the selected node, which is also done in $\mathcal{O}(logN)$

So, the complexity of this structure is $\mathcal{O}(\log N)$, which is better than $\mathcal{O}(N)$ from the previous structures for any value of N.

A.2 Spearman's ρ correlation coefficient

Optimizing the Spearman's ρ correlation function is extremely important since it is called every time that two rankings are to be compared, which occurs very frequently. We will first present the naive method for solving this problem, then present our solution and finally compare the computational speed of both implementations. We will conclude by explaining how we re-used some computations in the particular case of EGS and RGS.

A.2.1 Naive method

We will first present the naive version of the function's implementation. Although not time efficient, it has the benefit of being easy to understand, as it follows the procedure that one would intuitively apply to solve this problem.

The concept is very simple. It works in two loops. The first loop will remove objects which are not common in both rankings. Since each prediction is an ordering over the whole set of labels, it is impossible to find a label in the control which is not be present in the prediction, thus the search has to be performed

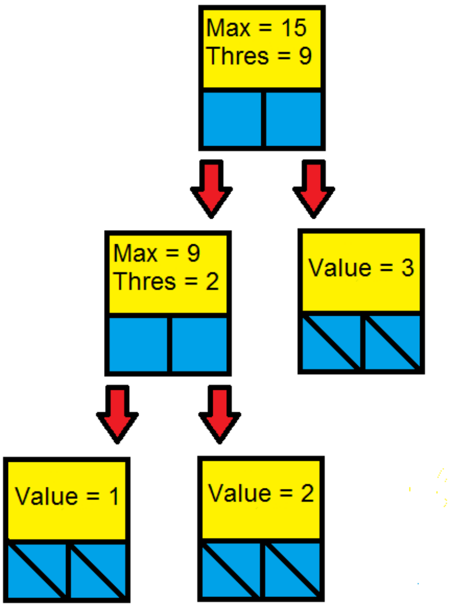


Figure A.3: A binary tree has a complexity of $\mathcal{O}(\log N)$ for picking and updating.

only in one direction. The second loop will compute the distance between those new rankings.

Algorithm 20 Spearman's ρ naive implementation

Input: A control ranking π of size $\#\pi$, a table of votes V of size $N \geq \#\pi$ representing another ranking
Output: The correlation measure between π and the ranking represented by V

```
$Ranking \leftarrow$ result of sorting the $N$ labels according to the votes in $V$
for every $i \in Ranking$ do
    $found \leftarrow false$
    for $j = 0; j < \#\pi$ && $\neg found; j++$ do
        if $Ranking[i] = Control[j]$ then
            $found \leftarrow true$
        end if
    end for
    if $\neg found$ then
        $Ranking \leftarrow Ranking \setminus \{i\}$
    end if
end for
$diff \leftarrow 0$
for every $i \in Ranking$ do
    for every $j \in Control$ do
        if $Ranking[i] = Control[j]$ then
            $diff \leftarrow diff + (i - j)^2$; BREAK
        end if
    end for
end for
$rho \leftarrow 1 - \frac{6*diff}{\#\pi*((\#\pi)^2-1)}$
return $rho$
```

A.2.2 Optimized method

Searching for matches using a nested loop is far from being the optimal solution. Ingeniously sorting can help us achieve the same goal in less iterations. Indeed, efficient sorting algorithms can perform in the order of $\mathcal{O}(N \log N)$. One could then sort the ranking according to the position of its elements (i.e. $Ranking[i]$ would not represent the i^{th} preferred object but the position of object i in the ranking) and one pass would be sufficient to compare the two rankings afterwards.

Algorithm 21 Spearman's ρ optimized implementation

```
Input: A control ranking $\pi$ of size $\#\pi$, a table of votes $V$ of size $N \geq \#\pi$ representing another ranking
Output: The correlation measure between $\pi$ and the ranking represented by $V$

sort $\pi$ according to the position of its elements
for every $i \in \{1, 2, ..., N\}$ do
    $NewTab[i] \leftarrow -1$
    if $i \in \pi$ then
        $NewTab[i] \leftarrow V[i]$
    end if
end for
$Ranking \leftarrow$ result of sorting the $N$ labels according to the values $NewTab$ and truncate it to the first $\#\pi$ top objects.
$diff \leftarrow 0$
sort $Ranking$ according to the position of its elements
for every $i \in Ranking$ do
    $diff \leftarrow diff + (Ranking[i] - control[i])^2$
end for
$rho \leftarrow 1 - \frac{6*diff}{\#\pi*((\#\pi)^2-1)}$
return $rho$
```

A.2.3 Computational gain

In order to measure the computational gain of using the optimized solution rather than the naive one, we performed several simulations. In each of these simulations, an ordering of size $\#\pi$, representing the control, was compared to an ordering of size N, representing the prediction. Both orderings were artificially created. We varied the value of parameters $\#\pi$ and N across the simulations, and compared the computational time of both implementations. As an attempt to search for best and worse cases, we performed three variations of these simulations. In the first case, items in the control were ranked identically to the prediction (correlation = 1); in the second case, the control was a partial ordering of the reverse prediction (correlation = -1); and in the third case both orderings were generated using an uniform distribution (correlation $\approx$ 0). The observed results are depicted in Table A.1.

Clearly, the optimized implementation outperforms the naive one. The com-

Parameters		Perfect correlation		Reverse correlation		Null correlation	
$\#\pi$	N	Naive	Optim.	Naive	Optim.	Naive	Optim.
10	10	3.68	11.22	3.59	11.2	3.85	13.31
10	100	1088.9	30.94	1062.4	29.97	1110.7	31.84
10	1000	960190	234	963460	229.3	961280	260.5
100	100	1090.4	168.5	1104.5	177.9	1143.5	219.9
100	1000	958630	408.7	958470	405.6	962060	464.9
1000	1000	956910	1937.5	966270	1964	963470	3163.7
736	1682	4548990	2160.6	4542750	1926.6	4549770	2600.6

Table A.1: Comparison between optimized and naive implementations of the Spearman's rho evaluation (time in milli-seconds).

putational time is approximately proportional to N^3 (without much influence of parameter $\#\pi$) in the latter, while the order of magnitude is hard to guess for the former, but which is at most linear with respect to $\#\pi$ and N.

The naive form does not seem to be much affected by the correlation of the rankings, while the optimized form requires significant additional time, yet in an acceptable range, to compute when the rankings are not correlated.

Also note that for very small values of $\#\pi$ and N, the naive version is more efficient and should be used instead. In this thesis, this would only concern the OMIB database, while the computational time would be reduced by (97.13% - 97.18%) for Sushi and (99.94% - 99.96%) for MovieLens.

A.2.4 Efficient computation of ρ in an EGS/RGS context

The EGS (and its derivate RGS) performs only a single comparison swap before recomputing the Spearman's ρ correlation coefficient. We could thus use a part of the computations performed in the previous iteration in order to compute this ρ in a more efficient manner than recomputing everything at each iteration.

Each comparison q_{kl} provides preference information about y_k and y_l for any given x. If we store the table of votes for each object, we can thus update this table at each swap $q_{kl} \leftrightarrow q_{ij}$ by removing the votes v_k and v_l from the table, and add v_i and v_j into it. Hence, the table will not have to be reconstructed at each iteration, and the update process is independent of T. Algorithm 22 describes the implementation of this protocol.

Algorithm 22 Optimized handling of the table of votes for RGS

```
*First Initialization*
**Input:** A set **LS** of $n$ objects and a set $Q$ of $T$ comparisons based on $N$ labels
**Output:** A table of votes

$V \leftarrow$ a table of size $n \times N$ filled with zeros
for every object $x^i \in$ LS do
    for every comparison $q_{kl} \in Q$ do
        $V[i][k] \leftarrow V[i][k] + v_k^i$
        $V[i][l] \leftarrow V[i][l] + v_l^i$
    end for
end for
return $V$

    *Update process*
**Input:** A table of votes $V$ of size $n \times N$ and a comparator swap $q_{kl} \leftrightarrow q_{jm}$
**Output:** An updated table ready for the Spearman's $\rho$ evaluation

for every $i \in \{1, 2, ..., n\}$ do
    $V[i][k] \leftarrow V[i][k] - v_k^i$
    $V[i][l] \leftarrow V[i][l] - v_l^i$
    $V[i][j] \leftarrow V[i][j] + v_j^i$
    $V[i][m] \leftarrow V[i][m] + v_m^i$
end for
return $V$
```

A.3 Pre-computing

Computing in advance some calculations in order to be able to re-use them one or several times later is called "pre-computing". The advantage is a gain in time since the same calculations are performed only once but, as a counterpart, it requires disk storage in order to maintain access to the results.

We performed pre-computing in two stages of our framework : tree building and rejected comparisons and objects.

A.3.1 Tree building

In addition to time gain, pre-computing the tree ensemble used as base learning reduces the variance as the observable accuracy variations are only caused by the trimming method and not by the base learner model.

A.3.2 Rejected comparisons

Depending on the manner of dealing with sparsity in partially ranked datasets, we might have to reject comparisons from the model construction if they do not provide any pairwise preference information. However, trying to select T comparisons in a set $Q^{'Full}$ of informative comparators would be an impossible task if $\#Q^{'Full} < T$. Thus, rather than iteratively selecting comparisons and later discover that these comparisons are not in a sufficient number, we could pre-compute the number of usable comparisons and select them all if this number is smaller than the desired T.

In the dual case, scanning the database in search for an object which provides preference information on the considered pair of labels can also be pre-computed.

These pre-computations must still be repeated for each value of the learning set size which we want to consider.

Appendix B

Result figures

Contents

B.1 Accuracy 210
B.1.1 Reverse alphanumerical order 210
B.1.1.1 Using dummy models 211
B.1.1.2 Dropping empty comparisons 215
B.1.2 Reverse alphanum / Mean order 219
B.1.2.1 Classification 219
B.1.2.2 Regression 221
B.1.2.3 Perfect models 222
B.1.3 Mean order 224
B.2 Correlation between LS and TS ranking scores . 226
B.3 Complexity 227
B.4 Robustness of RGS 229
B.4.1 Omib 229
B.4.2 Sushi_0, Dropping empty comparisons 231
B.4.3 Sushi_0, Using a dummy model 232
B.4.4 Sushi_3, Dropping empty comparisons 233
B.4.5 Sushi_3, Using a dummy model 235
B.4.6 MovieLens, Dropping empty comparisons 236
B.4.7 MovieLens, Using a dummy model 237

This Appendix contains all the result figures which are being discussed in Chapter 5, while some were not displayed in that chapter for readability purpose.

In Section B.1, you will find boxplot figures showing the accuracy, in terms of ranking score, of the tested methods in various settings, as discussed in Sections 5.3, 5.4, and 5.5. Section B.2 shows the correlation between ranking score on the **TS** and ranking score on the **LS** for a given set Q as discussed in Section 5.4.1. Section B.3 shows the result figures concerning time and space complexity of our presented algorithms, as discussed in Section 5.6. Finally, Section B.4 shows the effect of sub-optimal meta-parameters on the RGS algorithm, as discussed in Section 5.5.3.

B.1 Accuracy

In this section, we will present the result figures which we obtained over various experiments in which the parameters are : (i) The default order used to break the ties (the reverse alphanumerical order or the mean order) either during the learning phase or the prediction phase, (ii) the required $\#\mathbf{LS}_aux$ for a given comparison in order to be a candidate for selection, (iii) the mode of the model (Classification or Regression) as well as the fact that the "ObjectID" attribute is being used or not and (iv) the size T of the subset Q of comparisons. The tree parameters are set to the following: $n_{\min} = 2$, $K = \sqrt{\#A}$ and $M = 500$ for each experiment shown in this section.

This section will be subdivided in three parts, based on the default order used to break the ties. In section B.1.1, the reverse alphanumerical order is used at both learning and prediction stage. In section B.1.2, the reverse alphanumerical order is used at the learning stage and the mean order is used at the prediction stage and finally, in section B.1.3, the mean order is used at both learning and prediction stage.

B.1.1 Reverse alphanumerical order

In this section, the reverse alphanumerical order is used to break the ties at both learning at prediction stage. This section will be subdivided in two parts: in the first part, we use a dummy model[1] to represent empty comparisons and, in the second part, we drop empty comparisons, hence reducing the set of comparisons to draw from. Since both settings would act similarly on the OMIB database (which is completely ranked), we chose to display its related figures in the first part. In both subsections, figures on the left hand side are obtained by

[1]The dummy model which we use always outputs the value of 0.5.

training the models in classification mode, while figures on the right hand side are obtained by training the models in regression mode.

B.1.1.1 Using dummy models

The figures in this section were obtained after using a dummy model to represent empty comparisons.

B.1.1.1.1 Omib

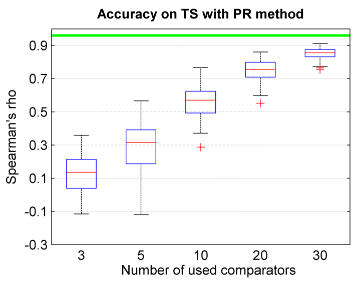


Figure B.1: Accuracy using the PR selection method in classification mode

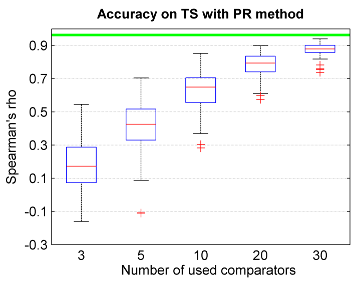


Figure B.2: Accuracy using the PR selection method in regression mode

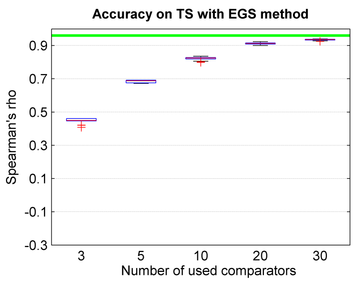


Figure B.3: Accuracy using the EGS selection method in classification mode

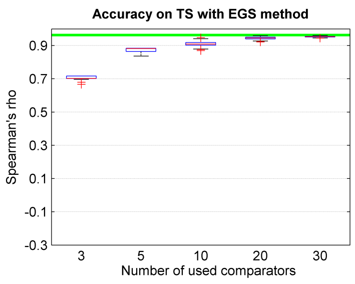


Figure B.4: Accuracy using the EGS selection method in regression mode

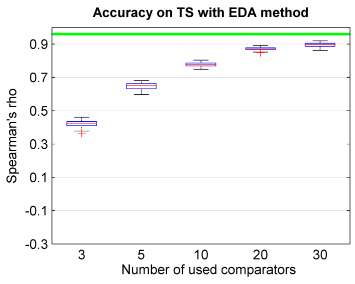


Figure B.5: Accuracy using the EDA selection method in classification mode

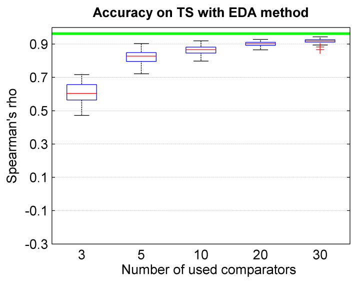


Figure B.6: Accuracy using the EDA selection method in regression mode

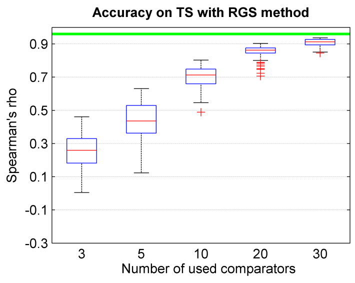


Figure B.7: Accuracy using the RGS selection method in classification mode

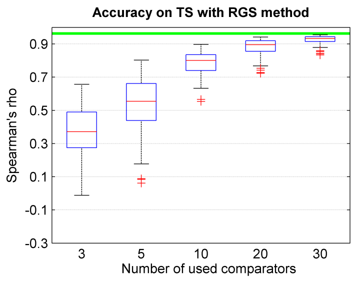


Figure B.8: Accuracy using the RGS selection method in regression mode

B.1.1.1.2 Sushi_0, Using a dummy model

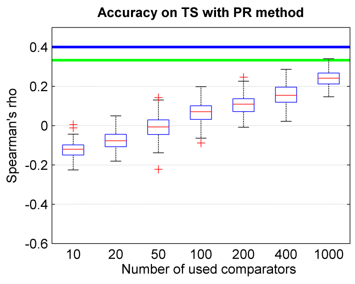


Figure B.9: Accuracy using the PR selection method in classification mode

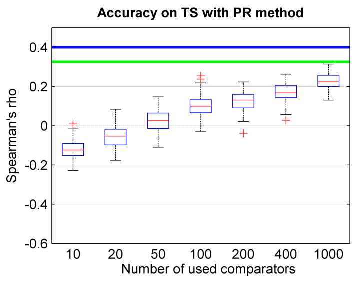


Figure B.10: Accuracy using the PR selection method in regression mode

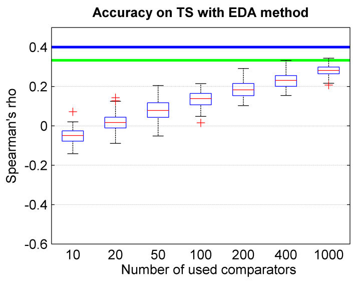


Figure B.11: Accuracy using the EDA selection method in classification mode

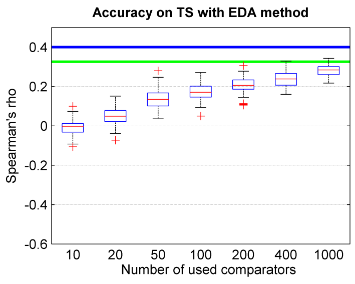


Figure B.12: Accuracy using the EDA selection method in regression mode

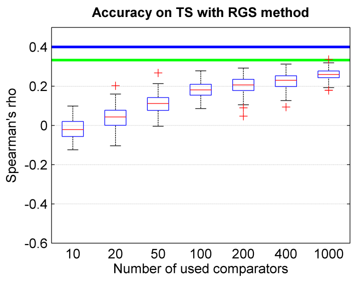


Figure B.13: Accuracy using the RGS selection method in classification mode

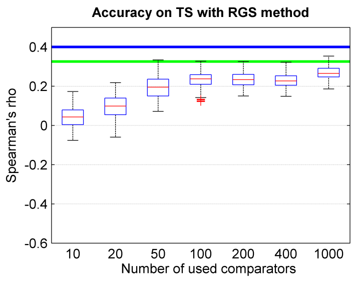


Figure B.14: Accuracy using the RGS selection method in regression mode

B.1.1.1.3 Sushi_3, Using a dummy model

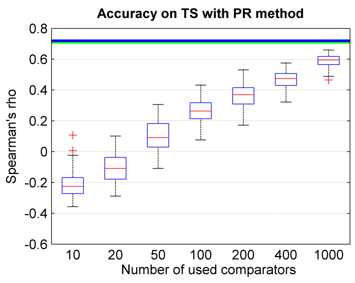


Figure B.15: Accuracy using the PR selection method in classification mode

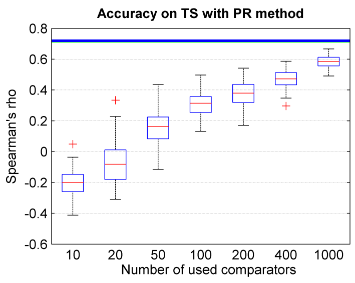
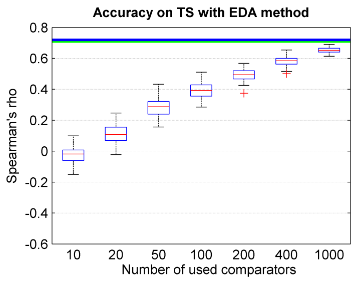


Figure B.16: Accuracy using the PR selection method in regression mode



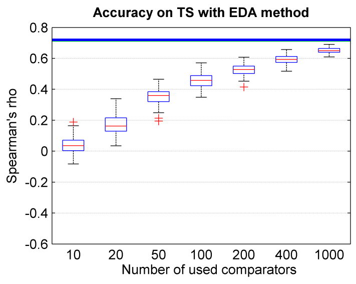
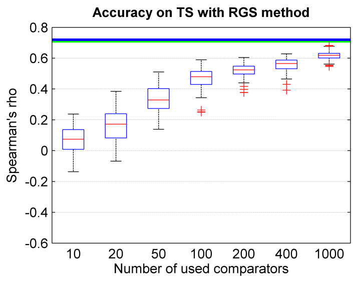


Figure B.17: Accuracy using the EDA selection method in classification mode

Figure B.18: Accuracy using the EDA selection method in regression mode



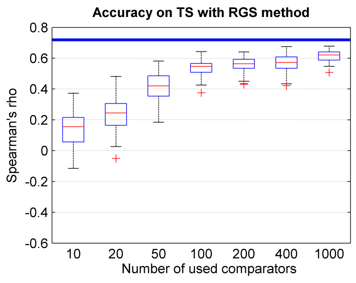
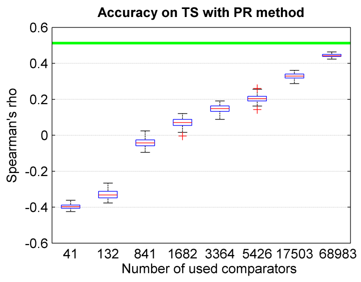


Figure B.19: Accuracy using the RGS selection method in classification mode

Figure B.20: Accuracy using the RGS selection method in regression mode

B.1.1.1.4 MovieLens, Using a dummy model



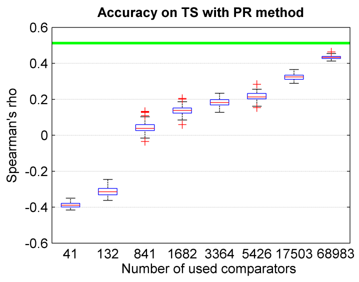


Figure B.21: Accuracy using the PR selection method in classification mode

Figure B.22: Accuracy using the PR selection method in regression mode

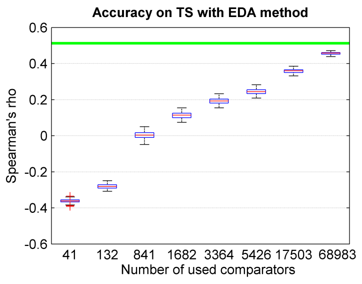


Figure B.23: Accuracy using the EDA selection method in classification mode

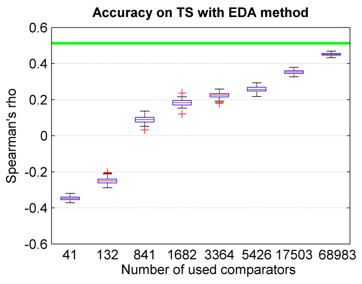


Figure B.24: Accuracy using the EDA selection method in regression mode

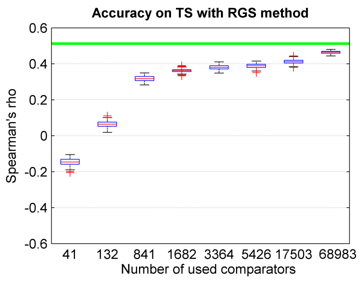


Figure B.25: Accuracy using the RGS selection method in classification mode

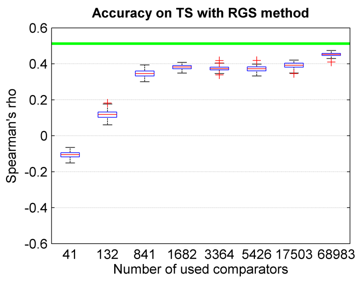


Figure B.26: Accuracy using the RGS selection method in regression mode

B.1.1.2 Dropping empty comparisons

The figures displayed in this section were obtained after dropping empty comparisons.

B.1.1.2.1 Sushi_0, Dropping empty comparisons

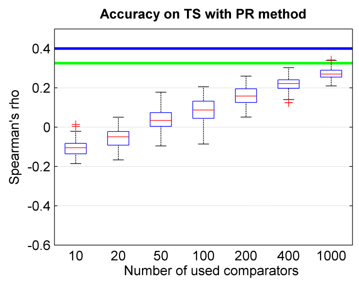


Figure B.27: Accuracy using the PR selection method in classification mode

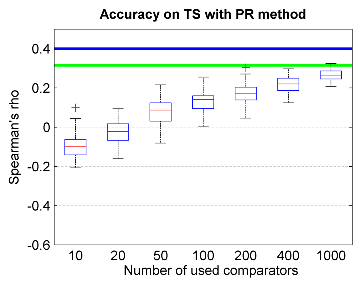


Figure B.28: Accuracy using the PR selection method in regression mode

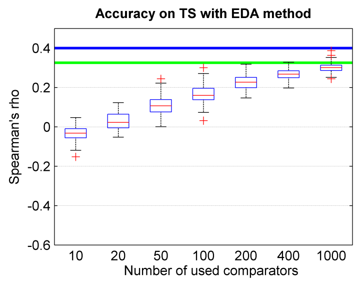


Figure B.29: Accuracy using the EDA selection method in classification mode

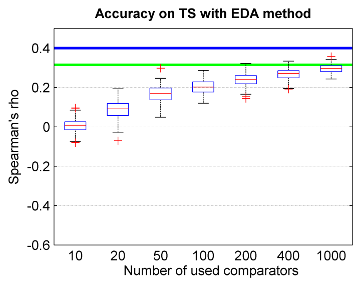


Figure B.30: Accuracy using the EDA selection method in regression mode

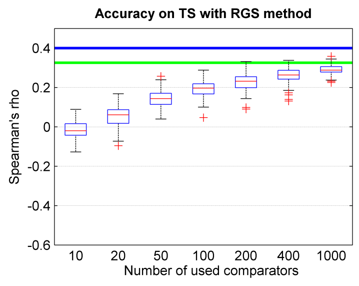


Figure B.31: Accuracy using the RGS selection method in classification mode

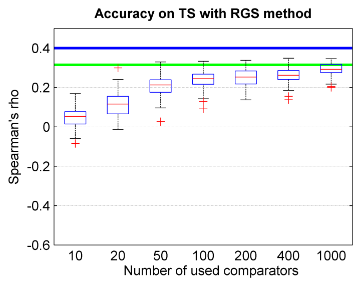


Figure B.32: Accuracy using the RGS selection method in regression mode

B.1.1.2.2 Sushi_3, Dropping empty comparisons

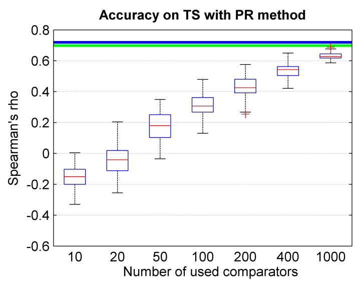


Figure B.33: Accuracy using the PR selection method in classification mode

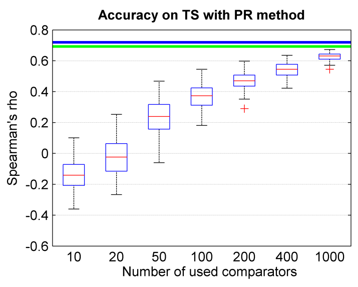


Figure B.34: Accuracy using the PR selection method in regression mode

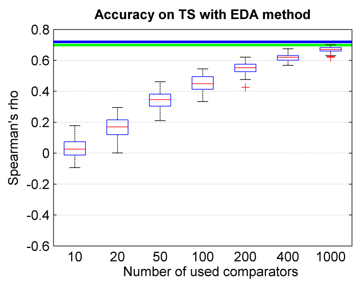


Figure B.35: Accuracy using the EDA selection method in classification mode

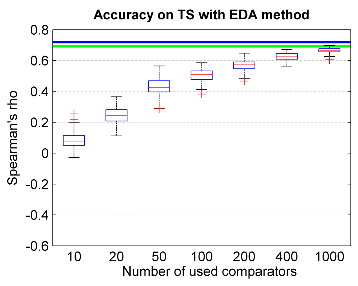


Figure B.36: Accuracy using the EDA selection method in regression mode

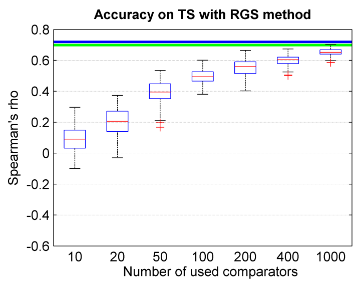


Figure B.37: Accuracy using the RGS selection method in classification mode

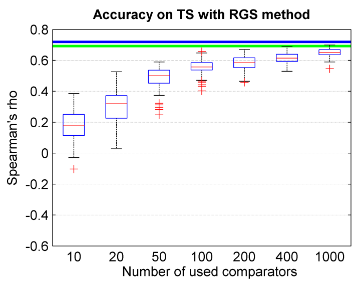


Figure B.38: Accuracy using the RGS selection method in regression mode

B.1.1.2.3 MovieLens, Dropping empty comparisons

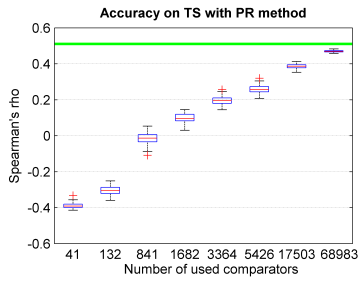


Figure B.39: Accuracy using the PR selection method in classification mode

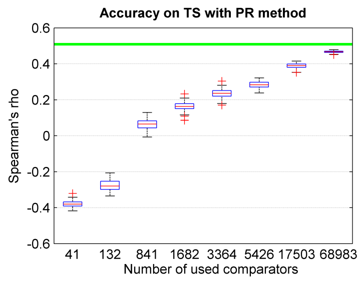


Figure B.40: Accuracy using the PR selection method in regression mode

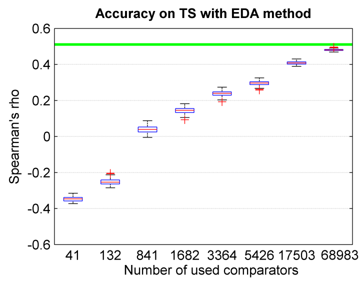


Figure B.41: Accuracy using the EDA selection method in classification mode

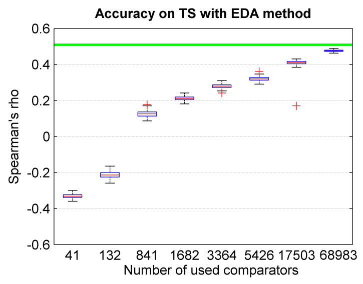


Figure B.42: Accuracy using the EDA selection method in regression mode

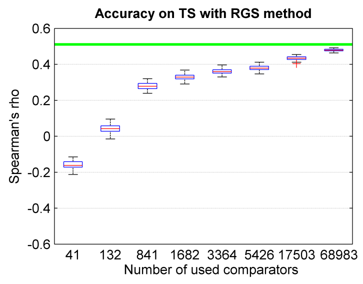


Figure B.43: Accuracy using the RGS selection method in classification mode

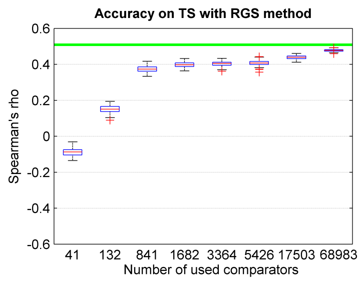


Figure B.44: Accuracy using the RGS selection method in regression mode

B.1.2 Reverse alphanum / Mean order

In this section, the reverse alphanumerical order is used to break the ties at the learning stage and the mean order is used to break the ties at the prediction stage. This section will be subdivided in three parts: in the first part, the models are built in classification mode; in the second part, the models are built in regression mode and, in the last part, we use simulated perfect models to represent our comparators. To select a subset Q of comparisons, we used EGS for the OMIB database, and RGS for MovieLens and both Sushi's. In the first two parts, we dropped empty comparisons and, in the last part, we also perform an experiment where $\#\mathbf{LS}_{aux} \geq 5$.

B.1.2.1 Classification

In this section, the models are trained in classification mode. The figures on the left hand side are obtained when the "objectID" attribute is used as feature in the model construction and, in the right hand side figures, this attribute is not considered in the model construction.

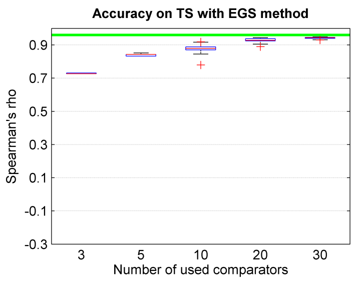


Figure B.45: Accuracy using EGS in classification mode with the "objectID" attribute (Omib)

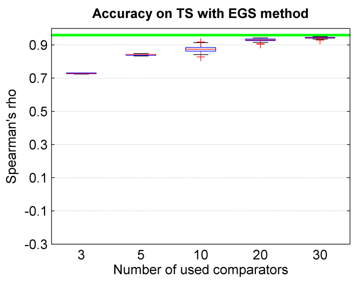


Figure B.46: Accuracy using EGS in classification mode without the "objectID" attribute (Omib)

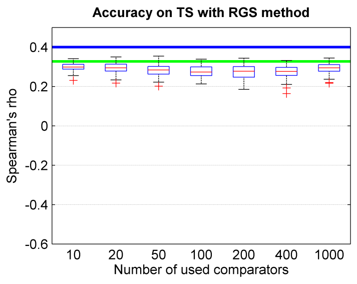


Figure B.47: Accuracy using RGS in classification mode with the "objectID" attribute (Sushi_0)

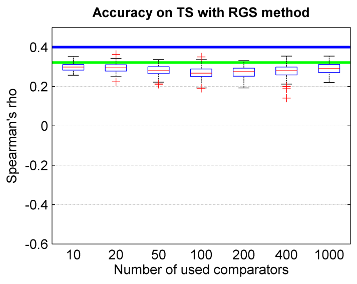


Figure B.48: Accuracy using RGS in classification mode without the "objectID" attribute (Sushi_0)

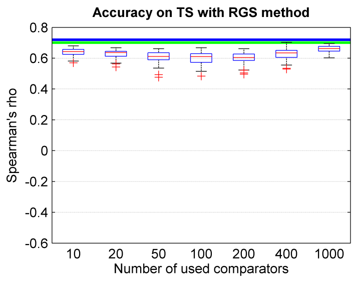


Figure B.49: Accuracy using RGS in classification mode with the "objectID" attribute (Sushi_3)

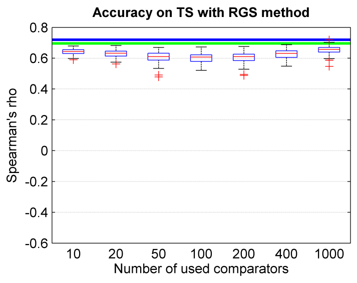


Figure B.50: Accuracy using RGS in classification mode without the "objectID" attribute (Sushi_3)

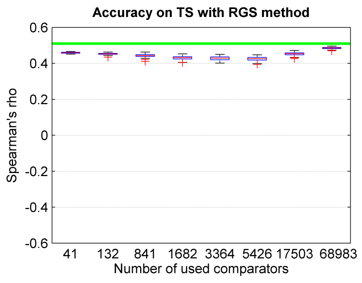


Figure B.51: Accuracy using RGS in classification mode with the "objectID" attribute (MovieLens)

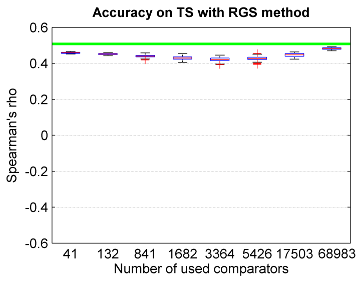


Figure B.52: Accuracy using RGS in classification mode without the "objectID" attribute (MovieLens)

B.1.2.2 Regression

In this section, the models are trained in regression mode. The figures on the left hand side are obtained when the "objectID" attribute is used as feature in the model construction and, in the right hand side figures, this attribute is not considered in the model construction.

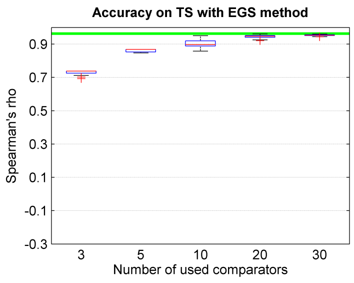


Figure B.53: Accuracy using EGS in regression mode with the "objectID" attribute (Omib)

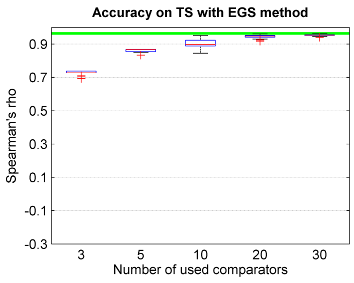


Figure B.54: Accuracy using EGS in regression mode without the "objectID" attribute (Omib)

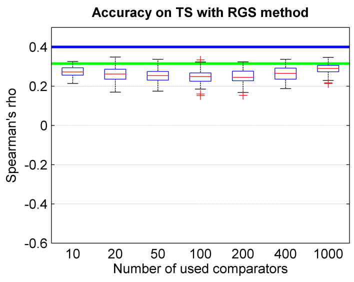


Figure B.55: Accuracy using RGS in regression mode with the "objectID" attribute (Sushi_0)

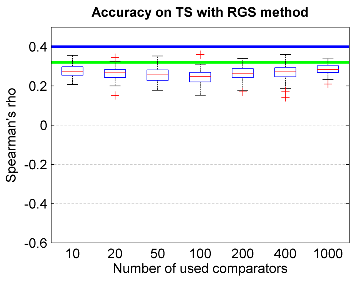


Figure B.56: Accuracy using RGS in regression mode without the "objectID" attribute (Sushi_0)

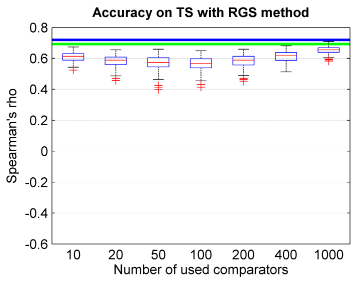


Figure B.57: Accuracy using RGS in regression mode with the "objectID" attribute (Sushi_3)

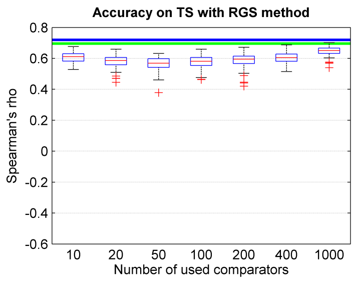


Figure B.58: Accuracy using RGS in regression mode without the "objectID" attribute (Sushi_3)

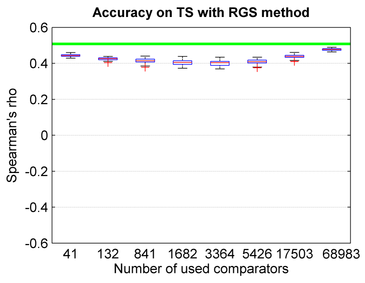


Figure B.59: Accuracy using RGS in regression mode with the "objectID" attribute (MovieLens)

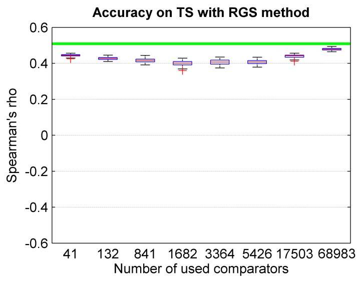


Figure B.60: Accuracy using RGS in regression mode without the "objectID" attribute (MovieLens)

B.1.2.3 Perfect models

In this section, we do not train any model. Instead, the "prediction" is obtained by scanning the supervision. The figures on the left hand side are obtained when $\#\mathbf{LS}_{aux} \geq 1$ and, in the right hand side figures, when $\#\mathbf{LS}_{aux} \geq 5$, except for the OMIB database which is not affected by this parameter.

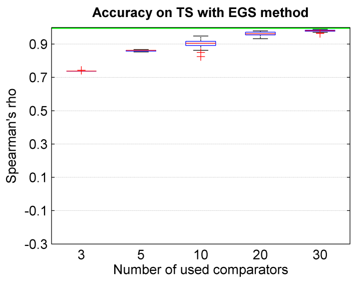


Figure B.61: Accuracy using EGS and perfect models (Omib)

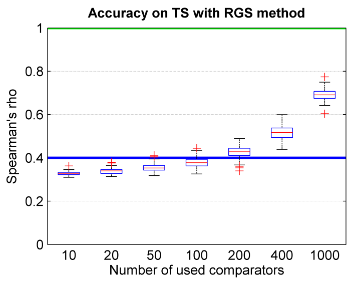


Figure B.62: Accuracy using RGS and perfect models with $\#\mathbf{LS}_{aux} \geq 1$ (Sushi_0)

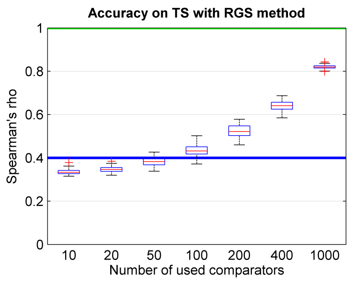


Figure B.63: Accuracy using RGS and perfect models with $\#\mathbf{LS}_{aux} \geq 5$ (Sushi_0)

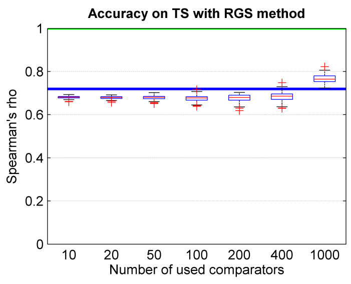


Figure B.64: Accuracy using RGS and perfect models with $\#\mathbf{LS}_{aux} \geq 1$ (Sushi_3)

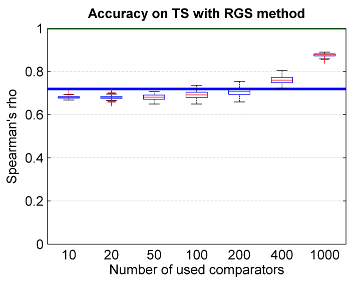


Figure B.65: Accuracy using RGS and perfect models with $\#\mathbf{LS}_{aux} \geq 5$ (Sushi_3)

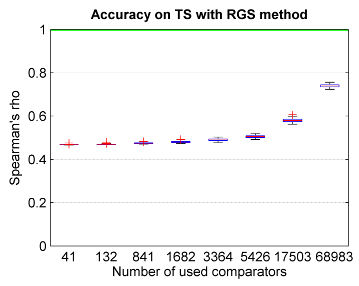


Figure B.66: Accuracy using RGS and perfect models with $\#\mathbf{LS}_{aux} \geq 1$ (MovieLens)

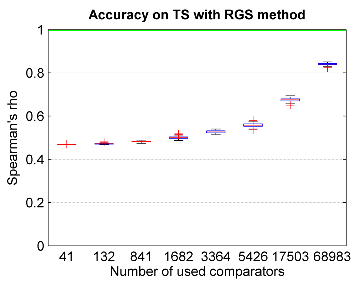


Figure B.67: Accuracy using RGS and perfect models with $\#\mathbf{LS}_{aux} \geq 5$ (MovieLens)

B.1.3 Mean order

In this section, the mean order is used to break the ties at both learning at prediction stage. In this experiment, we drop empty comparisons. Figures on the left hand side are obtained by training the models in classification mode, while figures on the right hand side are obtained by training the models in regression mode.

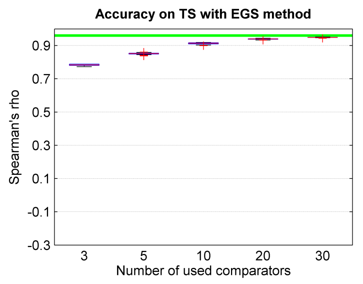


Figure B.68: Accuracy using EGS in classification mode (Omib)

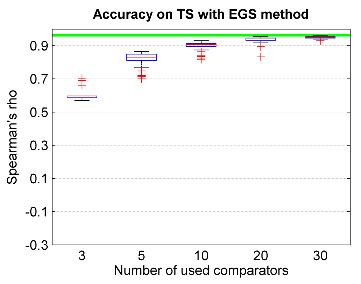


Figure B.69: Accuracy using EGS in regression mode (Omib)

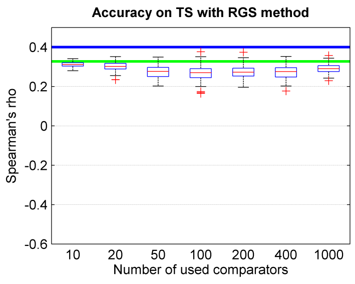


Figure B.70: Accuracy using RGS in classification mode (Sushi_0)

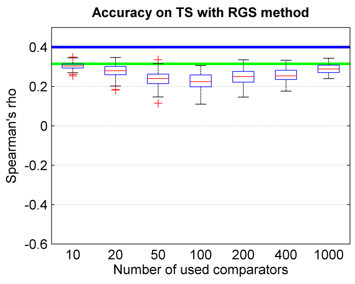


Figure B.71: Accuracy using RGS in regression mode (Sushi_0)

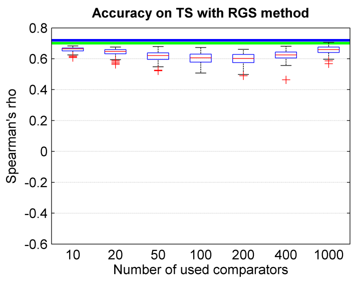


Figure B.72: Accuracy using RGS in classification mode (Sushi_3)

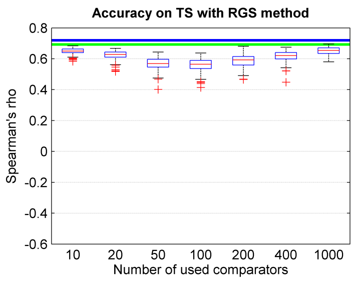


Figure B.73: Accuracy using RGS in regression mode (Sushi_3)

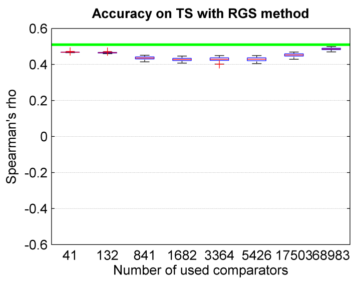


Figure B.74: Accuracy using RGS in classification mode (MovieLens)

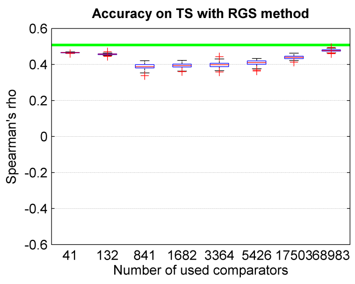


Figure B.75: Accuracy using RGS in regression mode (MovieLens)

B.2 Correlation between LS and TS ranking scores

In this section, one can find the figures, discussed in Section 5.4.1, representing the correlation between $S_{\mathbf{LS}}$ and $S_{\mathbf{TS}}$. On these graphs, each dot represents a set Q, obtained by the PR algorithm, in a given architecture, i.e. a combination of the tree parameters and the parameter T. We performed two variants of the experiment: in one experiment, we dropped empty comparisons (left hand side figures) and, in the second experiment, we used a dummy model to represent empty comparisons (right hand side figures). In each case, and for each database, there is a (more or less) linear correlation between $S_{\mathbf{LS}}$ and $S_{\mathbf{TS}}$, hence optimizing the set Q based on $S_{\mathbf{LS}}$ is relevant.

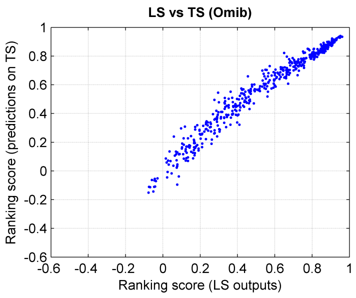


Figure B.76: LS vs TS correlation (Omib)

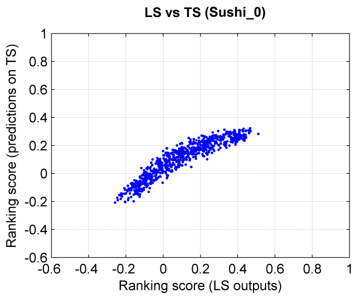


Figure B.77: LS vs TS correlation (Sushi_0, dropping empty comparisons)

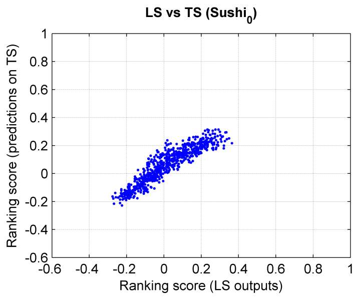


Figure B.78: LS vs TS correlation (Sushi_0, using a dummy model)

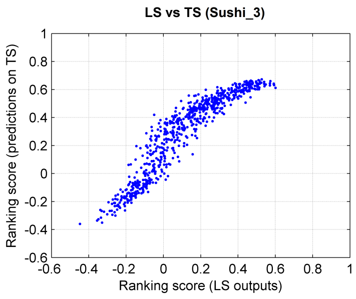


Figure B.79: LS vs TS correlation (Sushi_3, dropping empty comparisons)

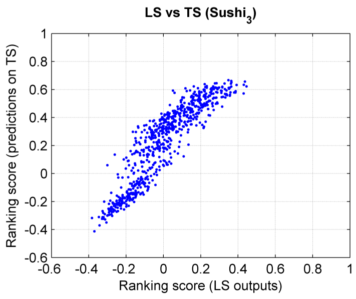


Figure B.80: LS vs TS correlation (Sushi_3, using a dummy model)

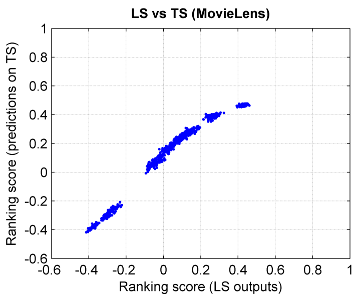


Figure B.81: LS vs TS correlation (MovieLens, dropping empty comparisons)

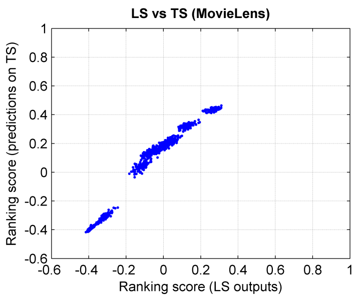


Figure B.82: LS vs TS correlation (MovieLens, using a dummy model)

B.3 Complexity

In this section, the figures concerning the complexity, as discussed in Section 5.6, can be found. Figures on the left hand side provide information about the time complexity and figures on the right hand side provide information about the space complexity. In both cases, each histogram bar represents the minimum required amount of resource (computational time or memory, depending on the case) for a given accuracy, which is given by the green line.

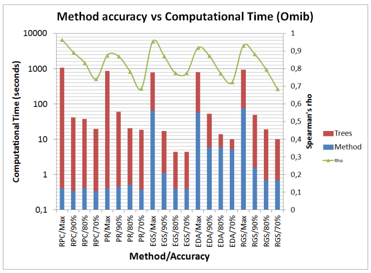


Figure B.83: Accuracy vs Computational time (Omib)

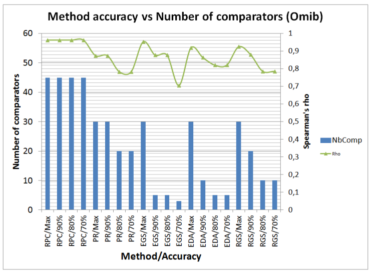


Figure B.84: Accuracy vs Number of classifiers (Omib)

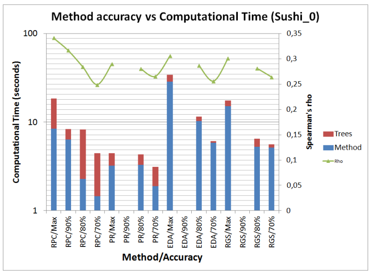


Figure B.85: Accuracy vs Computational time (Sushi_0)

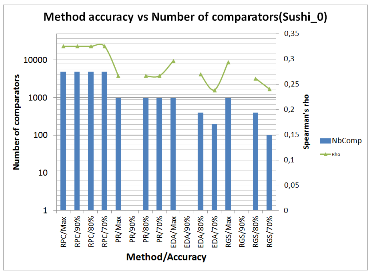


Figure B.86: Accuracy vs Number of classifiers (Sushi_0)

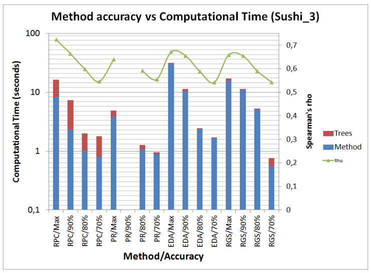


Figure B.87: Accuracy vs Computational time (Sushi_3)

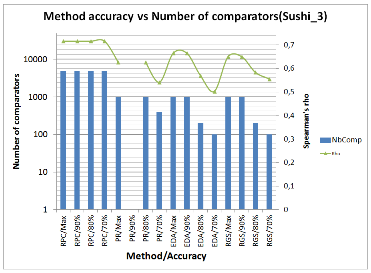


Figure B.88: Accuracy vs Number of classifiers (Sushi_3)

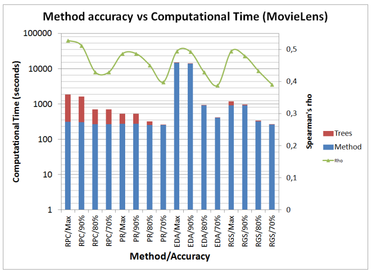


Figure B.89: Accuracy vs Computational time (MovieLens)

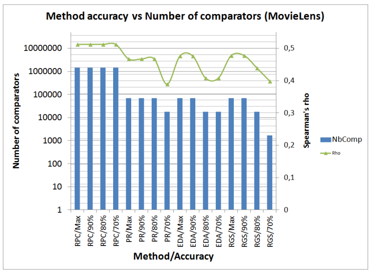


Figure B.90: Accuracy vs Number of classifiers (MovieLens)

B.4 Robustness of RGS

In this section, we provide the figures, discussed in Section 5.5.3, concerning the analysis of robustness of the RGS algorithm with respect to the tree parameters and to the parameter T. We also analyze the effect of #**LS**, which is also contained in this section although we cannot really speak about robustness in that case but rather about the learning effort w.r.t. parameter T. However, since varying #**LS** is very similar to varying the tree parameters, we chose to display the results of this experiment in the same section as the robustness experiments. We separate this section by database, and by the manner of dealing with sparsity (i.e. dropping empty comparisons or using a dummy model to represent them).

B.4.1 Omib

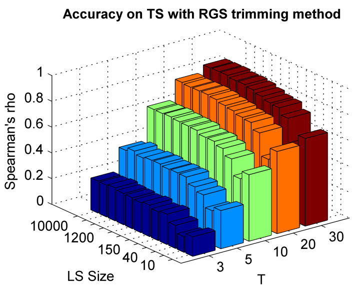


Figure B.91: Effect of #**LS** in classification

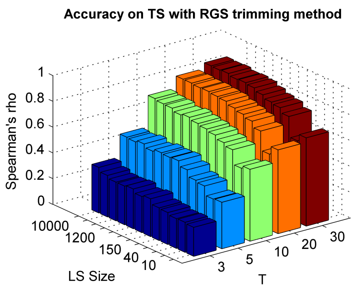


Figure B.92: Effect of #**LS** in regression

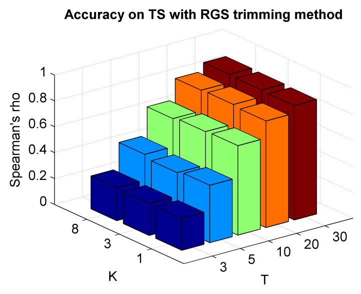


Figure B.93: Effect of parameter K in classification

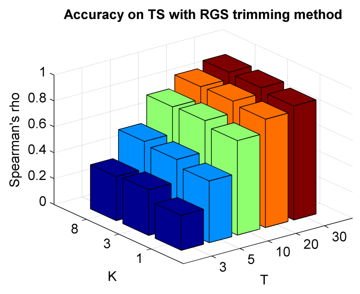


Figure B.94: Effect of parameter K in regression

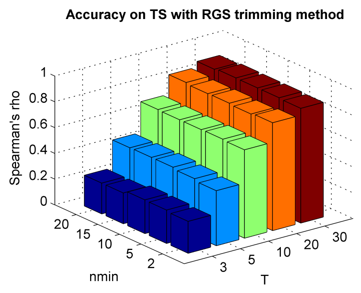


Figure B.95: Effect of parameter $n_{\min}$ in classification

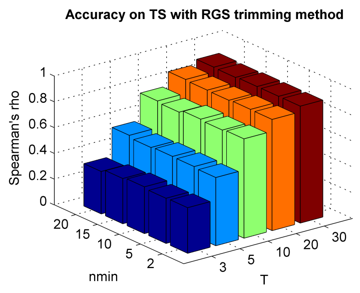


Figure B.96: Effect of parameter $n_{\min}$ in regression

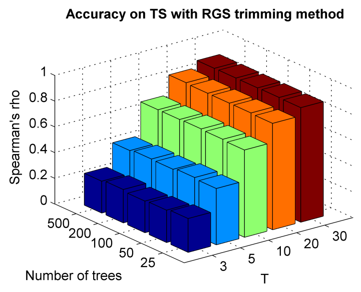


Figure B.97: Effect of parameter M in classification

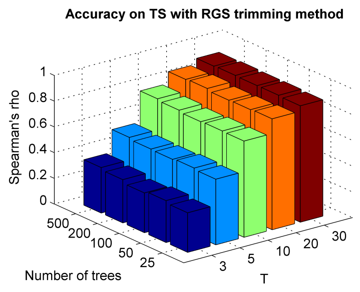


Figure B.98: Effect of parameter M in regression

B.4.2 Sushi_0, Dropping empty comparisons

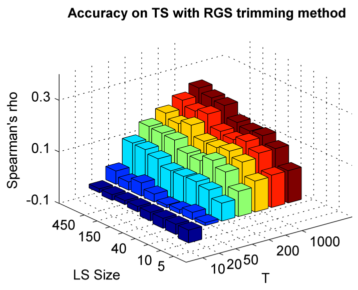


Figure B.99: Effect of #**LS** in classification

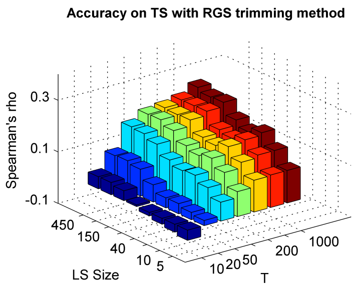


Figure B.100: Effect of #**LS** in regression

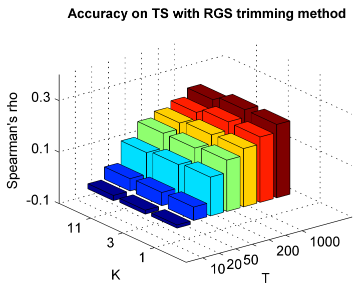


Figure B.101: Effect of parameter K in classification

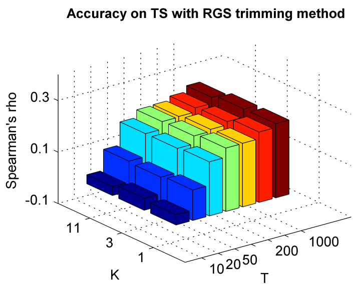


Figure B.102: Effect of parameter K in regression

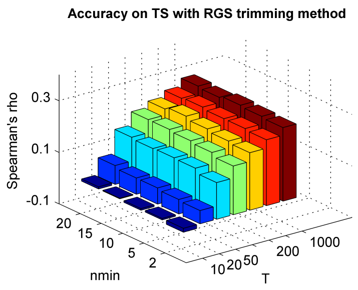


Figure B.103: Effect of parameter n_{min} in classification

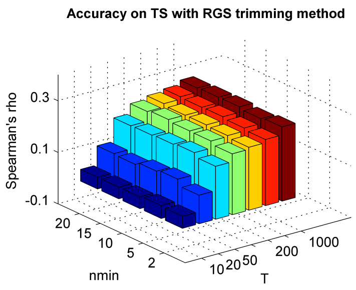


Figure B.104: Effect of parameter n_{min} in regression

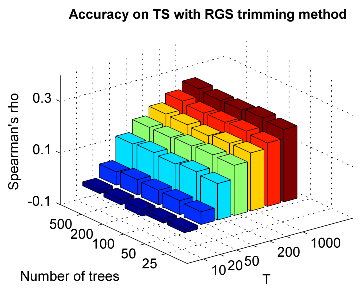


Figure B.105: Effect of parameter M in classification

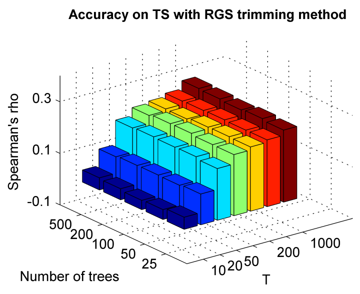


Figure B.106: Effect of parameter M in regression

B.4.3 Sushi_0, Using a dummy model

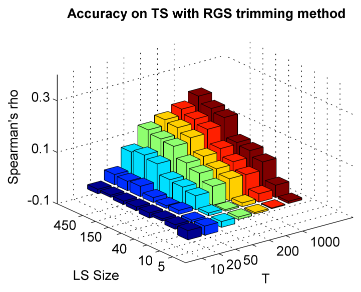


Figure B.107: Effect of #**LS** in classification

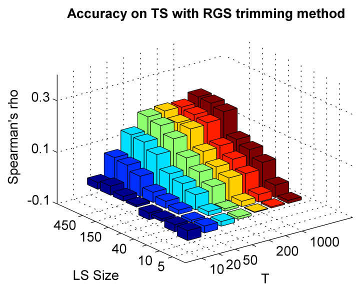


Figure B.108: Effect of #**LS** in regression

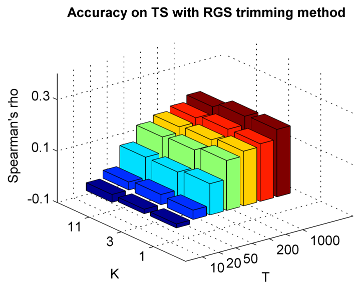


Figure B.109: Effect of parameter K in classification

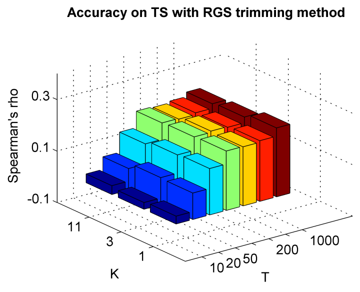


Figure B.110: Effect of parameter K in regression

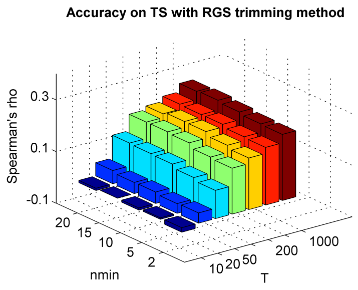


Figure B.111: Effect of parameter $n_{\min}$ in classification

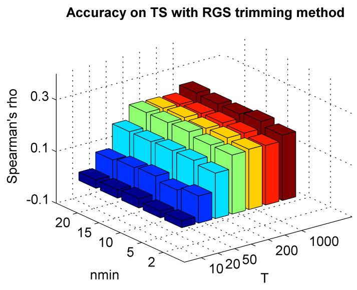


Figure B.112: Effect of parameter $n_{\min}$ in regression

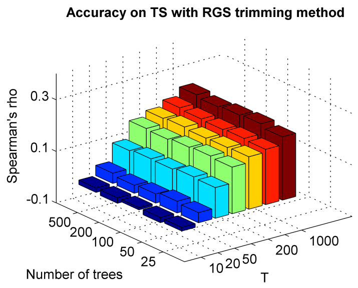


Figure B.113: Effect of parameter M in classification

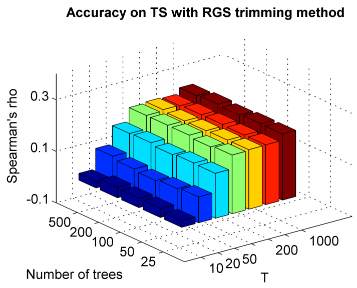


Figure B.114: Effect of parameter M in regression

B.4.4 Sushi_3, Dropping empty comparisons

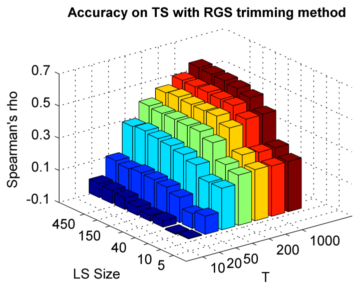


Figure B.115: Effect of #**LS** in classification

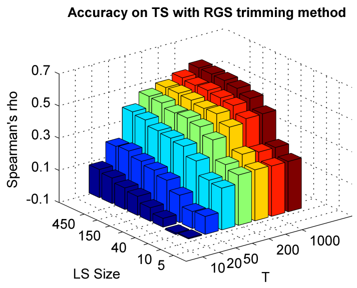


Figure B.116: Effect of #**LS** in regression

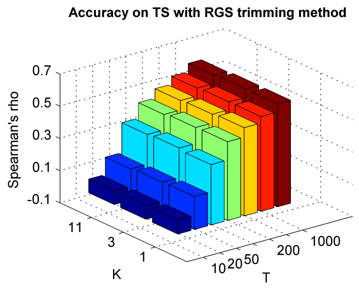


Figure B.117: Effect of parameter K in classification

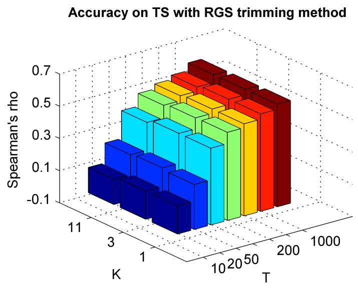


Figure B.118: Effect of parameter K in regression

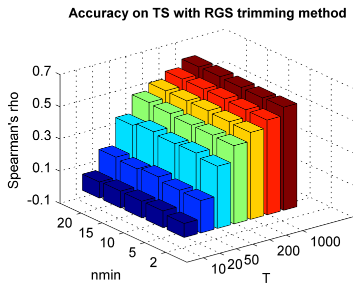


Figure B.119: Effect of parameter n_{min} in classification

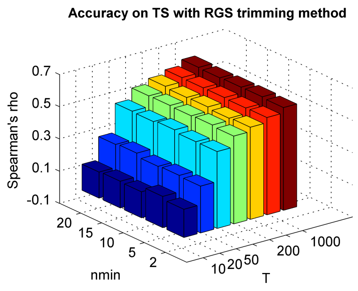


Figure B.120: Effect of parameter n_{min} in regression

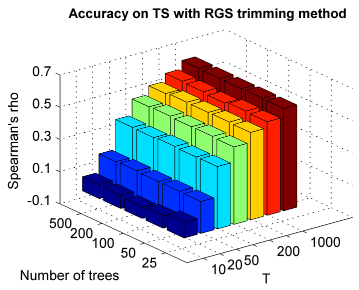


Figure B.121: Effect of parameter M in classification

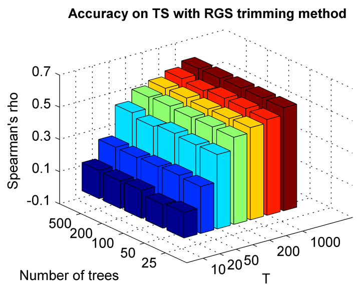


Figure B.122: Effect of parameter M in regression

B.4.5 Sushi_3, Using a dummy model

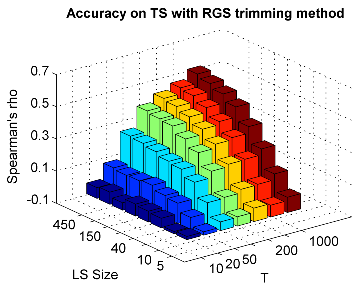


Figure B.123: Effect of #**LS** in classification

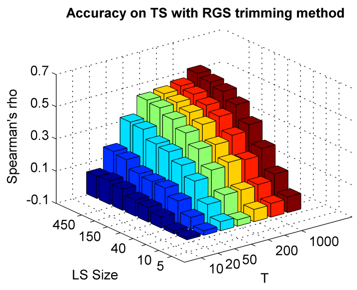


Figure B.124: Effect of #**LS** in regression

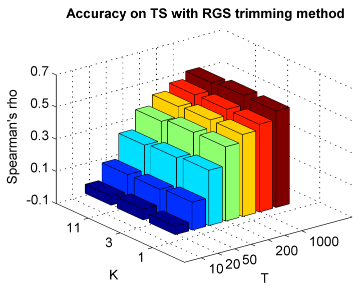


Figure B.125: Effect of parameter K in classification

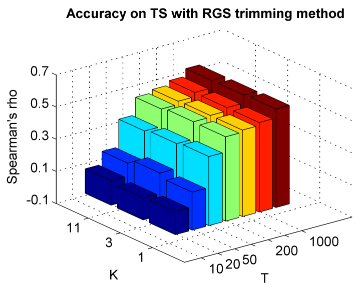


Figure B.126: Effect of parameter K in regression

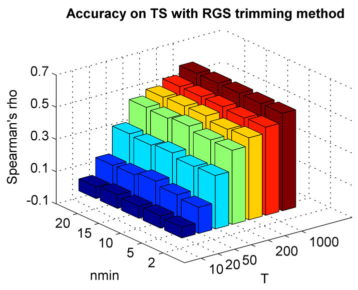


Figure B.127: Effect of parameter n_{min} in classification

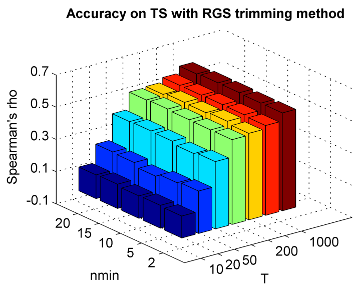


Figure B.128: Effect of parameter n_{min} in regression

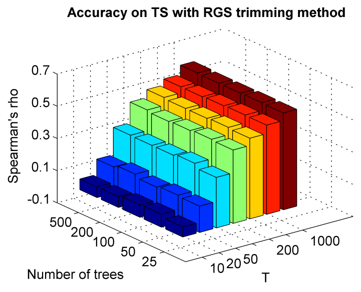


Figure B.129: Effect of parameter M in classification

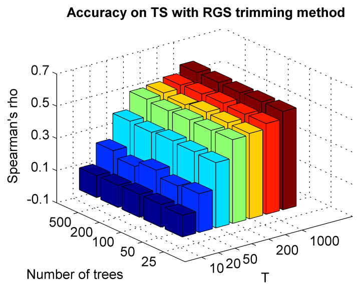


Figure B.130: Effect of parameter M in regression

B.4.6 MovieLens, Dropping empty comparisons

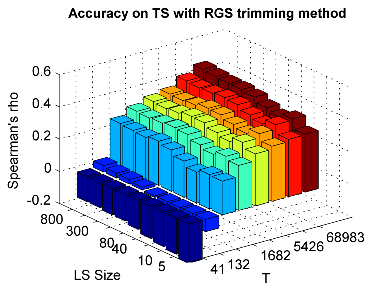


Figure B.131: Effect of #**LS** in classification

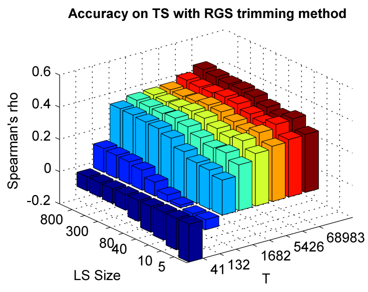


Figure B.132: Effect of #**LS** in regression

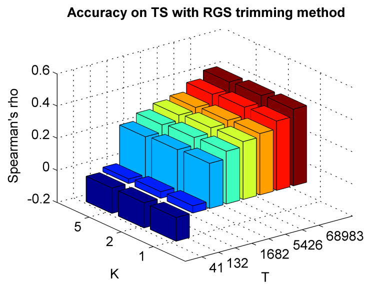


Figure B.133: Effect of parameter K in classification

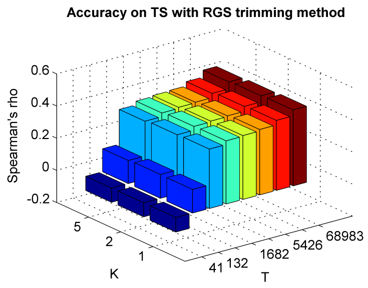


Figure B.134: Effect of parameter K in regression

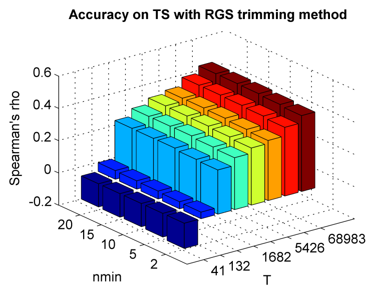


Figure B.135: Effect of parameter $n_{\min}$ in classification

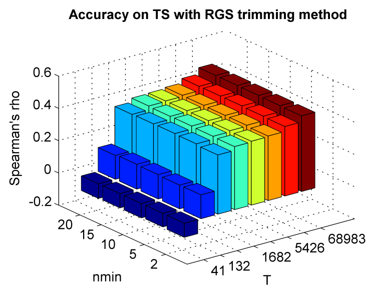


Figure B.136: Effect of parameter $n_{\min}$ in regression

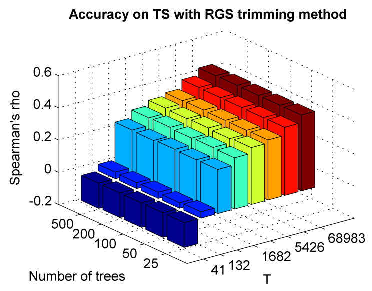


Figure B.137: Effect of parameter M in classification

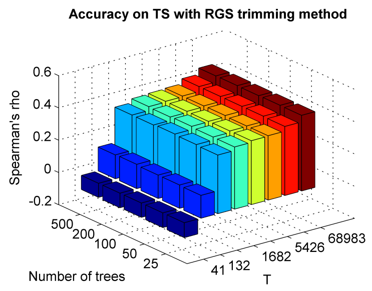


Figure B.138: Effect of parameter M in regression

B.4.7 MovieLens, Using a dummy model

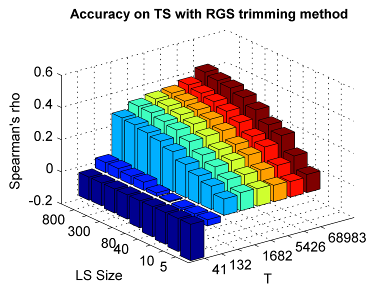


Figure B.139: Effect of #**LS** in classification

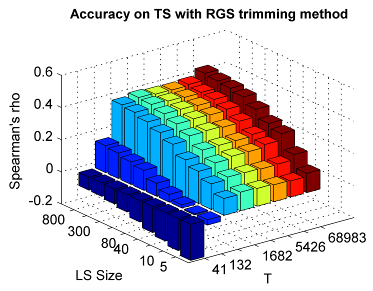


Figure B.140: Effect of #**LS** in regression

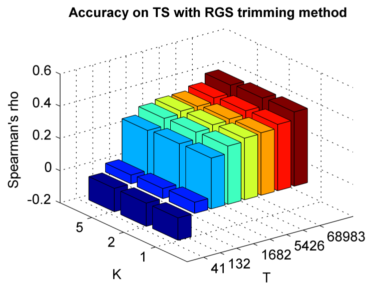


Figure B.141: Effect of parameter K in classification

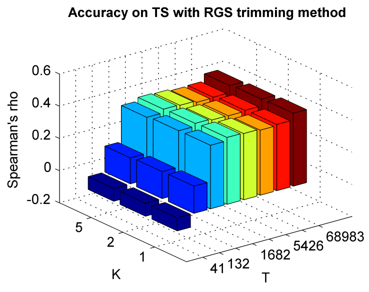


Figure B.142: Effect of parameter K in regression

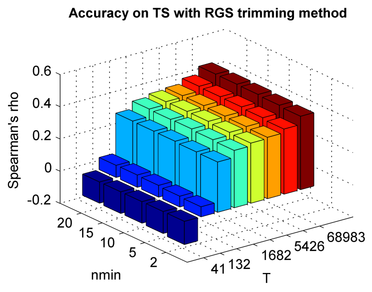


Figure B.143: Effect of parameter n_{min} in classification

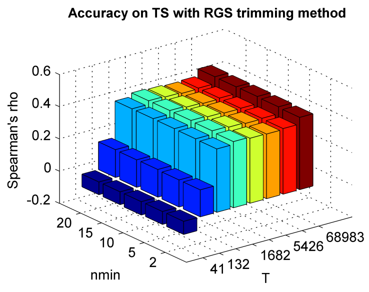


Figure B.144: Effect of parameter n_{min} in regression

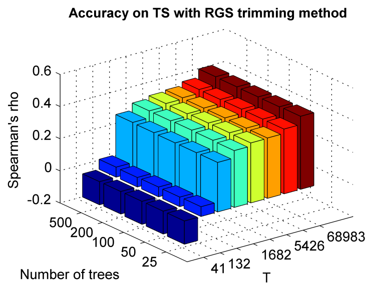


Figure B.145: Effect of parameter M in classification

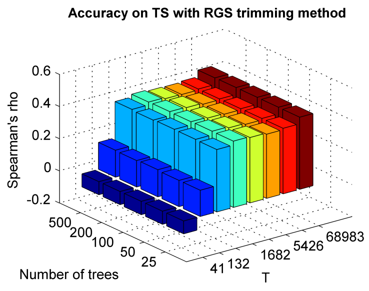


Figure B.146: Effect of parameter M in regression

Appendix C

Alternative approach

Contents

C.1 Introduction 239
C.2 LRANN loss function 240
C.3 Experimental setup 240
C.4 Results 241
C.5 Conclusion 242

Although our main task was to find a solution to the complexity burden of RPC by trimming the set of comparators, another completely different approach was considered and we will present this idea as a preliminary work.

C.1 Introduction

The application of ANNs to label ranking problems has already been studied in the literature. However, either the computation requires several models (one per class label [CDK+06] or per pair of labels [FISS03]) or uses an artifact, as proposed by Burges et al. in [BSR+05], where the authors perform two consecutive forward passes in order to compute the loss function. Indeed, when considering training examples of the form $\{x^i, y_k, y_l\}$, the network should output a greater score to y_k than to y_l when x^i is input (see 3.2.1.3 for more details about this procedure).

Crammer et al. proposed in [CDK+06] a methodology to assign a score to each label. However, they applied this technique to multi-label classification in such a way that the lowest score of relevant items should be higher than the highest score of irrelevant items.

Inspired by this previous research, we propose a new loss function for training a multiple output ANN, while coping with partially ranked datasets. The resulting method is called LRANN.

The subsequent sections are organized as follow : the proposed loss function will be given in Section C.2, the experimental setup will be described in Section C.3. Section C.4 will present the results and the conclusion for this part will be given in Section C.5.

C.2 LRANN loss function

The loss function that we propose is the following :

$$\forall i \in N, Loss[i] = \begin{cases} \left(1 - \frac{\tau(i)}{\#\tau - 1} - Output[i]\right)^2 & \text{if } i \in \tau \\ 0 & \text{Otherwise} \end{cases} \tag{C.1}$$

where τ is the (possibly partial) ranking associated to a given training example, N is the total number of class labels, $\#\tau$ is the number of class labels in the τ ranking, $Output[i]$ is the value of the i^{th} node in the output layer[1], and where $\tau(i)$ represents the position of label i in the τ ranking (from 0 to $\#\tau - 1$). Graphically, the representation of the expected prediction is shown in Figure C.1.

C.3 Experimental setup

We tested our LRANN method on the four previously depicted datasets. We vary the number of hidden layers from 1 to 3 and the number of nodes per layer in $\{5, 10, 20, 50, 100\}$. We used a sigmoid function ($f(x) = \frac{1}{1+e^{-x}}$) to model each neuron. The starting weights were randomly set in the interval $[-1, 1]$. Each layer was added a bias neuron b. We applied the back-propagation algorithm [Roj96] using the gradient descent to update the network weights. Since Wilson and Martinez discussed in [WM03] about the general inefficiency

[1]Each $Output[i]$ is constrained to stay between $[0, 1]$

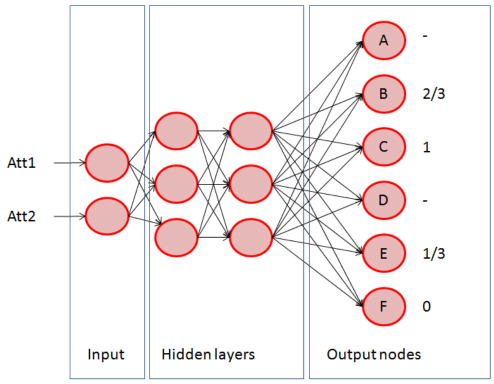


Figure C.1: Expected output value for a control ranking $\{C, B, E, F\}$. Nothing is expected for A and D, so the loss will be equal to 0.

of batch training in ANN, we updated the network weights after evaluating each object in the dataset. Each training consisted of 100 epochs. The learning rate α was set to 0.01.

C.4 Results

We compared the prediction of LRANN to predictions from the Ranking by Pairwise Comparison algorithm [HFCB08] using Extra-trees [GEW06] as base learner[2], as well as the best previously published value, when available[3]. For each dataset, we performed a 10-fold cross validation test, both with RPC and our method. For each training fold, we split the corresponding fold into two parts : 66.6% were used as training set and the remaining 33.3% were used as validation set to evaluate the error rate. Based on this split, we trained different architectures by varying, as parameters, the number of hidden layers and the number of nodes per layer as explained in Section C.3. Parameters yielding the lower error rate on the validation set determine the optimal architecture for this fold. A new model based on this architecture is trained on the combination of the previous training set and the validation set, and the error rate of this fold is estimated using the holdout. We then proceed with the next fold. However,

[2]We used standard parameter values. $K = \sqrt{\#A}$, $n_{\text{min}} = 2$, $M = 100$

[3]In this study, we only considered the values published in [KKA05]

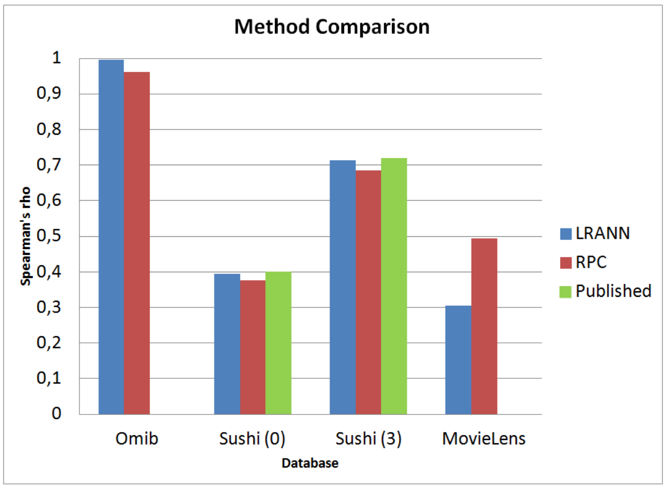


Figure C.2: Our preliminary results are promising

we did not perform such a parameter optimization on RPC, which could explain why RPC has lower scores. The comparison between two rankings was performed using the Spearman Rank Correlation coefficient [Spe04] and the performances are shown on Figure C.2, showing a slight advantage of the proposed approach with respect to the RPC method, on three out of 4 datasets, while on Movielens RPC remains significantly better.

C.5 Conclusion

We presented a new approach to solve label ranking problems using a single artificial neural network. Although our results are preliminary and they would require a deeper analysis, they are nevertheless promising. At this stage, the proposed algorithm is already competitive with state-of-the art methods on most of our tested datasets.

Bibliography

[ACN05] N. Ailon, M. Charikar, and A. Newman, *Aggregating inconsistent information: ranking and clustering*, Proceedings of the thirty-seventh annual ACM symposium on Theory of computing (New York, NY, USA), STOC '05, ACM, 2005, pp. 684–693.

[AIS93] R. Agrawal, T. Imieliński, and A. Swami, *Mining association rules between sets of items in large databases*, SIGMOD Rec. **22** (1993), no. 2, 207–216.

[ASS07] F. Aiolli, F. Sebastiani, and A. Sperduti, *Preference Learning for Category-Ranking based Interactive Text Categorization*, August 2007, pp. 2034–2039.

[Bay63] T. Bayes, *An essay towards solving a problem in the doctrine of chances*, Phil. Trans. of the Royal Soc. of London **53** (1763), 370–418.

[BFKM85] L. Brownston, R. Farrell, E. Kant, and N. Martin, *Programming expert systems in ops5: an introduction to rule-based programming*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1985.

[BFSO84] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*, 1 ed., Chapman and Hall/CRC, January 1984.

[BGH10] A. Birlutiu, P. Groot, and T. Heskes, *Multi-task preference learning with an application to hearing aid personalization*, Neurocomputing **73** (2010), 1177–1185.

[BH06] K. Brinker and E. Hüllermeier, *Case-based label ranking*, Proceedings of the 17th European Conference on Machine Learning (ECML-06) (Berlin, Germany) (Johannes Fürnkranz, Tobias Scheffer, and M. Spiliopoulou, eds.), Springer-Verlag, 2006, pp. 566–573.

[BH07] ______, *Case-Based Multilabel Ranking*, Proceedings of the 20th International Conference on Artificial Intelligence (IJCAI '07) (Hyderabad, India), 2007, pp. 702–707.

[BM98] A. Blum and T. Mitchell, *Combining labeled and unlabeled data with co-training*, Proceedings of the eleventh annual conference on Computational learning theory (New York, NY, USA), COLT' 98, ACM, 1998, pp. 92–100.

[BSR$^+$05] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, *Learning to rank using gradient descent*, Proceedings of the 22nd international conference on Machine learning (New York, NY, USA), ICML '05, ACM, 2005, pp. 89–96.

[BTT89] J. Bartholdi, C. A. Tovey, and M. A. Trick, *Voting schemes for which it can be difficult to tell who won the election*, Social Choice and Welfare **6** (1989), 157–165, 10.1007/BF00303169.

[CC00] Y. Cheng and G. M. Church, *Biclustering of expression data*, 2000.

[CDK$^+$06] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, *Online passive-aggressive algorithms*, J. Mach. Learn. Res. **7** (2006), 551–585.

[CH08] W. Cheng and E. Hüllermeier, *Instance-based label ranking using the mallows model.*, ECCBR Workshops (Martin Schaaf, ed.), 2008, pp. 143–157.

[CH09] ______, *A new instance-based label ranking approach using the mallows model*, ISNN (1) (Wen Yu, Haibo He, and Nian Zhang, eds.), Lecture Notes in Computer Science, vol. 5551, Springer, 2009, pp. 707–716.

[CHH09] W. Cheng, J. C. Huhn, and E. Hüllermeier, *Decision tree and instance-based learning for label ranking.*, ICML (Andrea Pohoreckyj Danyluk, Lon Bottou, and Michael L. Littman, eds.), ACM International Conference Proceeding Series, vol. 382, ACM, 2009, p. 21.

[CLR90] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, ch. 16: "Greedy algorithms", The MIT Press, Cambridge, MA, 1990.

[Cri09] N. Cristianini, *Are we there yet?*, European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, 2009.

[CV95] C. Cortes and V. Vapnik, *Support-vector networks*, Machine Learning, 1995, pp. 273–297.

[dGIL$^+$] M. de Gemmis, L. Iaquinta, P. Lops, C. Musto, F. Narducci, and G. Semeraro, *Preference Learning in Recommender Systems.*

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin, *Maximum likelihood from incomplete data via the em algorithm*, JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B **39** (1977), no. 1, 1–38.

[DWH12] K. Dembczynski, W. Waegeman, and E. Hüllermeier, *An analysis of chaining in multi-label classification*, ECAI, 2012, pp. 294–299.

[EGWL05] D. Ernst, P. Geurts, L. Wehenkel, and L. Littman, *Tree-based batch mode reinforcement learning*, Journal of Machine Learning Research **6** (2005), 503–556.

[EKJX96] M. Ester, H. Kriegel, S. Jrg, and X. Xu, *A density-based algorithm for discovering clusters in large spatial databases with noise*, AAAI Press, 1996, pp. 226–231.

[EW02] A. Elisseeff and J. Weston, *A kernel method for multi-labelled classification*, Advances in Neural Information Processing Systems 14 (NIPS-01) (T. G. Dietterich, S. Becker, and Z. Ghahramani, eds.), 2002, pp. 681–687.

[FH10] J. Fürnkranz and E. Hüllermeier, *Preference learning: An introduction*, Preference Learning (Johannes Fürnkranz and Eyke Hüllermeier, eds.), Springer-Verlag, 2010, pp. 1–17.

[FISS03] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, *An efficient boosting algorithm for combining preferences*, J. Mach. Learn. Res. **4** (2003), 933–969.

[FK99] A. M. Frieze and R. Kannan, *Quick approximation to matrices and applications.*, Combinatorica **19** (1999), no. 2, 175–220.

[FSST97] Y. Freund, H. S. Seung, E. Shamir, and N. Tishby, *Selective sampling using the query by committee algorithm*, Mach. Learn. **28** (1997), no. 2-3, 133–168.

[GEW06] P. Geurts, D. Ernst, and L. Wehenkel, *Extremely randomized trees*, Machine Learning **36** (2006), no. 1, 3–42.

[GRP73] University of Minnesota GroupLens Research Project, *Movielens data set*, http://www.grouplens.org/node/73.

[GV10] T. Gärtner and S. Vembu, *Label ranking algorithms: A survey*, Preference Learning (Eyke Hüllermeier Johannes Fürnkranz, ed.), Springer–Verlag, 2010.

[GWKS08] M. Gallagher, I. Wood, J. Keith, and G. Sofronov, *Bayesian inference in estimation of distribution algorithms*, 2008.

[HF04] E. Hüllermeier and J. Fürnkranz, *Ranking by pairwise comparison : A note on risk minimization*, the IEEE International Conference on Fuzzy Systems, 2004.

[HFCB08] E. Hüllermeier, J. Fürnkranz, W. Cheng, and K. Brinker, *Label ranking by learning pairwise preference*, Artificial Intelligence **172** (2008), 1897–1916.

[HGBSO98] R. Herbrich, T. Graepel, P. Bollmann-Sdorra, and K. Obermayer, *Learning preference relations for information retrieval*, Training, vol. 0, 1998, pp. 80–84.

[HpRZ02] S. Har-peled, D. Roth, and D. Zimak, *Constraint classification: A new approach to multiclass classification and ranking*, In Advances in Neural Information Processing Systems 15, 2002, pp. 365–379.

[HTF03] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning*, corrected ed., ch. 14.3.12 : Hierarchical clustering, Springer, July 2003.

[Hub06] M. Huber, *Fast perfect sampling from linear extensions.*, Discrete Mathematics (2006), 420–428.

[Kam03] T. Kamishima, *Clustering orders*, The 6th International Conference on Discovery Science, 2003, pp. 194–207.

[Ken38] M. Kendall, *A new measure of rank correlation*, Biometrika **30** (1938), no. 1-2, 81–89.

[KKA05] T. Kamishima, H. Kazawa, and S. Akaho, *Supervised ordering — an empirical survey*, The 5th IEEE International Conference on Data Mining, 2005, pp. 673–676.

[KMS07] C. Kenyon-Mathieu and W. Schudy, *How to rank with few errors*, STOC (D. S. Johnson and U. Feige, eds.), ACM, 2007, pp. 95–103.

[Knu98] D. E. Knuth, *Art of Computer Programming, Volume 3: Sorting and Searching (2nd Edition)*, 2 ed., ch. 5.2.2 : Sorting by Exchanging, Addison-Wesley Professional, May 1998.

[LHT07] L. Ling and L. Hsuan-Tien, *Ordinal regression by extended binary classification*, Advances in Neural Information Processing Systems 19 (B. Schölkopf, J. C. Platt, and T. Hofmann, eds.), 2007, pp. 865–872.

[LL01] P. Larranage and J. A. Lozano, *Estimation of distribution algorithms : A new tool for evolutionary computation*, Kluwer Academic Publishers, Dordrecht, Netherland, 2001.

[LMPF10] E. Loza Mencía, S. Park, and J. Fürnkranz, *Efficient voting prediction for pairwise multilabel classification*, Neurocomput. **73** (2010), no. 7-9, 1164–1176.

[LSY03] G. Linden, B. Smith, and J. York, *Amazon.com recommendations: item-to-item collaborative filtering*, Internet Computing, IEEE **7** (2003), no. 1, 76–80.

[Mac67] J. B. MacQueen, *Some methods for classification and analysis of multivariate observations*, Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability (L. M. Le Cam and J. Neyman, eds.), vol. 1, University of California Press, 1967, pp. 281–297.

[Mar71] A. Markov, *Extension of the Limit Theorems of Probability Theory to a Sum of Variables Connected in a Chain*, Dynamic Probabilistic Systems (Volume I: Markov Models) (R. Howard, ed.), John Wiley & Sons, Inc., New York City, 1971, pp. 552–577.

[McC79] P. McCorduck, *Machines who think*, Freeman, 1979.

[Men87] E. Mendelson, *Introduction to mathematical logic; (3rd ed.)*, Wadsworth and Brooks/Cole Advanced Books & Software, Monterey, CA, USA, 1987.

[Min74] M. Minsky, *A framework for representing knowledge*, Tech. report, Cambridge, MA, USA, 1974.

[Mit97] T. M. Mitchell, *Machine learning*, McGraw-Hill International Editions, 1997.

[Pea84] J. Pearl, *Heuristics: intelligent search strategies for computer problem solving*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[Pea85] ______, *Bayesian networks: A model of self-activated memory for evidential reasoning*, Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine, August 1985, pp. 329–334.

[PF07] S. Park and J. Fürnkranz, *Efficient pairwise classification*, ECML 2007, Springer, 2007, pp. 658–665.

[Pla98] J. C. Platt, *Sequential minimal optimization: A fast algorithm for training support vector machines*, 1998.

[Pla99] ______, *Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods*, ADVANCES IN LARGE MARGIN CLASSIFIERS, MIT Press, 1999, pp. 61–74.

[Qui92] J. R. Quinlan, *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning)*, 1 ed., Morgan Kaufmann, October 1992.

[Raf10] R. Rafter, *Evaluation and conversation in collaborative filtering*, Ph.D. Thesis, University College Dublin, School of Computer Science & Informatics, 2010.

[Roj96] R. Rojas, *Neural Networks: A Systematic Introduction*, 1 ed., ch. 7 : The backpropagation algorithm, Springer, July 1996.

[Ros56] M. Rosenblatt, *Remarks on Some Nonparametric Estimates of a Density Function*, The Annals of Mathematical Statistics **27** (1956), no. 3, 832–837.

[RPHF09] J. Read, B. Pfahringer, G. Holmes, and E. Frank, *Classifier chains for multi-label classification*, Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part II (Berlin, Heidelberg), ECML PKDD '09, Springer-Verlag, 2009, pp. 254–269.

[Rul99] C. M. Ruland, *Decision support for patient preference-based care planning: effects on nursing care and patient outcomes.*, Journal of the American Medical Informatics Association **6** (1999), no. 4, 304–312.

[SBZH07] C. Strobl, A. Boulesteix, A. Zeileis, and T. Hothorn, *Bias in random forest variable importance measures: Illustrations, sources and a solution*, BMC Bioinformatics **8** (2007).

[SE87] S. C. Shapiro and D. Eckroth, *Encyclopedia of artificial intelligence / stuart c. shapiro, editor in chief, david eckroth, managing editor*, Wiley, New York :, 1987 (English).

[Sha48] C. E. Shannon, *A mathematical theory of communication*, Bell System Technical Journal **27** (1948), 379–426 and 623–656.

[Sow91] J. F. Sowa, *Principles of semantic networks*, Morgan Kaufmann, 1991.

[Spe04] C. Spearman, *The proof and measurement of association between two things*, The American Journal of Psychology **15** (1904), no. 1, 72–101.

[Sut88] R. S. Sutton, *Learning to predict by the methods of temporal differences*, Machine Learning, 1988, pp. 9–44.

[Tok10] M. Tokic, *Adaptive ε-greedy exploration in reinforcement learning based on value differences*, Proceedings of the 33rd annual German conference on Advances in artificial intelligence (Berlin, Heidelberg), KI'10, Springer-Verlag, 2010, pp. 203–210.

[vR79] C. J. van Rijsbergen, *Information Retrieval*, 2 ed., Butterworths, London, 1979.

[VZPW07] A. Van Zuylen and D. P. Williamson, *Deterministic algorithms for rank aggregation and other ranking and clustering problems*, Proceedings of the Fifth International Workshop on Approximation and Online Algorithms, 2007.

[Weh98] L. Wehenkel, *Automatic learning techniques in power systems*, Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[WGIA] L. Wehenkel and P. Geurts, http://www.montefiore.ulg.ac.be/ lwh/AIA/, Applied inductive learning course, lesson 3 : Decision and regression trees, general principles.

[WK10] W. Wang and I. King, *Label ranking with semi-supervised learning*, Australian Journal of Intelligent Information Processing Systems **12** (2010), no. 1.

[WM03] D. R. Wilson and T. R. Martinez, *The general inefficiency of batch training for gradient descent learning*, Neural Networks **16** (2003), no. 10, 1429–1451.

[ZLTC07] Gang Zhang, Yue Liu, Songbo Tan, and Xueqi Cheng, *A novel method for hierarchical clustering of search results*, Web Intelligence and Intelligent Agent Technology, International Conference on **0** (2007), 181–184.