Object Feature Coding: A Decomposition Framework Unifying Object and Label Ranking

Diplomarbeit

Moritz Wissenbach

wissenba@stud.tu-darmstadt.de

4. Oktober 2010

Knowledge Engineering Group
TU Darmstadt

Prof. Dr. Johannes Fürnkranz

Ehrenwörtliche Erklärung

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

Darmstadt, den 4. Oktober 2010

Moritz Wissenbach

Abstract

Object ranking refers to the problem of learning to order a set of objects, each object described by an attribute tuple. *Label ranking* refers to the problem of learning to order a set of objects, each object only described by a nominal label; the order depends on a context, which is represented by an attribute tuple. The present work seeks to unify both approaches through the means of problem decomposition. To this end, a framework is proposed which allows to split the original problem into multiple sub-problems, which are then solved and their solutions aggregated into a solution to the original problem. The decomposition of the problem is a function of the object features; many different methods of decomposition are possible within the framework, and several are presented. The introduced methods are examined and compared to established or more obvious approaches. Along the way, several properties of the different types of ranking tasks are discussed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Automatic ranking has been an important field of study, and its importance only grows in a world of ever-increasing choices. Generally, ranking means the inference of an order on a given set of objects. These objects could be the results of a search engine query, product recommendations at an online shopping site, the possible actions of an artificial intelligence agent or messages to be ordered according to their priority. According to Kendall & Gibbons (1990, p.1),

> When objects are arranged in order according to some quality which they all possess to a varying degree, they are said to be *ranked* with respect to that quality. The arrangement as a whole is called a *ranking*. The rank of each object indicates its respective position in the ranking.

In order to clarify the use of the term *ranking* in this work, we will compare it to the above definition. In accordance with that definition is the notion of ranking as the assignment of a position to each object. Each object has a defined rank, and can thus be considered ranked higher or lower than any other given object. What cannot be done is the calculation of a meaningful distance between two objects. Speaking in terms of scales of measure, the objects lie on an ordinal scale, but not on an interval scale.

Ranking as defined by Kendall & Gibbons can be used as a method of observation. For example, in a survey, participants are not asked to *rate*, i.e. give a certain amount of points to each individual item, but to rank those items.

In contrast to the above definition, in this work, objects are not ranked with respect to a single quality, since the ranking is the result of a function of the multi-dimensional *context* in which the ranking takes place. The ranking function is induced by a machine learning algorithm and could be a black box, that is, difficult to comprehend by a human examiner.

In the general setting of machine learning, a predictive model is built from a set of *training data*. That model is then used to make a *prediction* on an instance of previously unseen data. In this work, *ranking* refers to the structure of the prediction, not to the structure of the training data. Instead, the training data is given in the form of *preferences*.

Preferences share with a ranking the characteristic that objects are brought into a relation with each other, without specifying an exact distance between them. They differ from a ranking in that not all objects need to be related to each other. For example, a participant of a survey asked to express her preferences regarding items A, B and C could say that she prefers A to B and A to C, but not make a statement about the relationship between B and C.

Observing preferences instead of ratings or rankings has a couple of advantages. Since it is not required to know about the relationship between each pair of objects, incomplete data can be used for training. Such data may be more readily available. Further, implicit data can be collected. Imagine a user being presented with a page listing ten results for a search engine query. If three of the results are clicked, it can be inferred that those are more relevant to the query than all others, and preference between those three could be determined by the order in which they are clicked. In this way, preferences can be collected even though there is no explicit rating or distance between the items given. Finally, according to Kamishima (2003), the method of collecting preferences inhibits a type of error that can be introduced because not all users share a universal scale when rating items. The same is true for ranking, and in a similar manner, (Kendall & Gibbons, 1990, p.2) write about ranking:

> However, what the ranking loses in accuracy it gains in generality, for if we stretch the scale of measurement (and even if we stretch it differently in different regions) the ranking remains unaltered; in mathematical language it is *invariant* under stretching of scale.

In some literature, for instance in Kamishima (2003), the term *order* is used instead of *ranking.* A (strict) total order is a binary relation that is transitive and trichotomous. Each ranking on a set of objects has a corresponding order and the terms are used synonymously in the following; *ranking* is predominantly used when the focus lies on the numerical rank of objects, and *order* is used when the focus lies on the pairwise preferences that make up the elements of a binary relation.

Object ranking refers to the task of ranking objects that are represented as a tuple of feature values. *Label ranking* refers to the task of ranking objects depending on a context, where the context is represented as a tuple of feature values. The objective of this work is to unify those scenarios and see if the unification brings about any benefits. The unified scenario would be a generalization of the two original scenarios. Each ranking task for the unified scenario could be transformed into a task for one of the original scenarios by stripping of certain information. In this way, a comparison between algorithms that implement either of the scenarios can be made. In chapter 2, each scenario is defined in a formal way.

Another aim is to devise a method for the decomposition of the unified ranking task. *Decomposition* in the data mining context means that a learning task is split into multiple sub-tasks, each of which is solved, and the solutions are then combined to form a solution to the original task. Decomposition in this sense is discussed in chapter 4.

Chapter 3 describes various methods that work on the defined scenarios. These methods have either been studied before or are obvious approaches to the problem, and they will provide a base for a comparison with the methods introduced in chapter 5. That chapter proposes a general decomposition framework through which any unified ranking task can be split into various sub-tasks. The decomposition is dependent on the features of the objects; as a matter of fact, the information of object features is exclusively used for the specification of the decomposition and does not contribute in any other way to the training information. For this reason, the proposed framework is called *object feature coding*. Within the framework, there are many possible methods of decomposition, and two are subsequently introduced.

The actual implementation of the evaluation environment, ranking methods and decomposition framework is briefly described in Appendix A.

Chapter 6 discusses the evaluation of the different methods. On the one hand, properties like algorithmic complexity are examined on a theoretical level. On the other hand, the premises for experimental evaluation are discussed, such as data sets to be used for testing and measures of prediction performance.

In chapter 7, the results of the conducted experiments are presented and interpreted; chapter 8 summarizes the work and draws a conclusion, and section 8.1 lists questions that have yet to be answered.

Chapter 2

Ranking Scenarios

This chapter describes and formalizes the scenarios or learning tasks to be treated in the course of this work. The definitions for Label Ranking and Object Ranking are taken from Fürnkranz & Hüllermeier (2010b); newly introduced notation is kept in accordance. For our purposes, it is being assumed that contexts and objects are represented as feature vectors, although that constraint is not made by Fürnkranz & Hüllermeier.

2.1 Label Ranking

Label ranking denotes a setting in which a finite number of labels is given, and the task is to order these labels depending on a context. The context is given by a tuple of attribute values.

Assuming m context attributes, let the ith attribute be in $\mathcal{X}_i$, then the context space $\mathcal{X}$ is denoted by

$$\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_m.$$

Let the finite set of labels be

$$\mathcal{Y} = \{y_1, y_2, \ldots, y_p\}.$$

The training data $\mathcal{T}$ consists of a set of instances $\{\boldsymbol{x}_\ell | \ell = 1, 2, \ldots, t\} \subseteq \mathcal{X}$, each such instance with an associated binary preference relation or set of binary preferences $\{y_i \succ_{\boldsymbol{x}_\ell} y_j, \ldots\}$.

From this data, a function

$$f : \mathcal{X} \to S_{\mathcal{Y}}$$

is learned which maps any $\boldsymbol{x} \in \mathcal{X}$ to a ranking or total order of $\mathcal{Y}$. The set of all total orders or permutations is denoted by $S_{\mathcal{Y}}$ following the common notation for the symmetric group S_A on set A.

In the following, $\boldsymbol{x}$ will be called the *context*, as it represents the general circumstances on which the order of the labels depends on. Note that $\mathcal{X}$ can possibly be infinite or will usually at least be very large due to the combinations, while $\mathcal{Y}$ is finite and will usually be sufficiently small. As $\mathcal{Y}$ is not structured in any way, every $y \in \mathcal{Y}$ must occur in the training data, or no prediction can be made about its rank.

The label ranking scenario finds its application wherever there is a large set of contexts to which features can be attributed, and a finite set of labels. One such application is an online shopping recommendation system, where a context is provided by the customer's attributes, and training data is provided by the choices that the customer made while browsing the site.

2.2 Object Ranking

In the *object ranking* scenario, a set of objects is given. Unlike the labels from the previous section, these objects are not merely nominal items taken from a finite set, but are described by attributes.

Assuming n object attributes, let the ith attribute be in $\mathcal{Z}_i$, then the object space or set of objects $\mathcal{Z}$ is denoted by

$$\mathcal{Z} = \mathcal{Z}_1 \times \mathcal{Z}_2 \times \ldots \times \mathcal{Z}_n$$

The training data $\mathcal{T}$ consists of a binary relation on $\mathcal{Z}$, or in other terms a set of binary preferences.

From this data, a function

$$f : Z \to S_Z \quad (Z \subseteq \mathcal{Z})$$

is learned, which orders any subset of objects given at ranking time. Note that, in contrast to label ranking, new objects which have not been part of the training data, can be ranked. The set of possible objects $\mathcal{Z}$ can be infinite; even with discrete-valued attributes, the set will usually be large due to the combinations of the n attributes.

As an example application, consider the task of ranking search engine results. By choosing some of the results to a query, but not others, a user implicitly expresses his or her preferences. These preferences can be used as training information to an object ranking algorithm, which learns how to rank the results of future queries.

2.3 Unified Object and Label Ranking

In contrast to label ranking, object ranking accounts for the features of the items to rank. On the other hand, it ranks the objects without regard for differences in context. A unifying approach would generalize from the two scenarios, so that each label ranking and object ranking problem would be a special case and processable by an implementation of the unified ranking. Of special interest is the question whether such an approach would achieve better results; specifically, whether label ranking, with the labels substituted by objects with features, could see an improvement (or decline) in prediction performance.

Let

$$\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_m$$

denote the context space and

the set of objects. Training data $\mathcal{T}$ consists of a set of contexts $\{\boldsymbol{x}_\ell | \ell = 1, 2, \ldots, t\} \subseteq \mathcal{X}$, each context $\boldsymbol{x}_\ell$ associated with a binary preference relation $\succ_{\boldsymbol{x}_\ell} = \{\boldsymbol{z}_i \succ_{\boldsymbol{x}_\ell} \boldsymbol{z}_j, \ldots\}$ on the set of objects.

From this data, a function

$$f : \mathcal{X} \times Z \to S_Z \quad (Z \subseteq \mathcal{Z})$$

is to be learned which takes as input any context $\boldsymbol{x} \in \mathcal{X}$ and subset Z of the objects and returns a total order of Z.

As the main goal of this work is a comparison with label ranking, the focus is on a special case of the unified scenario. In that special case, the constraint applies that every object $z \in Z$ that is to be ranked must be "known", that is, must have appeared in the training data $\mathcal{T}$.

For brevity, in the following the described unified object and label ranking scenario will also be called the *unified scenario*.

2.4 Approaches

In chapters 3 and 5, several ranking methods will be introduced. In order to provide a base for the discussion of these methods, two different elementary ways of approaching the problem will be pointed out here.

According to Fürnkranz & Hüllermeier (2010b), four general approaches can be identified when examining known techniques for object or label ranking. The two which are most relevant in the context of this work will be introduced in the following.

2.4.1 Utility Function

In this approach, a *utility function* is learned, which assigns an abstract degree of utility to each object or label. For example, in the label ranking case, for each label $y_i, 1 \leq i \leq p$, a utility function

$$f_i : \mathcal{X} \to \mathbb{R}$$

assigns a degree of utility dependent on the context. A variation of this learning of multiple models would be to learn a single function

$$f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$$

which maps any combination of a context and a label to a degree of utility.

After the utility of each label has been established, the labels can be ranked accordingly.

Generally speaking, the specific form of the input data used for learning imposes a certain set of constraints on the utility function. Therefore, in many cases, several utility functions are valid within those constraints and can possibly used for the learning task. The choice of one of these functions introduces an additional parameter that influences the outcome.

In practice, the described approach reduces the problem to one (in the single model variant) or many (in the multi-model variant) regression problems. Alternatively, if the utility scale is not numerical, but ordinal, the resulting problem is not one of regression, but of ordinal classification.

2.4.2 Preference Relations

In *preference relation* learning, the model to be learned operates on the notion of a binary preference relation. This means that for each pair of objects or labels, e.g. (y_i, y_j), a predicate $Q(y_i, y_j)$ is learned, which decides whether y_i is to be ranked higher than y_j or vice versa. It is easy to see that this approach adopts more naturally to training data given in the form of binary preferences.

As it has been hinted on by chapter 1 and defined in the preceding sections of this chapter, the present work is chiefly concerned with learning from preference data. Therefore, the preference relations approach is the one that will be most relevant to the further discussion.

Each ranking can be viewed as a binary preference relation which satisfies the conditions of transitivity and trichotomy, dubbed a *strict total order.* A measure between the similarity of arbitrary binary relations, including strict total orders, is developed in section 6.2.

All of the methods introduced throughout chapters 3 and 5 utilize the preference relation learning approach.

Chapter 3

Ranking Methods

In this chapter, several methods are introduced that implement the unified object and label ranking scenario. In one way or another, they reduce the ranking task to one of the standard machine learning problems of classification or regression. Classification and regression problems are common, extensively described in literature, well-understood, and there is a plethora of algorithms and implementations.

3.1 Baseline Ranking

The baseline ranking method transforms the ranking task into a single binary classification task. Each instance of that binary classification set represents the comparison of two objects in a specific context. A classifier $\mathcal{M}$ learns whether one object is to be preferred over another object in a given context. At ranking time, all members of the set of objects to be ranked Z are compared pairwise by $\mathcal{M}$, and the number of decisions in favor of any one object is the basis for the ranking.

The method is labeled Baseline ranking, since it seems to be a very obvious implementation of the task.

3.1.1 Training

Each instance in the original training data $\mathcal{T}$ comprises a context $\boldsymbol{x}_\ell$ and an associated preference relation $\succ_{\boldsymbol{x}_\ell}$. For each preference $p \in \succ_{\boldsymbol{x}_\ell}$, where $p = (\boldsymbol{z}_{p_1}, \boldsymbol{z}_{p_2})$, let $t_{p,1} = (\boldsymbol{x}_\ell, \boldsymbol{z}_{p_1}, \boldsymbol{z}_{p_2}, >)$ and $t_{p,2} = (\boldsymbol{x}_\ell, \boldsymbol{z}_{p_2}, \boldsymbol{z}_{p_1}, <)$ be training instances for $\mathcal{M}$. $\boldsymbol{z}_{p_2}, \boldsymbol{z}_{p_1}$ represent the object feature vectors, so if there are n object features and m context features, then $\mathcal{M}$ learns on $2n + m$ features. The class is $\{<, >\}$, and the concept to learn is whether the object represented by the first n features is to be ranked higher (">") or lower ("<") than the object represented by the second n features.

The following case illustrates the use of the method. Given the set of objects $\{\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3\}$ in four-dimensional feature space

$$\begin{array}{ccccccccc} & & & \mathcal{Z}_1 & \mathcal{Z}_2 & \mathcal{Z}_3 & \mathcal{Z}_4 & \\ \boldsymbol{z}_1 & = & (& 0 & 1 & 0 & 1 &) \\ \boldsymbol{z}_2 & = & (& 1 & 0 & 1 & 0 &) \\ \boldsymbol{z}_3 & = & (& 0 & 0 & 1 & 0 &) \end{array}$$

the set of contexts $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\}$ in two-dimensional feature space $\mathcal{X}$

$$\begin{array}{cccccc} & & & \mathcal{X}_1 & \mathcal{X}_2 & \\ \boldsymbol{x}_1 & = & (& 2 & 0 &) \\ \boldsymbol{x}_2 & = & (& 0 & 2 &) \\ \boldsymbol{x}_3 & = & (& \text{-2} & 0 &) \end{array}$$

and the training data $\mathcal{T} = \{t_1, t_2, t_3\}$

$$\begin{aligned} t_1 = (\boldsymbol{x}_1,\ \{&\boldsymbol{z}_1 \succ \boldsymbol{z}_2, \\ &\boldsymbol{z}_2 \succ \boldsymbol{z}_3\}) \\[2ex] t_2 = (\boldsymbol{x}_2,\ \{&\boldsymbol{z}_2 \succ \boldsymbol{z}_1, \\ &\boldsymbol{z}_3 \succ \boldsymbol{z}_1\}) \\[2ex] t_3 = (\boldsymbol{x}_3,\ \{&\boldsymbol{z}_2 \succ \boldsymbol{z}_3, \\ &\boldsymbol{z}_3 \succ \boldsymbol{z}_1\}) \end{aligned}$$

indicating that in context $\boldsymbol{x}_1$, object $\boldsymbol{z}_1$ is preferred over object $\boldsymbol{z}_2$ and object $\boldsymbol{z}_2$ is preferred over object $\boldsymbol{z}_3$, etc.

For baseline ranking, the training data $\mathcal{T}$ is transformed into training data $\mathcal{T}_{\boldsymbol{M}}$ for a binary classification problem. For example, the first preference $\boldsymbol{z}_1 \succ \boldsymbol{z}_2$ for context $\boldsymbol{x}_1$ yields

$$\begin{array}{ccccccccccccc} & \multicolumn{2}{c}{\overbrace{\hphantom{x_1\ x_2}}^{\boldsymbol{x}_1}} & \multicolumn{4}{c}{\overbrace{\hphantom{z_1\ z_2\ z_3\ z_4}}^{\boldsymbol{z}_1}} & \multicolumn{4}{c}{\overbrace{\hphantom{z_1\ z_2\ z_3\ z_4}}^{\boldsymbol{z}_2}} & & \\ & x_1 & x_2 & z_1 & z_2 & z_3 & z_4 & z_1 & z_2 & z_3 & z_4 & c & \\ (& 2 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & > &) \end{array}$$

and, by inversion of the two objects,

$$\begin{array}{ccccccccccccc} & \multicolumn{2}{c}{\overbrace{\hphantom{x_1\ x_2}}^{\boldsymbol{x}_1}} & \multicolumn{4}{c}{\overbrace{\hphantom{z_1\ z_2\ z_3\ z_4}}^{\boldsymbol{z}_2}} & \multicolumn{4}{c}{\overbrace{\hphantom{z_1\ z_2\ z_3\ z_4}}^{\boldsymbol{z}_1}} & & \\ & x_1 & x_2 & z_1 & z_2 & z_3 & z_4 & z_1 & z_2 & z_3 & z_4 & c & \\ (& 2 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & < &) \end{array}$$

The rest of the training data is generated analogously, and the classifier $\mathcal{M}$ is trained on it.

3.1.2 Prediction

To predict a ranking for a new context and a set of objects Z, all of the objects are mutually compared, resulting in $\frac{1}{2}|Z|(|Z|-1)$ comparisons. Each comparison involves a transformation into an instance of the learned problem and a subsequent classification through $\mathcal{M}$.

For example, let

$$\boldsymbol{x}_4 = \begin{pmatrix} 0 & -1 \end{pmatrix}$$

be a new context and

$$\begin{aligned} \boldsymbol{z}_4 &= \begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix} \\ \boldsymbol{z}_5 &= \begin{pmatrix} 0 & 1 & 1 & 0 \end{pmatrix} \end{aligned}$$

be two new objects to be compared. $\mathcal{M}$ will classify the instance

$$\begin{pmatrix} \overbrace{\overset{x_1}{0} \quad \overset{x_2}{-1}}^{\boldsymbol{x}_4} & \overbrace{\overset{z_1}{1} \quad \overset{z_2}{0} \quad \overset{z_3}{0} \quad \overset{z_4}{1}}^{\boldsymbol{z}_4} & \overbrace{\overset{z_1}{0} \quad \overset{z_2}{1} \quad \overset{z_3}{1} \quad \overset{z_4}{0}}^{\boldsymbol{z}_5} \end{pmatrix}.$$

$\mathcal{M}$ decides either '>', predicting that $\boldsymbol{z}_4 \succ_{\boldsymbol{x}_4} \boldsymbol{z}_5$, or '<', predicting that $\boldsymbol{z}_5 \succ_{\boldsymbol{x}_4} \boldsymbol{z}_4$. Points are given for the winner of each comparison, and after all comparisons, the objects are ranked according to their score.

3.2 Label Ranking

The label ranking *method* should not be confused with the label ranking *scenario* described in section 2.1. The label ranking method is still a method for the unified scenario. It carries its name as it functions by viewing any unified ranking task essentially as a label ranking task. That is, the features of the objects $\mathcal{Z} = \mathcal{Z}_1 \times \mathcal{Z}_2 \times \ldots \times \mathcal{Z}_n$ are being hidden from the learning algorithm. Each unique object $\boldsymbol{z} \in \mathcal{Z}$ is being mapped to a unique label $y \in \mathcal{Y}$, thus transforming the unified scenario into a label ranking scenario.

After this transformation, the method works similarly to the baseline method just described in section 3.1, in that it reduces the problem to a classification task learned by a single classifier $\mathcal{M}$ with each instance representing a pairwise comparison or preference.

For example, with the context

$$\boldsymbol{x}_1 = \begin{pmatrix} 2 & 0 \end{pmatrix}$$

and the preference

$$(\boldsymbol{x}_1, \{\boldsymbol{z}_1 \succ \boldsymbol{z}_2\})$$

in the training data, $\mathcal{M}$ would learn

$$\begin{array}{cccccc} & \overbrace{\mathcal{X}_1 \quad \mathcal{X}_2}^{\boldsymbol{x}_1} & \mathcal{Y}_1 & \mathcal{Y}_2 & & \\ (& 2 \quad\; 0 & y_1 & y_2 & > &) \end{array}$$

The features of $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ do not matter. They are ignored and the classifier only sees the labels y_1, y_2 as values for two nominal attributes.

At prediction time, just as in baseline ranking, the order of the objects is determined through pairwise comparison. As a consequence of the label ranking-like approach, the restriction applies that only objects which appeared in the training data can be ranked, but not new or unseen objects.

3.3 Default Ranking

The default ranking method is more of a measure for evaluation than it is a method to be considered for an actual ranking task. It is analogous to the *default classifier* that asserts *default accuracy* in the standard machine learning classification scenario. Please see section 6.3 for a more detailed description.

Chapter 4

Decomposition

Decomposition of a data mining task refers to the practice of dividing a given problem into many sub-problems. In order to solve the original problem, the sub-problems are solved first, and then these solutions are combined to form a solution to the original problem. This approach shares with ensemble methods such as bagging and boosting the similarity that the predictions of multiple models are combined. The difference is that ensemble methods generally learn the original problem multiple times, while in decomposition, the sub-problems are different from and are usually less complex than the original problem.

4.1 Advantages

Maimon & Rokach (2005, pp.125–128) assert the following advantages of decomposing a data mining problem:

Prediction performance or accuracy can be improved; this is arguably the most important advantage. The most frequent cause for this effect is a more optimal bias-variance tradeoff. As the problem is decomposed into smaller sub-problems on each of which a model is learned, the instance space becomes less complex, and enables many algorithms to find a better-suited function.

Large datasets can become more tractable by only processing a reduced portion of the data at a time.

Comprehensibility can be increased if the smaller problems have a smaller and more defined scope than the original problem.

Modularity as a property of the decomposed task makes it possible to rebuild only parts of the model when new data affects only a portion of the built sub-models.

Parallel processing of the sub-problems is another important advantage. Especially at a time where the single-processor paradigm increasingly reaches its limits, the possibility of processing different parts of a problem in parallel becomes interesting, whether the computation be distributed over a network or multiple processors in one machine. Of course, to facilitate such a distribution, the decomposed sub-problems must be independent of each other.

Flexibility in techniques arises from the possibility of learning each sub-problem utilizing a different (or differently parametrized) algorithm.

The two advantages that seem to be the most promising for the problem at hand are increased prediction performance and the ability to process the decomposed sub-problems in parallel.

The former will be examined in chapter 7, where the prediction performance of methods that utilize decomposition techniques will be experimentally compared to methods that do not utilize decomposition.

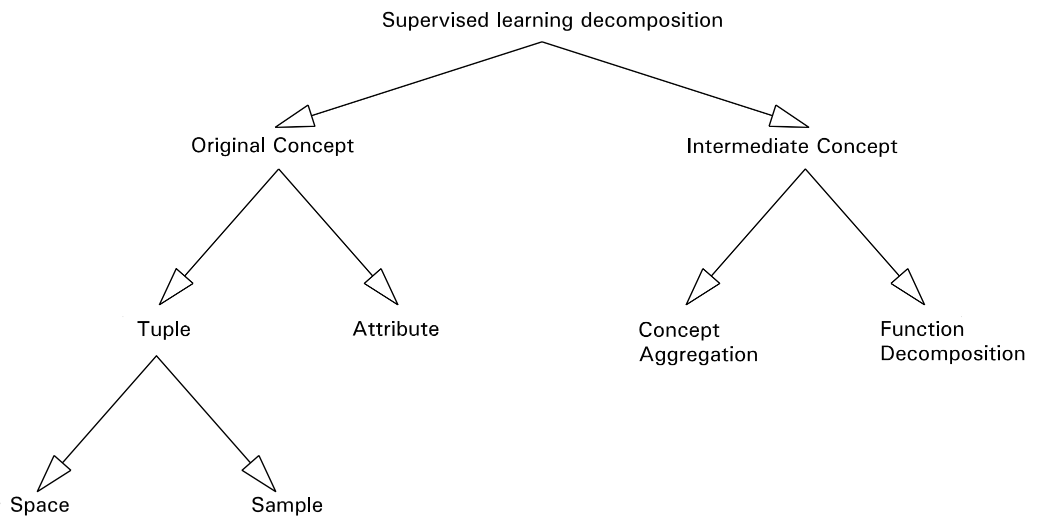


Figure 4.1: Taxonomy of decomposition methods according to Maimon & Rokach (2005)

The latter will be subject of chapter 6, where the implications of decomposition on computation complexity will be discussed.

4.2 Taxonomy of Decomposition Methods

A taxonomy of decomposition methods is introduced by Maimon & Rokach (2005, pp.128–135). Although it was written with the decomposition of classification tasks in mind, it will be described briefly in order to discuss its applicability to the ranking decomposition presented later on.

Figure 4.1 shows a tree view of the taxonomy. The first division is between those methods that learn the *original concept*, that is, the same concept that is learned for the original problem, and those methods that learn *intermediate concepts.*

Intermediate concept decomposition splits into *concept aggregation* and *function decomposition.* In concept aggregation, the target attribute y is replaced with a function so that the domain of the new target attribute $y*$ is smaller than the domain of the original attribute:

$$f(y) : \mathrm{dom}(y) \mapsto \mathrm{dom}(y*) \qquad (|\mathrm{dom}(y*)| < |\mathrm{dom}(y)|)$$

As examples, Maimon & Rokach cite Buntine (1996), which is concerned with text classification. Instead of classifying the text into a topic immediately, it is first classified into a broader group of topics, and in a second step classified within that group. The other example given is the *error-correcting output coding (ECOC)* approach introduced by Dietterich & Bakiri (1995). This approach decomposes a multi-class classification problem into several binary classification problems and is of some relevance to the present work; it will be discussed in more detail later.

Function decomposition looks for intermediate concepts, which are then used as features for further classification, so that a hierarchy of concepts is constructed.

Original concept decomposition learns the original concept and is in that somewhat similar to ensemble methods, but uses always only a sub-sample of the original training data. Sampling can happen across the *attributes*, meaning each of the sub-samples has only a portion of the original attributes, or across the instances, called *tuple decomposition.* Tuples can in turn either be decomposed with regard to their *position in space* or without regard to their position, called *sample decomposition.*

chapter 5 proposes a method that decomposes the ranking task into a number of binary classification tasks. According to the above taxonomy, this can be viewed as a concept aggregation approach for the function

$$f : \mathcal{X} \times Z \to S_Z \quad (Z \subseteq \mathcal{Z})$$

introduced in section 2.3.

4.3 Classification

To illustrate the application of problem decomposition in machine learning and to introduce two techniques which will have some relevance to the further discussion, the work of Dietterich & Bakiri (1995) and Allwein *et al.* (2001) will be described briefly.

The former is the ECOC technique, which was already mentioned above and is given by Maimon & Rokach as an example for concept aggregation. One original multiclass classification problem is reduced to l binary problems. This is achieved by *encoding* each class by means of a binary *code word* of length l, in which each entry decides if an example with the respective class is a positive or negative training example for a particular binary classifier. The coding words make up a *coding matrix* of binary entries.

At prediction time, each of the binary classifiers predicts a binary value, and all of the values make up a binary vector of length l, which can then be *decoded* into one of the class values by comparison to the code words.

Allwein *et al.* generalize the approach by allowing the entries of the code words and thus the coding matrix to take on three values. Additional to a value signifiying a positive class value for the decomposed classifiers and one signifying a negative class value, there is a value signifying that the particular examples should be ignored.

A more detailed description of the approach will be given in chapter 5, where it will be transferred to the ranking scenario.

4.4 Pairwise Label Ranking

Pairwise label ranking (Hüllermeier *et al.* , 2008) can serve as an example for problem decomposition applied to ranking.

In this method, a binary classifier is learned for each pair of labels. Assuming p labels, the approach learns $\frac{1}{2} \cdot p \cdot (p-1)$ binary classifiers, one for each unordered pair or 2-combination of labels. A classifier for any given pair of labels y_i, y_j takes as positive examples all context instances $\boldsymbol{x}_\ell$ with the associated preference $y_i \succ_{\boldsymbol{x}_\ell} y_j$, and as negative examples all context instances $\boldsymbol{x}_\ell$ with the associated preference $y_j \succ_{\boldsymbol{x}_\ell} y_i$.

To make a prediction, all of the binary classifiers are queried and either make a positive or negative prediction for their associated pair of labels. A positive prediction would indicate that the first object is ranked higher than the second object, and a negative prediction would indicate the opposite. From the single predictions, a rank of all labels can be aggregated.

Pairwise label ranking is brought up again in subsection 5.1.4, where it serves as an example to illustrate the use of a coding matrix for ranking.

Chapter 5

Object Feature Coding

As described above in section 4.3, Allwein *et al.* (2001) introduce a framework for multiclass classification by reduction to multiple binary classification problems, unifying and generalizing from previous approaches such as one-against-all classification or error-correcting output codes.

A *coding matrix*

$$\boldsymbol{M} \in \{-1, 0, +1\}^{k \times l}$$

commands the decomposition of the multiclass learning problem into multiple binary problems and the aggregation of these results to an answer to the original problem. The number of rows k of the matrix equals the number of classes; l is the number of columns and equals the number of binary classifiers $\mathcal{M}_s$ that are trained for the problem, each of which is learning a function f_s. l and the entries of the matrix, taken from $\{-1, 0, 1\}$ are chosen arbitrarily and determine the specific decomposition approach within the framework.

At training time, for each binary classifier $\mathcal{M}_s, (s \in \{1 \dots l\})$ the associated column s is inspected. For row r, a 1 entry means the class associated with r marks positive training examples for $\mathcal{M}_s$; -1 marks negative training examples, and 0 means that each example of class r is to be ignored and not among the training data for classifier f_s.

To classify a new example $\boldsymbol{x}$, the predictions of the classifiers $\mathcal{M}_s$ make up a vector

$$\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_l(\boldsymbol{x})).$$

This vector is then compared to each row r of the coding matrix. Let $\boldsymbol{M}(r)$ denote row r of the matrix, then a distance function $d(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{M}(r))$ is minimized over r. The class associated with the row that is closest to $\boldsymbol{f}(\boldsymbol{x})$ is predicted.

5.1 Coding Matrix for Ranking

The idea of using a coding matrix can be transferred to the scenario of ranking by associating each row of the coding matrix $\boldsymbol{M}$ not with a class, but with a binary preference between two objects.

To clarify the use of the terms *binary preference* and *pairwise comparison*, consider that a pairwise comparison of objects $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ can have two outcomes: $\boldsymbol{z}_1 \succ \boldsymbol{z}_2$ and $\boldsymbol{z}_2 \succ \boldsymbol{z}_1$. Each of these outcomes is called a binary preference. This means that there are twice as many binary preferences as pairwise comparisons. On a given set of objects Z of size p, the number of pairwise comparisons equals the number of 2-*combinations*, which are unordered subsets of size 2. There are $\binom{p}{2} = \frac{1}{2} \cdot p \cdot (p-1)$ pairwise comparisons on Z. The number of pairwise preferences is the number of unordered 2-*variations*, which is $p \cdot (p-1)$.

When we say there is a row for each binary preference, this means there is one row for $\boldsymbol{z}_1 \succ \boldsymbol{z}_2$, and another row for $\boldsymbol{z}_2 \succ \boldsymbol{z}_1$. The total number of rows is $k = p \cdot (p-1)$. In the terms of ECOC, there is a code word assigned to each binary preference. In this respect, the binary preferences assume the place of the classes in the original approach.

It might deserve some consideration whether it wouldn't suffice to introduce a row for each *pairwise comparison* only, thus cutting in half the number of rows. This would be a welcome optimization, but as we will see below, the chosen approach allows to state the algorithm in an elegant manner; thus, the pairwise comparison version is given no consideration here.[1]

The number of *columns* l is arbitrarily chosen and depends on the particular method within the framework. Below, examples of various methods will be given.

5.1.1 Training

Each column of the matrix is associated with a binary classifier. Each row of the matrix is associated with a binary preference. Assume a training example with the binary preference $\boldsymbol{z}_i \succ \boldsymbol{z}_j$. The row r associated with $\boldsymbol{z}_i \succ \boldsymbol{z}_j$ is inspected. For each column s, the matrix entry $\boldsymbol{M}_{r,s}$ decides if the example is positive (an entry of 1), negative (an entry of -1), or to be ignored (an entry of 0) by the classifier $\mathcal{M}_s$. The relation that the classifiers work on is the context relation, as will be expanded upon below.

In this way, the original training data set is converted into l sets of training data for the single classifiers.

5.1.2 Prediction

Each of the l functions classifies a new instance $\boldsymbol{x}$. The single classifications must then be aggregated to gain a prediction for the original problem.

[1]An inversion of sign of each entry of the row could render unnecessary the storage of a complementary row, but this would not decrease the number of distance calculations that need to be performed. Enough consideration.

5.1.3 Aggregation

The classification vector $f(\boldsymbol{x})$ of the l functions is compared to the rows $\boldsymbol{M}(r)$, $(r \in \{1 \dots k\})$, and the distance to each is measured using an apt *distance function*. A high distance means that the row is very far from the prediction vector, and thus that the binary preference for which the row stands is unlikely for context $\boldsymbol{x}$. Assume, row r is associated with the preference $\boldsymbol{z}_i \succ \boldsymbol{z}_j$. A high distance of r to $f(\boldsymbol{x})$ means that in context $\boldsymbol{x}$, $\boldsymbol{z}_i$ should not be ranked higher than $\boldsymbol{z}_j$.

By comparing the prediction vector to each row, a score can be accumulated for each object. Here, too, different functions can be used for scoring.

Two of these will be shown as examples. They build on the rank of the rows, which in turn is derived from the distance of each row to the classification vector: The rows are ranked depending on their distance to $f(\boldsymbol{x})$, such that the row closest to $f(\boldsymbol{x})$ is ranked first and the row most distant to $f(\boldsymbol{x})$ is ranked last.

The row ranked first adds $p-1$ points to the score of the preferred object it represents, the row ranked second adds $p-2$ points and so on. The score awarded to object $\boldsymbol{z}_i$ can be expressed as

$$\text{score}_A(\boldsymbol{z}_i) = \sum_{j \neq i} p - \text{rank}(\boldsymbol{z}_i \succ \boldsymbol{z}_j).$$

This is the function used for the experiments described in chapter 7.

Another way of score calculation only considers the top-ranked half of binary preferences and counts the occurrence of decisions in favor of $\boldsymbol{z}_i$.

$$\text{score}_B(\boldsymbol{z}_i) = \sum_{j \neq i, \text{rank}(\boldsymbol{z}_i \succ \boldsymbol{z}_j) < \frac{k}{2}} 1.$$

The idea behind this way of scoring is that if e.g. preference $\boldsymbol{z}_1 \succ \boldsymbol{z}_2$ is in the top half, it is reasonable to assume that its inversion, $\boldsymbol{z}_2 \succ \boldsymbol{z}_1$ would be in the bottom half. If this assumption holds true, for each pairwise comparison, one object will “win” and be given one point.

After a score is calculated for each object, the objects can be ordered according to that score. This order or ranking of the objects is the prediction for $\boldsymbol{x}$.

Note that ranking rows and calculating scores is a particular way of deriving a final order of the objects. The general idea is that *for each binary preference, a distance can be calculated to any given context.* From this information, an order of the objects can be found, and there is certainly room for optimization in finding a good mapping function.

5.1.4 Example: Pairwise Label Ranking

Employing the described scheme, a coding matrix can be utilized for the decomposition of ranking tasks. Note that the pairwise comparisons that make up the rows are 2-permutations on an established, finite set of objects; for now, assume that no unseen objects will be ranked. The ranking of unseen objects will be discussed in the further course of this chapter.

The relation on which the single decomposed classifiers learn is supposed to be the context relation. The instances which the classifiers see for training and classification are context instances. The features of the objects are not part of that relation; rather, the object features will used exclusively to determine the decomposition; this will be expanded on in section 5.2.

To give an example of which kinds of decomposition can be modelled with the help of a coding matrix, and to illustrate the approach discussed above, pairwise label ranking (Hüllermeier *et al.* , 2008), already introduced in section 4.4. To reiterate, in this method, for p labels, $\frac{1}{2} \cdot p \cdot (p-1)$ binary classifiers are learned, one for each unordered pair or 2-combination of labels. A classifier $\mathcal{M}_{i,j}$ for labels y_i, y_j learns a function $f_{i,j}$ and takes as positive examples all context instances $\boldsymbol{x}_\ell$ which have the preference $y_i \succ_{\boldsymbol{x}_\ell} y_j$, and as negative examples all context instances $\boldsymbol{x}_\ell$ which have the preference $y_j \succ_{\boldsymbol{x}_\ell} y_i$.

Since there are no object features involved in label ranking, it is important to note that we are not talking about object feature coding yet. The object feature coding method uses a coding matrix and builds it from the object features; but the general approach of using a coding matrix for ranking encompasses a broader range of applications, such as this one.

As an example, consider the set of objects $\mathcal{Z} = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3\}$. Since (pairwise) label ranking views objects only as labels, object features are disregarded and each distinct object is assigned a label. The coding matrix for pairwise label ranking would be

	$\mathcal{M}_{1,2}$	$\mathcal{M}_{1,3}$	$\mathcal{M}_{2,3}$
$1 \succ 2$	1	0	0
$1 \succ 3$	0	1	0
$2 \succ 1$	-1	0	0
$2 \succ 3$	0	0	1
$3 \succ 1$	0	-1	0
$3 \succ 2$	0	0	-1

Three classifiers are trained; $\mathcal{M}_{1,2}$ for example is associated with the first column and takes for positive training instances all contexts with the preference $\boldsymbol{z}_1 \succ \boldsymbol{z}_2$, for negative instances all contexts with the associated preference $\boldsymbol{z}_2 \succ \boldsymbol{z}_1$, while ignoring all other context instances.

Assume the training data $\mathcal{T} = \{t_1, t_2, t_3\}$

$$\begin{aligned} t_1 = (\boldsymbol{x}_1, \ \{&\boldsymbol{z}_1 \succ \boldsymbol{z}_2, \\ &\boldsymbol{z}_2 \succ \boldsymbol{z}_3\}) \end{aligned}$$

$$\begin{aligned} t_2 = (\boldsymbol{x}_2, \ \{&\boldsymbol{z}_2 \succ \boldsymbol{z}_1, \\ &\boldsymbol{z}_3 \succ \boldsymbol{z}_1\}) \end{aligned}$$

$$\begin{aligned} t_3 = (\boldsymbol{x}_3, \ \{&\boldsymbol{z}_2 \succ \boldsymbol{z}_3, \\ &\boldsymbol{z}_3 \succ \boldsymbol{z}_1\}) \end{aligned}$$

The first preference $\boldsymbol{z}_1 \succ \boldsymbol{z}_2$ of t_1 is associated with the first row of the matrix. The entry for $\mathcal{M}_{1,2}$ is 1, thus, $\boldsymbol{x}_1$ is used as a positive training example by $\mathcal{M}_{1,2}$. The entries for $\mathcal{M}_{1,3}$ and $\mathcal{M}_{2,3}$ are both 0, which means that t_1 is neither a positive nor a negative training example for those classifiers.

The second preference $\boldsymbol{z}_2 \succ \boldsymbol{z}_3$ of t_1 is associated with the fourth row of the matrix. In this row, the first two entries are empty, signifying that $\boldsymbol{x}_1$ is neither a positive nor a negative example for the first two classifiers $\mathcal{M}_{1,2}$ or $\mathcal{M}_{1,3}$. The entry in the third column is 1; therefore, $\boldsymbol{x}_1$ is used as a positive example by $\mathcal{M}_{2,3}$.

t_2 provides a negative example to $\mathcal{M}_{1,2}$: The first preference $\boldsymbol{z}_2 \succ \boldsymbol{z}_1$ is associated with row three. Here, the entry in the first row is -1, meaning $\boldsymbol{x}_2$ will be a negative example to $\mathcal{M}_{1,2}$.

It becomes apparent how this specific matrix leads to the decomposition method introduced above as pairwise label ranking. Applying the matrix to each example yields the following training data for the single decomposed classifiers:

	$+$	$-$
$\mathcal{M}_{1,2}$	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$
$\mathcal{M}_{1,3}$		$\boldsymbol{x}_3$
$\mathcal{M}_{2,3}$	$\boldsymbol{x}_1, \boldsymbol{x}_3$	$\boldsymbol{x}_2$

For a prediction example, assume that an order for a new context x_4 is to be calculated. Each of the learned functions gives a classification. Assume that $\mathcal{M}_{1,2}$ classifies x_4 as -1, $\mathcal{M}_{1,3}$ yields -1(note that $\mathcal{M}_{1,3}$ only has a negative and no positive example in the table above, so it is likely to only predict negatively), and $\mathcal{M}_{2,3}$ yields 1. Thus, the classification vector is

$$f(\boldsymbol{x}_4) \quad = \quad (\quad \text{-1} \quad \text{-1} \quad 1 \quad).$$

Assuming a euclidean distance function, and using the scoring method score$_B$ that only considers the top half of binary preferences, we get

	dist.	score
$1 \succ 2$	2.4	inc. $score(\boldsymbol{z}_1)$ by 1
$1 \succ 3$	2.4	inc. $score(\boldsymbol{z}_1)$ by 1
$3 \succ 2$	2.4	inc. $score(\boldsymbol{z}_3)$ by 1
$2 \succ 1$	1.4	–
$2 \succ 3$	1.4	–
$3 \succ 1$	1.4	–

The scores are 2 for $\boldsymbol{z}_1$, 1 for $\boldsymbol{z}_3$ and 0 for $\boldsymbol{z}_2$. Thus, the order

$$\boldsymbol{z}_1 \succ \boldsymbol{z}_3 \succ \boldsymbol{z}_2$$

will be predicted for the context $\boldsymbol{x}_4$.

5.2 Object Feature Coding

The previous section of this chapter introduced an approach to ranking by means of a coding matrix. An example given for pairwise label ranking showed how the approach can be applied to the label ranking scenario described in section 2.1.

This section introduces *object feature coding*, a method that assumes the unified ranking scenario introduced in section 2.3. It constructs a coding matrix

$$\boldsymbol{M} \in \{-1, 0, +1\}^{k \times l}$$

from the features of the objects to be ranked.

k is given through the number of objects p by $k = p(p-1)$. l is the number of classifiers. It must be chosen and defines, together with the construction of the matrix entries, the method of decomposition.

A function

$$v : \mathcal{Z} \times \mathcal{Z} \rightarrow E^l$$

maps each combination of objects $\boldsymbol{z}_i$, $\boldsymbol{z}_j \in \mathcal{Z}$ to an l-dimensional vector that is the matrix row associated with the preference $\boldsymbol{z}_i \succ \boldsymbol{z}_j$.

E is the domain of the matrix entries. In the examples given until now, and in the experiments in chapter 7, it is $E = \{-1, 0, 1\}$ but other domains are possible. For example, a continuous value could express the level of confidence of the classification.

Recall that the objects are given by feature vectors:

$$\mathcal{Z} = \mathcal{Z}_1 \times \mathcal{Z}_2 \times \ldots \times \mathcal{Z}_n$$

and thus, $v(\cdot)$ can be expressed in terms of the domains for the single features

$$v : \mathcal{Z}_1 \times \mathcal{Z}_2 \times \ldots \times \mathcal{Z}_n \times \mathcal{Z}_1 \times \mathcal{Z}_2 \times \ldots \times \mathcal{Z}_n \rightarrow E^l.$$

It becomes apparent that the coding matrix is a function on the object features; thus the name of the approach, object feature coding.

5.2.1 Per Attribute

In the following, a method that utilizes one classifier per object attribute is described. As discussed above, each row of the coding matrix represents a binary preference. For two objects $\boldsymbol{z}_i, \boldsymbol{z}_j$, let

$$v(\boldsymbol{z}_i, \boldsymbol{z}_j) = \boldsymbol{\delta}_{i,j} = \boldsymbol{z}_i - \boldsymbol{z}_j$$

be the difference between the object feature vectors, and the coding matrix

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{\delta}_{1,2} \\ \boldsymbol{\delta}_{1,3} \\ \vdots \\ \boldsymbol{\delta}_{2,1} \\ \vdots \\ \boldsymbol{\delta}_{3,1} \\ \vdots \\ \boldsymbol{\delta}_{p,p-1} \end{bmatrix}$$

for all $\boldsymbol{\delta}_{i,j}$ for which $i \neq j$.

As an example, consider the objects in four-dimensional feature space

$$\begin{array}{lcl} \boldsymbol{z}_1 & = & (\ 0\ \ 1\ \ 0\ \ 1\ ) \\ \boldsymbol{z}_2 & = & (\ 1\ \ 0\ \ 1\ \ 0\ ) \\ \boldsymbol{z}_3 & = & (\ 0\ \ 0\ \ 1\ \ 0\ ) \end{array}$$

Given the following contexts

$$\begin{array}{lcl} \boldsymbol{x}_1 & = & (\ 2\ \ 0\ ) \\ \boldsymbol{x}_2 & = & (\ 0\ \ 2\ ) \\ \boldsymbol{x}_3 & = & (\ \text{-2}\ \ 0\ ) \end{array}$$

and the training data $\mathcal{T} = \{t_1, t_2, t_3\}$

$$\begin{aligned} t_1 = (\boldsymbol{x}_1,\ &\{\boldsymbol{z}_1 \succ \boldsymbol{z}_2, \\ &\ \ \boldsymbol{z}_2 \succ \boldsymbol{z}_3\}) \end{aligned}$$

$$\begin{aligned} t_2 = (\boldsymbol{x}_2,\ &\{\boldsymbol{z}_2 \succ \boldsymbol{z}_1, \\ &\ \ \boldsymbol{z}_3 \succ \boldsymbol{z}_1\}) \end{aligned}$$

$$\begin{aligned} t_3 = (\boldsymbol{x}_3,\ &\{\boldsymbol{z}_2 \succ \boldsymbol{z}_3, \\ &\ \ \boldsymbol{z}_3 \succ \boldsymbol{z}_1\}), \end{aligned}$$

$\boldsymbol{\delta}_{ij} = \boldsymbol{z}_i - \boldsymbol{z}_j$ yields the following coding matrix $\boldsymbol{M}$:

$$\begin{array}{cccrrrrc}
 & & & \mathcal{M}_1 & \mathcal{M}_2 & \mathcal{M}_3 & \mathcal{M}_4 & \\
\hline
\boldsymbol{\delta}_{12} & = & (& -1 & 1 & -1 & 1 &) \\
\boldsymbol{\delta}_{13} & = & (& 0 & 1 & -1 & 1 &) \\
\boldsymbol{\delta}_{21} & = & (& 1 & -1 & 1 & -1 &) \\
\boldsymbol{\delta}_{23} & = & (& 1 & 0 & 0 & 0 &) \\
\boldsymbol{\delta}_{31} & = & (& 0 & -1 & 1 & -1 &) \\
\boldsymbol{\delta}_{32} & = & (& -1 & 0 & 0 & 0 &)
\end{array}$$

Applying the algorithm described above, for each of the four object attributes or columns of the coding matrix, a binary function f_s is learned by $\mathcal{M}_s$. It uses $\boldsymbol{x}$ as a positive instance if there is an example $(\boldsymbol{x}, \boldsymbol{z}_i \succ \boldsymbol{z}_j)$ and there is a matrix entry $\boldsymbol{M}_{rs} = 1$ for which row r is associated with the comparison $\boldsymbol{z}_i \succ \boldsymbol{z}_j$. Analogous, for $\boldsymbol{M}_{rs} = -1$, $\boldsymbol{x}$ would be a negative example. Entries $\boldsymbol{M}_{rs} = 0$ are ignored.

For the given example, this yields:

	$+$	$-$
$\mathcal{M}_1$	$\boldsymbol{x}_2(t_2), \boldsymbol{x}_1(t_1), \boldsymbol{x}_3(t_3)$	$\boldsymbol{x}_1(t_1)$
$\mathcal{M}_2$	$\boldsymbol{x}_1(t_1)$	$\boldsymbol{x}_2(t_2), \boldsymbol{x}_2(t_2), \boldsymbol{x}_3(t_3)$
$\mathcal{M}_3$	$\boldsymbol{x}_2(t_2), \boldsymbol{x}_2(t_2), \boldsymbol{x}_3(t_3)$	$\boldsymbol{x}_1(t_1)$
$\mathcal{M}_4$	$\boldsymbol{x}_1(t_1)$	$\boldsymbol{x}_2(t_2), \boldsymbol{x}_2(t_2), \boldsymbol{x}_3(t_3)$

For a new context, $\boldsymbol{x}_4 = (1, 1)$, let the classifications by f_1 to f_4 be $+1$, -1, $+1$, -1. The resulting vector $(+1 - 1 + 1 - 1)$ is compared with the rows of $\boldsymbol{M}$, using an apt distance function. A low difference means that the corresponding preference can be predicted with a high confidence. For example, the resulting vector is equal to row 3 of $\boldsymbol{M}$, meaning that y_2 is likely to be ranked higher than y_1. All comparisons are aggregated to determine an order of the objects.

In the training data above, note that $\boldsymbol{x}_1$ is both a positive and a negative example for $\mathcal{M}_1$. This can be resolved by assigning each instance to the class in which it occurs most frequently. For the case at hand, $\boldsymbol{x}_1$ occurs once positive and once negative, thus it would not be used for $\mathcal{M}_1$ at all. The result is the final training data

	$+$	$-$
$\mathcal{M}_1$	$\boldsymbol{x}_2(t_2), \boldsymbol{x}_3(t_3)$	
$\mathcal{M}_2$	$\boldsymbol{x}_1(t_1)$	$\boldsymbol{x}_2(t_2), \boldsymbol{x}_2(t_2), \boldsymbol{x}_3(t_3)$
$\mathcal{M}_3$	$\boldsymbol{x}_2(t_2), \boldsymbol{x}_2(t_2), \boldsymbol{x}_3(t_3)$	$\boldsymbol{x}_1(t_1)$
$\mathcal{M}_4$	$\boldsymbol{x}_1(t_1)$	$\boldsymbol{x}_2(t_2), \boldsymbol{x}_2(t_2), \boldsymbol{x}_3(t_3)$

This means that $\mathcal{M}_1$ has no negative examples and would likely always predict $+$.

The example uses binary attributes which take their values from $0, 1$. The difference of two such values then is in $\{-1, 0, 1\}$ and can be used in the coding matrix in the same fashion as by Allwein *et al.* (2001). If the object attributes are not binary, there are several possible solutions.

First, transform the non-binary object attributes to binary object attributes. Usually, the number of new binary attributes should be higher to convey the same amount of information.

Secondly, the entries of the coding matrix can be transformed to be in $\{-1, 0, 1\}$.

Thirdly, learn l functions not with a binary codomain, but with a real-valued codomain. This decomposes the problem not into l classification tasks, but into l numeric prediction or regression tasks.

In the experiments listed in chapter 7, the second option has been chosen. The matrix entries are simply the sign of the differences of the real-valued object attributes. This is a rather simplistic approach, as information that lies in the magnitude of the values is discarded. More sophisticated approaches might increase the quality of prediction.

Finally, we can take a look at the semantics of the per-attribute decomposition, that is, we can try to describe in words the predicate that the single classifiers learn.

There is one classifier per attribute. The classifier for attribute s learns the class + for the preference $\boldsymbol{z}_i \succ \boldsymbol{z}_j$ if attribute s is greater in $\boldsymbol{z}_i$ than in $\boldsymbol{z}_j$. Thus, it learns to rank if a high value for attribute s contributes to a higher rating or conversely to a lower rating. To give an even a more concrete example, consider the sushi dataset. Here, for instance, a classifier could learn if a person with certain attributes likes an oily sushi item or, conversely, will give such an item a low rating.

There are two problems with this approach: First, the underlying utility function might not be monotone. A sushi item might be rated high if it is just oily enough, but not if it is either not oily at all or too oily.

Secondly, it might not be sufficient to consider the object attributes independently. For example, a certain type of person might like sushi that is just oily or sushi that is just made with seafood, but not sushi that is both oily and made with seafood.

The second problem provides the motivation for the following method.

5.2.2 Cross Attribute

This method uses the difference between each pair of distinct attributes of an object pair, and is thus called cross-attribute. Let $\boldsymbol{z}_i(q)$ denote the value of the qth attribute for object $\boldsymbol{z}_i$, p the number of objects and n the number of object attributes. Then the coding matrix can be defined as

$$\boldsymbol{M} = \begin{array}{c} \overbrace{\hspace{36em}}^{\text{object attribute pairs}} \\ \begin{bmatrix} \boldsymbol{z}_1(1)-\boldsymbol{z}_2(2) & \boldsymbol{z}_1(1)-\boldsymbol{z}_2(3) & \dots & \boldsymbol{z}_1(2)-\boldsymbol{z}_2(1) & \dots & \boldsymbol{z}_1(3)-\boldsymbol{z}_2(1) & \dots & \boldsymbol{z}_1(n)-\boldsymbol{z}_2(n-1) \\ \boldsymbol{z}_1(1)-\boldsymbol{z}_3(2) & \boldsymbol{z}_1(1)-\boldsymbol{z}_3(3) & \dots & \boldsymbol{z}_1(2)-\boldsymbol{z}_3(1) & \dots & \boldsymbol{z}_1(3)-\boldsymbol{z}_3(1) & \dots & \boldsymbol{z}_1(n)-\boldsymbol{z}_3(n-1) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \boldsymbol{z}_2(1)-\boldsymbol{z}_1(2) & \boldsymbol{z}_2(1)-\boldsymbol{z}_1(3) & \dots & \boldsymbol{z}_2(2)-\boldsymbol{z}_1(1) & \dots & \boldsymbol{z}_2(3)-\boldsymbol{z}_1(1) & \dots & \boldsymbol{z}_2(n)-\boldsymbol{z}_1(n-1) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \boldsymbol{z}_3(1)-\boldsymbol{z}_1(2) & \boldsymbol{z}_3(1)-\boldsymbol{z}_1(3) & \dots & \boldsymbol{z}_3(2)-\boldsymbol{z}_1(1) & \dots & \boldsymbol{z}_3(3)-\boldsymbol{z}_1(1) & \dots & \boldsymbol{z}_3(n)-\boldsymbol{z}_1(n-1) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \boldsymbol{z}_p(1)-\boldsymbol{z}_{p-1}(2) & \boldsymbol{z}_p(1)-\boldsymbol{z}_{p-1}(3) & \dots & \boldsymbol{z}_p(2)-\boldsymbol{z}_{p-1}(1) & \dots & \boldsymbol{z}_p(3)-\boldsymbol{z}_{p-1}(1) & \dots & \boldsymbol{z}_p(n)-\boldsymbol{z}_{p-1}(n-1) \end{bmatrix} \end{array}$$

This means that there are $n(n-1)$ classifiers trained, one for each pair of attributes. A matrix entry for any pair of objects is the difference of the first attribute of the first object and the second attribute of the second object.

The motivation for this method is, as mentioned above, the ability to learn predicates across attributes, so that dependent attributes can be taken into consideration.

The matrix construction works differently; in all other respects, the method works just as the one described above.

5.2.3 Unseen objects

Although the focus of this work lies on a scenario where the set of objects is fixed an known at training time, it should be noted that the object feature coding approach can rank unseen objects.

Recall the function that is learned in the unified ranking scenario from section 2.3:

$$f : \mathcal{X} \times Z \to S_Z \quad (Z \subseteq \mathcal{Z})$$

The function ranks any subset Z of the object space $\mathcal{Z}$. According to the definition, there is no requirement that any of the objects in Z have been in the training data.

Such a set of unknown objects can be ranked by generating the corresponding rows of the matrix at prediction time. Recall how the rankings are generated: Each row of the matrix is rated by its difference to the prediction, and then, the objects that appear in the rows' preferences are ranked accordingly. A row with a preference that contains an irrelevant object, that is, an object that is not in Z, can be discarded.

The conclusion of this reflection is that, in the general case, matrix generation for learning and matrix generation for prediction are two different steps. This differentiation has not been made so far, because the scenario of unseen objects has not been a primary concern.

5.3 Generality

In this chapter, it has been demonstrated how a coding matrix can be used for ranking tasks. Within this general approach, several parameters determine the specific ranking method. In this final section of the chapter, the particular choices that have been made will be highlighted, in order to distinguish what is specific and what is generic, and to reiterate the underlying general approach.

At the base is the idea that for each possible preference between objects, a vector is assigned. This vector – in the terms used above, a row of the matrix – can be arbitrarily chosen, and could in fact be random. Each component of the vector is learned to be predicted by a single machine learning algorithm. The first choice is from which domain the values of the vector are taken. Depending on this is the choice for the predicting functions and the algorithms that learn them. The focus in the above chapter was on binary values, but real values or some measure of confidence for a binary classification are possible.

The next generic condition is the existence of a distance function between the prediction vector and the original vectors, which is another parameter to be chosen. Independently from that, a function can be chosen that aggregates the difference value of each row to an order of the objects. This must be done because each row is associated with a binary preference, not an object.

On this base, the approach dubbed *object feature coding* by this work is built. The construction of the matrix, and thus the single vectors for each preference, is done via the object features that are available in the unified scenario, which is the focus of this work. Nevertheless, object features are not a prerequisite for the use of a coding matrix, as has been shown in the pairwise label ranking example.

Finally, there is the choice within the object feature coding framework of a function that maps the feature vectors of a 2-variation of objects to the vector of the associated preference, determining the specific decomposition method. Two such methods have been introduced above, per-attribute ranking and cross-attribute ranking.

The motivation for this emphasis of the general possibilities of the framework is the belief that, although two specific methods have been described, there might still be potential in finding other methods that conform to more strictly defined optimality criteria.

Chapter 6

Evaluation

On the one hand, this chapter discusses the premises for experimental evaluation of the introduced scenario and methods, which will be performed in chapter 7. On the other hand, algorithmic complexity is discussed on a theoretical level.

6.1 Test Data

To assess the quality of the different methods experimentally, apt testing data is needed. While there are numerous data sets available for standard data mining tasks such as classification, and to a smaller degree label or object ranking, little data is available for the scenario presented here. For this reason, and in order to better highlight different properties of the various algorithms, generated data is employed in addition to real-world data.

6.1.1 Sushi Dataset

The sushi data,[1] used in Kamishima (2003) and extended in Kamishima *et al.* (2005) is based on a Japanese survey of preference between types of sushi. It consists of two different datasets.[2]

For the first, called *Sushi A* in the following, participants were asked to order ten different types of sushi according to their preferences. These ten items were the same for each participant.

For the second, called *Sushi B*, participants werde asked to order ten different types of sushi, but these ten items were chosen randomly from a larger set of a hundred sushi types. Thus, each context (user) is associated with a partial order on the complete set of a hundred objects $\mathcal{Z}$.

Both datasets record 11 features of the users, shown in Table 6.1. The sushi items have 9 features, shown in Table 6.2.

6.1.2 Generated Data

The generated data set consists of a hundred context items and ten object items. Contexts and attributes have each ten features, with randomly generated values. For each context, an order of objects is established by rating each object and then ordering objects according to their rating.

The rating function $r(\boldsymbol{x}, \boldsymbol{z})$ for an object $\boldsymbol{z}$ in context $\boldsymbol{x}$

$$r(\boldsymbol{x}, \boldsymbol{z}) = \nu \cdot N_{\boldsymbol{x},\boldsymbol{z}} + (\nu - 1) \cdot \sum_{i=1}^{m} \sum_{j=1}^{n} M_{i,j} \cdot \boldsymbol{x}(i) \cdot \boldsymbol{z}(j)$$

[1]`http://www.kamishima.net/sushi/`

[2]Strictly speaking, it consists of three: The original paper's point was to highlight the differences between *rating* and *ordering*, so the participants were asked to first rate, then order the different items. For the purposes of this work, however, the rating data is not used.

user attributes
id
gender
age
writing speed
childhood prefecture
childhood region
childhood coast (east/west)
current prefecture
current region
current coast (east/west)
childhood / current equal?

Table 6.1: User attributes of the sushi dataset

item attributes
id
name
style (maki/other)
major group (seafood/other)
minor group
oiliness
eating frequency
normalized price
sale frequency

Table 6.2: Item attributes of the sushi dataset

uses an $m \times n$ matrix M of random values; m is the number of context features, n is the number of object features (as mentioned above, both variables have a value of 10 for the data used in the experiments). $M_{i,j}$ determines the influence of the ith context feature $\boldsymbol{x}(i)$ and the jth object feature $\boldsymbol{z}(j)$ on the overall rating of object $\boldsymbol{z}$. N is a matrix with random entries, supposed to introduce noise. $N_{\boldsymbol{x},\boldsymbol{z}}$ is the matrix entry for the combination of context $\boldsymbol{x}$ and object $\boldsymbol{z}$; $\nu \in [0, 1]$ determines the strength of the noise.

After the objects have been ordered by their rating value, 40 preferences of the form $\boldsymbol{z}_1 \succ \boldsymbol{z}_2$ are chosen randomly for each context and made visible in the training data.

For the experiment described in section 7.2, the original object features that determine the ranking have been replaced with random feature values not correlated to the ranking. This is explained in detail in the description of the experiment.

6.2 Prediction Performance

To compare the quality of prediction that the different algorithms deliver, a measure of performance is needed. This section reviews two measures commonly employed in connection with ranking, Spearman's rank coefficient and Kendall's tau coefficient, and discusses their applicability to the scenario at hand.

As a third measure, Kendall's tau is generalized to be able to handle arbitrary binary relations instead of just total orders.

The measure introduced has a similar function to *prediction accuracy* in classification problems. Since the term *accuracy* has that very specific meaning in classification, it will not be used. In order to avoid confusion, the term *prediction performance* will be used instead.

6.2.1 Spearman's Rho

Spearman's rank coefficient, also known as *Spearman's rho* (Salkind, 2006, vol.3, p.927) is an established rank coefficient that has been introduced in the context of psychology by Spearman (1904). It is, for instance, used as a measure of dissimilarity between orders by Kamishima (2003).

Spearman's rank operates on the difference in rank between two total orders on the same set. *Rank* denotes the position of an object in the order, where the first object of the order has rank 1, the second rank 2 etc. For two orders $\succ_1, \succ_2$ on a set of objects $|Z|$, Spearman's rank is given by

$$\rho = 1 - \frac{6\sum_{\boldsymbol{z}\in Z}\left(\text{rank}(\succ_1, \boldsymbol{z}) - \text{rank}(\succ_2, \boldsymbol{z})\right)^2}{|Z|^3 - |Z|}.$$

To give an example, assume $\succ_1$ to be

$$a \succ_1 b \succ_1 c \succ_1 d,$$

yielding the following rank values

object	a	b	c	d
rank	1	2	3	4

and $\succ_2$ to be

$$b \succ_2 a \succ_2 c \succ_2 d,$$

yielding the following rank values

object	a	b	c	d
rank	2	1	3	4

For the example, Spearman's rho is

$$\rho = 1 - \frac{6((1-2)^2 + (2-1)^2 + (3-3)^2 + (4-4)^2)}{4^3 - 4} = 1 - \frac{6(1+1)}{60} = 0.8.$$

A positive value close to 1 indicates a strong positive correlation between the orders. When the orders are identical, the sum of squares of the differences becomes 0, and thus $\rho = 1$. On the other hand, $\rho = -1$ if $\succ_2$ is $\succ_1$reversed.[3]

6.2.2 Kendall's Tau

The *Kendall rank correlation coefficient* (Kendall & Gibbons, 1990, p.3), also known as *Kendall's tau*, is another measure of similarity between two orders on the same set of objects. It takes into account the number of different pairwise preferences between the two orders of the set.

Each total order on a set of objects Z has $\frac{1}{2}|Z|(|Z|-1)$ elements. For example, the order $\succ_1$,

$$a \succ_1 b \succ_1 c \succ_1 d$$

when viewed as a binary relation or set of pairwise preferences, is

$$\succ_1 = \{(a \succ b), (a \succ c), (a \succ d), (b \succ c), (b \succ d), (c \succ d)\}.$$

For a comparison of two orders on the same set Z, the number of different pairs is counted. To compare a second order $\succ_2$,

$$b \succ_2 a \succ_2 c \succ_2 d$$

to the order above, consider the pairs

$$\succ_2 = \{(b \succ a), (b \succ c), (b \succ d), (a \succ c), (a \succ d), (c \succ d)\}.$$

The *symmetric difference* between the two orders is the set of elements that are not common to both,

$$\succ_1 \Delta \succ_2 = (\succ_1 \cup \succ_2) \setminus (\succ_1 \cap \succ_2) = \{(a \succ b), (b \succ a)\}.$$

[3]See Kendall & Gibbons (1990, pp.8-9) for a proof.

The size of that set is the *symmetric difference distance* $|\succ_1 \Delta \succ_2|$, which is 2 in the example.

The Kendall rank correlation coefficient τ is the symmetric difference distance normalized to the interval $[-1, 1]$. A value of 1 signifies the minimal distance possible, corresponding to identical orders, in which case $| \succ_1 \Delta \succ_2 | = 0$. A value of -1 signifies the maximum distance between $\succ_1$ and $\succ_2$.

A total order on Z has

$$\frac{1}{2} \cdot |Z| \cdot (|Z| - 1)$$

elements. For two orders $\succ_1$, $\succ_2$ on Z, the maximum of $| \succ_1 \Delta \succ_2 |$ is therefore

$$|Z| \cdot (|Z| - 1).$$

Used to normalize $| \succ_1 \Delta \succ_2 |$, this yields Kendall's rank correlation coefficient

$$\tau = \frac{\frac{1}{2} \cdot |Z| \cdot (|Z| - 1) - | \succ_1 \Delta \succ_2 |}{\frac{1}{2} \cdot |Z| \cdot (|Z| - 1)} = 1 - \frac{2 \cdot | \succ_1 \Delta \succ_2 |}{|Z| \cdot (|Z| - 1)}.$$

With $| \succ_1 \Delta \succ_2 | = 2$ and $|Z| = 4$ for the example, $\tau = 1 - \frac{2 \cdot 2}{4 \cdot 3} = \frac{2}{3} \approx 0.67$. This positive value close to 1 signifies a relatively strong positive correlation between the two orders. A more defined interpretation of τ can be expressed in probabilistic terms (Salkind, 2006, vol.2, p.508): Suppose choosing two random objects from Z. Let $P(\mathrm{S})$ be the probability that the objects are in the same order in both O_1 and O_2, and $P(\mathrm{D})$ the probability that they are in a different order. Then, τ expressed in those terms is

$$\tau = P(\mathrm{S}) - P(\mathrm{D}).$$

6.2.3 Generalization of Kendall's Tau

Spearman's rho and Kendall's tau compare two rankings or total orders on the same set of objects. For a discussion of their applicability to the unified ranking scenario, recall the definition from section 2.3. The task is to learn a function

$$f : \mathcal{X} \times Z \to S_Z \quad (Z \subseteq \mathcal{Z})$$

from a set of contexts

$$\{\boldsymbol{x}_\ell | l = 1, 2, \ldots, n\} \subseteq \mathcal{X},$$

where each context $\boldsymbol{x}_\ell$ is associated with a binary preference relation

$$\succ_{\boldsymbol{x}_\ell} = \{\boldsymbol{z}_i \succ_{\boldsymbol{x}_\ell} \boldsymbol{z}_j, \ldots\}.$$

That binary preference relation is not necessarily a total order. For evaluation, a dataset is split in training and test data (see section 6.4). A model is built on the training portion, which then predicts an order on the test portion. The predicted order is always total, but the true preference relation, to which the predicted order is compared, may be partial.

For this reason, both Spearman's rho and Kendall's tau cannot be applied to the unified ranking task without modification. But Kendall's tau builds on the notion of similar pairs, and thus can be generalized to fit the purpose at hand.

Let $\succ_1$ be a total order on the set of objects Z and $\succ_p$ be a binary relation on Z. Let $Z_p \subseteq Z$ be the subset of objects used by $\succ_p$, that is, object $\boldsymbol{z}_i \in Z_p$ iff there is a j for which $(\boldsymbol{z}_i \succ \boldsymbol{z}_j) \in \succ_p$ or $(\boldsymbol{z}_j \succ \boldsymbol{z}_i) \in \succ_p$.

Let $C \subseteq \succ_p$ be the subset of pairs in $\succ_p$ that are consistent with $\succ_1$, that is,

$$C = \{(\boldsymbol{z}_i \succ \boldsymbol{z}_j) : (\boldsymbol{z}_i \succ \boldsymbol{z}_j) \in \succ_p \wedge (\boldsymbol{z}_j \succ \boldsymbol{z}_i) \notin \succ_1\}.$$

In the setting of evaluation of unified ranking, the total order $\succ_1$ is predicted by the learned function, and the binary relation or partial order $\succ_p$ is given in the test data for a context.

Note that a partial order is a binary relation that is antisymmetric. In the definition of the unified scenario in section 2.3, it is not a prerequisite that the preference relation attached to a context be antisymmetric.[4] With the definition of C above, if there is a pair of objects $\boldsymbol{z}_i, \boldsymbol{z}_j$ for which both $\boldsymbol{z}_i \succ_p \boldsymbol{z}_j$ and $\boldsymbol{z}_j \succ_p \boldsymbol{z}_i$, then only one of the two preferences will be counted as correct. It could be a another valid approach to define C so that such a pair would be exempted from the measure altogether – in this case, the learning algorithm under test would always be given the "benefit of the doubt" and score higher with inconsistent data.

If C is the subset of true preferences of $\succ_p$ in accordance to the predicted order $\succ_1$, then

$$\frac{|C|}{|\succ_p|}$$

is the ratio of correctly predicted preferences. It is in the interval $[0, 1]$, and normalized to $[-1, 1]$ it yields

$$\tau_p = \frac{2 \cdot |C|}{|\succ_p|} - 1.$$

This measure will be used in the following experiments to determine the performance of prediction of the various algorithms.

To see that it is a generalization of Kendall's tau, consider the special case where $\succ_p = \succ_2$ is a total order. In this case,

$$|\succ_p| = |\succ_2| = \frac{1}{2} \cdot |Z| \cdot (|Z| = 1)$$

and

$$|C| = \frac{1}{2} \cdot |Z| \cdot (|Z| - 1) - \frac{1}{2} \cdot |\succ_1 \Delta \succ_2|$$

[4]A partial order is also transitive, which is not a requirement either. The special case of a partial order is referred to a lot only because it is an illustrative example.

and by substituting into the formula for τ_p above,

$$\begin{aligned}
\tau_p &= \frac{2 \cdot (\frac{1}{2} \cdot |Z| \cdot (|Z|-1) - \frac{1}{2} \cdot |\succ_1 \Delta \succ_2|)}{\frac{1}{2} \cdot |Z| \cdot (|Z|-1)} - 1 \\
&= \frac{2 \cdot |Z| \cdot (|Z|-1)}{|Z| \cdot (|Z|-1)} - \frac{2 \cdot |\succ_1 \Delta \succ_2|}{|Z| \cdot (|Z|-1)} - 1 \\
&= 1 - \frac{2 \cdot |\succ_1 \Delta \succ_2|}{|Z| \cdot (|Z|-1)} && = \tau
\end{aligned}$$

it yields Kendall's tau.

To give an example of the calculation, imagine a predicted order $\succ_1$

$$b \succ_1 a \succ_1 c \succ_1 d$$

and the true preferences to be

$$\succ_p = \{(a \succ b), (a \succ c), (a \succ d), (b \succ c), (c \succ d)\}.$$

Of these, 5 are in accordance with $\succ_1$

$$C = \{(a \succ c), (a \succ d), (b \succ c), (c \succ d)\}.$$

Then,

$$\tau_p = \frac{2 \cdot |C|}{|\succ_p|} - 1 = \frac{2 \cdot 5}{6} - 1 \approx 0.67.$$

6.3 Default Prediction Performance

In multi-label classification, *default accuracy* is the accuracy that is achieved by always predicting the largest class; that is, for each example in the test data, the same class is predicted: that which occurred most frequently in the training data.

The resulting default accuracy can serve as a benchmark for evaluating the performance of a classification algorithm. Clearly, any algorithm worth considering should perform better than this simplistic *default classifier*.

The target domain in the ranking scenario is not a set of classes, but the set of all possible orders. It should be clear that simply predicting the most frequent order will not result in an equivalent default accuracy. In search for a measure that provides a comparable assessment for the ranking scenario, we reconsider the idea of default accuracy to be the highest accuracy achievable by building a predictive model only from the distribution of the dependent variable.

One approach is counting how often a label is preferred over any other label in any training context and order the labels accordingly. The resulting order is the one predicted by the default ranking. This approach was chosen for the computation of a default prediction performance shown in the experimental results in chapter 7.

Another approach would be the construction of a total order for each training context. Given n labels, the greatest label would be awarded $n - 1$ points for the context, the second-greatest $n - 2$ points etc., and the smallest label 0 points. The points would then be added over all contexts, and the resulting order predicted by the default ranking.

6.4 Cross Validation

For classification problems, a widely-used evaluation method is *cross validation*(Witten & Frank, 2005, pp.149-151). In cross validation, the test data is partitioned into a fixed number of portions or *folds*. Given n folds, both learning and classification are performed n times, with the respective fold, one nth of the data, used for testing, and the remainder, $n - 1$ nth of the data, used for training. The results are averaged over the n runs.

Cross validation for the unified ranking scenario works similar. The set of contexts is partitioned into n folds, each of which is used as test data for one run. The remainder of the data is used for training the model. Which subset of objects is used for the training and the test stage depends only on the preferences present in the training/test context subsets.

For the experiments conducted in this work, 10-fold cross validation is used. This choice of n represents a common tradeoff between accuracy of the assessment and computation speed.

6.5 Complexity

In this section, the run-time and memory requirements of the methods introduced in chapter 5 will be discussed. These methods work by decomposition, that is, by reducing the problem to one or more classification problems. This has two major implications for the discussion of the computation time and memory requirements.

First, as the decomposition method relies on an underlying classification algorithm, the execution time and memory requirements of that classification algorithm need to be considered. Without identifying the particular classification algorithm that does the actual learning, time and space complexity cannot be definitely determined, but only be stated depending on the time and space complexity of the classifier.

Secondly, since the decomposed sub-problems can be solved independently, they can be processed in parallel. This can reduce computation time greatly and has thus been identified as one of the major advantages of decomposition in chapter 4.

The two parameters that affect the algorithmic complexity of the object feature coding approach are the number of object attributes n and the number of objects p.

It has been described in chapter 5 how a training example is constructed from each pairwise object comparison for each decomposed learning algorithm; with the reservation that the example can be ignored (resulting in a 0 entry in the coding matrix). Thus, the number of objects poses an upper bound of $p(p-1)$ on the number of training examples that need to be used for training by any of the single induction algorithms.

Note that the number of training instances for the decomposed problems is directly dependent on the number of objects, not on the number of original training instances. Let t_{pref} be the size of the original training data defined as the total number of binary preferences given. Then, the number of original training instances t_{pref} poses an upper bound on the number of objects p such that

$$p \leq 2 \cdot t_{pref}$$

as each training example can contain two objects. In practice, the number of objects may be much smaller, as the same object will likely appear multiple times in the training data.

As it was stated in chapter 2 and chapter 5, the particular scenario that we are interested in assumes a fixed set of objects that is independent in the number of training examples. Thus, for the remaining discusion of algorithmic complexity, the focus lies on the number of object attributes.

The number of object attributes n and the method of coding matrix construction determine the number of single classifiers that have to be learned.

6.5.1 Training Time

In chapter 5, two ranking methods based on object feature coding have been proposed: per-attribute (subsection 5.2.1) and cross-attribute(subsection 5.2.2).

Per-attribute learns as many models as there are object attributes. Thus, if n is the number of object attributes, the time complexity for learning depending on n is $O(n)$.

However, it needs to be considered that through the decomposition, the single problems become smaller on average with respect to the training instances for each sub-problem. This can shorten training time remarkably, especially in cases where a learning algorithm is used where the training time grows faster than linear in the training instances.

Cross-attribute learns $n(n-1)$ functions. This means that computation time grows quadratic in the number of object attributes. The increase compared to the the per-attribute approach is considerable, and it is accepted in the hope that this approach will result in a higher prediction performance.

Again, the sub-problems are smaller in the number of instances, which can mean a lower training time for each individual learner.

Finally, the sub-problems are independent of each other and can be solved in parallel. Assuming a high enough number of processing units, such that there is at least one processing unit per sub-problem, the distributed training time with respect to the number of object attributes is in $O(1)$.

6.5.2 Prediction Time

When predicting an order, each of the sub-models needs to be consulted, resulting in an asymptotic prediction time in $O(n)$ for per-attribute ranking and $O(n^2)$ for cross-attribute ranking.

The reduction of the problem size in terms of training instances is not relevant here. Many classification algorithms have a constant time requirement in the number training instances. If the run-time for prediction depends on the number of training instances at all, as it is the case e.g. with instance-based models, it usually grows sub-linearly.

Especially for per-attribute ranking or other methods within the object feature coding framework that use a similarly high number of sub-models, the prediction time can be considerable when the number of object attributes is high. The expectation is that this is the cost of buying a better prediction.

From these premises, it can be reasoned that the ideal problem for which the utilization of object feature coding should be considered is one where the number of object features is relatively small or can be reduced to a small number, and the prediction quality that has been achieved with different methods is not satisfactory.

6.5.3 Memory Requirements

After learning, all of the sub-models need to be stored: n in the case of per-attribute ranking and $n(n-1)$ in the case of cross-attribute ranking.

This results in an asymptotic memory requirement of $O(n)$ and $O(n^2)$, respectively. The point made in the previous discussion of prediction time is valid mutatis mutandis, i.e. the higher memory use is expected to provide for a more accurate prediction.

Again, it is worth pointing out the independence of each decomposed problem. This means, even if the total amount of memory needed may be huge for large problems, each individual classification function only needs access to the data of its own model. This allows for an architecture which can distribute the problem efficiently among different machines.

Chapter 7

Experiments

This chapter details the setup and results of the conducted experiments. The fundamental basis for experimental evaluation has been laid by the previous chapter 6; in the following there will additionally be more specific introductions to the particular experiment, as each experiment is supposed to illuminate a different aspect of the problem and of the ranking methods under test.

Thus, the experiments are grouped into three sections: prediction performance on real-world and synthetic data (section 7.1), effects of varying correlation strength between context and object features (section 7.2), and effects of noisy data (section 7.3).

All of the ranking methods described in this work reduce the ranking task to one or more standard classification tasks. This means that any classification algorithm can be substituted to do the actual machine learning. For the experiments described in this chapter, the *C4.5* decision tree generator introduced by Quinlan (1993) has been used.

7.1 Prediction performance

This experiment measures the accuracy of the ranking algorithms over three data sets: sushi A, sushi B, and the generated data (see section 6.1 for a detailed description of these data sets).

The results are depicted in Figure 7.1 and the corresponding Table 7.1. The figure is a grouped bar chart of the prediction performance of five ranking methods. Each data set is represented by a group: The group to the left contains the results for the sushi A data set, the middle group for the sushi B data set, and the group to the right contains the results for the generated data. The ranking method is indicated by the pattern and color of the particular bar.

For each group, the two leftmost bars, marked with a grid pattern, represent the non-decomposed methods of baseline ranking and label ranking. The two rightmost bars, marked with a stripe pattern, represent the decomposed methods of cross attribute ranking and classifier-per-attribute ranking. The solid bar in the middle represents default ranking.

As a first observation it is to be noted that the decomposed methods show better results than the non-decomposed methods over all three data sets. These results provide a positive answer to the initially posed question whether decomposition approaches could increase the quality of prediction.

Of the two decomposed methods, the cross attributes methods beats the classifier-per-attribute method in two of the three cases. This result is not unexpected, since the cross-attribute method was introduced as an alternative to the classifier-per-attribute method and was supposed to fix certain shortcomings of it. The price for this increase in prediction performance has been pointed out in section 6.5: a higher computational complexity.

Of the non-decomposed methods, baseline ranking achieves better results in two of the three cases, indicating a positive answer to the initial question

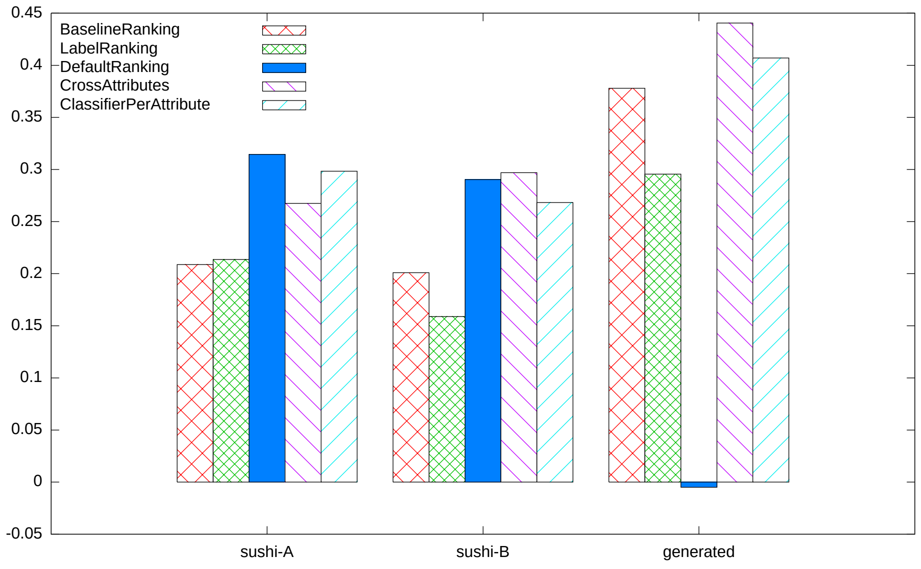


Figure 7.1: τ_p across different datasets

	BaselineRanking	LabelRanking	DefaultRanking	CrossAttributes	ClassifierPerAttribute
sushi-A	0.21	0.21	0.31	0.27	0.30
sushi-B	0.20	0.16	0.29	0.30	0.27
generated	0.38	0.30	−0.00	0.44	0.41

Table 7.1: τ_p across different datasets

whether taking into account the object features – even without decomposition – could improve prediction accuracy over label ranking.

The results for sushi A and sushi B are relatively similar, while results for the generated data differ from these. The most noticeable difference between the generated data and the sushi data is the difference in the default ranking. The default ranking measures the default prediction performance. As described in section 6.3, this is a measure for the distribution of preference among the different objects, independent from a specific context. The employed method of data generation has distributed preference equally over all objects, while the experiments show that there are sushi items that are more universally preferred or put simply, more popular.

Recall that the default ranking, in analogy to *default accuracy* in classification, was introduced as a way of providing a minimum level of predictive performance that every good ranking method should outdo. Usually, a high default prediction performance means the dataset should be easier to learn, as some of the objects are universally (over all contexts) rated higher than others. In this respect, the results for the sushi data put the quality of the other ranking methods into perspective: While the default ranking is on par with the two decomposed methods, it outperforms the non-decomposed methods by a clear margin.

7.2 Number of relevant attributes

Label ranking only represents objects as nominal labels, without being concerned with their features. The decomposed methods introduced in this work, on the other hand, use the object features as an information to define the way they decompose the problem. To put it differently, label ranking correlates the context features only with a label, while the decomposed methods correlate context features with object features. What if object features were chosen that show no correlation to the context features?

Figure 7.2 and Table 7.2 show the results of an according experiment. The test data is generated as described in section 6.1. Then, attributes are gradually replaced with new attributes with random values. In other words, for each instance, each value for the particular attribute is assigned a new random value unrelated to anything. Is is done for one attribute after another, with the experiment repeated between each attribute substitution. In the end, all of the object attributes are completely random without any correlation to the context attributes. The x-axis of Figure 7.2 shows the number of correlated or relevant attributes. The curves represent the prediction performance of the five ranking methods.

As expected, the performance of the decomposed predictors drops. The drop of the per-attribute method is somewhat less dramatic than the drop of the cross-attributes method. Here, the more simplistic per-attribute method seems to be slightly more stable against this kind of noise in the data.

Label ranking is not at all affected by the change in attribute values. When recalling the way the method works, it is easy to explain why: Label ranking strips the attributes from the objects so that they seem like nominal labels. Thus, the problem presents itself as identical for each iteration from the label ranking perspective.

The baseline method appears stable. Even with completely random object attributes, it is able to predict a ranking no worse than with the original data. This is somewhat surprising; the expectation was that, with random object attributes, the prediction performance should drop to the levels of label ranking.

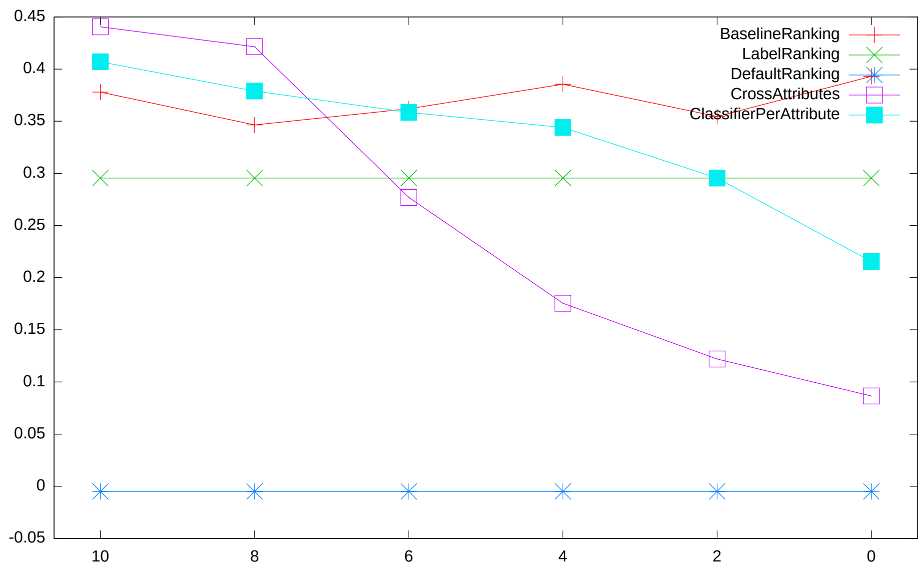


Figure 7.2: τ_P against number of relevant attributes

	BaselineRanking	LabelRanking	DefaultRanking	CrossAttributes	ClassifierPerAttribute
10	0.38	0.30	-0.00	0.44	0.41
8	0.35	0.30	-0.00	0.42	0.38
6	0.36	0.30	-0.00	0.28	0.36
4	0.39	0.30	-0.00	0.18	0.34
2	0.35	0.30	-0.00	0.12	0.30
0	0.39	0.30	-0.00	0.09	0.22

Table 7.2: τ_P against number of relevant attributes

7.3 Noise

To test how the different ranking methods handle noisy data, different kinds of noise are introduced to the generated data, and the methods are tested on it.

7.3.1 Rank Noise

For this experiment, the generated dataset is used, the generation of which has been described in section 6.1. To reiterate, a fixed number of context and object items are initialized with random values. An $m \times n$ matrix M with random entries governs the rating of each object in each context. A rating is a value in $\mathbb{R}$, which is calculated by the following function:

$$r(\boldsymbol{x}, \boldsymbol{z}) = \nu \cdot N_{\boldsymbol{x},\boldsymbol{z}} + (\nu - 1) \cdot \sum_{i=1}^{m} \sum_{j=1}^{n} M_{i,j} \cdot \boldsymbol{x}(i) \cdot \boldsymbol{z}(j).$$

After each object has been rated, an order is established according to the rating. Then, 40 binary preferences of the form $\boldsymbol{z}_1 \succ \boldsymbol{z}_2$ are randomly picked for each context and put into the data set.

In the formula above, $N_{\boldsymbol{x},\boldsymbol{z}}$ represents a random noise component for the rating of object $\boldsymbol{z}$ in context $\boldsymbol{x}$, and $\nu, 0 \leq \nu \leq 1$ the ratio of noise. For the present experiment, ν is gradually increased.

The results are shown Figure 7.3 and Table 7.3. The x-axis shows the amount of noise, that is ν, in percent. It can be noticed that the effects only begin to show with a great amount of noise, from 80 percent onward. This means that only a high amount of noise in the rating has an effect on the actual ranking. It can be noted that it would probably be a a better method for ranking noise generation to modify the ranking itself, e.g. by inverting an amount of randomly selected preferences in the data.

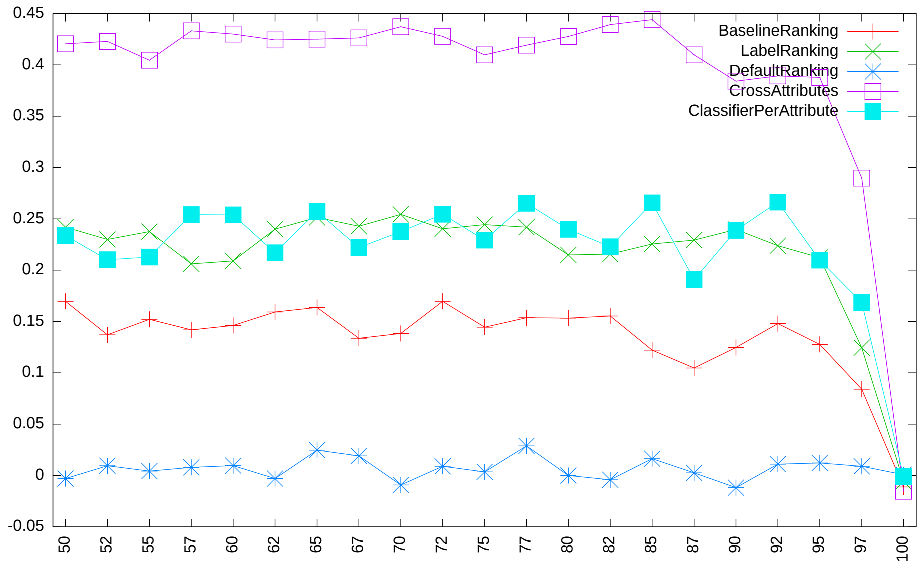


Figure 7.3: τ_P against rank noise [%]

	BaselineRanking	LabelRanking	DefaultRanking	CrossAttributes	ClassifierPerAttribute
50	0.17	0.24	−0.00	0.42	0.23
52	0.14	0.23	0.01	0.42	0.21
55	0.15	0.24	0.00	0.40	0.21
57	0.14	0.21	0.01	0.43	0.25
60	0.15	0.21	0.01	0.43	0.25
62	0.16	0.24	−0.00	0.42	0.22
65	0.16	0.25	0.02	0.42	0.26
67	0.13	0.24	0.02	0.43	0.22
70	0.14	0.25	−0.01	0.44	0.24
72	0.17	0.24	0.01	0.43	0.25
75	0.14	0.24	0.00	0.41	0.23
77	0.15	0.24	0.03	0.42	0.27
80	0.15	0.21	−0.00	0.43	0.24
82	0.16	0.22	−0.00	0.44	0.22
85	0.12	0.23	0.02	0.44	0.27
87	0.10	0.23	0.00	0.41	0.19
90	0.12	0.24	−0.01	0.38	0.24
92	0.15	0.22	0.01	0.39	0.27
95	0.13	0.21	0.01	0.39	0.21
97	0.08	0.12	0.01	0.29	0.17
100	−0.01	−0.01	0.00	−0.02	−0.00

Table 7.3: τ_P against rank noise [%]

From the results can be seen that the quality of prediction decreases for all methods; the decrease is roughly equal among all methods, and there is no one that performs obviously better or worse than the other.

7.3.2 Object Feature Noise

Figure 7.4 and Table 7.4 show the results of an experiment where noise was added to the object features. That means, after data generation, an amount of noise was added to each object feature value. This amount of noise was gradually increased, until the object features are completely random and have no correlation to the context features anymore.

The x-axis of the graph shows the amount of noise that was added to the object features. An inspection of the results for the various ranking methods reveals that label ranking and the default ranking remain constant, while all other methods decline. Just as in the relevant-attributes experiment in section 7.2, neither label ranking nor the default ranking are influenced by a change in object attributes.

While default ranking is not considered to be a real ranking method and was only introduced as a benchmark, it is still interesting to observe how it behaves under the different types of introduced noise. Noise introduced to the correlation between context and object features, as in this and in the following experiment, does not affect it at all; it scores exactly the same constant prediction performance. Noise introduced to the ranking, as in the relevant-attributes experiment, does affect it. In that experiment, the preferences change randomly. Default ranking uses the overall preference for an object independent from a context for prediction. For that reason, the curve for default ranking shows a random jitter without showing a trend.

The baseline ranking method's results are also relatively constant. In con-

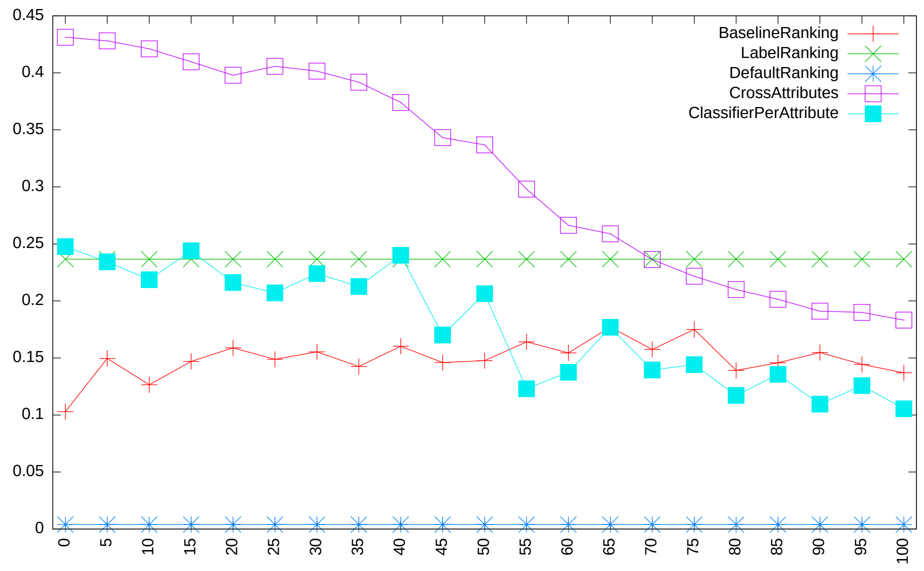


Figure 7.4: τ_P against object feature noise [%]

	BaselineRanking	LabelRanking	DefaultRanking	CrossAttributes	ClassifierPerAttribute
0	0.10	0.24	0.00	0.43	0.25
5	0.15	0.24	0.00	0.43	0.23
10	0.13	0.24	0.00	0.42	0.22
15	0.15	0.24	0.00	0.41	0.24
20	0.16	0.24	0.00	0.40	0.22
25	0.15	0.24	0.00	0.41	0.21
30	0.16	0.24	0.00	0.40	0.22
35	0.14	0.24	0.00	0.39	0.21
40	0.16	0.24	0.00	0.37	0.24
45	0.15	0.24	0.00	0.34	0.17
50	0.15	0.24	0.00	0.34	0.21
55	0.16	0.24	0.00	0.30	0.12
60	0.15	0.24	0.00	0.27	0.14
65	0.18	0.24	0.00	0.26	0.18
70	0.16	0.24	0.00	0.24	0.14
75	0.18	0.24	0.00	0.22	0.14
80	0.14	0.24	0.00	0.21	0.12
85	0.15	0.24	0.00	0.20	0.14
90	0.15	0.24	0.00	0.19	0.11
95	0.14	0.24	0.00	0.19	0.13
100	0.14	0.24	0.00	0.18	0.11

Table 7.4: τ_P against object feature noise [%]

trast, the object feature coding methods of cross-attribute and classifier-per-attribute show a clear decrease in prediction quality, with cross-attribute ranking staying well above classifier-per-attribute ranking. Similarly to the relevant-attributes experiment in section 7.2, the results show that an approach like object feature coding, which utilizes the correspondence between context features and object features exclusively, relies on the presence of such a correspondence. A proposal to work around this shortcoming is given in section 8.1.

7.3.3 Context Feature Noise

The final experiment adds noise to the values of the context attributes. Figure 7.5 and Table 7.5 show the results. The x-axis of the graph shows the noise ratio in percent.

The decrease in prediction performance is clearly visible. All methods seem to be equally affected; the two decomposed methods show the best results, with the cross-attribute approach on top of all others.

In contrast to the previous experiment, label ranking also sees a loss in prediction performance.

7.4 Summary

The experiments show that the decomposition approaches can increase the accuracy of rank prediction in comparison to both label ranking and non-decomposed methods which take into account context and object features.

It becomes apparent, however, that the decomposition approach depends especially on a good modeling of the object features. Otherwise, even label ranking can outperform the decomposition approach.

It would be another interesting experiment to model a label ranking problem

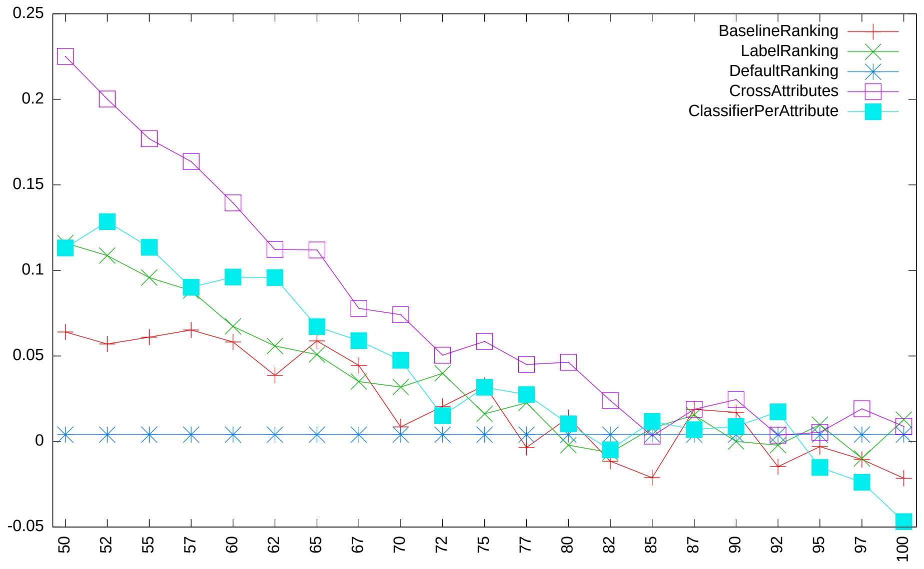


Figure 7.5: τ_P against context feature noise [%]

	BaselineRanking	LabelRanking	DefaultRanking	CrossAttributes	ClassifierPerAttribute
50	0.06	0.12	0.00	0.23	0.11
52	0.06	0.11	0.00	0.20	0.13
55	0.06	0.10	0.00	0.18	0.11
57	0.07	0.09	0.00	0.16	0.09
60	0.06	0.07	0.00	0.14	0.10
62	0.04	0.06	0.00	0.11	0.10
65	0.06	0.05	0.00	0.11	0.07
67	0.04	0.04	0.00	0.08	0.06
70	0.01	0.03	0.00	0.07	0.05
72	0.02	0.04	0.00	0.05	0.02
75	0.03	0.02	0.00	0.06	0.03
77	−0.00	0.02	0.00	0.05	0.03
80	0.01	−0.00	0.00	0.05	0.01
82	−0.01	−0.01	0.00	0.02	−0.00
85	−0.02	0.01	0.00	0.00	0.01
87	0.02	0.02	0.00	0.02	0.01
90	0.02	0.00	0.00	0.02	0.01
92	−0.01	−0.00	0.00	0.00	0.02
95	−0.00	0.01	0.00	0.01	−0.02
97	−0.01	−0.01	0.00	0.02	−0.02
100	−0.02	0.01	0.00	0.01	−0.05

Table 7.5: τ_P against context feature noise [%]

for the decomposition methods, with one object feature per label, which has a positive value if the object has the corresponding label and a negative value otherwise. The results could then be compared to label ranking.

Chapter 8

Conclusion

The objective of the present work was to find and evaluate an approach that unifies the known scenarios of object ranking and label ranking.

The unified ranking scenario was defined as a generalization of both label ranking and object ranking. After a general consideration of the common approaches to ranking problems, several basic methods have been selected and described in order to provide a benchmark for evaluation.

A technique well-known in multi-class classification, the decomposition of a problem by means of a coding matrix, has been transferred to the ranking scenario. In order to demonstrate the versatility of the technique, the pairwise label ranking method has been expressed in terms of a coding matrix. Then, an approach named object feature coding has been proposed, which specifies a coding matrix through a function of the features of the objects that should be ranked.

This technique is a method that utilizes decomposition. This means, the method decomposes the problem into multiple smaller sub-problems, which are then solved and their solutions combined to form a solution to the original problem. For this reason, the general properties of the decomposition of data mining tasks have been discussed; two benefits of decomposition have been identified that were expected to apply to the proposed methods, and thus were expected to become visible in the further evaluation. The first is the ability to distribute computation among different processing units, thus allowing for a faster execution. This benefit has been evaluated in the course of the analysis of algorithmic complexity, and it could be shown that the proposed ranking methods could indeed be distributed among multiple computers, since the single sub-problems are independent of each other.

The second expected benefit was an increase in prediction performance.

The described methods were implemented and subsequently tested on real-word data as well as on generated data. First, a comparison between label ranking and the baseline ranking method for the unified scenario has shown that the information of object features can indeed improve prediction accuracy.

Secondly, the it could be shown that the introduced decomposition approaches can further increase performance.

Nonetheless, it could be demonstrated that the introduced decomposition method also has weaknesses when the object features do not show a high correlation to the context features. In extreme cases, even label ranking performed better. An approach was suggested that could alleviate or solve this problem by modeling object id in the object features.

Finally, it is to be said that the two methods of per-attribute ranking and cross-attribute ranking that were implemented and tested within the object feature coding framework are only two examples of a multitude of many possible methods and parametrizations.

8.1 Future Work

8.1.1 Data

The only real-world data used for evaluation is the sushi dataset (see section 6.1). This is largely due to the reason that data fit for the specific scenario is much harder to find than data for standard data mining scenarios, and surveying new data is outside the scope of this work.

Ideally, there should be several real-world datasets on which the ranking methods are tested to gain a sound evaluation. It remains an open task to find such data and evaluate the proposed methods on them.

Likewise, the automatically generated data used in the present work is generated ad-hoc. A more sophisticated data generation approach would deliver more general results. But it is obvious that a good method for data generation should generalize from the regularities found in real-world data. Thus, real-world data might be needed either way.

8.1.2 Experiments

In cases where there is no correlation between context and object features, it could be helpful to augment objects with features that identify or label them. One such experiment is sketched in section 7.4.

8.1.3 Margin-based Decoding

Allwein *et al.* (2001) introduce a method of decomposing multi-class learning tasks into multiple binary learning tasks. This method takes into account the *margin*, which is an inherent property of the classification of many binary classification algrorithms such as support-vector machines, AdaBoost, regression, logistic regression and decision-tree algorithms. It is a real value whose magnitude can be interpreted to be a measure of confidence for the particular decision. Allwein et al. show how the learning of the mentioned algorithms minimize a loss function on the margin of the training examples, and go on to prove how the loss of the individual binary classifiers bounds the loss of the original problem.

A similar approach could be used with the algorithm described in this work, where only the binary predictions of the single classifiers are taken into account. Such an approach is expected to increase prediction performance.

Appendix A

Implementation

The ranking methods to be tested are implemented in Java. Figure A.1 shows a class diagram of the ranking methods. As can be seen, `LabelRanking`, `DefaultRanking` and `ClassifierPerAttribute` inherit from `AbstractLabelRanking`. This is the superclass for all methods that need to know the complete set of objects at training time, as described in section 2.3.[1] `BaselineRanking`, on the other hand, can make predictions for previously unseen objects. The `AbstractLabelRanking` class provides facilities for establishing object identity between objects of the training and test sets, which are utilized by the subclasses.

The Weka toolkit(Hall *et al.* , 2009) provides a library for standard machine learning tasks such as classification and regression. As described in chapter 3, the ranking task is reduced to one or more classification tasks. In the implementation, classification is performed by a Weka learner. By programming against the interface provided by Weka, any of the pre-implemented machine

[1]Although the general object feature decoding approach can rank unseen objects, this is not part of the experiments and has therefore not been implemented.

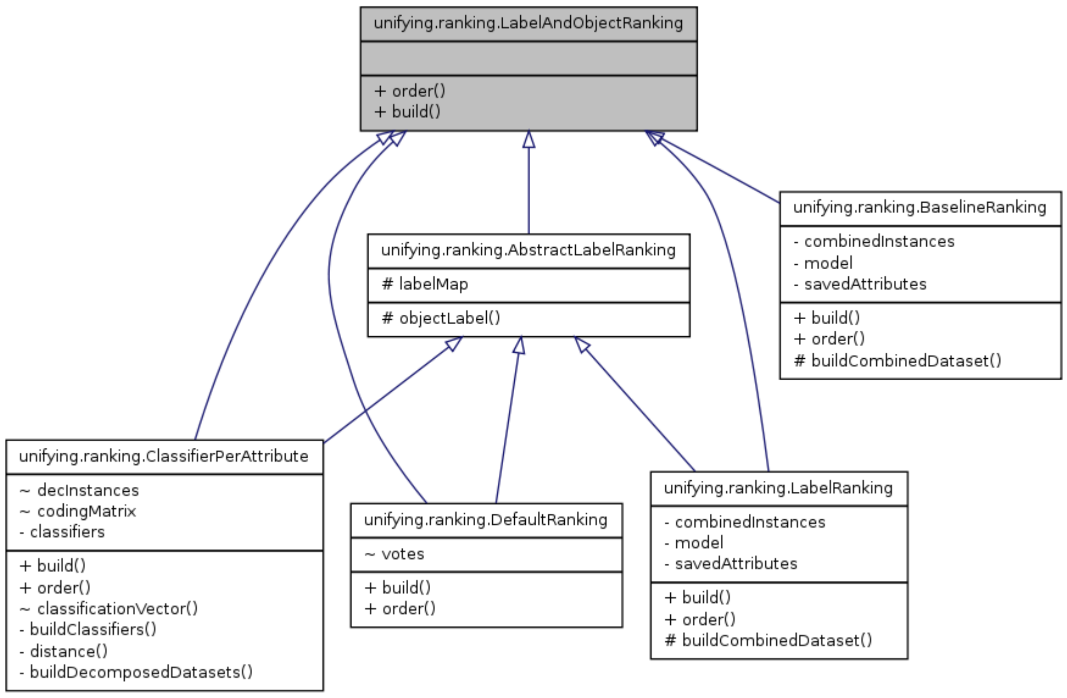


Figure A.1: Class diagram of the ranking methods

learning algorithms can be used. The actual algorithm that was used for the experiments described in chapter 7 is the *C4.5* decision tree generator introduced by Quinlan (1993), which is implemented in Weka under the name of `J48`.

A.1 Data File Format

A data format for the unified ranking scenario must store two relations, one for the context set and one for the object set. Additionally, each member of the context set must be associated with a set of pairwise object preferences.

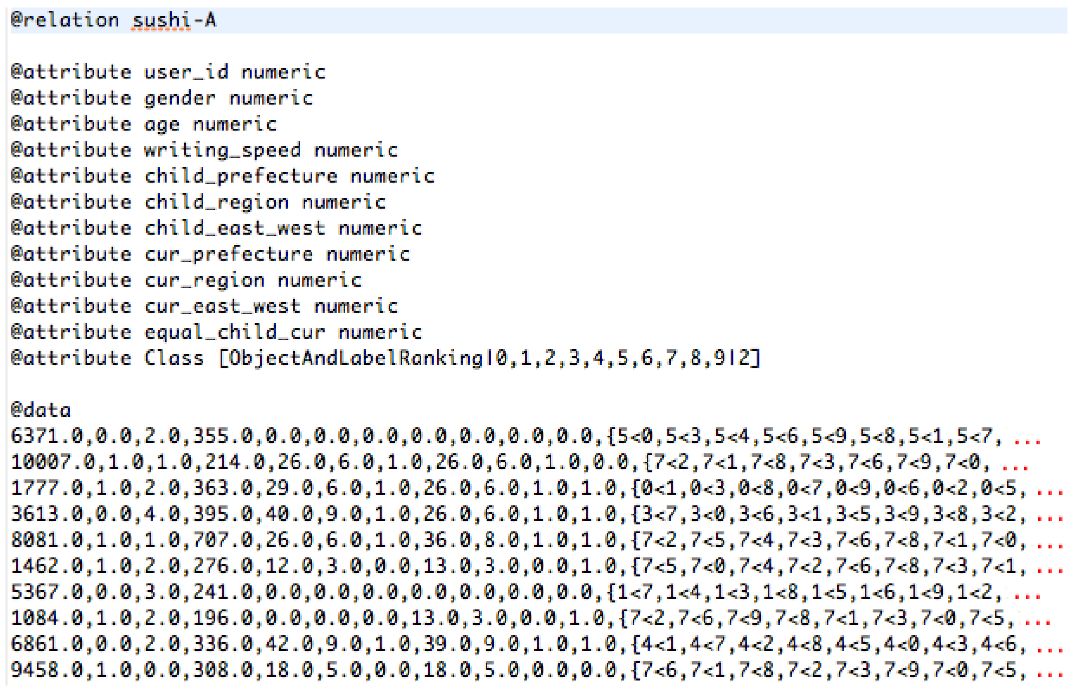
The LPCforSOS (Learning by Pairwise Comparison for Problems with Structured Output Spaces) framework[2] developed in cooperation between the Knowledge Engineering Group [3] and the Knowledge Engineering & Bioinformatics Lab of Philipps-Universität Marburg[4] aims at learning to predict structures such as orders through pairwise comparisons.

LPCforSOS extends the ARFF (Attribute-Relation File Format) used by Weka. In ARFF, every instance of the data set is associated with a single class, as it fits the standard classification problem. The EARFF (Extended ARFF) used by LPCforSOS allows for the association of multiple classes with an instance. By using preferences in the place of classes, it is possible to attach a preference relation to each instance. Note that there are no restrictions on the preference relation such as antisymmetry, meaning it may be $\boldsymbol{z}_1 \succ \boldsymbol{z}_2$ and $\boldsymbol{z}_2 \succ \boldsymbol{z}_1$ for $\boldsymbol{z}_1 \neq \boldsymbol{z}_2$. This is intended and in line with the problem formulation in chapter 2.

[2]`http://www.ke.tu-darmstadt.de/projects/lpcforsos`

[3]`http://www.ke.tu-darmstadt.de/`

[4]`http://www.uni-marburg.de/fb12/kebi/`

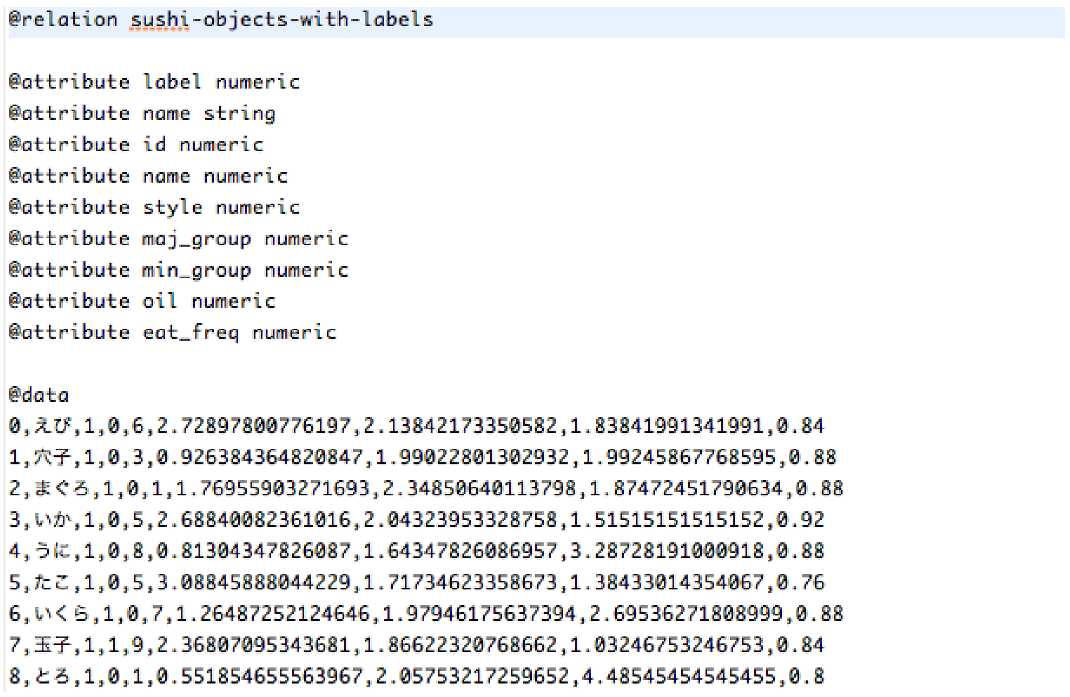


```
@relation sushi-A

@attribute user_id numeric
@attribute gender numeric
@attribute age numeric
@attribute writing_speed numeric
@attribute child_prefecture numeric
@attribute child_region numeric
@attribute child_east_west numeric
@attribute cur_prefecture numeric
@attribute cur_region numeric
@attribute cur_east_west numeric
@attribute equal_child_cur numeric
@attribute Class [ObjectAndLabelRanking|0,1,2,3,4,5,6,7,8,9|2]

@data
6371.0,0.0,2.0,355.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,{5<0,5<3,5<4,5<6,5<9,5<8,5<1,5<7, ...
10007.0,1.0,1.0,214.0,26.0,6.0,1.0,26.0,6.0,1.0,0.0,{7<2,7<1,7<8,7<3,7<6,7<9,7<0, ...
1777.0,1.0,2.0,363.0,29.0,6.0,1.0,26.0,6.0,1.0,1.0,{0<1,0<3,0<8,0<7,0<9,0<6,0<2,0<5, ...
3613.0,0.0,4.0,395.0,40.0,9.0,1.0,26.0,6.0,1.0,1.0,{3<7,3<0,3<6,3<1,3<5,3<9,3<8,3<2, ...
8081.0,1.0,1.0,707.0,26.0,6.0,1.0,36.0,8.0,1.0,1.0,{7<2,7<5,7<4,7<3,7<6,7<8,7<1,7<0, ...
1462.0,1.0,2.0,276.0,12.0,3.0,0.0,13.0,3.0,0.0,1.0,{7<5,7<0,7<4,7<2,7<6,7<8,7<3,7<1, ...
5367.0,0.0,3.0,241.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,{1<7,1<4,1<3,1<8,1<5,1<6,1<9,1<2, ...
1084.0,1.0,2.0,196.0,0.0,0.0,0.0,13.0,3.0,0.0,1.0,{7<2,7<6,7<9,7<8,7<1,7<3,7<0,7<5, ...
6861.0,0.0,2.0,336.0,42.0,9.0,1.0,39.0,9.0,1.0,1.0,{4<1,4<7,4<2,4<8,4<5,4<0,4<3,4<6, ...
9458.0,1.0,0.0,308.0,18.0,5.0,0.0,18.0,5.0,0.0,0.0,{7<6,7<1,7<8,7<2,7<3,7<9,7<0,7<5, ...
```

Figure A.2: Context File Format



```
@relation sushi-objects-with-labels

@attribute label numeric
@attribute name string
@attribute id numeric
@attribute name numeric
@attribute style numeric
@attribute maj_group numeric
@attribute min_group numeric
@attribute oil numeric
@attribute eat_freq numeric

@data
0,えび,1,0,6,2.72897800776197,2.13842173350582,1.83841991341991,0.84
1,穴子,1,0,3,0.926384364820847,1.99022801302932,1.99245867768595,0.88
2,まぐろ,1,0,1,1.76955903271693,2.34850640113798,1.87472451790634,0.88
3,いか,1,0,5,2.68840082361016,2.04323953328758,1.51515151515152,0.92
4,うに,1,0,8,0.81304347826087,1.64347826086957,3.28728191000918,0.88
5,たこ,1,0,5,3.08845888044229,1.71734623358673,1.38433014354067,0.76
6,いくら,1,0,7,1.26487252124646,1.97946175637394,2.69536271808999,0.88
7,玉子,1,1,9,2.36807095343681,1.86622320768662,1.03246753246753,0.84
8,とろ,1,0,1,0.551854655563967,2.05753217259652,4.48545454545455,0.8
```

Figure A.3: Object File Format

The implementation uses an EARFF file for the set of contexts. A sample can be seen in Figure A.2. In the `@data` section at the bottom, the preferences can be seen at the right, surrounded by curly braces. Although in this example the objects are denoted by numerals, generally they can be denoted by any label. The labels are specified in the `@Class` attribute, and their order corresponds to the order of object instances in the object data file. Figure A.3 shows the corresponding object data. It is specified in the ARFF format.

The original raw data was converted into the target format with the use of some Python scripts.

Bibliography

ALLWEIN, ERIN L., SCHAPIRE, ROBERT E., & SINGER, YORAM. 2001. Reducing multiclass to binary: a unifying approach for margin classifiers. *Journal of Machine Learning Research*, **1**, 113–141.

BUNTINE, WRAY. 1996. Graphical models for discovering knowledge. *Advances in knowledge discovery and data mining*, 59–82.

DIETTERICH, THOMAS G., & BAKIRI, GHULUM. 1995. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, **2**, 263–286.

FÜRNKRANZ, JOHANNES, & HÜLLERMEIER, EYKE (eds). 2010a. *Preference Learning.* Springer-Verlag.

FÜRNKRANZ, JOHANNES, & HÜLLERMEIER, EYKE. 2010b. Preference Learning: An Introduction. *In:* Fürnkranz & Hüllermeier (2010a). To appear.

FÜRNKRANZ, JOHANNES, & HÜLLERMEIER, EYKE. 2010c. Preference Learning and Ranking by Pairwise Comparison. *In:* Fürnkranz & Hüllermeier (2010a). To appear.

HALL, MARK, FRANK, EIBE, HOLMES, GEOFFREY, PFAHRINGER, BERNHARD, REUTEMANN, PETER, & WITTEN, IAN H. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, **11**(1).

Hüllermeier, Eyke, Fürnkranz, Johannes, Cheng, Weiwei, & Brinker, Klaus. 2008. Label ranking by learning pairwise preferences. *Artificial Intelligence*, **172**(16-17), 1897 – 1916.

Kamishima, Toshihiro. 2003. Nantonac Collaborative Filtering: Recommendation Based on Order Responses. *Pages 583–588 of: KDD '03: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* New York: ACM.

Kamishima, Toshihiro, Kazawa, Hideto, & Akaho, Shotaro. 2005. Supervised Ordering — An Empirical Survey. *IEEE International Conference on Data Mining*, **0**, 673–676.

Kendall, Maurice, & Gibbons, Jean D. 1990. *Rank Correlation Methods.* 5 edn. London: Arnold.

Maimon, Oded Z., & Rokach, Lior. 2005. *Decomposition Methodology For Knowledge Discovery And Data Mining: Theory And Applications (Machine Perception and Artificial Intelligence).* World Scientific Publishing Company.

Quinlan, J. Ross. 1993. *C4.5: programs for machine learning.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Salkind, Neil (ed). 2006. *Encyclopedia of Measurement and Statistics.* 1 edn. Thousand Oaks: Sage Publications.

Spearman, Charles. 1904. The Proof and Measurement of Association between Two Things. *The American Journal of Psychology*, **15**, 72–101.

Witten, Ian H., & Frank, Eibe. 2005. *Data Mining: Practical Machine Learning Tools and Techniques.* 2 edn. San Francisco: Morgan Kaufmann.