Binary Decomposition Methods for Multipartite Ranking

Johannes Fürnkranz[1], Eyke Hüllermeier[2], and Stijn Vanderlooy[3]

[1] Department of Computer Science, TU Darmstadt
juffi@ke.tu-darmstadt.de
[2] Department of Mathematics and Computer Science, Marburg University
eyke@mathematik.uni-marburg.de
[3] Department of Knowledge Engineering, Maastricht University
s.vanderlooy@maastrichtuniversity.nl

Abstract. Bipartite ranking refers to the problem of learning a ranking function from a training set of positively and negatively labeled examples. Applied to a set of unlabeled instances, a ranking function is expected to establish a total order in which positive instances precede negative ones. The performance of a ranking function is typically measured in terms of the AUC. In this paper, we study the problem of multipartite ranking, an extension of bipartite ranking to the multi-class case. In this regard, we discuss extensions of the AUC metric which are suitable as evaluation criteria for multipartite rankings. Moreover, to learn multipartite ranking functions, we propose methods on the basis of binary decomposition techniques that have previously been used for multi-class and ordinal classification. We compare these methods both analytically and experimentally, not only against each other but also to existing methods applicable to the same problem.

1 Introduction

There are several connections between "learning to rank", a topic of increasing interest in machine learning research, and conventional classifier learning. First, some ranking problems such as *label ranking* [16] can be seen as direct extensions of (multi-class) classification. Second, many learning methods for ranking essentially reduce the original problem to a standard classification problem. Third, aspects of ranking and sorting are also of interest for classification itself. A notable case is ROC analysis, which evaluates the ability of classifiers to sort positive and negative instances in terms of the area under the ROC curve, abbreviated as AUC [7]. The problem to learn classifiers with high AUC (instead of low error rate) is called *bipartite ranking.*

In this paper, we are interested in extending bipartite ranking from the binary to the multi-class case. This problem, that we shall refer to as *multipartite ranking*, is closely related to *ordinal classification*, that is, classification with a totally ordered set of classes. Yet, just like in bipartite ranking, the goal is not to learn a good classifier, but a good ranker, that is, a function that systematically ranks "high" classes ahead of "low" classes. Note that a ranking is in a sense a refinement of the order information provided by an ordinal classifier, as the latter does not distinguish between objects within the same category. As an example, compare the task of two reviewers: one has to make an ordinal prediction for each paper (reject, weak reject, weak accept, accept), and the other has to rank the papers according to quality. Obviously, the latter approach provides more detailed information that can be used not only for partitioning papers into the above four categories.

To solve the multipartite ranking problem, we explore the use of binary decomposition techniques. Such techniques have already been applied quite successfully in conventional classification, ordinal classification, and label ranking [1,8,16], but have not yet been explored in prior work on multipartite ranking [20,22]. Our main result is that, compared to existing methods applicable for the multipartite ranking problem, binary decomposition techniques are at least competitive in terms of predictive accuracy, and presumably even superior, while being computationally much more efficient.

In the next section, we introduce the multipartite ranking problem, discuss its relation to bipartite ranking and ordinal classification, and also address the question of how to evaluate multipartite ranking functions. In Section 3, we propose two methods for multipartite ranking which are based on binary decomposition techniques. Section 4 is devoted to an experimental analysis of these methods. The paper ends with some concluding remarks in Section 5.

2 Ordinal Classification and Multipartite Ranking

In this section, we introduce the problem of multipartite ranking and discuss corresponding performance metrics. Beforehand, we recall the related problems of ordinal classification and bipartite ranking.

2.1 Ordinal Classification

In ordinal classification, also called ordinal regression in statistics, the set of class labels $\mathcal{L} = \{\lambda_1, \lambda_2 \ldots \lambda_m\}$ is endowed with a natural (total) order relation $\lambda_1 \prec \lambda_2 \prec \cdots \prec \lambda_m$. This distinguishes ordinal from conventional classification, where $\mathcal{L}$ is an unordered set.

From a learning point of view, the ordinal structure of $\mathcal{L}$ is *additional* information that a learner should try to exploit, and this is what existing methods for ordinal classification essentially seek to do [8,4]. In fact, the problem of ordinal classification is in a sense in-between classification and regression, two problems that have been extensively studied. Like in classification, the output space is finite, and like in regression, the elements of this space are ordered.

Thus, it is hardly surprising that both classification and regression algorithms have been used to tackle ordinal classification problems, even though both approaches are obviously problematic: A simple classification method will neglect information about the class order, whereas a regression method will make too strong assumptions, because it does not only exploit the order relation but assumes meaningful distances between output values [19]. To avoid these problems, new algorithms have been developed in the machine learning field in recent years, which are able to exploit class order information in a meaningful way [13,8,5,4].

2.2 Bipartite Ranking

In the problem of *bipartite ranking*, training data consists of a set of positively and negatively labeled instances, just like in conventional binary classification. However, instead of learning a classifier that can be used for assigning instances to one of the two classes, the goal is to learn a ranking function $f(\cdot)$ that can be used for ordering a set of instances from most likely positive to most likely negative. Thus, given a set X of instances with unknown class labels, the learned ranking function outputs a linear order of these instances. Typically, this is accomplished by *scoring* the instances, i.e., $f(\cdot)$ is implemented as an $\mathbb{X} \to \mathbb{R}$ mapping that assigns a real-valued score $f(\boldsymbol{x})$ to each instance $\boldsymbol{x}$ from an instance space $\mathbb{X}$. The instances are then ranked according to their respective scores.

The most commonly used metric to evaluate a predicted ranking is the area under the ROC curve (AUC) [7]:

$$\mathrm{AUC}(f, X) = \frac{1}{|P||N|} \sum_{\boldsymbol{x} \in P} \sum_{\boldsymbol{x}' \in N} S(f(\boldsymbol{x}'), f(\boldsymbol{x})) \ , \tag{1}$$

where $P \subset X$ and $N \subset X$ are the positive and negative instances in X (hence $X = P \cup N$, $P \cap N = \emptyset$). The mapping $S(\cdot, \cdot)$ outputs 1 when the positive instance is ranked before the negative one, and 0 in the reverse case. An output of 1/2 is given when the instances are assigned the same score.

2.3 Multipartite Ranking

Given that class labels are ordered, as in ordinal classification, the idea of bipartite ranking can obviously be generalized from the binary to the multi-class case. Given a set of instances X with class labels in $\mathcal{L} = \{\lambda_1, \lambda_2 \dots \lambda_m\}$, the goal is to order them in such a way that, ideally, the instances from λ_m precede those from λ_{m-1}, which in turn precede those from class λ_{m-2}, etc. Subsequently, we shall refer to the problem of learning a corresponding ranking function from a training set $T = \{(\boldsymbol{x}_i, \ell_{\boldsymbol{x}_i})\}_{i=1}^n \subset \mathbb{X} \times \mathcal{L}$ of labeled instances as *multipartite ranking*.

A common approach to learning the scoring function f consists of turning the original training data into a set of *order constraints* on $f(\cdot)$, and then finding a function that is as much as possible in agreement with these constraints. More specifically, each pair of observed examples $(\boldsymbol{x}_i, \ell_{\boldsymbol{x}_i})$ and $(\boldsymbol{x}_j, \ell_{\boldsymbol{x}_j})$ with $\ell_{\boldsymbol{x}_i} \succ \ell_{\boldsymbol{x}_j}$ gives rise to a constraint $f(\boldsymbol{x}_i) > f(\boldsymbol{x}_j)$. To make the problem amenable to existing learning algorithms, the idea is to express such constraints as classification examples. Suppose, for example, that $f(\cdot)$ is a linear function $\boldsymbol{x} \mapsto \langle \alpha, \boldsymbol{x} \rangle$. Then,

$$f(\boldsymbol{x}) > f(\boldsymbol{x}') \quad \Leftrightarrow \quad \langle \alpha, \boldsymbol{x} \rangle - \langle \alpha, \boldsymbol{x}' \rangle > 0 \quad \Leftrightarrow \quad \langle \alpha, \boldsymbol{x} - \boldsymbol{x}' \rangle > 0 \ ,$$

which is equivalent to saying that $\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{x}'$ should be classified as positive (or $-\boldsymbol{z}$ as negative) by a standard binary classifier [13]. The number of pairwise constraints, and hence the size of the training data for the binary classifier, will typically be much larger than the original training data (cf. Section 3.5).

2.4 Evaluation Metrics for Multipartite Ranking

To evaluate the performance of a predicted multipartite ranking of a set X of instances, different metrics have been proposed in the literature. An obvious generalization of the AUC, which estimates the probability that a randomly chosen positive instance is ranked higher than a randomly chosen negative instance, is to consider the probability that a randomly chosen pair of instances from different classes is ranked correctly by $f(\cdot)$, i.e.

$$\mathbf{P}\left(f(\boldsymbol{x}) < f(\boldsymbol{x}') \,|\, \lambda_{\boldsymbol{x}} \prec \lambda_{\boldsymbol{x}'}\right) \ .$$

Assuming that the training examples are drawn independently from an underlying probability distribution on $\mathbb{X} \times \mathcal{L}$, an unbiased estimate of this probability is obtained by the C-index

$$C(f, X) = \frac{1}{\sum_{i<j} n_i n_j} \sum_{1 \le i < j \le m} \ \sum_{(\boldsymbol{x}, \boldsymbol{x}') \in X_i \times X_j} S(f(\boldsymbol{x}'), f(\boldsymbol{x})) \ , \qquad (2)$$

where X_i is the subset of instances $\boldsymbol{x} \in X$ whose true class is λ_i, and $n_i = |X_i|$. The C-index is commonly used as a metric of concordance in statistics [11]. It is essentially equivalent to the pairwise ranking error introduced in [13]. Obviously, the AUC defined in (1) is a special case of (2) with $X_1 = P$ and $X_2 = N$.

A related metric is the Jonckheere-Terpstra statistic [14], which is closely related to a multi-class extension of the AUC that has been proposed in [12]:

$$U(f, X) = \frac{2}{m(m-1)} \sum_{1 \le i < j \le m} \mathrm{AUC}(f, X_i \cup X_j) \ . \qquad (3)$$

The key difference between (2) and (3) is that in (2), the contribution of a class is proportional to the size of the class, while each class has the same weight in (3). In fact, (2) can be written as a weighted sum of pairwise AUCs:

$$C(f, X) = \frac{1}{\sum_{i<j} n_i n_j} \sum_{1 \le i < j \le m} n_i n_j \ \mathrm{AUC}(f, X_i \cup X_j) \qquad (4)$$

An alternative proposal for extending the AUC has recently been made in [22]. This alternative is motivated by the observation that decomposing the evaluation of a multipartite ranking into several pairwise evaluations may arguably come along with a certain loss of information and, moreover, can violate desirable transitivity properties. For example, pairwise AUCs can violate stochastic transitivity: $\mathrm{AUC}(f, X_1 \cup X_2) \geq 1/2$, $\mathrm{AUC}(f, X_2 \cup X_3) \geq 1/2$ but $\mathrm{AUC}(f, X_1 \cup X_3) < 1/2$.

Therefore, to evaluate rankings in a more global way, the idea is to consider the probability

$$\mathbf{P}\left(f(\boldsymbol{x}_1) < f(\boldsymbol{x}_2) < \ldots < f(\boldsymbol{x}_m) \,|\, \ell_{\boldsymbol{x}_1} = \lambda_1, \ldots, \ell_{\boldsymbol{x}_m} = \lambda_m \right) \ ,$$

where the $(\boldsymbol{x}_i, \ell_{\boldsymbol{x}_i})$, $i = 1 \ldots m$, are again drawn independently from an underlying probability distribution on $\mathbb{X} \times \mathcal{L}$. This gives rise to the following evaluation metric for the ranking of a finite set X of instances:

$$W(f, X) \; = \; \frac{1}{\prod_{i=1}^{m} n_i} \sum_{\boldsymbol{x}_1 \in X_1, \ldots, \boldsymbol{x}_m \in X_m} \mathbb{I}(f(\boldsymbol{x}_1) < f(\boldsymbol{x}_2) < \ldots < f(\boldsymbol{x}_m)) \ , \qquad (5)$$

where the indicator function $\mathbb{I}(\cdot)$ maps truth values of predicates to $\{0, 1\}$. Thus, the basic idea is to select m instances, one from each class, and to check whether they are correctly ordered or not. The metric (5) is simply the average over all possible m-tuples that can be verified in this way.

Despite having some potential advantages, we believe that (5) does also exhibit some questionable properties. In particular, evaluating a ranking of m elements in terms of a simple 0/1 loss, as done by the indicator function $\mathbb{I}(\cdot)$, does not distinguish between very poor rankings (e.g., a reversal of the correct ranking) and rankings that are "almost correct" (in which, for example, only two adjacent classes are swapped). To take an extreme example, suppose that $X_1 = \{\boldsymbol{x}_1\}$ and $X_2 = \{\boldsymbol{x}_2\}$ each only contain a single instance, and that the predicted ranking swaps these two instances ($f(\boldsymbol{x}_1) > f(\boldsymbol{x}_2)$), while the instances from all other classes are always ordered correctly by $f(\cdot)$. Still, (5) will yield the worst evaluation $W(f, X) = 0$.

This strong sensitivity toward small mistakes could in principle be avoided by replacing the indicator function $\mathbb{I}(\cdot)$ in (5) with a more tolerant metric, for example the (normalized) sum of concordant pairs

$$\frac{2}{m(m-1)} \sum_{1 \leq i < j \leq m} \mathbb{I}(f(\boldsymbol{x}_i) < f(\boldsymbol{x}_j)) \ .$$

This would obviously yield a metric closely related to the pairwise variants (2) and (3). Still, when expressing the metric thus obtained in terms of pairwise AUCs, another questionable property of (5) becomes obvious. In fact, one will again obtain a weighted combination of pairwise AUCs, just like (4), but now the AUC of classes λ_i and λ_j is weighted by $\prod_{i=1}^{m} n_i / (n_i n_j)$. In other words, the weighing is now *inversely* related to the size of the classes. This is because, to produce all m-tuples in (5), a pair of instances $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is combined with all instances from all other classes. In our above example, where $|X_1| = |X_2| = 1$, the instance pair $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ has an extreme influence, since it will necessarily appear in *all* m-tuples.

Subsequently, we shall focus on the pairwise evaluation metrics (2) and (3), which are intuitively appealing and widely adopted in the literature.

2.5 From Multipartite Ranking to Ordinal Classification

We end this section with a few comments on the relationship between multipartite ranking and ordinal classification, the problem that we also started with at the beginning of the section. Obviously, an ordinal classification function can be used as a ranking function. In fact, note that an ordinal classifier becomes a scoring function by interpreting its predictions, namely class labels, as scores (i.e., predicting class λ_i for instance $\boldsymbol{x}$ is considered as scoring $\boldsymbol{x}$ by i). Needless to say, however, this type of scoring will produce a large number of ties, which is why one cannot expect ordinal classifiers to be good rankers.

Conversely, a ranking function $f(\cdot)$ can be turned into an ordinal classifier by *thresholding*: Given $m+1$ threshold values t_i, $i = 1 \ldots m+1$, class λ_i is predicted for an instance $\boldsymbol{x}$ if the score $f(\boldsymbol{x})$ is between t_i and t_{i+1}. The main problem here is to find optimal thresholds, i.e., thresholds that lead to an optimal classification performance [21]. However, this problem is beyond the scope of this paper.

3 Binary Decomposition for Multipartite Ranking

As mentioned in Section 2.3, existing ranking methods are mostly based on the idea of transforming the original ranking problem into a binary classification problem. Roughly speaking, each pair of instances (from different classes) gives rise to a training example. It is worth mentioning that these methods are indeed applicable though not specifically designed for multipartite ranking: Their input is a set of order constraints on instances ($f(\boldsymbol{x}_i) > f(\boldsymbol{x}_j)$), which *can* originate from class information ($\ell_{\boldsymbol{x}_i} \succ \ell_{\boldsymbol{x}_j}$), but may also come from other sources; in fact, the existence of classes is not even assumed. In this section, we propose an alternative strategy which is specialized to the multipartite ranking problem and exploits class information in a more explicit way.

3.1 Exploiting Class Information by Binary Decomposition

Instead of transforming the original problem to a *single* binary classification problem, we decompose it into several such problems, resorting to binary decomposition techniques that have already been used successfully in multi-class classification. A decomposition technique specifically designed for ordinal classification has been proposed in [8]. Since ordinal classification and multipartite ranking are closely related (cf. Section 2.5), the idea to adapt this method to the latter problem suggests itself, and we will do so in Section 3.2. Another promising decomposition technique is the all-pairs learning scheme that has already been used in conventional and ordinal classification, as well as in other types of ranking problems such as label ranking [9,10,16,15]. In fact, this technique is especially motivated by (3) and (4), which show that the quality of a multipartite ranking is in direct correspondence with the quality of the associated bipartite rankings. Thus, it should, in principle, be possible to optimize the former by optimizing the latter, and this is what the all-pairs approach is seeking to do. We shall elaborate on this approach in Section 3.3.

One key advantage of binary decomposition is that the related problems are simpler and usually much smaller than a single binary problem, as they avoid the combinatorial explosion caused by considering all pairs of the original training examples (cf. Section 3.5). Roughly speaking, by exploiting class information, a large number of order constraints can be satisfied in an implicit way: A classifier that separates n_0 negative from n_1 positive instances, and which is hence trained on $n_0 + n_1$ examples, automatically satisfies $n_0 n_1$ order constraints.

On the other hand, binary decomposition also produces an additional problem, namely the need to *aggregate* the predictions of the different models into a single ranking: Binary decomposition techniques learn a *set* of models $\mathcal{M}_i$ on different subproblems. Given a query instance $\boldsymbol{x}$, this instance is submitted to all models, and the predictions $\mathcal{M}_i(\boldsymbol{x})$ have to be combined into an overall prediction. In the context of multipartite ranking, aggregation can essentially be realized at two different levels: (1) *Aggregation of rankings:* In this case, each model $\mathcal{M}_i$ produces a ranking, and these rankings are combined into a *consensus ranking.* (2) *Aggregation of scoring functions:* If all models $\mathcal{M}_i$ are based on scoring functions $f_i(\cdot)$, a second option is to combine these functions into a single function $f(\cdot)$, which means combining the scores $f_i(\boldsymbol{x})$ into a single score $f(\boldsymbol{x})$. Computationally, the first alternative is more complex, since, depending on the concrete criterion used, optimal rank aggregation may become very expensive. We shall therefore adopt the second approach.

3.2 The Approach of Frank and Hall

A simple and intuitively appealing approach to ordinal classification has been proposed by Frank and Hall [8]. The idea of this method, subsequently referred to as F&H, is to decompose the original problem involving m classes $\mathcal{L} = \{\lambda_1, \lambda_2 \dots \lambda_m\}$ into $m-1$ binary problems. The i-th problem is defined by the "meta-classes" $C_- = \{\lambda_1, \lambda_2 \dots \lambda_i\}$ and $C_+ = \{\lambda_{i+1}, \lambda_{i+2} \dots \lambda_m\}$ playing the role, respectively, of the negative and positive class.

Let $\mathcal{M}_i$, $i = 1 \dots m-1$, denote the model learned on this problem. Given a query instance $\boldsymbol{x}$, a prediction $\mathcal{M}_i(\boldsymbol{x})$ is interpreted as an estimation of the probability $\mathbf{P}(\ell_{\boldsymbol{x}} \succ \lambda_i)$ that the class of $\boldsymbol{x}$, denoted $\ell_{\boldsymbol{x}}$, is in C_+. Consequently, the models must guarantee outputs in the unit interval. From these probabilities, a probability distribution on $\mathcal{L}$ is derived as follows:

$$\begin{aligned}
&\mathbf{P}(\ell_{\boldsymbol{x}} = \lambda_1) = 1 - \mathbf{P}(\ell_{\boldsymbol{x}} \succ \lambda_1) \\
&\mathbf{P}(\ell_{\boldsymbol{x}} = \lambda_i) = \max\{\, \mathbf{P}(\ell_{\boldsymbol{x}} \succ \lambda_{i-1}) - \mathbf{P}(\ell_{\boldsymbol{x}} \succ \lambda_i),\, 0 \,\}, \quad i = 2, \dots, m-1 \\
&\mathbf{P}(\ell_{\boldsymbol{x}} = \lambda_m) = \mathbf{P}(\ell_{\boldsymbol{x}} \succ \lambda_{m-1})
\end{aligned} \tag{6}$$

Eventually, the class with the highest probability is predicted.

Coming back to the problem of multipartite ranking, recall that we seek to aggregate the scoring functions associated with individual models into an overall scoring function that defines the ranking. Here, these functions are given by the predictions of the models $\mathcal{M}_i$, that is, $f_i(\boldsymbol{x}) = \mathcal{M}_i(\boldsymbol{x})$, and each of them induces an individual ranking. An obvious aggregation function is given by

$$f(\boldsymbol{x}) = \sum_{i=1}^{m-1} f_i(\boldsymbol{x}) = \sum_{i=1}^{m-1} \mathcal{M}_i(\boldsymbol{x}) \ . \tag{7}$$

This aggregation is meaningful as it systematically assigns higher scores to instances from higher classes. For example, an instance $\boldsymbol{x}$ of class λ_1 will be in the negative meta-class of all $m-1$ binary classifiers, i.e., all $\mathcal{M}_i$ should return a low score $f_i(\boldsymbol{x})$, and hence the cumulative score $f(\boldsymbol{x})$ will be low.

Formally, it is worth mentioning that (7) is in direct correspondence with the *expected class index* of an instance, given that the models $\mathcal{M}_i$ yield reasonable probability estimations and are consistent in the sense that $\mathcal{M}_i(\boldsymbol{x}) \geq \mathcal{M}_{i+1}(\boldsymbol{x})$. In fact, the mapping $i \mapsto \mathcal{M}_i(\boldsymbol{x})$ is nothing else than a decumulative distribution function, and hence, with $\mathbb{E}(i)$ the expected class index of $\boldsymbol{x}$:

$$f(\boldsymbol{x}) = \sum_{i=1}^{m-1} \mathbf{P}(\ell_{\boldsymbol{x}} \succ \lambda_i) = \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \mathbf{P}(\ell_{\boldsymbol{x}} = \lambda_j) = \sum_{i=1}^{m-1} (i-1)\mathbf{P}(\ell_{\boldsymbol{x}} = \lambda_i) = \mathbb{E}(i) - 1$$

3.3 Learning by Pairwise Comparison

Learning by pairwise comparison (LPC), also known as all-pairs or round robin learning [9], is a popular binarization technique for multi-class classification. LPC trains a separate model $\mathcal{M}_{i,j}$ for each pair of classes $(\lambda_i, \lambda_j) \in \mathcal{L} \times \mathcal{L}$, $1 \leq i < j \leq m$; thus, a total number of $m(m-1)/2$ models is needed. At classification time, a query $\boldsymbol{x}$ is submitted to all models, and each prediction $\mathcal{M}_{i,j}(\boldsymbol{x})$ is interpreted as a vote for a label. More specifically, assuming scoring classifiers that produce normalized scores $f_{i,j} = \mathcal{M}_{i,j}(\boldsymbol{x}) \in [0,1]$, the *weighted voting* technique interprets $f_{i,j}$ and $f_{j,i} = 1 - f_{i,j}$ as weighted votes for classes λ_i and λ_j, respectively, and predicts the class λ^* with the highest sum of weighted votes, i.e., $\lambda^* = \arg\max_i \sum_{j \neq i} f_{i,j}$.

LPC has been used successfully for conventional multi-class classification, but has also been shown to produce strong results for ordinal classification [10,15]. To derive a ranking function from the ensemble of models $\mathcal{M}_{i,j}$, we again need a proper scoring function. Imitating the derivation of (7), a reasonable candidate is the sum of the predictions "in favor of a higher class", that is

$$f(\boldsymbol{x}) = \sum_{1 \leq i < j \leq m} f_{j,i}(\boldsymbol{x}) = \sum_{1 \leq i < j \leq m} \mathcal{M}_{j,i}(\boldsymbol{x}) \ . \tag{8}$$

A variant of this aggregation, motivated by the fact that the C-index (2) is a *weighted* combination of the pairwise AUCs, is

$$f(\boldsymbol{x}) = \sum_{1 \leq i < j \leq m} p_i p_j \, f_{j,i}(\boldsymbol{x}) = \sum_{1 \leq i < j \leq m} p_i p_j \, \mathcal{M}_{j,i}(\boldsymbol{x}) \ , \tag{9}$$

where p_i is the probability of class λ_i estimated by the relative frequency in the training data. In fact, one may expect that (9) is more suitable to optimize the C-index (2), while (8) might be preferable when using (3) as evaluation metric.

3.4 Comparing LPC and F and H

A critical issue of the LPC approach is the so-called "non-competence" problem. Even though the pairwise models $\mathcal{M}_{i,j}$ are trained only on examples from classes λ_i and λ_j, they have to be queried by all instances at prediction time. Thus, if $\boldsymbol{x}$ neither belongs to λ_i nor to λ_j, then $\mathcal{M}_{i,j}$ is actually not "competent" and the prediction $\mathcal{M}_{i,j}(\boldsymbol{x})$ becomes arguably questionable.

This problem is well-known from standard classification, and is also relevant for other binary decomposition techniques which are based on a generalized version of error correcting output codes that allows classifiers to be trained on proper subsets of the label set [1]. Despite this problem, LPC is known to perform extremely well for classification and often outperforms other decomposition techniques that do not suffer from the non-competence problem, including the one-vs-rest decomposition and the F&H method. A key advantage of LPC is a *simplification effect* produced by the all-pairs decomposition: Two-class problems are maximally simple from a learning point of view, and the predictions of models trained on these problems are hence more accurate. Since methods like one-vs-rest and F&H train each model on *all* classes, it is true that they only produce competent models. On the other hand, however, the problems are more difficult (typically requiring more complex decision boundaries), and hence the model predictions presumably less accurate. Besides, there is another potential advantage of LPC, namely a kind of *redundancy* due to the training of a larger (namely quadratic) number of models. Thanks to this redundancy, is is easier to compensate for prediction errors of individual models.

For the following reason, however, one may expect the non-competence problem to be more severe for multipartite ranking than for classification: In classification, one score is derived for each class label λ_i by combining the predictions $\mathcal{M}_{i,j}(\boldsymbol{x})$, $1 \leq i \neq j \leq m$. Thus, at least the computation of the score for the true label $\ell_{\boldsymbol{x}}$ does not involve any incompetent prediction, and as long as these predictions are correct, the true class will be the winner of the voting scheme, regardless of all other (non-competent) predictions. In multipartite ranking, on the other hand, the score of an instance $\boldsymbol{x}$ depends on *all* models $\mathcal{M}_{i,j}$, and most of them are not competent for $\boldsymbol{x}$.

Fortunately, due to the ordinal structure, there is still reason to hope that even the non-competent models will make reasonable predictions: Given that the ordinal structure of the set of class labels $\mathcal{L}$ is also reflected in the topology of the instance space $\mathbb{X}$, an assumption which is implicitly made by all ordinal classification methods [15], a model $\mathcal{M}_{i,j}$ will *indirectly* also be trained for classes λ_k, $i \neq k \neq j$. For example, when the model $M_{2,3}$ is queried with an instance $\boldsymbol{x}$ whose true class is λ_4, a vote for the higher class λ_3 (which is supposed to be "closer" to λ_4) should be more likely than a vote for the lower class λ_2.

In summary, LPC has the disadvantage of partially non-competent models, but the advantage of simplicity and redundancy. Which of these effects will dominate in practice is a question that cannot be answered in general, especially since the answer will strongly depend on the data set and learning algorithm used. First insights will be offered by our empirical study presented in Section 4.

3.5 Computational Complexity

The F&H method trains $m-1$ models on the complete set of training examples whose size is $|T| = n$. Thus, the total number of training examples for the transformed problem is $(m-1)n$, and when using a base learner with complexity $O(k^{\alpha})$, the overall complexity for training an F&H ranking model is $O(mn^{\alpha})$.

LPC requires a total of $(m-1)n$ training examples, too, because each original example $(\boldsymbol{x}_i, \ell_{\boldsymbol{x}_i})$ is used in exactly $m-1$ binary models $\mathcal{M}_{i,j}$. Correspondingly, the training complexity is $O(m^2 n^{\alpha})$ for a base learner with complexity $O(k^{\alpha})$. It should be noted, however, that the pairwise problems, which involve only the instances from two classes, are typically much smaller than n. For example, for a uniform class distribution where each class has $\approx n/m$ examples, the complexity is $O(m^{2-\alpha} n^{\alpha})$. Thus, for base learners with super-linear complexity ($\alpha > 1$), LPC is typically even more efficient than F&H.

For methods that construct a single binary classification problem, with one order constraint for each pair of instances with different class labels, the number of training examples can become as large as $O(n^2)$. With a classifier whose complexity is $O(k^{\alpha})$, the overall complexity is hence $O(n^{2\alpha})$ and increases much faster with the size of the training set T than for the decomposition methods.

However, it is worth mentioning that, for support vector machines, a quadratic growth in the number of examples can be avoided under certain conditions. In [18], Joachims proposes a sophisticated cutting plane algorithm for SVM training that exploits the sparsity of a data set and scales with $O(s \cdot n \cdot \log(n))$, where s is the average number of non-zero features. Unfortunately, an implementation of this approach (`http://svmlight.joachims.org/`) is offered only for the case of two ranks ($m = 2$), which precludes its use in our experimental analysis.

4 Experimental Analysis

In this section, we provide an extensive empirical evaluation and comparison between methods for multipartite ranking. Our primary interest is to show that a reduction to multiple binary classifiers is at least competitive to state-of-the-art ranking methods in terms of predictive accuracy, while being much more efficient from a computational point of view. Besides, we are also interested in comparing the two decomposition methods, LPC and F&H, amongst each other.

4.1 Data Sets, Ranking Methods, and Experimental Setup

Due to a lack of ordinal benchmark data sets, several previous studies including [8,10,15] have resorted to discretized regression data sets for experimental purposes. This is reasonable and has the advantage that, by changing the discretization, ordinal data sets can be produced in a quite flexible manner. We have used twenty-one regression data sets from the UCI repository [2]. These data sets vary strongly in size and number and type of features, and hence, they are representative for a wide range of data that may occur in practice; see Table 1. To obtain an ordinal class attribute, the numerical output attributes were

Table 1. The twenty-five data sets used in the experiments (name, number of instances, number of numerical features, number of nominal features). The last four data sets are truly ordinal, and the number of classes is given in brackets after the name.

#	name	size	num	nom	#	name	size	num	nom
1	Abalone	4177	7	1	14	House 8L	22784	8	0
2	Ailerons	13750	40	0	15	House 16H	22754	16	0
3	Auto Mpg	398	40	0	16	Kinematics	8192	8	0
4	Auto Price	159	14	1	17	MV Artificial	40768	7	3
5	Bank 8FM	8192	8	0	18	Pumadyn 8NH	8192	8	0
6	Bank 32NH	8192	32	0	19	Pumdayn 32H	8192	32	0
7	Boston Housing	506	12	1	20	Servo	167	0	4
8	California Housing	20640	8	0	21	Stocks	950	9	0
9	CPU Act	8192	21	0					
10	CPU Small	8192	12	0	22	ERA (9)	1000	40	0
11	Delta Ailerons	7129	5	0	23	ESL (9)	488	40	0
12	Elevators	16599	18	0	24	Eucalyptus (5)	736	14	5
13	Friedman Artificial	40768	10	0	25	LEV (5)	1000	40	0

discretized into $m = 5$ classes using equal-frequency binning. It is worth noting that this did not always lead to classes of equal size, since in many data sets, one or more values of the numerical output attribute occurs many times. We only report results for $m = 5$, because other values produced quite similar results. We complemented the discretized regression data sets with four truly ordinal classification data sets taken from [3], namely those that have reasonable size and at least five classes.

As a baseline, we selected SVMRank [17], a state-of-the-art method for ranking problems which is based on support vector learning (`http://svmlight.joachims.org/`). We used this method in its default setting with a linear kernel. F&H and LPC were implemented with logistic regression as a base learner, which comes down to fitting a linear model and using the logistic link function ($\text{logit}(x) = \log(x/(1-x))$) to derive $[0,1]$-valued scores, the type of model output requested by both methods. Essentially, all three methods are therefore based on linear models so that the comparison is fair from this point of view. For LPC, we tried both aggregation strategies, the unweighted version (8) and the weighted variant (9); we shall refer to these methods as LPC-U and LPC-W, respectively. As a side remark, we mention that we also tried the classifier versions of F&H and LPC (ordering instances by the predicted class). As expected, however, these were consistently outperformed by their ranking variants, and therefore we excluded them from the analysis that we will present below.

We report averages of several test statistics over five repetitions of a stratified ten-fold cross validation, each time with a different random permutation of the data. The standard deviations were very small (often of the magnitude 10^{-4}) and, for the ease of exposition, are therefore omitted in the tables.

4.2 Experimental Results

The ranking performance of the methods in terms of the C-index (2) and the Jonckheere-Terpstra statistic (3) are shown in Table 3. To analyze the results, we followed the two-step statistical test procedure recommended in [6], consisting of a Friedman test of the null hypothesis that all learners have equal performance and, in case this hypothesis is rejected, a Nemenyi test to compare learners in a pairwise way. Both tests are based on the average ranks over all data sets, which are shown at the bottom of the table. The ranks on the individual data sets are indicated by the numbers in brackets (the best method has rank 1, the worst rank 4; average ranks are assigned in case of ties).

The results show that F&H is significantly better than the rest, while no significant difference is detected among the others (the critical rank difference is 0.88 for a significance level of 10%). Note that, as expected, the performance of LPC-U increases in comparison to LPC-W when using (3) instead of (2) as a performance metric. Overall, however, these differences are small.

The main conclusion from this experiment is that the binary decomposition methods are quite competitive to SVMRank in terms of ranking performance, and in fact, the ranking variant of F&H seems to be even superior. (We are somewhat cautious here, since the computational complexity of SVMRank prevented from a truly thorough parameter optimization.) In terms of computational complexity, the decomposition methods are much more efficient, outperforming SVMRank by several orders of magnitude on some data sets. In fact, training of a single SVMRank model may easily take hours to days, and running the experiments for this method was only feasible on a parallel computer. Obviously, this precludes a direct comparison of run-times. We can mention, though, that the run-times of F&H and LPC are comparable to the run-times of their corresponding classification variants, and for both methods, 5 repetitions of a 10-fold cross validation never took more than one hour of CPU time.

As to the direct comparison of F&H and LPC, we complemented the ranking results by the performance of the respective classifier-versions in terms of classification accuracy; see Table 2. This study essentially confirms previous results [10,15], which have shown that LPC, even though it is not specifically designed

Table 2. Classification accuracy for the classifier-variants of LPC and F&H

#	LPC	F&H	#	LPC	F&H	#	LPC	F&H	#	LPC	F&H
1	.5248	.5157	8	.5490	.5463	14	.5040	.4814	20	.7425	.7199
2	.6084	.6118	9	.7195	.7183	15	.4967	.4603	21	.8307	.7840
3	.6836	.6846	10	.6686	.6676	16	.4044	.4161	22	.4036	.4062
4	.5987	.5902	11	.5716	.5587	17	.9149	.8837	23	.7729	.7741
5	.7897	.7885	12	.5727	.5707	18	.4860	.4575	24	.6378	.6619
6	.4851	.4855	13	.5547	.5547	19	.3863	.3933	25	.5972	.6022
7	.6608	.6561									

Table 3. Ranking performance of the different methods

	C-index				Jonckheere-Terpstra statistic			
#	LPC-U	LPC-W	F&H	SVMRank	LPC-U	LPC-W	F&H	SVMRank
1	.8398 (3.0)	.8401 (2.0)	.8417 (1.0)	.8361 (4.0)	.8383 (2.5)	.8383 (2.5)	.8401 (1.0)	.8352 (4.0)
2	.9171 (1.0)	.9170 (2.0)	.9167 (4.0)	.9168 (3.0)	.9121 (1.0)	.9120 (2.0)	.9118 (3.0)	.9117 (4.0)
3	.9468 (4.0)	.9470 (3.0)	.9495 (2.0)	.9583 (1.0)	.9471 (4.0)	.9473 (3.0)	.9498 (2.0)	.9590 (1.0)
4	.9142 (3.0)	.9137 (4.0)	.9217 (2.0)	.9342 (1.0)	.9130 (3.0)	.9125 (4.0)	.9210 (2.0)	.9367 (1.0)
5	.9764 (3.5)	.9764 (3.5)	.9774 (1.0)	.9766 (2.0)	.9764 (3.5)	.9764 (3.5)	.9774 (1.0)	.9766 (2.0)
6	.8422 (3.5)	.8422 (3.5)	.8431 (1.0)	.8427 (2.0)	.8418 (3.5)	.8418 (3.5)	.8427 (1.0)	.8423 (2.0)
7	.9129 (4.0)	.9130 (3.0)	.9242 (1.0)	.9241 (2.0)	.9132 (3.5)	.9132 (3.5)	.9245 (1.0)	.9240 (2.0)
8	.8782 (2.5)	.8782 (2.5)	.8784 (1.0)	.8734 (4.0)	.8782 (2.5)	.8782 (2.5)	.8784 (1.0)	.8734 (4.0)
9	.9249 (4.0)	.9250 (3.0)	.9528 (1.0)	.9518 (2.0)	.9238 (4.0)	.9239 (3.0)	.9519 (1.0)	.9509 (2.0)
10	.9105 (4.0)	.9108 (3.0)	.9349 (1.0)	.9324 (2.0)	.9090 (4.0)	.9093 (3.0)	.9334 (1.0)	.9310 (2.0)
11	.8722 (1.0)	.8721 (2.0)	.8715 (3.0)	.8704 (4.0)	.8594 (1.0)	.8592 (2.0)	.8585 (3.0)	.8575 (4.0)
12	.8696 (2.0)	.8697 (1.0)	.8686 (3.0)	.8640 (4.0)	.8695 (2.0)	.8696 (1.0)	.8686 (3.0)	.8639 (4.0)
13	.8839 (3.5)	.8839 (3.5)	.8853 (1.0)	.8849 (2.0)	.8839 (3.5)	.8839 (3.5)	.8853 (1.0)	.8849 (2.0)
14	.8317 (1.5)	.8317 (1.5)	.8243 (3.0)	.8169 (4.0)	.8317 (1.5)	.8317 (1.5)	.8244 (3.0)	.8170 (4.0)
15	.8239 (1.0)	.8238 (2.0)	.8164 (3.0)	.8123 (4.0)	.8239 (1.5)	.8239 (1.5)	.8164 (3.0)	.8123 (4.0)
16	.7741 (2.5)	.7741 (2.5)	.7753 (1.0)	.7734 (4.0)	.7741 (2.5)	.7741 (2.5)	.7753 (1.0)	.7734 (4.0)
17	.9780 (2.5)	.9780 (2.5)	.9797 (1.0)	.9505 (4.0)	.9780 (2.5)	.9780 (2.5)	.9797 (1.0)	.9505 (4.0)
18	.7892 (2.5)	.7892 (2.5)	.7895 (1.0)	.7806 (4.0)	.7892 (2.5)	.7892 (2.5)	.7895 (1.0)	.7806 (4.0)
19	.6721 (3.5)	.6721 (3.5)	.6724 (2.0)	.6732 (1.0)	.6721 (3.5)	.6721 (3.5)	.6724 (2.0)	.6732 (1.0)
20	.9031 (4.0)	.9052 (3.0)	.9272 (1.0)	.9219 (2.0)	.9058 (4.0)	.9068 (3.0)	.9271 (1.0)	.9231 (2.0)
21	.9325 (2.0)	.9324 (3.0)	.9636 (1.0)	.9320 (4.0)	.9324 (2.5)	.9324 (2.5)	.9635 (1.0)	.9320 (4.0)
22	.7415 (3.5)	.7415 (3.5)	.7418 (2.0)	.7434 (1.0)	.7263 (3.0)	.7262 (4.0)	.7265 (2.0)	.7277 (1.0)
23	.9645 (4.0)	.9648 (2.0)	.9654 (1.0)	.9647 (3.0)	.9662 (3.0)	.9664 (2.0)	.9672 (1.0)	.9655 (4.0)
24	.8908 (4.0)	.8954 (3.0)	.9346 (2.0)	.9395 (1.0)	.8895 (4.0)	.8936 (3.0)	.9341 (2.0)	.9384 (1.0)
25	.8635 (4.0)	.8647 (3.0)	.8660 (2.0)	.8679 (1.0)	.8775 (1.0)	.8748 (4.0)	.8757 (3.0)	.8764 (2.0)
avg. rank	2.96	2.72	1.68	2.64	2.78	2.78	1.68	2.76

Table 4. C-index statistics of the individual models M_{ij} and M_i for LPC and F&H, respectively. We report the average min, mean, median, and max. For comparison, we also add the performance of the complete (aggregated) model.

#	LPC					F&H				
	min	mean	median	max	overall	min	mean	median	max	overall
1	.6553	.8011	.8185	.8356	.8401	.8185	.8295	.8311	.8372	.8417
2	.8917	.9099	.9121	.9156	.9170	.9110	.9135	.9138	.9154	.9167
3	.7851	.8718	.8703	.9113	.9470	.8546	.9034	.9153	.9283	.9495
4	.7253	.8169	.8342	.8882	.9137	.8103	.8663	.8752	.9043	.9217
5	.9433	.9657	.9732	.9763	.9764	.9726	.9746	.9747	.9764	.9774
6	.7936	.8266	.8310	.8378	.8422	.8239	.8351	.8379	.8407	.8431
7	.7887	.8450	.8653	.8971	.9130	.8741	.8906	.8912	.9059	.9242
8	.8533	.8656	.8688	.8732	.8782	.8606	.8664	.8665	.8721	.8784
9	.8746	.9042	.8797	.9492	.9250	.9435	.9476	.9484	.9502	.9528
10	.8576	.8933	.8930	.9280	.9108	.9243	.9273	.9273	.9304	.9349
11	.8477	.8590	.8590	.8662	.8721	.8580	.8649	.8668	.8681	.8715
12	.7719	.8313	.8396	.8631	.8697	.7991	.8435	.8563	.8623	.8686
13	.8753	.8834	.8844	.8852	.8839	.8820	.8841	.8847	.8852	.8853
14	.7804	.8101	.8136	.8262	.8317	.7945	.8131	.8165	.8251	.8243
15	.7309	.7878	.7956	.8104	.8238	.7866	.8018	.8024	.8156	.8164
16	.7450	.7631	.7661	.7723	.7741	.7650	.7691	.7691	.7730	.7753
17	.7056	.8760	.8996	.9424	.9780	.8186	.8976	.9176	.9368	.9797
18	.5884	.7367	.7647	.7814	.7892	.7147	.7581	.7689	.7797	.7895
19	.5544	.6401	.6638	.6683	.6721	.6701	.6706	.6702	.6719	.6724
20	.6968	.7850	.7913	.8457	.9052	.7322	.8274	.8438	.8900	.9272
21	.5115	.7420	.7618	.8650	.9324	.7154	.8295	.8478	.9069	.9636
22	.7043	.7345	.7383	.7419	.7415	.7392	.7413	.7418	.7423	.7418
23	.8960	.9393	.9492	.9622	.9648	.9521	.9596	.9619	.9625	.9654
24	.7205	.8007	.7827	.8939	.8954	.8412	.8757	.8735	.9146	.9346
25	.7726	.8465	.8526	.8673	.8647	.8441	.8590	.8622	.8674	.8660

for ordinal problems, is at least competitive and often superior to F&H: The classification version of LPC wins 15 times against the classification variant of F&H and looses only 9 times; see Table 2. All the more interesting is the question why F&H performs better for the ranking problem.

In Section 3.3, some possible answers to this question have already been anticipated. To get more insight, we looked at the ranking performance of the *individual* models. More precisely, we let each of the models $\mathcal{M}_i$ in F&H and $\mathcal{M}_{i,j}$ in LPC rank all examples, and compared this ranking to the true ordinal classification of the examples. Table 4 shows the minimum, mean, median, and maximum C-index among these models for both LPC and F&H. It can be seen that F&H is consistently better in terms of the minimum, i.e., the worst model $\mathcal{M}_i$ is still better than the worst model $\mathcal{M}_{i,j}$. This may not be too surprising, because the pairwise approach learns a much higher number of models. However, the same can also be observed for the mean and the median (with a single exception), and even in terms of the maximum, F&H is better in 20 of the 25 data sets. Less surprisingly, the variability among the LPC models $\mathcal{M}_{i,j}$ is higher than among the F&H models $\mathcal{M}_i$, as can be seen from the difference between the maximum and the minimum. We take these results as strong evidence for the dominance of the non-competence problem of LPC models, as discussed in Section 3.3, and consider this problem as a reasonable explanation for the superiority of F&H.

Worth mentioning is finally a kind of ensemble effect revealed by the results in Table 4: For both methods, LPC and F&H, the overall performance of the aggregated model is consistently better than the best performance of an individual model.

5 Conclusions

We have elaborated on the use of binary decomposition methods in the context of multipartite ranking, a generalization of bipartite ranking to the multi-class case. The use of such methods is motivated by their successful application to related problems, including multi-class classification, ordinal classification, and label ranking. Our results have shown that decomposition methods are competitive, if not even superior, to state-of-the-art ranking methods in terms of predictive accuracy, while being much more efficient from a computational point of view. In any case, they offer a viable alternative to hitherto existing methods.

In future work, we plan to further improve the performance of the methods proposed in this paper, for example by means of alternative aggregation schemes. Moreover, only simple classification methods such as logistic regression have been used as base learners so far. Such methods seek to maximize classification accuracy in the first place, and hence optimize ranking performance (in terms of AUC) only indirectly. Therefore, the use of AUC-optimizing classifiers as base learners is likely to yield improved results.

Acknowledgements. This work was supported by the *German Science Foundation (DFG)* and the *Dutch Organisation for Scientific Research (NWO).*

References

1. Allwein, E., Schapire, R., Singer, Y.: Reducing multiclass to binary: A unifying approach for margin classifiers. JMLR 1, 113–141 (2001)
2. Asuncion, A., Newman, D.: UCI machine learning repository (2007), `http://www.ics.uci.edu/~mlearn/MLRepository.html`
3. Ben David, A.: Ordinal real-world data sets repository (2008), `http://www.cs.waikato.ac.nz/ml/weka/index_datasets.html`
4. Cardoso, J., da Costa, J.P.: Learning to classify ordinal data: The data replication method. JMLR 8, 1393–1429 (2007)
5. Chu, W., Keerthi, S.: New approaches to support vector ordinal regression. In: Proceedings ICML, pp. 145–152. ACM, New York (2005)
6. Demsar, J.: Statistical comparisons of classifiers over multiple data sets. JMLR 7, 1–30 (2006)
7. Fawcett, T.: An introduction to ROC analysis. Pattern Recognition Letters 27(8), 861–874 (2006)
8. Frank, E., Hall, M.: A simple approach to ordinal classification. In: Flach, P.A., De Raedt, L. (eds.) ECML 2001. LNCS (LNAI), vol. 2167, pp. 145–156. Springer, Heidelberg (2001)
9. Fürnkranz, J.: Round robin classification. JMLR 2, 721–747 (2002)
10. Fürnkranz, J.: Round robin ensembles. Intelligent Data Analysis 7(5), 385–404 (2003)
11. Gnen, M., Heller, G.: Concordance probability and discriminatory power in proportional hazards regression. Biometrika 92(4), 965–970 (2005)
12. Hand, D., Till, R.: A simple generalization of the area under the ROC curve for multiple class problems. Machine Learning 45(2), 171–186 (2001)
13. Herbrich, R., Graepel, T., Obermayer, K.: Large margin rank boundaries for ordinal regression. In: Advances in Large Margin Classifiers, pp. 115–132. MIT Press, Cambridge (2000)
14. Higgins, J.: Introduction to Modern Nonparametric Statistics. Duxbury Press (2004)
15. Hühn, J., Hüllermeier, E.: Is an ordinal class structure useful in classifier learning? Int. J. Data Mining, Modelling and Management 1(1), 45–67 (2009)
16. Hüllermeier, E., Fürnkranz, J., Cheng, W., Brinker, K.: Label ranking by learning pairwise preferences. Artificial Intelligence 172, 1897–1917 (2008)
17. Joachims, T.: Optimizing Search Engines Using Clickthrough Data. In: Proceedings ACM SIGKDD, Edmonton, Alberta, Canad, pp. 133–142. ACM Press, New York (2002)
18. Joachims, T.: Training linear SVMs in linear time. In: Proceedings ACM SIGKDD, Philadelphia, PA, USA, pp. 217–226. ACM Press, New York (2006)
19. Kramer, S., Widmer, G., Pfahringer, B., De Groeve, M.: Prediction of ordinal classes using regression trees. Fundamenta Informaticae 34(1-2), 1–15 (2000)
20. Rajaram, S., Agarwal, S.: Generalization bounds for k-partite ranking. In: Proceedings of the NIPS 2005 Workshop on Learning to Rank, Whistler, BC, Canada, pp. 28–23 (2005)
21. Shashua, A., Levin, A.: Ranking with large margin principle: Two approaches. In: Advances in NIPS, vol. 15, pp. 937–944. MIT Press, Cambridge (2002)
22. Waegeman, W., Baets, B.D., Boullart, L.: ROC analysis in ordinal regression learning. Pattern Recognition Letters 29(1), 1–9 (2008)