Decision Rule-based Algorithm for Ordinal Classification based on Rank Loss Minimization

Krzysztof Dembczyński[1] and Wojciech Kotłowski[1,2]

[1] Institute of Computing Science, Poznań University of Technology,
60-965 Poznań, Poland
kdembczynski@cs.put.poznan.pl
[2] Centrum Wiskunde & Informatica, NL-1097 CA Amsterdam, The Netherlands
kotlowsk@cwi.nl

Abstract. Many classification problems have in fact an ordinal nature, i.e., the class labels are ordered. We introduce a decision rule algorithm, called RankRules, tailored for this type of problems, that is based on minimization of the rank loss. In general, the complexity of the rank loss minimization is quadratic with respect to the number of training examples, however, we show that the introduced algorithm works in linear time (plus sorting time of attribute values that is performed once in the pre-processing phase). The rules are built using a boosting approach. The impurity measure used for building single rules is derived using one of four minimization techniques often encountered in boosting. We analyze these techniques focusing on the trade-off between misclassification and coverage of the rule. RankRules is verified in the computational experiment showing its competitiveness to other algorithms.

1 Introduction

Given an object described by attribute values $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, the goal in *ordinal classification* consists in predicting an unknown class label y taken from an ordered set $\mathcal{K} = \{1, \ldots, K\}$. We assume that there is a meaningful order between classes which corresponds to the natural order between class labels.

This type of problem is very common in real applications, mainly in the areas related to exploiting user's preferences. For example, in recommender systems, users are asked to rate items on finite value scale, e.g. one to five "stars". The task is to predict users' rates. Another example is classification of emails to groups like: "very important", "important", "normal", and "later".

There are two recently introduced approaches to ordinal classification that gain the main attention. The first one consists in constructing a scoring function that properly orders the training examples. This is done by minimization of the so-called rank loss [1]. The classes are then determined by thresholds defined on the range of the scoring function. The construction of the scoring function requires, however, that all examples are compared pairwise. Thus, the complexity of this approach is in general quadratic with respect to the number N of training examples. In order to reduce the complexity, the approach based on threshold

loss has been introduced [2, 3]. The scoring function is computed simultaneously with thresholds in such a way that training examples are compared to thresholds only. In the following, however, we focus on the former approach despite of its complexity, introducing an algorithm that works in linear time.

We consider a classifier in which the scoring function is a rule ensemble, i.e., linear combination of decision rules being logical patterns of the form: if [condition] $then$ [decision]. Such a rule can be treated as a simple classifier that gives a constant response to examples satisfying the condition part, and abstains from the response for all other examples. The main advantage of rules is their simplicity and human-readable form. The introduced algorithm, called RankRules, builds the rule ensemble by using the boosting approach in which the exponential rank loss is greedily minimized.

The contribution of the paper can be seen from two perspectives. On the one hand, we thoroughly analyze a boosting algorithm applied with a specific base classifier – decision rule. In this context our algorithm is similar to RankBoost [4] that was already analyzed in ordinal classification settings [3]. However, the main conclusion in [3] is that an approach based on the rank loss performs worse than an approach based on the threshold loss. We have obtained an opposite result which states that both approaches are competitive. We also show that the complexity of the rank loss minimization can be linear in the number of training examples in the ordinal classification settings, similarly as in the case of threshold loss minimization (assuming that the attribute values are already sorted). This extends results obtained for two-class problems. It was shown that RankBoost [4] scales linearly. In the case of SVM with linear kernel function, the rank loss minimization scales with $N \log N$ [5].

From the other perspective, RankRules can be seen as a novel approach to rule generation. The classical algorithms were mainly based on the sequential covering procedure [6]. Recently, several algorithms based on boosting have been introduced that can be seen as generalization of the sequential covering [7–9]. RankRules is distinguished by the fact that it minimizes the rank loss. In the case of binary classification, this is closely related to maximization of the so-called AUC (area under the ROC curve) criterion. Thus, RankRules for two-class problems can be treated as an efficient decision rule algorithm maximizing AUC. From this point of view, it is related to [10].

In rule induction, we need to specify the impurity measure that controls the procedure constructing a single rule. In the introduced approach, the impurity measure is naturally derived from the empirical risk formula using one of four minimization techniques. Since the rule can abstain from classification, we analyze the impurity measures in the context of a trade-off between misclassification and coverage (i.e., the number of covered training examples) of the rule. We show how the minimization techniques influence this trade-off. A similar analysis has already been performed for the rule ensembles in binary classification case [11], for margin loss functions. This is also related to [12] in which the trade-off is discussed for some classic rule impurity measures.

The paper is organized in the following way. Section 2 states the ordinal classification problem. Section 3 describes and analyzes the RankRules algorithm. Section 4 reports the experiments on artificial and benchmark data sets. The last section concludes the paper.

2 Ordinal Classification

In order to solve the ordinal classification problem one has to construct a function $F(\mathbf{x})$ using a set of training examples $\{y_i, \mathbf{x}_i\}_1^N$ that predicts accurately an ordered label y. Since y is discrete, for a given $\mathbf{x}$ it obeys a multinomial distribution, $p_k(\mathbf{x})$, $k = 1, \ldots, K$, where $p_k(\mathbf{x}) = \Pr(y = k|\mathbf{x})$. Thus, the optimal function is clearly given by:

$$F^*(\mathbf{x}) = \arg\min_{F(\mathbf{x})} \sum_{k=1}^{K} p_k(\mathbf{x}) L(y, F(\mathbf{x})),$$

where $L(y, F(\mathbf{x}))$ is the *loss function* which gives the penalty for predicting $F(\mathbf{x})$ when the actual value is y. We define the loss function $L(y, F(\mathbf{x}))$ as matrix:

$$\mathbf{L}(y, \hat{y}) = (l_{y,\hat{y}})_{K \times K} \tag{1}$$

where $\hat{y}$ is a prediction made by $F(\mathbf{x})$ (i.e., $\hat{y} = F(\mathbf{x})$). It is assumed that the matrix satisfies $l_{y,\hat{y}} = 0$, for $y = \hat{y}$, and $l_{y,\hat{y}} > 0$, otherwise. Moreover, since we want loss matrix to be consistent with the labels' order, each row of $\mathbf{L}$ has to be *v-shaped*: $l_{y,\hat{y}-1} \geq l_{y,\hat{y}}$, if $\hat{y} \leq y$ and $l_{y,\hat{y}} \leq l_{y,\hat{y}+1}$, for $\hat{y} \geq y$ (larger deviation of predicted label $\hat{y}$ from observed label y is more penalized). A natural choice is the absolute-error loss for which $l_{y,\hat{y}} = |y - \hat{y}|$, since the optimal function in this case is median over class distribution:

$$F^*(\mathbf{x}) = \text{median}_{p_k(\mathbf{x})}(y).$$

Median does not depend on a distance between class labels, so the scale of the decision attribute does not matter, only the order is taken into account.

We can conclude that in order to solve the ordinal classification problem with loss matrix $\mathbf{L}$, one can first estimate $p_k(\mathbf{x})$, $k \in \mathcal{K}$. A final prediction then can be computed as median over estimated probabilities.

As it was observed in [13], any loss matrix (1) can be decomposed into an aggregation of margin 0-1 loss functions (defined as $L_{0-1}(f) = [\![f < 0]\!]$). [3] Let y_k be an auxiliary class label, $k = 1, \ldots, K-1$, such that $y_k = 1$, if $y > k$, and $y_k = -1$, otherwise. Moreover, let $f_k(\mathbf{x}) \in \mathbb{R}$ be a function such that $f_k(\mathbf{x}) > 0$, if $F(\mathbf{x}) > k$, and $f_k(\mathbf{x}) < 0$, otherwise. Then, the following decomposition holds:

$$l_{y,\hat{y}} = \sum_{k=1}^{K-1} |l_{y,k+1} - l_{y,k}| L_{0-1}(y_k f_k(\mathbf{x})).$$

[3] $[\![\pi]\!]$ is the Boolean test, equal to 1 if predicate π is true, and 0 otherwise.

For the absolute-error we have $|l_{y,k+1} - l_{y,k}| = 1$. It is usually desired to reconstruct $F(\mathbf{x})$ by monotonic $f_k(\mathbf{x})$:

$$f_k(\mathbf{x}) \geq f_{k+1}(\mathbf{x}), \text{ for } k = 1, \ldots, K-1. \tag{2}$$

In order to satisfy (2), one can use function $f(\mathbf{x})$ and introduce thresholds $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_K)$ such that

$$\theta_0 = -\infty \leq \theta_1 \leq \ldots \leq \theta_{K-1} \leq \theta_K = \infty. \tag{3}$$

In this case, we have that $f_k(\mathbf{x}) = f(\mathbf{x}) - \theta_k$.

Function $f(\mathbf{x})$ can be constructed simultaneously with estimation of thresholds $\boldsymbol{\theta}$ by minimizing the so-called *threshold loss* [2, 13]:

$$L(y, f(\mathbf{x}), \boldsymbol{\theta}) = \sum_{k=1}^{K-1} L(y_k(f(\mathbf{x}) - \theta_k)), \tag{4}$$

where $L(f)$ is a typical margin loss function used in binary classification problems, like the 0-1 loss, exponential loss, $L_{\exp}(f) = \exp(-f)$, or logit loss, $L_{\log}(f) = \log(1+\exp(-f))$. In general, this approach yields in upper bounding the absolute-error.

In this paper, however, we introduce the algorithm that solves the problem in a similar way to [1]. We first construct the function $f(\mathbf{x})$. To this end we use the *rank loss* that is defined on pairs of examples:

$$L\left(y_\bullet, y_\circ, f(\mathbf{x}_\bullet), f(\mathbf{x}_\circ)\right) = L\left(y_{\bullet\circ}(f(\mathbf{x}_\bullet) - f(\mathbf{x}_\circ))\right), \tag{5}$$

where $y_{\bullet\circ} = \operatorname{sgn}(y_\bullet - y_\circ)$ and $L(f)$ is one of margin loss functions. The learning algorithm minimizes this loss over the pairs of training examples $i, j = 1, \ldots, N$:

$$R(f) = \sum_{y_{ij}>0} L(f(\mathbf{x}_i) - f(\mathbf{x}_j)),$$

where y_{ij} denotes $y_{\bullet\circ}$ defined for the i-th and j-th training example. In the next step, thresholds $\boldsymbol{\theta}$ are computed by minimizing the threshold loss function for known $f(\mathbf{x})$:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i), \boldsymbol{\theta}), \tag{6}$$

subject to (3). Thus, this approach is also suited for minimization of the absolute-error loss.

Let us remark that in the case of binary classification, the above formulation is closely related to the so-called AUC (area under the "Receiving Operator Characteristic" curve) criterion, becoming a popular measure for evaluating the performance of classifiers. The AUC criterion is defined as:

$$AUC(f) = \Pr(f(\mathbf{x}_\bullet) > f(\mathbf{x}_\circ) | y_\bullet = 1, y_\circ = -1).$$

It is easy to see that maximizing the AUC criterion boils down to minimizing the rank loss, since:

$$AUC(f) = 1 - \frac{\mathbb{E}_{y_\bullet, y_\circ, \mathbf{x}_\bullet, \mathbf{x}_\circ}[L_{0-1}(y_{ij}(f(\mathbf{x}_\bullet) - f(\mathbf{x}_\circ)))]}{2\Pr(y_\bullet = 1, y_\circ = -1))}.$$

3 RankRules

We start the description of RankRules with the formal definition of a decision rule. Let X_p be a domain of attribute $p \in \{1, \ldots, n\}$. Condition part of the rule, denoted by Φ, consists of a conjunction of elementary conditions of the general form

$$x_p \in S_p,$$

where x_p is the value of $\mathbf{x}$ on attribute p and S_p is a subset of X_p. In particular, elementary conditions are of the form $x_p \geq s_p$, $x_p \leq s_p$, for quantitative attributes, and $x_p = s_p$, $x_p \neq s_p$, for qualitative attributes, where s_p is taken from a domain of p-th attribute. Let $\Phi(\mathbf{x})$ be a function indicating whether an object satisfies the condition of the rule. In other words, $\Phi(\mathbf{x})$ defines an arbitrary axis-parallel region in the attribute space. We say that a rule *covers* object $\mathbf{x}$ if it belongs to this region, i.e., $\Phi(\mathbf{x}) = 1$, otherwise $\Phi(\mathbf{x}) = 0$. The number of training examples covered by the rule is referred to as *rule coverage*. Decision (also called response), denoted by α, is a real non-zero value assigned to the region defined by Φ. Therefore, we define a decision rule as:

$$r(\mathbf{x}) = \alpha\Phi(\mathbf{x}).$$

We assume that the rule ensemble is a linear combination of M decision rules:

$$f_M(\mathbf{x}) = \sum_{m=1}^{M} r_m(\mathbf{x}).$$

In order to make prediction one determines $\boldsymbol{\theta}$ and computes

$$F(\mathbf{x}) = \sum_{k=1}^{K} k[\![f_M(\mathbf{x}) \in [\theta_{k-1}, \theta_k)]\!].$$

The rule ensemble is constructed by minimizing the rank loss based on the exponential loss. We use the exponential loss, since it is a convex function, which makes the minimization process easier to cope with. Moreover, due to modularity of the exponential function, minimization of rank loss can be performed in an efficient way.

We follow the boosting approach that results in an iterative procedure in which rules are added one by one to the ensemble greedily minimizing the loss over training examples. In the m-th iteration, the rule is obtained in the following way:

$$r_m = \arg\min_{r} R(f_{m-1} + r) = \arg\min_{\Phi, \alpha} \sum_{y_{ij} > 0} w_{ij}^{(m)} e^{-\alpha(\Phi_m(\mathbf{x}_i) - \Phi_m(\mathbf{x}_j))}, \qquad (7)$$

where f_{m-1} is rule ensemble after $m-1$ iterations, and

$$w_{ij}^{(m)} = e^{-(f_{m-1}(\mathbf{x}_i) - f_{m-1}(\mathbf{x}_j))}$$

can be treated as weights associated with pairs of training examples. Let us remark that the overall loss changes only for pairs in which one example is covered by the rule and the other is not (i.e., $\Phi(\mathbf{x}_i) \neq \Phi(\mathbf{x}_j)$). The details how the rule is generated are given in the next subsection.

After generating the rule ensemble, the algorithm determines values of thresholds by solving (6) based on the exponential loss:

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \sum_{k=1}^{K-1} e^{-y_{ik}(f_M(\mathbf{x}_i) - \theta_k)}, \tag{8}$$

subject to (3). We use y_{ik} to denote the auxiliary variable y_k for the i-th training example. The solution of (8) is then given, for k = 1,. . . ,K - 1, by:

$$\theta_k = \frac{1}{2} \log \frac{\sum_{i=1}^{N} [\![y_{ik} > 0]\!] e^{f_M(\mathbf{x}_i)}}{\sum_{i=1}^{N} [\![y_{ik} < 0]\!] e^{-f_M(\mathbf{x}_i)}}.$$

The condition (3) is satisfied by this solution, as proved in [13].

3.1 Single Rule Generation

In the remainder, we use the following notation: $\Phi(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_j)$, $W_> = \sum_{y_{ij}>0} [\![\Phi(\mathbf{x}_i, \mathbf{x}_j) > 0]\!] w_{ij}^{(m)}$, $W_< = \sum_{y_{ij}>0} [\![\Phi(\mathbf{x}_i, \mathbf{x}_j) < 0]\!] w_{ij}^{(m)}$, $W_0 = \sum_{y_{ij}>0} [\![\Phi(\mathbf{x}_i, \mathbf{x}_j) = 0]\!] w_{ij}^{(m)}$, and $W = W_> + W_< + W_0$.

Thus, we can rewrite the term minimized in (7) to $e^{-\alpha} W_> + e^{\alpha} W_< + W_0$. A decision rule is obtained by an approximate solution of this minimization problem. We separately treat the problem of finding α_m and Φ_m.

Let us remark that for given Φ_m the problem of finding α_m becomes much simpler. After simple calculations, one obtain a closed-form solution:

$$\alpha_m = \frac{1}{2} \ln \frac{W_>}{W_<}. \tag{9}$$

To avoid division by zero, we add to the numerator and denominator a small positive value that regularizes the estimate of the response.

The challenge is to find Φ_m. To this end, we derive an impurity measure $\mathcal{L}_m(\Phi)$ from (7) in such a way that the minimization problem does not longer depend on α. For given $\mathcal{L}_m(\Phi)$ the procedure constructing Φ_m resembles the way in which decision trees are built. It constructs only one path from the root to the leaf. At the beginning, Φ_m is empty and in each subsequent step an elementary condition $x_p \in S_p$ minimizing $\mathcal{L}_m(\Phi)$ is added to Φ_m. This procedure ends if $\mathcal{L}_m(\Phi)$ cannot be decreased. Let us underline that minimal value of $\mathcal{L}_m(\Phi)$ is a natural stop criterion of the procedure and no other parameters have to be

specified. This is due to the fact that rules represent a natural trade-off between misclassification and coverage of the rule.

Below, we analyze four techniques for deriving the form of the impurity measure that have been often used in the boosting algorithms. We show the relations between them by focusing on the above mentioned trade-off. Because of the nature of the algorithm, we analyze the coverage with respect to pairs of training examples. Moreover, these pairs are represented mainly by the weights w_{ij}. However, there is a clear relationship between coverage of pairs and single examples. The presented results generalize to some extent results obtained for binary classification [11].

Simultaneous minimization. Putting optimal value of α_m given by (9) into (7) results in the following impurity measure:

$$\mathcal{L}_m(\Phi) = 2\sqrt{W_> W_<} + W_0.$$

This can be simplified by using short multiplication formulas and replacing W_0 by $W - W_> - W_<$ (remark that W is constant in a given iteration) to:

$$\mathcal{L}_m(\Phi) = -|\sqrt{W_>} + \sqrt{W_<}|. \tag{10}$$

We called this technique *simultaneous minimization*, since for both parameters the closed-form solution exists.

Gradient descent. This technique approximates (7) up to the first order with respect to α:

$$r_m \simeq \arg\min_{\Phi,\alpha} \sum_{y_{ij}>0} \left(w_{ij}^{(m)} - \alpha\Phi(\mathbf{x}_i, \mathbf{x}_j) w_{ij}^{(m)} \right). \tag{11}$$

It is easy to see that the optimal solution with respect to Φ is obtained by minimizing:

$$\mathcal{L}_m(\Phi) = -\Big| \sum_{y_{ij}>0} \Phi(\mathbf{x}_i, \mathbf{x}_j) w_{ij}^{(m)} \Big| = -|W_> + W_<|, \tag{12}$$

since sign and magnitude of α may be established afterwards. We can now state the following relation between simultaneous minimization and gradient descent:

Theorem 1. *Consider minimization of the rank loss based on the exponential loss on the training set. Let Φ_m^{GD} be the optimal condition part obtained by minimization of (12):*

$$\Phi_m^{GD} = \arg\min_{\Phi} -|W_> + W_<|, \tag{13}$$

and let Φ_m^{SM} be the optimal condition part obtained by minimization of (10):

$$\Phi_m^{SM} = \arg\min_{\Phi} -|\sqrt{W_>} + \sqrt{W_<}|. \tag{14}$$

Then, the following holds:

$$W_{>}^{SM} + W_{<}^{SM} \leq W_{>}^{GD} + W_{<}^{GD},$$

where $W_{>}^{SM}$, $W_{<}^{SM}$, $W_{>}^{GD}$, $W_{<}^{GD}$ *denote the sum of respective weights in optimal solutions for* Φ_m^{SM} *and* Φ_m^{GD}.

Proof. Denoting by $W_+ = \max(W_>, W_<)$ and by $W_- = \min(W_>, W_<)$, we have from (13) that:

$$-W_+^{GD} + W_-^{GD} \leq -W_+^{SM} + W_-^{SM}, \tag{15}$$

and from (14), we have that:

$$-\sqrt{W_+^{GD}} + \sqrt{W_-^{GD}} \geq -\sqrt{W_+^{SM}} + \sqrt{W_-^{SM}}. \tag{16}$$

Since all the sums of weights are positive, we can use $\sqrt{W_-^{GD}} \geq \sqrt{W_+^{GD}} - \sqrt{W_+^{SM}} + \sqrt{W_-^{SM}}$ in $-W_+^{GD} \leq -W_+^{SM} + W_-^{SM} + W_-^{GD}$. In result we obtain that $\sqrt{W_+^{GD}} \geq \sqrt{W_+^{SM}}$. From this and (16) we have immediately that $\sqrt{W_-^{GD}} \geq \sqrt{W_-^{SM}}$. Finally, we get $W_+^{SM} + W_-^{SM} \leq W_+^{GD} + W_-^{GD}$, as claimed. □

Value of $W_> + W_<$ is the (weighted) number of pairs of objects affected by the rule; therefore it can be regarded as a measure of rule coverage. Thus, Theorem 1 shows that gradient descent produces more "general" rules (with larger coverage) than simultaneus minimization.

We can also easily prove that the gradient descent technique determines a precise trade-off between misclassified and uncovered pairs of training examples.

Theorem 2. *Let* $W_- = \min(W_>, W_<)$*, then minimization of (12) is equivalent to minimization of:*

$$W_- + \frac{1}{2} W_0. \tag{17}$$

Proof. Using W_- in (12), and substituting $\max(W_>, W_<)$ by $W - W_- - W_0$ results in $W_- + \frac{1}{2} W_0$, since W is constant in a given iteration. □

This theorem has a nice interpretation: the term W_- corresponds to pairs "misclassified" by the rule, while the second term – to pairs that are not "classified" by the rule at all. Value $\frac{1}{2}$ plays the role of a penalty for abstaining from classification and establishes a trade-off between not classified and misclassified pairs.

Gradient boosting. Gradient boosting is similar to gradient descent, but in this case we fit a rule to the negative gradient by minimizing the squared-error criterion:

$$r_m(\mathbf{x}) \simeq \arg\min_{\Phi,\alpha} \sum_{y_{ij}>0} \left(w_{ij}^{(m)} - \alpha \Phi(\mathbf{x}_i, \mathbf{x}_j) \right)^2. \tag{18}$$

This problem can be solved for $\alpha = \frac{W_> + W_<}{\sum_{y_{ij}>0} |\Phi_m(\mathbf{x}_i, \mathbf{x}_j)|}$. Therefore, we finally obtain:

$$\mathcal{L}_m(\Phi) = -\frac{\left|W_> + W_<\right|}{\sqrt{\sum_{y_{ij}>0} |\Phi_m(\mathbf{x}_i, \mathbf{x}_j)|}}, \tag{19}$$

which is equivalent to (12) normalized by the square root of the rule coverage. In other words, this technique produces more specific rules than gradient descent.

Constant-step minimization. In this technique, we restrict α in (7) to $\alpha \in \{-\beta, \beta\}$, where β is a fixed parameter of the algorithm. In result we obtain:

$$\mathcal{L}_m(\Phi) = e^{\mp\beta} W_> + e^{\pm\beta} W_< + W_0. \tag{20}$$

We can easily show that the constant-step minimization generalizes the gradient descent technique.

Theorem 3. *Solution of (20) for the exponential loss and step length β is equivalent to minimization of*

$$W_- + \ell W_0, \tag{21}$$

where W_- is defined as before and $\ell = \frac{1-e^{-\beta}}{e^{\beta}-e^{-\beta}}$.

Proof. Using W_- in (20), and substituting $\max(W_>, W_<)$ by $W - W_- - W_0$, we get

$$(e^{\beta} - e^{-\beta})W_- + (1 - e^{-\beta})W_0 + e^{-\beta}W.$$

The last element does not change the solution, so it suffices to minimize the first two terms. Moreover, dividing by $(e^{\beta} - e^{-\beta})$ we obtain: $\mathcal{L}_m(\Phi) = W_- + \ell W_0$, where $\ell = \frac{1-e^{-\beta}}{e^{\beta}-e^{-\beta}}$, as claimed. □

It is easy to see that for $\beta > 0$, $\ell \in [0, 0.5)$. Expression (21) has a similar interpretation as (17), but with varying value of ℓ. Increasing ℓ (or decreasing β) results in more general rules, covering more pairs. For $\beta \to 0$ we get the gradient descent technique. This means that gradient descent produces the most general (in the sense of coverage) rules.

3.2 Fast Implementation of RankRules

We can notice that the complexity of all impurity measures defined above is quadratic $O(N^2)$ with respect to the number of training examples N. However, we can reduce the complexity to $O(KN)$. First of all, let us remark that the minimization problem defined in m-th iteration (7) can be rewritten to:

$$\begin{aligned} r_m &= \arg\min_{\Phi,\alpha} \sum_{y_{ij}>0} w_{ij}^{(m)} e^{-\alpha(\Phi_m(\mathbf{x}_i) - \Phi_m(\mathbf{x}_j))} = \arg\min_{\Phi,\alpha} e^{-\alpha} W_> + e^{\alpha} W_< + W_0 \\ &= \arg\min_{\Phi,\alpha} \sum_{k=1}^{K-1} \left(e^{-\alpha} W_k^+ W_k^{0-} + e^{\alpha} W_k^- W_k^{0+} + W_k^+ W_k^- + W_k^{0+} W_k^{0-} \right), \end{aligned}$$

where $W_k^{0-} = \sum_{\Phi(\mathbf{x}_i)=0} [\![y_{ik} < 0]\!] w_i^{(m-)}$, $W_k^{-} = \sum_{\Phi(\mathbf{x}_i)=1} [\![y_{ik} < 0]\!] w_i^{(m-)}$, $W_k^{+} = \sum_{\Phi(\mathbf{x}_i)=1} [\![y_{ik} > 0]\!] w_i^{(m)}$, $W_k^{0+} = \sum_{\Phi(\mathbf{x}_i)=0} [\![y_{ik} > 0]\!] w_i^{(m)}$, with $w_i^{(m)} = e^{-f_{m-1}(\mathbf{x}_i)}$, $w_j^{(m-)} = e^{f_{m-1}(\mathbf{x}_j)}$. These values depend on single examples (not pairs) and can be easily computed and updated in each iteration.

Computation of α requires then a linear time with respect to the number of training examples:

$$\alpha_m = \frac{1}{2} \ln \frac{\sum_{k=1}^{K-1} W_k^{+} W_k^{0-}}{\sum_{k=1}^{K-1} W_k^{-} W_k^{0+}}.$$

To reduce the complexity, we also have to find the way in which $\mathcal{L}_m(\Phi)$ can be efficiently computed. This requires the training examples to be sorted on each attribute, since the elementary conditions to be added to Φ_m are tested consecutively for all values (midpoints between training examples) on all condition attributes. The sorting phase can be performed once before generating any rule. For each consecutive candidate elementary condition, the coverage of new Φ then differs with one training example.

Let us focus on the gradient descent technique, for simplicity, but the main idea is the same for other techniques. In this case, one minimizes $-|W_> + W_<|$. Let us observe that we can write:

$$W_> = \sum_{k=1}^{K-1} W_k^{+} W_k^{0-} \quad W_< = \sum_{k=1}^{K-1} W_k^{-} W_k^{0+}.$$

All these values can be easily updated by adding or subtracting $w_j^{(m)}$ or $w_j^{(m-)}$, for any consecutive candidate Φ.

3.3 Regularization

The generalization ability of the rule ensemble can be increased by performing three steps regularizing the solution.

The first one consists in shrinking [14] a newly generated rule $r_m(\mathbf{x}) = \alpha_m \Phi_m(\mathbf{x})$ towards rules already present in the ensemble:

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \nu \cdot r_m(\mathbf{x}),$$

where $\nu \in (0, 1]$ is a shrinkage parameter that can be regarded as controlling the learning rate. For small ν we obtain accurate ensemble, but less interpretable, since we need to generate more rules.

The algorithm works better when decision rules are less correlated. That is why the procedure for finding Φ_m can work on a subsample of original data that is a fraction ζ of all training examples, drawn without replacement [15]. Such an approach leads to a set of rules that are more diversified and less correlated. Moreover, finding Φ_m on a subsample reduces the computational costs. However, we pay once again the price of the interpretability.

Independently of the fact whether Φ_m was found using a subsample or not, value of α_m is calculated on *all* training examples in the introduced algorithm.

This usually decreases $|\alpha_m|$, and avoids overfitting the rule to the training set. These three elements (shrinking, sampling, and calculating α_m on the entire training set) constitute an alternative technique to post-pruning often used in rule induction algorithms.

4 Experimental Results

We thoroughly tested the RankRules algorithm in ordinal classification tasks. The first experiment was performed on artificial data. Based on the results of this experiment, we selected the default parameters for the algorithm. In the second experiment, we compared RankRules to other methods using benchmark data sets. In all the experiments, we measured the mean absolute error (MAE).

4.1 Artificial Data

The artificial data were generated by the following model. Examples $\mathbf{x} \in \mathbb{R}^n$ are drawn according to the normal distribution, $\mathbf{x} \sim N(0, \mathbf{I})$, where $\mathbf{I}$ is a unit matrix of size n. Each example belongs to one of the classes $y \in \{1, \ldots, 10\}$. Assume that the function $f(\mathbf{x}) \in \mathbb{R}$ can be transformed to conditional probabilities $\Pr(y > k|\mathbf{x})$ by $\log \frac{\Pr(y>k|\mathbf{x})}{\Pr(y\leq k|\mathbf{x})} = \pi(f(\mathbf{x}) - \theta_k), \quad k = 1, \ldots, 9$, where π corresponds to the level of noise, measured by the Bayes risk of the absolute-error that we set to 1. We defined function $f(\mathbf{x})$ to be: $f(\mathbf{x}) = x_1 - x_2 + 0.2(x_3 - x_4) + 5e^{-(x_5^2 + x_6^2 + 0.2x_7^2)} - \prod_{j=8}^{10} I(-0.5 \leq x_j \leq 0.5) + I(x_{11} \geq 0 \wedge x_{12} \geq 0) - I(x_{13} \geq 0 \wedge x_{14} \geq 0)$. Notice that this function contains linear terms (which are hard to approximate by trees and rules), Gaussian term (ball in the coordinate origin), cube and two rectangles. Thresholds $\boldsymbol{\theta} = \{\theta_k\}_0^{10}$ are chosen so that the prior probabilities of all classes are approximately equal. We generated 30 training and testing sets according to this model. Each of them consisted of 1000 examples.

First, we examined the effect of regularization. We set for the regularized algorithm some ad hoc values corresponding to high regularization. These are $\nu = 0.1$ (shrinkage parameter), and $\zeta = 0.25$ (fraction of training examples). We used 6 instances of the algorithm, each using different minimization technique: simultaneous minimization (SM), gradient descent (GD), gradient boosting (GB), and constant-step minimization with $\beta = 0.1$, 0.2, and 0.5. We generated up to $M = 1000$ rules. Top panels of Fig. 1 present the error curves, i.e., error as a function of M. The results are also summarized in Table 1, in which MAE for $M = 1000$, standard errors and computing times are given. We see that regularization improves results. The error curves are much smoother. Simultaneous minimization seems to perform the best. We can observe that higher values of β increase the error. This is, however, partially due to the greedy nature of the procedure building the single rule. Taking a subsample of training examples speeds up the algorithm. In bottom left panel of Fig. 1, we also show the computing time as a function of N (size of the training set) for two instances of RankRules (for $\zeta = 1.0$ and $\zeta = 0.5$). We see that the algorithm scales linearly.

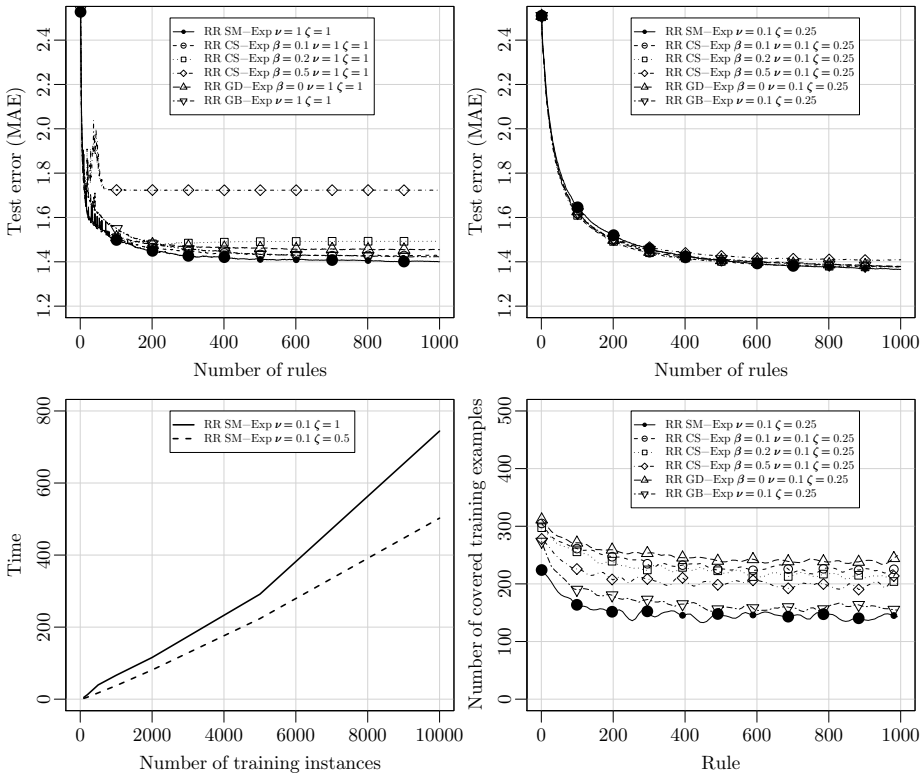

Fig. 1. Comparison of unregularized (on top left) and regularized (on top right) RankRules on artificial data. Computing time (bottom left panel) as a function of the size of the training set. Coverage of rules (bottom right panel) for the regularized algorithms with different minimization techniques (lines are smoothed).

We also verified the theoretical results concerning the relation between minimization techniques. Bottom right panel of Fig. 1 shows the coverage of the rules for different variants of RankRules. Since the algorithm works on pairs of examples, we plot the number of examples covered by the rule or placed outside the rule, dependently on which value is smaller. We see that gradient descent generates the most general rules. Length of the step in constant-step minimization controls the coverage.

Finally, we selected the default parameters for the algorithm. We tested 6 variants of RankRules as before with all combinations of the following values of the parameters: $\nu \in \{1, 0.5, 0.2, 0.1\}$, $\zeta \in \{1, 0.75, 0.5, 0.25\}$. For each algorithm, an error curve over 30 trials was drawn showing MAE for M up to 1000. Using these curves, the best variant of the algorithm was chosen. This is simultaneous minimization (SM) with $\nu = 0.1$ and $\zeta = 0.5$.

Table 1. Test errors (MAE) and standard errors for regularized and unregularized RankRules. Computing time in seconds is given obtained on AMD Opteron 250 Processor 2.5 GHz with 8 GB of RAM running MS Windows Server 2003.

RankRules	Unregularized		Regularized	
	MAE	Time	MAE	Time
SM	1.40±0.010	76.75	1.37±0.008	20.45
CS $\beta = 0.1$	1.43±0.010	86.61	1.38±0.008	18.11
CS $\beta = 0.2$	1.49±0.011	66.94	1.38±0.007	18.45
CS $\beta = 0.5$	1.72±0.019	46.52	1.41±0.008	26.14
GD $\beta = 0.0$	1.46±0.008	66.00	1.38±0.008	15.69
GB	1.42±0.009	83.57	1.38±0.008	21.80

Table 2. Data sets used in the experiment. First columns describe data sets, n denotes number of attributes, N – number of training examples, and T – number of testing examples. The first eight data sets were used in [16, 3]

.

Data set	n	N	T	Data set	n	N	T
Pyrim	27	50	24	Auto Price	15	100	59
Machine cpu	6	150	59	Bank8FM	8	3000	5192
Housing	13	300	206	CPU small	12	4000	4192
Abalone	8	1000	3177	Delta ailerons	5	3500	3629
Bank32nh	32	3000	5192	Elevators	18	4000	4752
CPU act	22	4000	4192	House 8L	8	6000	16784
Cal housing	8	5000	15640	Stock	9	450	500
House 16H	16	6000	16784	Triazines	60	100	86
2dplanes	10	10000	30768	Wisconsin	31	150	44
AutoMpg	7	250	148				

4.2 Benchmark Data Sets

In the second part of the experiment, RankRules is tested on 19 benchmark data sets[4]. These data concern originally metric regression problems. We transform and prepare these data in a similar way as in [16, 3]. The dependent variables are discretized into ten ordered class labels using equal-frequency binning. Tests are performed on 20 random splits for each file. The size of training and test partitions are the same as in the referred papers. Data sets are summarized in Table 2.

First we compared the following algorithms:

– RankBoost AE [3]: the boosting algorithm based on the exponential rank loss; the thresholds are computed in order to minimize the absolute-error loss by dynamic programming; the base classifiers are perceptron and sigmoid functions, the algorithm generates 2000 base classifier.

[4] Data sets are taken from `http://www.liacc.up.pt/~ltorgo/Regression/DataSets.html`.

Table 3. Comparison of MAE between RankRules and algorithms introduced in [16, 3]. Means over 20 trials with ranks (in parentheses) are given.

Data set	RankRules	RankBoost AE (percept.)	RankBoost AE (sigmoid)	SVM-IMC	ORBoost-All (percept.)	ORBoost-All (sigmoid)
Pyrim	1.423(4)	1.619(6)	1.59(5)	1.294(1)	1.36(2)	1.398(3)
Machine CPU	0.903(2)	1.573(6)	1.282(5)	0.99(4)	0.889(1)	0.969(3)
Housing	0.811(4)	0.842(5)	0.892(6)	0.747(1)	0.791(3)	0.777(2)
Abalone	1.259(1)	1.517(5)	1.738(6)	1.361(2)	1.432(4)	1.403(3)
Bank32nh	1.608(4)	1.867(5)	2.183(6)	1.393(1)	1.49(2)	1.539(3)
CPU act	0.573(1)	0.841(5)	0.945(6)	0.596(2)	0.626(3)	0.634(4)
Cal housing	0.948(2)	1.528(6)	1.251(5)	1.008(4)	0.977(3)	0.942(1)
House 16h	1.156(1)	2.008(6)	1.796(5)	1.205(3)	1.265(4)	1.198(2)
Ave. Rank	(2.375)	(5.5)	(5.5)	(2.25)	(2.75)	(2.625)

- ORBoost-All [3]: the boosting algorithm applied to threshold loss based on the exponential loss; the base classifiers are perceptron and sigmoid functions; the algorithm generates 2000 base classifier.
- SVM-IMC [16]: this is SVM that minimizes threshold loss based on the hinge loss; the Gaussian kernel is used, the optimal parameters are determined in 5-fold cross-validation.
- RankRules: we use $\nu = 0.1$, $\zeta = 0.5$, and $M = 500$ (above this value the algorithm did not give significant improvement in the experiment on artificial data sets).

All these algorithms are tailored for minimizing the absolute-error loss. Table 3 contains MAE averaged over 20 trials and ranks of the algorithms on the first eight data sets from Table 2 that were used in the referred papers. Friedman test shows that there is a significant difference between classifiers, since Friedman statistics gives 27.786 which exceeds the critical value 11.07 (for confidence level 0.05). This is due to a poor performance of RankBoost AE. All other algorithms are significantly better than this classifier (critical difference is CD = 2.666, computed according to Nemenyi's test), while there is no significant difference among them. It is rather curious that RankBoost AE performs so poorly, since RankRules is quite similar and obtains much better results. It follows that we got an opposite result than [3], in which a conclusion is given that rank loss minimization performs worse than threshold loss minimization. The main difference between RankBoost AE and RankRules stems from the choice of base classifiers, and in the way in which the thresholds are computed. ORBoost-All, using the same base classifiers as RankBoost AE, gets better results; that is why we conclude that the second difference can be more important.

From the above result, we can see that there is no significant difference between various approaches to ordinal classification. We tried to verify this in an enlarged experiment that we constrained to one type of classifier. We used other three rule ensemble algorithms that work in similar way to RankRules, but minimize different loss functions:

– ENDER AE [17] – this algorithm directly minimizes absolute error using the gradient descent technique.
– Ordinal ENDER – the ENDER algorithm [11] being similar to MLRules [9] is used to $K-1$ binary problems in which the probabilities of $\Pr(y > k|\mathbf{x})$ are estimated. From those probabilities, the distribution $p_k(\mathbf{x})$ is obtained and the median over this distribution is computed.
– ORDER [18] – this algorithm minimizes the exponential threshold loss using the constant-step minimization.

The parameters of these algorithms were obtained on the artificial data set in the same way as in the case of RankRules.

Test errors and ranks of four rule ensemble are given in Table 4. The Friedman statistics gives 25.844 which exceeds the critical value 7.814 (for confidence level 0.05), and the critical difference rises CD = 1.076. In result we have that ENDER AE is significantly outperformed by all other algorithms, and there is no significant difference between Ordinal ENDER, ORDER and RankRules. This is due to the fact that ENDER AE is mainly suited to ordinary regression tasks, but the rest of algorithms is suited to ordinal classification problems. Trying to compare further Ordinal ENDER, ORDER and RankRules, we can mention some qualitative differences. RankRules can minimize the exponential loss in simultaneous way, but ORDER has to perform approximation (this causes that ORDER-E, which performed the best in the experiment reported in [18], is much slower; other techniques like constant-step are much faster). Moreover, RankRules is directly related to maximization of AUC, while ENDER and MLRules to probability estimation.

Table 4. Comparison of four rule ensemble algorithms. Test errors (MAE) and ranks (in parenthesis) are given.

Data set	ENDER AE		Ordinal ENDER		ORDER		RankRules	
2dplanes	0.581	(4)	0.508	(1)	0.513	(2)	0.517	(3)
Abalone	1.34	(4)	1.265	(3)	1.248	(1)	1.259	(2)
AutoMpg	0.814	(4)	0.743	(3)	0.729	(2)	0.707	(1)
Auto Price	0.814	(4)	0.777	(3)	0.748	(2)	0.744	(1)
Bank32nh	1.671	(4)	1.591	(2)	1.545	(1)	1.608	(3)
Bank8FM	0.624	(4)	0.479	(2)	0.485	(3)	0.464	(1)
Cal housing	1.17	(4)	0.922	(1)	0.994	(3)	0.948	(2)
CPU act	0.741	(4)	0.567	(1)	0.602	(3)	0.573	(2)
CPU small	0.827	(4)	0.661	(1)	0.689	(3)	0.67	(2)
Delta ailerons	0.943	(4)	0.862	(1)	0.885	(2)	0.895	(3)
Elevators	1.407	(4)	1.238	(2)	1.251	(3)	1.233	(1)
House 16H	1.264	(4)	1.116	(1)	1.181	(3)	1.156	(2)
House 8L	1.242	(4)	1.135	(1)	1.183	(3)	1.161	(2)
Housing	0.901	(4)	0.768	(1)	0.795	(2)	0.811	(3)
Machine cpu	0.956	(4)	0.84	(1)	0.871	(2)	0.903	(3)
Pyrim	1.43	(3)	1.431	(4)	1.26	(1)	1.423	(2)
Stock	0.467	(4)	0.28	(1)	0.329	(2)	0.338	(3)
Triazines	1.975	(4)	1.878	(3)	1.874	(2)	1.808	(1)
Wisconsin	2.338	(1)	2.572	(4)	2.46	(2)	2.526	(3)
Average Rank		(3.789)		(1.895)		(2.211)		(2.105)

5 Conclusions

We introduced a learning algorithm called RankRules, constructing an ensemble of decision rules for ordinal classification problems. This algorithm employs the notion of the rank loss. We considered four minimization techniques that result in different forms of the impurity measure used for construction of a single decision rule. We showed the relations between different impurity measures, emphasizing the trade-off between misclassification and coverage of the rule. We also showed that RankRules minimizes the rank loss in linear time (if the attribute values are already sorted), despite of the fact that in general this problem is quadratic. In the experiment, we reported that rank loss minimization is competitive to two other approaches used in the ordinal classification settings. However, there is no significant difference between those approaches.

References

1. Herbrich, R., Graepel, T., Obermayer, K.: Regression Models for Ordinal Data: A Machine Learning Approach. Technical report, TU Berlin (1999)
2. Rennie, J., Srebro, N.: Loss functions for preference levels: Regression with discrete ordered labels. In: IJCAI 2005 M-PREF. (2005)
3. Lin, H.T., Li, L.: Large-Margin Thresholded Ensembles for Ordinal Regression: Theory and Practice. In: ALT. Volume 4264 of LNAI. (2006) 319–333
4. Freund, Y., Iyer, R., Schapire, R.E., Singer, Y.: An Efficient Boosting Algorithm for Combining Preferences. JMLR **6**(4) (2003) 933–969
5. Joachims, T.: Training Linear SVMs in Linear Time. In: ACM KDD. (2006) 217–226
6. Fürnkranz, J.: Separate-and-Conquer Rule Learning. AI Review **13**(1) (1996) 3–54
7. Cohen, W.W., Singer, Y.: A Simple, Fast, and Effective Rule Learner. In: AAAI. (1999) 335–342
8. Friedman, J.H., Popescu, B.E.: Predictive Learning via Rule Ensembles. Ann. Appl. Stat. **2**(3) (2008) 916–954
9. Dembczyński, K., Kotłowski, W., Słowiński, R.: Maximum Likelihood Rule Ensembles. In: ICML. (2008) 224–231
10. Flach, P.A., Fürnkranz, J.: ROC'n'Rule Learning - Towards a Better Understanding of Covering Algorithms. Mach. Learn. **58** (2005) 39–77
11. Dembczyński, K., Kotłowski, W., Słowiński, R.: A General Framework for Learning an Ensemble of Decision Rules. In: LeGo, ECML/PKDD 2008 Workshop. (2008)
12. Janssen, F., Fürnkranz, J.: An empirical quest for optimal rule learning heuristics. Technical report, TU Darmstadt (2008)
13. Lin, H.T., Li, L.: Ordinal Regression by Extended Binary Classifications. In: NIPS. (2007) 865–872
14. Hastie, T., Tibshirani, R., Friedman, J.H.: Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer (2003)
15. Friedman, J.H., Popescu, B.E.: Importance Sampled Learning Ensembles. Technical report, Stanford University (2003)
16. Chu, W., Keerthi, S.S.: New Approaches to Support Vector Ordinal Regression. In: ICML. (2005) 321–328

17. Dembczyński, K., Kotłowski, W., Słowiński, R.: Solving Regression by Learning an Ensemble of Decision Rules. In: Artificial Intelligence and Soft Computing. Volume 5097 of LNAI., Springer-Verlag (2008) 533–544
18. Dembczyński, K., Kotłowski, W., Słowiński, R.: Ordinal Classification with Decision Rules. In: Mining Complex Data 2007. Volume 4944 of LNAI., Springer (2008) 169–181