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

Krzysztof Dembczyński\textsuperscript{1,2}, Wojciech Kotłowski\textsuperscript{1,3}

\textsuperscript{1}Institute of Computing Science, Poznań University of Technology
\textsuperscript{2}KEBI, Philipps-Universität in Marburg
\textsuperscript{3}Centrum Wiskunde & Informatica, Amsterdam

PL-09, Bled, September 11, 2009
1 Ordinal Classification

2 RankRules

3 Conclusions
1 Ordinal Classification

2 RankRules

3 Conclusions
Ordinal classification consists in predicting a label taken from a finite and ordered set for an object described by some attributes.

This problem shares some characteristics of multi-class classification and regression, but:

- the order between class labels cannot be neglected,
- the scale of the decision attribute is not cardinal.
Recommender system predicting a rating of a movie for a given user.
Email filtering to ordered groups like: important, normal, later, or spam.
**Denotation:**

- $K$ – number of classes
- $y$ – actual label
- $x$ – attributes
- $\hat{y}$ – predicted label
- $F(x)$ – prediction function
- $f(x)$ – ranking or utility function
- $\theta = (\theta_0, \ldots, \theta_K)$ – thresholds
- $L(\cdot)$ – loss function
- $[\cdot]$ – Boolean test
- $\{y_i, x_i\}_{1}^{N}$ – training examples
Ordinal Classification:

- Since $y$ is discrete, it obeys a **multinomial distribution** for a given $x$:

  $$p_k(x) = \Pr(y = k|x), \quad k = 1, \ldots, K.$$ 

- The **optimal prediction** is clearly given by:

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

  where $L(y, \hat{y})$ is the **loss function** defined as a matrix:

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

  with **v-shaped rows** and zeros on diagonal.

$$L(y, \hat{y}) = \begin{pmatrix}
0 & 1 & 2 \\
1 & 0 & 1 \\
2 & 1 & 0
\end{pmatrix}$$
Ordinal Classification:

- A natural choice of the loss matrix is the **absolute-error loss** for which
  \[ l_{y,\hat{y}} = |y - \hat{y}|. \]

- The optimal prediction in this case is **median** over class distribution:
  \[ F^*(x) = \text{median}_{p_k(x)}(y). \]

- Median **does not depend** on a **distance** between **class labels**, so the scale of the decision attribute does not matter; the order of labels is taken into consideration only.
Two Approaches to Ordinal Classification:

- Threshold Loss Minimization (SVOR, ORBoost-All, MMMF),
- Rank Loss Minimization (RankSVM, RankBoost).

In both approaches, one assumes existence of:

- **ranking (or utility)** function $f(x)$, and
- **consecutive thresholds** $\theta = (\theta_0, \ldots, \theta_K)$ on a range of the ranking function,

and the final prediction is given by:

$$F(x) = \sum_{k=1}^{K} k\mathbb{1}[f(x) \in [\theta_{k-1}, \theta_k]].$$
Threshold Loss Minimization:

- **Threshold loss** function is defined by:

\[
L(y, f(x), \theta) = \sum_{k=1}^{K-1} [y_k(f(x) - \theta_k) \leq 0],
\]

where

\[
y_k = 1, \text{ if } y > k, \text{ and } y_k = -1, \text{ otherwise}.
\]

\[
\begin{align*}
\theta_0 &= -\infty & \cdots & \theta_1 &= -3.5 & \theta_2 &= -1.2 & \cdots & \theta_{k-1} &= 1.2 & \theta_{k-2} &= 3.8 & \cdots & \theta_K &= \infty
\end{align*}
\]
Rank Loss Minimization:

- **Rank loss** function is defined over pairs of objects:

\[
L(y_\circ, f(x_\circ), f(x_\bullet)) = \mathbb{I}[y_\circ (f(x_\circ) - f(x_\bullet)) \leq 0],
\]

where

\[
y_\circ = \text{sgn}(y_\circ - y_\bullet).
\]

- **Thresholds** are computed afterwards with respect to a given loss matrix.

\[
\begin{align*}
y_{i_1} & > y_{i_2} > y_{i_3} > \ldots > y_{i_{N-1}} > y_{i_N} \\
f(x_{i_1}) & > f(x_{i_3}) > f(x_{i_2}) > \ldots > f(x_{i_{N-1}}) > f(x_{i_N})
\end{align*}
\]
Comparison of the two approaches:

Threshold loss:
- Comparison of an object to thresholds instead to all other training objects.
- Weighted threshold loss can approximate any loss matrix.

Rank loss:
- Minimization of the rank loss on training set has quadratic complexity with respect to a number of object, however, in the case of $K$ ordered classes, the algorithm can work in linear time.
- Rank loss minimization is closely related to maximization of AUC criterion.
1 Ordinal Classification

2 RankRules

3 Conclusions
RankRules:

- Ranking function is an ensemble of decision rules:

\[ f(x) = \sum_{m=1}^{M} r_m(x), \]

where

\[ r_m(x) = \alpha_m \Phi_m(x) \]

is a decision rule defined by a response \( \alpha_m \in \mathcal{R} \), and an axis-parallel region in attribute space \( \Phi_m(x) \in \{0, 1\} \).

- Decision rule can be seen as logical pattern:

  \[ if \ [condition] \ then \ [decision]. \]
RankRules:

• RankRules follows the rank loss minimization.
• We use the boosting approach to learn the ensemble.
• The rank loss is upper-bounded by the exponential function:

\[ L(y, f) = \exp(-yf). \]

• This is a convex function, which makes the minimization process easier to cope with.
• Due to modularity of the exponential function, minimization of the rank loss can be performed in a fast way.
RankRules:

- In the $m$-th iteration, the rule is computed by:

$$r_m = \arg \min_{\Phi, \alpha} \sum_{y_{ij} > 0} w_{ij} e^{-\alpha(\Phi_m(x_i) - \Phi_m(x_j))},$$

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

$$w_{ij} = e^{-(f_{m-1}(x_i) - f_{m-1}(x_j))}$$

can be treated as weights associated with pairs of training examples.

- The overall loss changes only for pairs in which one example is covered by the rule and the other is not ($\Phi(x_i) \neq \Phi(x_j)$).
RankRules:

- Thresholds are **computed** by:

\[
\theta = \arg \min_{\theta} \sum_{i=1}^{N} \sum_{k=1}^{K-1} e^{-y_{ik}(f(x_i)-\theta_k)},
\]

subject to

\[
\theta_0 = -\infty \leq \theta_1 \leq \ldots \leq \theta_{K-1} \leq \theta_K = \infty.
\]

- The problem has a **closed-form solution**:

\[
\theta_k = \frac{1}{2} \log \frac{\sum_{i=1}^{N} [y_{ik} > 0] e^{f(x_i)}}{\sum_{i=1}^{N} [y_{ik} < 0] e^{-f(x_i)}}, \quad k = 1, \ldots, K - 1.
\]

- The monotonicity condition is **satisfied** by this solution as proved by Lin and Li (2007).
Single Rule Generation:

- The $m$-th rule is obtained by solving:

$$r_m = \arg \min_{\Phi, \alpha} \sum_{y_{ij} > 0} w_{ij} e^{-\alpha(\Phi_m(x_i) - \Phi_m(x_j))}.$$ 

- For given $\Phi_m$ the problem of finding $\alpha_m$ has a closed-form solution:

$$\alpha_m = \frac{1}{2} \ln \frac{\sum_{y_{ij} > 0 \land \Phi_m(x_i) > \Phi_m(x_j)} w_{ij}}{\sum_{y_{ij} > 0 \land \Phi_m(x_i) < \Phi_m(x_j)} w_{ij}}.$$ 

- The challenge is to find $\Phi_m$ by deriving the **impurity measure** $\mathcal{L}(\Phi_m)$ in such a way that the optimization problem does not longer depend on $\alpha_m$. 
**Boosting Approaches and Impurity Measures:**

- **Simultaneous minimization:** finds the closed-form solution for $\Phi$ (Confidence-rated AdaBoost, SLIPPER, RankBoost).

- **Gradient descent:** relies on approximation of the loss function up to the first order (AdaBoost, AnyBoost).

- **Gradient boosting:** minimizes the squared-error between rule outputs and the negative gradient of the loss function (Gradient Boosting Machine, MART).

- **Constant-step minimization:** restricts $\alpha \in \{-\beta, \beta\}$, with $\beta$ being a fixed parameter.
Boosting Approaches and Impurity Measures:

- Each of the boosting approaches provides another impurity measure that represents different trade-off between misclassification and coverage of the rule.
- **Gradient descent** produces the most general rules in comparison to other techniques.
- **Gradient descent** represents $\frac{1}{2}$ trade-off between misclassification and coverage of the rule.
- **Constant-step minimization** generalizes the gradient descent technique to obtain different trade-offs between misclassification and coverage of the rule, namely $\ell \in [0, 0.5)$, with
  \[
  \beta = \ln \frac{1 - \ell}{\ell}.
  \]
Fast Implementation:

• We rewrite the minimization problem of complexity $O(N^2)$:

$$r_m = \arg \min_{\Phi, \alpha} \sum_{y_{ij}>0} w_{ij} e^{-\alpha(\Phi_m(x_i) - \Phi_m(x_j))},$$

to the problem that can be solved in $O(KN)$.

• We use the fact that

$$w_{ij} = e^{-(f_{m-1}(x_i) - f_{m-1}(x_j))} = e^{-f_{m-1}(x_i)} e^{f_{m-1}(x_j)} = w_i w_j^-,$$

and use denotation:

$$W_k = \sum_{y_i=k \land \Phi(x_i)=1} w_i^-, \quad W_k^0 = \sum_{y_i=k \land \Phi(x_i)=0} w_i^-.$$
Fast Implementation:

• The minimization problem can be rewritten to

\[ r_m = \arg \min_{\Phi, \alpha} \sum_{i=1}^{N} w_i e^{-\alpha(\Phi_m(x_i))} \sum_{y_i > y_j} w_j e^{\alpha \Phi_m(x_j)}, \]

where the inner sum can be given by:

\[ \sum_{y_i > y_j} w_j e^{\alpha \Phi_m(x_j)} = e^{\alpha} \sum_{y_i > k} W_k + \sum_{y_i > k} W_0^k. \]

• The values

\[ W_k \quad \text{and} \quad W_0^k, \quad k = 1, \ldots, K, \]

can be easily computed and updated in each iteration.
Fast Implementation

![Graph showing the relationship between the number of training instances and time. The graph includes two lines representing different scenarios: one for RR SM−Exp $\nu = 0.1 \; \zeta = 1$ and another for RR SM−Exp $\nu = 0.1 \; \zeta = 0.5$.](image)
Regularization:

- The rule is **shrinked** (multiplied) by the amount \( \nu \in (0, 1] \) towards rules already present in the ensemble:

  \[
  f_m(x) = f_{m-1}(x) + \nu \cdot r_m(x).
  \]

- Procedure for finding \( \Phi_m \) works on a **fraction** \( \zeta \) of original data, drawn without replacement.

- Value of \( \alpha_m \) is calculated on **all** training examples – this usually decreases \(|\alpha_m|\) and plays the role of **regularization**.
Regularization:

![Graph showing test error (MAE) versus number of rules for different regularization methods and parameters. The graph includes lines for RR SM-Exp, RR CS-Exp, RR CS-Exp with different values of \( \beta \), RR GD-Exp, and RR GB-Exp. Each line represents a different combination of parameters, with labels indicating the values of \( \nu \) and \( \zeta \).]
Experimental Results:

RankRules vs. SVOR (Chu and Keerthi, 2005), RankBoost-AE and ORBoost-All (Lin and Li, 2006).

<table>
<thead>
<tr>
<th>Data set</th>
<th>RankRules (percpt.)</th>
<th>RankBoost AE (sigmoid)</th>
<th>SVOR (percpt.)</th>
<th>ORBoost-All (sigmoid)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pyrim</td>
<td>1.423(4)</td>
<td>1.619(6)</td>
<td>1.590(5)</td>
<td>1.360(2)</td>
</tr>
<tr>
<td>Machine CPU</td>
<td>0.903(2)</td>
<td>1.573(6)</td>
<td>1.282(5)</td>
<td>0.889(1)</td>
</tr>
<tr>
<td>Housing</td>
<td>0.811(4)</td>
<td>0.842(5)</td>
<td>0.892(6)</td>
<td>0.791(3)</td>
</tr>
<tr>
<td>Abalone</td>
<td>1.259(1)</td>
<td>1.517(5)</td>
<td>1.738(6)</td>
<td>1.432(4)</td>
</tr>
<tr>
<td>Bank32nh</td>
<td>1.608(4)</td>
<td>1.867(5)</td>
<td>2.183(6)</td>
<td>1.490(2)</td>
</tr>
<tr>
<td>CPU act</td>
<td>0.573(1)</td>
<td>0.841(5)</td>
<td>0.945(6)</td>
<td>0.626(3)</td>
</tr>
<tr>
<td>Cal housing</td>
<td>0.948(2)</td>
<td>1.528(6)</td>
<td>1.251(5)</td>
<td>0.977(3)</td>
</tr>
<tr>
<td>House 16H</td>
<td>1.156(1)</td>
<td>2.008(6)</td>
<td>1.796(5)</td>
<td>1.265(4)</td>
</tr>
<tr>
<td>Ave. Rank</td>
<td>(2.375)</td>
<td>(5.5)</td>
<td>(5.5)</td>
<td>(2.25)</td>
</tr>
</tbody>
</table>

- Ensembles of decision rules are **competitive** to the state-of-the-art algorithms.
- **Poor** performance of RankBoost AE (!?).
- Rank loss minimization performs **similarly** to the threshold loss minimization (opposite result to Lin and Li (2006)).
1 Ordinal Classification

2 RankRules

3 Conclusions
Conclusions:

- Two approaches to ordinal classification: **threshold loss** and **rank loss** minimization.
- Boosting-like algorithm for learning of **rule ensemble**.
- **Rule coverage** analysis of different boosting techniques.
- **Fast** implementation.
- RankRules are **competitive** to the state-of-the-art algorithms.
- **Nature of ordinal classification?**