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

Krzysztof Dembczyński[1,2], Wojciech Kotłowski[1,3]

[1]Institute of Computing Science, Poznań University of Technology
[2]KEBI, Philipps-Universität in Marburg
[3]Centrum Wiskunde & Informatica, Amsterdam

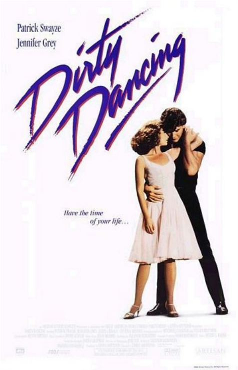
PL-09, Bled, September 11, 2009

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:

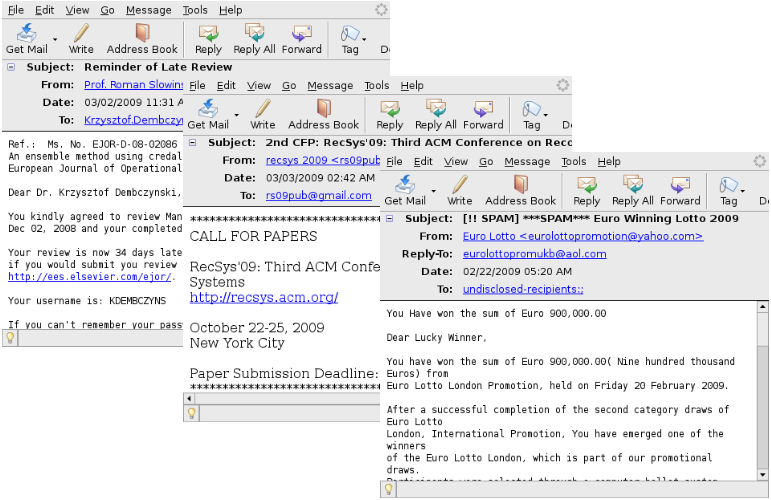
- the **order** between class labels **cannot** be **neglected**,
- the **scale** of the decision attribute is **not cardinal**.

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
- $\mathbf{x}$ – attributes
- $\hat{y}$ – predicted label
- $F(\mathbf{x})$ – prediction function
- $f(\mathbf{x})$ – ranking or utility function
- $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_K)$ – thresholds
- $L(\cdot)$ – loss function
- $[\![\cdot]\!]$ – Boolean test
- $\{y_i, \mathbf{x}_i\}_1^N$ – training examples

Ordinal Classification:

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

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

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

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

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

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

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

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

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:
$$F^*(\mathbf{x}) = \mathrm{median}_{p_k(\mathbf{x})}(y).$$
- Median **does not depend** on a **distance** between **class labels**, so the scale of the decision attribute does not matter; 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 $f(\mathbf{x})$, and
- **consecutive thresholds** $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_K)$ on a range of the ranking function,

and the final prediction is given by:

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

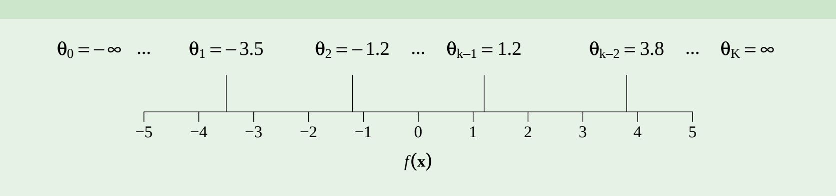
Threshold Loss Minimization:

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

$$L(y, f(\mathbf{x}), \boldsymbol{\theta}) = \sum_{k=1}^{K-1} [\![y_k(f(\mathbf{x}) - \theta_k) \leqslant 0]\!],$$

where

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



Rank Loss Minimization:

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

$$L\left(y_{\circ\bullet}, f(\mathbf{x}_\circ), f(\mathbf{x}_\bullet)\right) = [\![y_{\circ\bullet}(f(\mathbf{x}_\circ) - f(\mathbf{x}_\bullet)) \leqslant 0]\!],$$

  where

$$y_{\circ\bullet} = \operatorname{sgn}(y_\circ - y_\bullet).$$

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

$$\begin{array}{ccccccccccc} y_{i_1} & > & \color{red}{y_{i_2}} & > & \color{red}{y_{i_3}} & > \ldots > & y_{i_{N-1}} & > & y_{i_N} \\ f(\mathbf{x}_{i_1}) & > & \color{red}{f(\mathbf{x}_{i_3})} & > & \color{red}{f(\mathbf{x}_{i_2})} & > \ldots > & f(\mathbf{x}_{i_{N-1}}) & > & f(\mathbf{x}_{i_N}) \end{array}$$

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**.

RankRules:

- Ranking function is an **ensemble of decision rules**:

$$f(\mathbf{x}) = \sum_{m=1}^{M} r_m(\mathbf{x}),$$

  where

$$r_m(\mathbf{x}) = \alpha_m \Phi_m(\mathbf{x})$$

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

- Decision rule can be seen as **logical pattern**:

$$if \text{ [condition] } then \text{ [decision].}$$

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:

$$r_m = \arg\min_{\Phi,\alpha} \sum_{y_{ij}>0} w_{ij} e^{-\alpha(\Phi_m(\mathbf{x}_i)-\Phi_m(\mathbf{x}_j))},$$

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

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

  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 $(\Phi(\mathbf{x}_i) \neq \Phi(\mathbf{x}_j))$.

RankRules:

- Thresholds are **computed** by:

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

  subject to

$$\theta_0 = -\infty \leqslant \theta_1 \leqslant \ldots \leqslant \theta_{K-1} \leqslant \theta_K = \infty.$$

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

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

- 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:

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

- For given Φ_m the problem of finding α_m has a **closed-form solution**:

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

- The challenge is to find Φ_m by deriving the **impurity measure** $\mathcal{L}(\Phi_m)$ in such a way that the optimization problem does not longer depend on α_m.

Boosting Approaches and Impurity Measures:

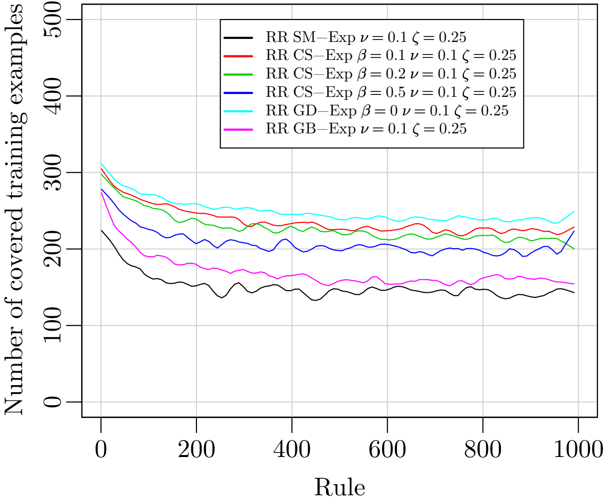
- **Simultaneous minimization**: finds the closed-form solution for Φ (Confidence-rated AdaBoost, SLIPPER, RankBoost).
- **Gradient descent**: relies on approximation of the loss function up to the first order (AdaBoost, AnyBoost).
- **Gradient boosting**: minimizes the squared-error between rule outputs and the negative gradient of the loss function (Gradient Boosting Machine, MART).
- **Constant-step minimization**: restricts $\alpha \in \{-\beta, \beta\}$, with β 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

$$\beta = \ln \frac{1 - \ell}{\ell}.$$

Rule Coverage (artificial data)



Fast Implementation:

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

$$r_m = \arg\min_{\Phi,\alpha} \sum_{y_{ij}>0} w_{ij} e^{-\alpha(\Phi_m(\mathbf{x}_i)-\Phi_m(\mathbf{x}_j))},$$

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

- We use the fact that

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

  and use denotation:

$$W_k = \sum_{y_i=k \wedge \Phi(\mathbf{x}_i)=1} w_i^-, \qquad W_k^0 = \sum_{y_i=k \wedge \Phi(\mathbf{x}_i)=0} w_i^-.$$

Fast Implementation:

- The minimization problem can be **rewritten** to

$$r_m = \arg\min_{\Phi,\alpha} \sum_{i=1}^{N} w_i e^{-\alpha(\Phi_m(\mathbf{x}_i))} \sum_{y_i > y_j} w_j^- e^{\alpha \Phi_m(\mathbf{x}_j)},$$

  where the **inner** sum can be given by:

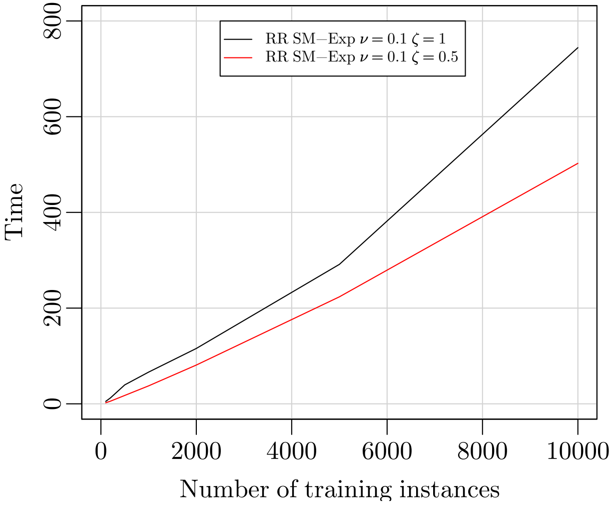
$$\sum_{y_i > y_j} w_j^- e^{\alpha \Phi_m(\mathbf{x}_j)} = e^{\alpha} \sum_{y_i > k} W_k + \sum_{y_i > k} W_k^0.$$

- The values

$$W_k \quad \text{and} \quad W_k^0, \quad k = 1, \ldots, K,$$

  can be **easily** computed and updated in each iteration.

Fast Implementation



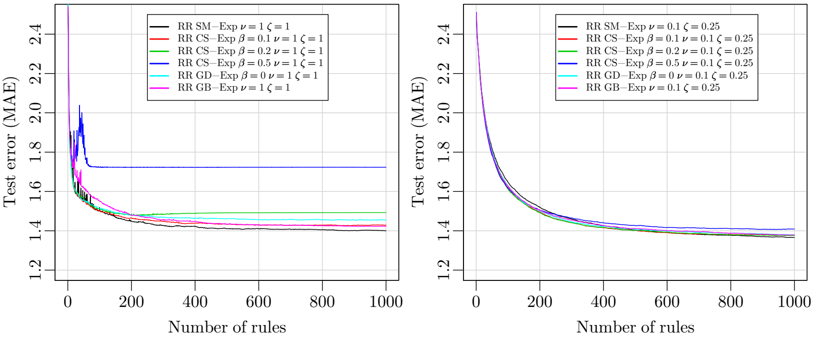
Regularization:

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

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

- Procedure for finding Φ_m works on a **fraction** ζ of original data, drawn without replacement.
- Value of α_m is calculated on **all** training examples – this usually decreases $|\alpha_m|$ and plays the role of **regularization**.

Regularization:



Experimental Results:

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

Data set	RankRules	RankBoost AE (percpt.)	RankBoost AE (sigmoid)	SVOR	ORBoost-All (percpt.)	ORBoost-All (sigmoid)
Pyrim	1.423(4)	1.619(6)	1.590(5)	1.294(1)	1.360(2)	1.398(3)
Machine CPU	0.903(2)	1.573(6)	1.282(5)	0.990(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)	1.490(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)

- 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)).

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**?