CS-E4710 Machine Learning: Supervised Methods

Lecture 12: Preference learning

Riikka Huusari & Juho Rousu

November 29, 2022

Department of Computer Science
Aalto University

Preference learning[1]

Preferences play a key role in various fields of application:

- Social networks (facebook, twitter,...)
- Recommender systems (Netflix,last.fm,...)
- Review web sites (tripadvisor,goodpubguide,...)
- Internet banner advertizing
- Electronic commerce (Amazon,...)
- Adaptive retrieval systems (e.g. Google personalized search)

[1] Huellermeyer & Fuernkrantz, Preference Learning: An Introduction, 2010

Preference learning

- Goal: learn a predictive preference model from observed preference information.
- Notation: A is preferred over B: $A \succ B$, alternatively we can say A is ranked above B

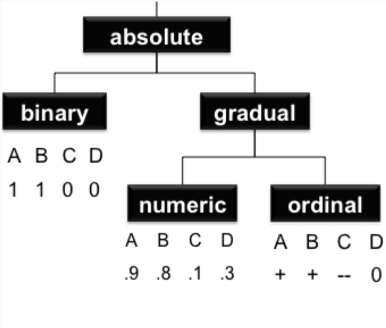
Preference learning tasks

- Object ranking: Given a set of inputs (objects), predict their order. Example: web search ranks results based on predicted relevance to a query
- Label ranking: Given an input, and a set of potential labels, predict the (relevance) order of the labels - generalization of multi-class classification
- Rating (also called Instance ranking): Given an input, assign it to one of pre-ordered categories, e.g. (very good, good, neutral, bad, very bad) - this task is otherwise known as ordinal regression

Representing preferences

Absolute preferences

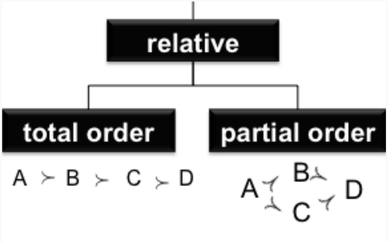
- Absolute preferences: each object has a preference score
- Binary preferences: object is preferred/not preferred (c.f. binary classification)
- Ordinal scale preferences: order or objects is defined ("very satisfied" $\succ$ "satisfied") but distance is not
- Numeric scale: order and distance is defined



(Source: Huellermeyer & Fuernkrantz, 2010)

Relative preferences

- Relative preferences: Preference information comes as known pairwise comparisons: $A \succ B$
- Total order: all objects are ranked from the most preferred to the least preferred (e.g. ranking for all lunch restaurants in Otaniemi)
- Partial order: order is known only for a subset of objects: (e.g. "Fat Lizard" $\succ$ "Maukas")



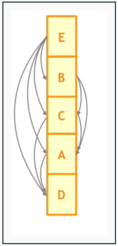
(Source: Huellermeyer & Fuernkrantz, 2010)

Representing rankings

- Assume a set of objects (inputs) $S = \{\mathbf{x}_i\}_{i=1}^m$
- Ranking function for S is a bijective function

$$\sigma : S \mapsto \{1, \ldots, m\}$$

  that assigns a unique rank $1 \leq \sigma(\mathbf{x}) \leq m$ to each object in S
- The inverse mapping $\sigma^{-1}(j) : \{1, \ldots, m\} \mapsto S$ gives the object of S at given rank j

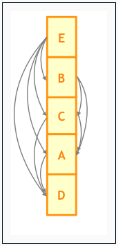


In the Figure:

- $\sigma(A) = 4, \sigma(B) = 2, \sigma(C) = 3, \sigma(D) = 5, \sigma(E) = 1$
- $\sigma^{-1}(1) = E, \sigma^{-1}(2) = B, \sigma^{-1}(3) = C, \sigma^{-1}(4) = A, \sigma^{-1}(5) = D$

Representing rankings

- A ranking for S is a permutation of S sorted in ascending order of σ: $\sigma^{-1}(1), \sigma^{-1}(2), \dots \sigma^{-1}(m)$
- The ranking corresponds to a sequence of pairwise preferences: $\sigma^{-1}(1) \succ \sigma^{-1}(2) \succ \cdots \succ \sigma^{-1}(m)$
- Note: high preference equals low ranking and vice versa; the most preferred object has rank 1, the least preferred rank m



In the Figure:

- $\sigma^{-1}(1) = E, \sigma^{-1}(2) = B, \sigma^{-1}(3) = C, \sigma^{-1}(4) = A, \sigma^{-1}(5) = D$
- $E \succ B \succ C \succ A \succ D$

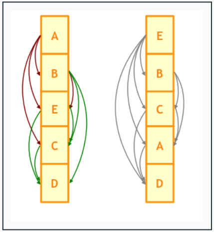
Kendall's distance

- Kendall's distance compares a predicted ranking $\sigma'(\mathbf{x})$ to a ground truth ranking $\sigma(\mathbf{x})$
- It counts the pairs that are inverted in the predicted ranking

$$d_K(\sigma, \sigma') = |\{(j, l) | \sigma(\mathbf{x}_j) > \sigma(\mathbf{x}_l) \text{ and } \sigma'(\mathbf{x}_j) < \sigma'(\mathbf{x}_l)\}|$$

- d_k takes values between $d_K(\sigma, \sigma') = 0$ and $d_K(\sigma, \sigma') = m(m-1)/2$, where m is the number of items

- Figure:
  - Predicted ranking σ' (left) has four inverted pairs $(A, B), (A, E), (A, C)$, (B, E) compared to ground truth
  - Kendall's distance $d_K(\sigma, \sigma') = 4$



Other loss functions for ranking

- Spearman's footrule: sum of absolute distances in ranks

$$d_{SF}(\sigma, \sigma') = \sum_{i=1}^{m} |\sigma(\mathbf{x}_i) - \sigma'(\mathbf{x}_i)|$$

- Position error: the number of wrong items that are predicted before the target item $\mathbf{x}_*$:

$$d_{PE}(\sigma, \sigma') = \sigma'(\mathbf{x}_*) - 1, \text{ where } \sigma(\mathbf{x}_*) = 1$$

- Discounted error: down-weights ranking errors of items with a lower true rank, with some factor v_i

$$d_{DE}(\sigma, \sigma') = \sum_{i=1}^{m} v_i d_{\mathbf{x}_i}(\sigma, \sigma'),$$

where is some distance of rankings of single item x_i in σ and σ'

Object ranking

Object ranking

Given a training set of (input) objects $\{\mathbf{x}_i\}_{i=1}^m$ and set of pairwise preferences $\mathcal{P} = \{(i,j) | \mathbf{x}_i \succ \mathbf{x}_j\}$ our aim is to learn a ranking function σ that can order new sets of objects $\{\mathbf{x}'_j\}_{j=1}^n$

Training

$$
\begin{array}{ccc}
(0.74, 1, 25, 165) & \succ & (0.45, 0, 35, 155) \\
(0.47, 1, 46, 183) & \succ & (0.57, 1, 61, 177) \\
(0.25, 0, 26, 199) & \succ & (0.73, 0, 46, 185)
\end{array}
$$

 $\succ$ 

Pairwise preferences between objects (instances)

Prediction (ranking a new set of objects)

$$\mathcal{Q} = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}\}$$

$$x_{10} \succ x_4 \succ x_7 \succ x_1 \succ x_{11} \succ x_2 \succ x_8 \succ x_{13} \succ x_9 \succ x_3 \succ x_{12} \succ x_5 \succ x_6$$

Two-step scheme for object ranking

We can approach object ranking through a two-step process:

1. Learn a model that assigns preference score $f(\mathbf{x}, \mathbf{x}')$ for the preferences $\mathbf{x} \succ \mathbf{x}'$ for any pair of inputs $(\mathbf{x}, \mathbf{x}')$
2. For a set of new points to be ranked $\{\mathbf{x}_i\}_{i=1}^n$ find the ranking σ that maximizes the agreement between the ranking and the predicted preference score:

$$AGREE(\sigma, f) = \sum_{\sigma(\mathbf{x}_i) < \sigma(\mathbf{x}_j)} f(\mathbf{x}_i, \mathbf{x}_j),$$

that is, the sum of preference scores consistent with σ'

Cohen, W.W., Schapire, R.E. and Singer, Y., 1999. Learning to order things. Journal of artificial intelligence research, 10, pp.243-270.

First step: Learning to order pairs

- We can convert the problem of ordering pairs into a binary classification problem with input data given by the pairs of objects
- As training data we assume a set of inputs $\{\mathbf{x}_i\}_{i=1}^m$ and set of preferences $\mathcal{P} = \{(i, j) | \mathbf{x}_i \succ \mathbf{x}_j\}$.
- A classifier should predict for a given a pair of inputs $(\mathbf{x}, \mathbf{x}')$

$$h(\mathbf{x}, \mathbf{x}') = \begin{cases} 1 & \text{if } \mathbf{x} \succ \mathbf{x}' \\ -1 & \text{if } \mathbf{x}' \succ \mathbf{x} \end{cases}$$

- We can use any classification algorithm on the pairwise data to learn the predictor
- If the classifier outputs real valued scores (e.g. probabilities, margins, etc.) $f(\mathbf{x}_i, \mathbf{x}_j)$, we can use the scores instead of the predicted binary labels

Second step: Extracting a ranking

- For a set of new inputs $\mathbf{x}_1, \ldots, \mathbf{x}_n$ we will obtain a pairwise preference $f(\mathbf{x}_i, \mathbf{x}_j)$ for each pair $(\mathbf{x}_i, \mathbf{x}_j)$
- These predictions can be contradictory, e.g. we may have cycle $A \succ B \succ C \succ A$
- To extract a ranking for the objects, pairwise predictions that are not consistent with the chosen order need to be ignored
- The problem is to find a ranking $\hat{\sigma}$ that maximizes the agreement with f: $\hat{\sigma} = \mathrm{argmax}_\sigma AGREE(\sigma, f)$
- However: Finding the highest scoring ranking is a NP-hard optimization problem (Cohen et al. 1999)

Cohen, W.W., Schapire, R.E. and Singer, Y., 1999. Learning to order things. Journal of artificial intelligence research, 10, pp.243-270.

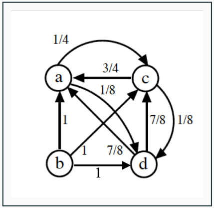
Second step: Extracting a ranking

- A approximate solution can be found by a graph based solution
- In the graph, objects correspond to nodes and pairwise preferences to directed edges
- Edge weights are preference scores $f(\mathbf{x}_i, \mathbf{x}_j)$ which are scaled to interval $[0, 1]$ and satisfy $f(\mathbf{x}_i, \mathbf{x}_j) + f(\mathbf{x}_j, \mathbf{x}_i) = 1$

  - Our goal is to maximize the agreement between the preference scores and the chosen ranking

$$AGREE(\sigma', f) = \sum_{\sigma'(\mathbf{x}_i) < \sigma'(\mathbf{x}_j)} f(\mathbf{x}_i, \mathbf{x}_j),$$

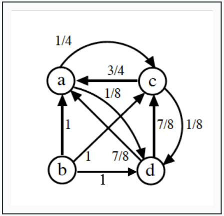
  - This amounts to keeping all edges consistent with the chosen order and ignoring the conflicting ones



Cohen's algorithm

Cohen's algorithm (Cohen et al. 1999) builds a preference graph with nodes corresponding to the input data points (in figure: $S = \{a, b, c, d\}$)

- Weighted edges correspond to the predicted preference scores $f(\mathbf{x}, \mathbf{x}')$ and $f(\mathbf{x}', \mathbf{x})$
- The algorithm maintains for each node the net preference score $\pi(\mathbf{x}) = \sum_{\mathbf{x}'} f(\mathbf{x}, \mathbf{x}') - \sum_{\mathbf{x}'} f(\mathbf{x}', \mathbf{x})$ which is the sum of outgoing edge weights (pairwise preferences $\mathbf{x} \succ \mathbf{x}'$) minus the sum of incoming edge weights (pairwise preferences $\mathbf{x}' \succ \mathbf{x}$)

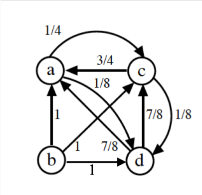


Cohen, W.W., Schapire, R.E. and Singer, Y., 1999. Learning to order things. Journal of artificial intelligence research, 10, pp.243-270.

Cohen's algorithm

- The net preference scores for the full graph are:

$$\pi(a) = 0 + 1/4 + 1/8 - (1 + 3/4 + 7/8) = -18/8$$
$$\pi(b) = 1 + 1 + 1 - 0 = 3$$
$$\pi(c) = 0 + 3/4 + 1/8 - (1 + 1/4 + 7/8) = -10/8$$
$$\pi(d) = (0 + 7/8 + 7/8) - (1 + 1/8 + 1/8) = 4/8$$



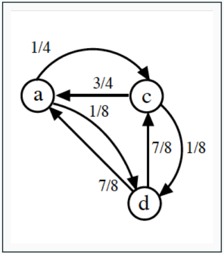
- The most preferred node is computed, it is b
- We set $\sigma'(b) = 1$

Cohen's algorithm

- The most preferred node is deleted and the net preference scores $\pi(\mathbf{x})$ are updated to reflect the new graph

$$\pi(a) = -18/8 + (1 - 0) = -10/8$$
$$\pi(c) = -10/8 + (1 - 0) = -2/8$$
$$\pi(d) = 4/8 + (1 - 0) = 12/8$$

- The most preferred node is again computed: (d) and it gets the first available rank: $\sigma(d) = 2$

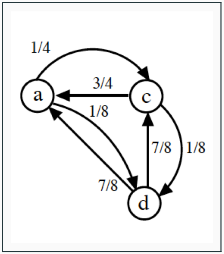


Cohen's algorithm

- The most preferred node d is deleted and the net preference scores are updated to reflect the new graph

  $$\pi(a) = -10/8 + (7/8 - 1/8) = -2/4$$
  $$\pi(c) = -2/8 + (7/8 - 1/8) = 2/4$$

- The most preferred node is c, we set $\sigma(c) = 3$

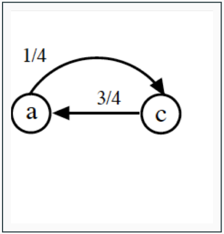


Cohen's algorithm

- One node a remains in the graph with net preference score

$$\pi(a) = -2/4 + (3/4 - 1/4) = 0$$

- We set $\sigma(a) = 4$, and terminate the algorithm
- The extracted total order is then $b \succ d \succ c \succ a$



Cohen's algorithm: pseudocode

Input: A set of objects $S = \{\mathbf{x}_i\}_{i=1^n}$, preference function $f(\mathbf{x}, \mathbf{x}')$

t=1

Set $\pi(\mathbf{x})$ as the net preference score of for all $\mathbf{x} \in S$:

$\pi(\mathbf{x}) = \sum_{\mathbf{x}' \in S} f(\mathbf{x}, \mathbf{x}') - \sum_{\mathbf{x}' \in S} f(\mathbf{x}', \mathbf{x})$

while $S \neq \emptyset$ **do**

Find the object with largest net preference:

$\mathbf{x}^* = \mathrm{argmax}_{\mathbf{x} \in S} \pi(\mathbf{x})$

$S = S - \mathbf{x}^*$

$\sigma(\mathbf{x}^*) = t$;

Remove the contribution of $\mathbf{x}^*$ from the net preference scores:

$\pi(\mathbf{x}) = \pi(\mathbf{x}) + (f(\mathbf{x}^*, \mathbf{x}) - f(\mathbf{x}, \mathbf{x}^*))$ for all $\mathbf{x} \in S$

$t = t + 1$;

end while

Output: $(\sigma(\mathbf{x}_1), \sigma(\mathbf{x}_2), \ldots, \sigma(\mathbf{x}_n))$

Preference learning through ranking loss minimization

Preference learning through ranking loss minimization

- The above described scheme is two-step preference learning scheme (binary classification and post-processing to extract a ranking)
- Although it simple and can be effective, it does not directly optimize a loss function for ranking
- In the following we examine algorithms that directly to optimize the quality of the ranking

Preference learning through linear models

- Consider learning a linear model $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ that assigns a preference score $f(\mathbf{x})$ to each input $\mathbf{x}$
- As training data we assume a set of inputs $\{\mathbf{x}_i\}_{i=1}^m$ and set of preferences $\mathcal{P} = \{(i,j)|\mathbf{x}_i \succ \mathbf{x}_j\}$.
- The pair $(\mathbf{x}_i, \mathbf{x}_j)$, $(i,j) \in \mathcal{P}$ is consistently predicted if and only if

$$f(\mathbf{x}_i) \geq f(\mathbf{x}_j)$$

or alternatively if and only if

$$f(\mathbf{x}_i) - f(\mathbf{x}_j) = \mathbf{w}^T(\mathbf{x}_i - \mathbf{x}_j) = \mathbf{w}^T\Delta\mathbf{x}_{ij} \geq 0$$

where $\Delta\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$ is the difference vector of $\mathbf{x}_i$ and $\mathbf{x}_j$

Preference learning through linear models

- We can denote the preferences by labels

$$y_{ij} = \begin{cases} +1 & \text{if } (i,j) \in \mathcal{P} \\ -1 & \text{if}(j,i) \in \mathcal{P} \\ 0 & \textit{otherwise} \end{cases}$$

- Then a pair is consistently predicted if it has a non-negative margin

$$y_{ij}\mathbf{w}^T\Delta\mathbf{x}_{ij} \geq 0$$

- This is a hyperplane classifier with difference vectors $\Delta\mathbf{x}_{ij}$ as inputs and the preferences encoded into the labels y_{ij}
- Data points with $y_{ij} = 0$ correspond to the pairs with no preferred order. They are always consistently classified.

Preference learning through linear discrimination

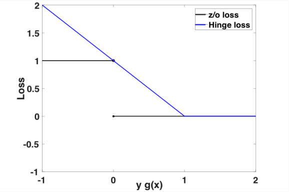
- Recall that finding the hyperplane that minimizes the zero-one loss of training set is NP-hard
- In our case, an error happens when the pair has a negative margin

$$y_{ij}\mathbf{w}^T\Delta\mathbf{x}_{ij} < 0$$

  in other words when the model puts the pair in inverted order $\mathbf{w}^T\mathbf{x}_i < \mathbf{w}^T\mathbf{x}_j$, $\mathbf{x}_i \succ \mathbf{x}_j$
- Thus, minimizing the number of inverted pairs - the Kendall distance - is hard as well

Hinge loss for preference learning

- Similarly to the binary classification, replacing the zero-one loss with a convex upper bound, such as Hinge loss, leads to efficient optimization



- Hinge loss for a pair (i, j):

$$\max(0, 1 - y_{ij}\mathbf{w}^T \Delta \mathbf{x}_{ij})$$

- Loss is incurred if the functional margin $y_{ij}\mathbf{w}^T \Delta \mathbf{x}_{ij} < 1$
- Average Hinge loss over all pairs:

$$\frac{1}{m(m-1)} \sum_{(i,j), i \neq j} \max(0, 1 - y_{ij}\mathbf{w}^T \Delta \mathbf{x}_{ij})$$

- RankSVM minimizes the above loss, while controlling the norm of the weight vector

RankSVM

- RankSVM (Joachims, 2002) solves the following regularised learning problem:

$$\min_{\mathbf{w},\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{|P|}\sum_{(i,j)\in\mathcal{P}} \xi_{ij}$$

$$\text{s.t. } \mathbf{w}^T\mathbf{x}_i - \mathbf{w}^T\mathbf{x}_j \geq 1 - \xi_{ij}, \text{ for all } (i,j) \in \mathcal{P}$$

$$\xi_{ij} \geq 0, \text{ for all } (i,j) \in \mathcal{P}$$

- The objective is to minimize the combination of the norm is of the weight weight vector (regularizer) and the loss (given by ξ_{ij})
- Note that only the preferred order $(i,j) \in \mathcal{P}$ is considered, not the opposite order (j,i). This is ok, since:

$$y_{ij}\mathbf{w}^T\Delta\mathbf{x}_{ij} = -y_{ij}\mathbf{w}^T\Delta\mathbf{x}_{ji} = y_{ji}\mathbf{w}^T\Delta\mathbf{x}_{ji}$$

- That is, satisfying the constraints for $(i,j) \in \mathcal{P}$, the constraints for (j,i) are automatically satisfied

T. Joachims: Optimizing search engines using clickthrough data, KDD 2002

RankSVM with kernels

- We can use kernel functions to perform non-linear ranking
- This is solved by the dual RankSVM problem:

$$\max_{\boldsymbol{\alpha}} g(\boldsymbol{\alpha}) = \sum_{(i,j)\in\mathcal{P}} \alpha_{ij} - \frac{1}{2} \sum_{(i,j)\in\mathcal{P}} \sum_{(r,s)\in\mathcal{P}} \alpha_{ij} \Delta\mathbf{x}_{ij}^T \Delta\mathbf{x}_{rs} \alpha_{rs}$$

$$\text{s.t.} 0 \leq \alpha_{ij} \leq \frac{C}{|P|}, \text{ for all } i \succ j$$

- It is a constrained Quadratic Programme
- The inner product $\Delta\mathbf{x}_{ij}^T \Delta\mathbf{x}_{rs}$ can be replaced with any kernel $\kappa(\Delta\mathbf{x}_{ij}, \Delta\mathbf{x}_{rs})$ acting on the difference vectors $\Delta\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$
- The number of dual variables is proportional to the set of pairwise preferences, at worst quadratic in number of objects

Boosting for ranking

RankBoost algorithm

- RankBoost is an algorithm that applies the AdaBoost framework to the ranking problem
- It gets as input a training sample $S = \{(x_i, x_i', y_i)\}$ where

$$y_i = \begin{cases} +1 & \text{if } x_i' \succ x_i \\ 0 & \text{if } x_i', x_i \text{ have the same preference or are incomparable} \\ -1 & \text{if } x_i \succ x_i' \end{cases}$$

- It learns a linear combination

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

  of base rankers or weak rankers h_t
- Base rankers are assumed to output a binary preference (preferred/not preferred): $h_t(x) \in \{0, 1\}$ learned by minimizing the weighted ranking errors $D_t(i)\mathbf{1}_{y_i(h_t(x_i') - h_t(x_i)) < 0}$ in the training set

Weak ranker?

- The weak learning assumption that the base rankers are assumed to satisfy is that they rank correctly more pairs than incorrectly
- Denote by

$$\epsilon_t^+ = \sum_{i=1}^{m} D_t(i)\mathbf{1}_{y_i(h_t(x_i')-h_t(x_i))\geq 0}$$

  the proportion of correctly ranked pairs, by

$$\epsilon_t^- = \sum_{i=1}^{m} D_t(i)\mathbf{1}_{y_i(h_t(x_i')-h_t(x_i))<0}$$

  the proportion of the incorrectly ranked pairs and by

$$\epsilon_t^0 = \sum_{i=1}^{m} D_t(i)\mathbf{1}_{y_i(h_t(x_i')-h_t(x_i))=0}$$

  the proportion of the non-ranked pairs
- A weak ranker is thus required to satisfy: $\epsilon_t^+ - \epsilon_t^- > 0$

Weights of the weak rankers

- The weights of the weak learner is given by $\alpha_t = \frac{1}{2}\log\frac{\epsilon_t^+}{\epsilon_t^-}$ which represents the log-odds ratio between the weak learner being correct or incorrect on the training sample
- When the weak ranking assumption $\epsilon_t^+ - \epsilon_t^- > 0$ is satisfied, we have $\frac{\epsilon_t^+}{\epsilon_t^-} > 1$
- Thus $\alpha_t > 0$ in this case

Re-weighting of examples

- The weight distribution of examples is updated by

$$D_{t+1}(i) = \frac{D_t(i)e^{-\alpha_t y_i (h_t(x_i') - h_t(x_i))}}{Z_t}$$

- The exponent will be positive when the weak ranker (with $\alpha_t > 0$) makes a ranking mistake $(y_i(h_t(x_i') - h_t(x_i)) < 0)$ on the pair $\Rightarrow$ up-weighting the example for the next iteration
- Correctly classified pairs result in down-weighting
- For pairs for which the weak ranker cannot decide on ranking $(h(x_i) - h(x_i') = 0)$, weights are unchanged
- $Z_t = \sum_{i=1}^{m} D_t(i)e^{-\alpha_t y_i (h_t(x_i') - h_t(x_i))} = \epsilon_t^0 + 2(\epsilon_t^+ \epsilon_t^-)^{1/2}$ is a normalization factor

RankBoost pseudocode

$\textsc{RankBoost}(S = ((x_1, x_1', y_1) \ldots, (x_m, x_m', y_m)))$

for $i \leftarrow 1$ **to** m **do**

  $\mathcal{D}_1(i) \leftarrow \frac{1}{m}$

for $t \leftarrow 1$ **to** T **do**

  $h_t \leftarrow$ base ranker in $\mathcal{H}$ with smallest $\epsilon_t^- - \epsilon_t^+ = -\mathop{\mathbb{E}}_{i \sim \mathcal{D}_t}\left[y_i\big(h_t(x_i') - h_t(x_i)\big)\right]$

  $\alpha_t \leftarrow \frac{1}{2} \log \frac{\epsilon_t^+}{\epsilon_t^-}$

  $Z_t \leftarrow \epsilon_t^0 + 2[\epsilon_t^+ \epsilon_t^-]^{\frac{1}{2}}$ ▷ normalization factor

  for $i \leftarrow 1$ **to** m **do**

    $\mathcal{D}_{t+1}(i) \leftarrow \frac{\mathcal{D}_t(i) \exp\left[-\alpha_t y_i \left(h_t(x_i') - h_t(x_i)\right)\right]}{Z_t}$

$f \leftarrow \sum_{t=1}^{T} \alpha_t h_t$

return f

RankBoost error

- RankBoost can be shown to minimize the loss function

$$\sum_{i=1}^{m} e^{-y_i(f_N(x_i') - f_N(x_i))}$$

- It is a convex upper bound of the empirical risk defined as the number of inverted pairs $\hat{R}(h) = \sum_{i=1}^{m} \mathbf{1}_{f_N(x_i') - f_N(x_i) \leq 0}$ i.e. Kendall's distance
- If all weak rankers satisfy $\frac{\epsilon_t^+ - \epsilon_t^-}{2} \geq \gamma \geq 0$ then $\hat{R}(h) \leq \exp(-2\gamma^2 T)$
- The empirical risk goes exponentially down in the boosting iterations T,
- A larger edge $\epsilon_t^+ - \epsilon_t^-$ - how many more pairs are correct than incorrect - gives faster decrease of the risk

Summary

- Preference learning covers a number of machine learning tasks where the aim is to order, rank or rate objects
- In object ranking the goal is to rank new objects with a ranking function learned from existing preference data
- Two-stage approach for object ranking consists of using a binary classifier to order pairs, followed by a phase where the best consistent order for the whole dataset is extracted
- RankSVM and RankBoost are examples of models that aim to directly minimize a ranking loss function