An Efficient Boosting Algorithm for Combining Preferences by Y. Freund, R. Iyer, R. E. Schapire, Y. Singer

Presented by: Aivi Kaljuvee
RankBoost – boosting algorithm for combining multiple ranking functions to predict the target function

Boosting – method of producing highly accurate prediction rules by combining many “weak” rules which may be only moderately accurate
Example problems:

- movie recommendation service – based on user's ratings to movies h/she has already seen and ratings from other service users, h/she is presented a list of new movies he/she should like (collaborative filtering task)
- meta-search – results form several search engine queries are combined into one
Formal framework

- $X$ – instance space

- $f_i : X \rightarrow \overline{\mathbb{R}}, i = 1..n, \overline{\mathbb{R}} = \mathbb{R} \cup \{\tau\}$ – n ranking features
  - $f_i(x) = \tau$ – no ranking given to $x$ by $f_i$
  - $f_i(x_1) > f_i(x_0)$ – $x_1$ is preferred over $x_0$ by $f_i$
  - Ties are allowed

- $H : X \rightarrow \mathbb{R}$ – combined ranking
Formal framework

• $\Phi : X \times X \rightarrow \mathbb{R} \quad -$ feedback function
  $\Phi (x_0, x_1) > 0$ – $x_1$ is preferred over $x_0$
  $\Phi (x_0, x_1) < 0$ – the opposite
  $\Phi (x_0, x_1) = 0$ – no preference
  $|\Phi (x_0, x_1)|$ – importance of preference

• $D(x_0, x_1) = c \cdot \max \{0, \Phi(x_0, x_1)\}$
  $c$ is a constant so that $\sum x_0, x_1 D(x_0, x_1) = 1$

• $rloss_D(H) = \sum x_0, x_1 D(x_0, x_1) [H(x_1) \leq H(x_0)]$
  loss function ($\left\lceil \pi \right\rceil - 1$ if predicate $\pi$ holds, 0 otherwise)
Pseudocode for RankBoost

Given: initial distribution \( D \) over \( X \times X \).
Initialize: \( D_1 = D \).

For \( t = 1, \ldots, T \):
  • Train weak learner using distribution \( D_t \).
  • Get weak ranking: \( h_t: X \to \mathbb{R} \).
  • Choose \( \alpha_t \in \mathbb{R} \).
  • Update: 
    \[
    D_{(t+1)}(x_0, x_1) = \frac{(D_t(x_0, x_1) \exp(\alpha_t(h_t(x_0) - h_t(x_1))))}{Z_t}
    \]
    where \( Z_t \) is a normalization factor (so that \( D_{t+1} \) is a distribution).

Output the final ranking: 
\[
H(x) = \sum_{t=1..T} (\alpha_t h_t(x))
\]
Finding $\alpha$

- $rloss_D(H) \leq \prod_{t=1..T} Z_t$

- $Z_t = \sum_{x_0, x_1} (D_t(x_0, x_1) \exp(\alpha_t(h_t(x_0) - h_t(x_1))))$
Finding a weak ranking

\begin{itemize}
  \item $h(x) = \{(f_i(x) \text{ if } f_i(x) \in \mathbb{R}) \} \cdot (q_{\text{default}} \in \mathbb{R})$
  \begin{align*}
    q_{\text{default}} \text{ if } f_i(x) = \ell
  \end{align*}

  based on actual values from ranking features, not used

  \item
  \begin{align*}
    h(x) &= \begin{cases} 
      1 & \text{if } f_i(x) > \theta \\
      0 & \text{if } f_i(x) \leq \theta \\
      q_{\text{default}} & \text{if } f_i(x) = \ell
    \end{cases}
  \end{align*}

  \begin{align*}
    (q_{\text{default}}, \theta \in \mathbb{R})
  \end{align*}

  uses relative-ordering information, used for producing weak rankings
\end{itemize}
Experiments with the meta-search task

- queries for finding homepages for machine learning researchers and universities
- base query and extended query
- instances are pairs of base queries and URLs
- a ranking feature is the ordered list received from a request with one of the extended queries
# Results for the meta-search task

<table>
<thead>
<tr>
<th>ML Domain</th>
<th>Top 1</th>
<th>Top 2</th>
<th>Top 5</th>
<th>Top 10</th>
<th>Top 20</th>
<th>Top 30</th>
<th>Avg</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>RankBoost</td>
<td>102</td>
<td>144</td>
<td>173</td>
<td>184</td>
<td>194</td>
<td>202</td>
<td>4.38</td>
<td></td>
</tr>
<tr>
<td>Best (Top 1)</td>
<td><strong>117</strong></td>
<td>137</td>
<td>154</td>
<td>167</td>
<td>177</td>
<td>181</td>
<td>6.80</td>
<td></td>
</tr>
<tr>
<td>Best (Top 10)</td>
<td>112</td>
<td><strong>147</strong></td>
<td><strong>172</strong></td>
<td><strong>179</strong></td>
<td>185</td>
<td>187</td>
<td><strong>5.33</strong></td>
<td></td>
</tr>
<tr>
<td>Best (Top 30)</td>
<td>95</td>
<td>129</td>
<td>159</td>
<td>178</td>
<td><strong>187</strong></td>
<td><strong>191</strong></td>
<td>5.68</td>
<td></td>
</tr>
<tr>
<td>University Domain</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RankBoost</td>
<td>95</td>
<td>141</td>
<td>197</td>
<td>215</td>
<td>247</td>
<td>263</td>
<td>7.74</td>
<td></td>
</tr>
<tr>
<td>Best single query</td>
<td>112</td>
<td>144</td>
<td>198</td>
<td>221</td>
<td>238</td>
<td>247</td>
<td>8.17</td>
<td></td>
</tr>
</tbody>
</table>
Experiments with movie recommendation service

- users' ratings are ranking features, ratings from a single target user are used to construct the feedback function (half of the films rated for training, half for testing)

- compared with the following algorithms:
  - regression
  - nearest-neighbour
  - vector similarity
Performance measures:

- **disagreement**
  - $c$ – ordering of test movies
  - $N$ – number of test movies
  - \[ \frac{1}{N} \sum_{x_0, x_1: c(x_0) < c(x_1)} \left[ H(x_0) > H(x_1) \right] \]

- **average precision**
  - $K$ – number of movies in the feedback ordering
  - $t_k$ – movie
  - $k$ – its rank in the feedback ordering
  - $\text{rank}(t_k)$ – movie's rank in the learned ranking
  - \[ \frac{1}{K} \sum_{k=1..K} \left( \frac{k}{\text{rank}(t_k)} \right) \]
Performance measures:

• predicted rank of top  
  \[ \frac{1}{\text{rank}(t_1)} \]

• coverage  
  \[ \frac{1}{\text{rank}(t_K)} \]
Directions for future work

- reimplement algorithms used for comparison using relative-ordering information
- compare performance with AdaBoost
- apply RankBoost to information retrieval problems
- apply RankBoost to language processing tasks