Lecture 11 – Learning-to-rank (1)
Classic Retrieval Models (before 2005)

We’ve discussed many retrieval models so far

• VSM and TF-IDF
• Probabilistic retrieval models, e.g., BIM, BM25
• Language modeling for IR, e.g., QL, KL-divergence
• There are other popular models, but we don’t have time to introduce all
  • e.g., divergence from randomness (DFR)
Classic Retrieval Models (before 2005)

They all implemented very similar ideas

- TF, IDF
- Penalizing repeated occurrences of the same term
- Penalizing TF for longer documents

\[
\text{VSM cosine: } \cos(\vec{q}, \vec{d}) \propto \frac{\sum_{i=1}^{k} q_i d_i}{\sqrt{\sum_{i=1}^{k} d_i^2}} = \frac{\sum_{t \in q} w(t, q) w(t, d)}{\sqrt{\sum_{i=1}^{k} d_i^2}} = \frac{\sum_{t \in q} q tf \cdot t f \cdot idf^2}{\sqrt{\sum_{i=1}^{k} d_i^2}}
\]

\[
\text{BM25: } w_i = \frac{(k_3 + 1) \cdot qtf}{k_3 + qtf} \cdot \frac{(k_1 + 1) \cdot t f}{k_1 \left(1 - b + b \cdot \frac{dl}{avdl}\right) + tf} \cdot \log \frac{N - n + 0.5}{n + 0.5}
\]

\[
\text{LM Dirichlet: } \sum_{t \in q} \log \frac{tf + \mu \cdot P(t \mid \text{Corpus})}{dl + \mu}
\]
Classic Retrieval Models (before 2005)

The general way of designing these models

- Intuition & ideas, e.g., comparing query-document similarity
- Sometimes human observations of data, e.g., 2-Poisson model
- Hand-designed ranking/scoring functions
- Just a few parameters (usually does not have a huge impact as long as the values stay in a reasonable range)
Learning-to-rank (2003–now)

Automatically learn a ranking function from data

• What’s left for human?
  • Hand-designed features (ranking signals)
  • Define the basic model structure (e.g., linear)

• Advantages: flexibility; scalability

• Disadvantages: requires training data (more complex model requires more)
Deep ranking models (2013~now)

Using deep neural nets for learning-to-rank

- Does not require feature engineering
  - Usually just takes a sequence of text as input
  - Good neural nets will learn feature-like structure
- Advantages: flexibility; scalability; no feature engineering (not yet fully successful at this moment)
- Disadvantages: requires training data (a lot of!)
How to learn from data to rank results?

One seemingly possible solution

• IR as binary classification (query-dependent)
• Train a binary classification model based on user feedback
• Several issues …
  • Training data (user feedback)
  • Generalizability to unseen queries (and even old queries)
  • User control
• HW3 Q3 is an example
How to learn from data to rank results?

What is an ideal solution?

• A query-independent solution
• A set of query-document match features
  • Nothing specifically related to a particular query
  • These features can be variants of TF-IDF scores, etc.
  • Can include query or document features, e.g., is it a long document?
• Learn from past queries and relevance judgments
  • Learn the correct way of combining multiple ranking feats
• To rank future (unseen) queries
  • Because the features are generalizable …
Outline

• Learning-to-rank Basics
• Pointwise, pairwise, listwise approaches
• Features & benchmarks
• RankLib Tutorial
Classic models vs. Learning-to-rank

Classic Retrieval Models

- Features (factors): only a few, e.g., TF, IDF, |D|, P(t|Corpus) etc.
- Structure: optimized for the a few particular features
- Parameter & training
  - Often 1-2; not every factor has a parameter controlling its influence
  - Hand-tuning or data-based; can exhaustive since just 1-2 parameters

Learning-to-rank

- Features: can include up to hundreds, thousands, or even more
- Define the basic structure of a model
  - Quite generic: such as a weighted linear combination of all features
- Parameters & training
  - Many; control the influence of each feature and their combinations
  - Impossible to tune by hand; Must be data-driven
Why Learning-to-rank?

• Modern systems – especially on the Web – use a great number of features:
  • Arbitrary useful features – not a single unified model
  • Log frequency of query word in anchor text?
  • Query word in color on page?
  • # of images on page?
  • # of (out) links on page?
  • PageRank of page?
  • URL length?
  • URL contains “~”?  
  • Page length?

• The New York Times (2008-06-03) quoted Amit Singhal as saying Google was using over 200 such features.
Why weren’t early attempts very successful/influential?

- Sometimes an idea just takes time to be appreciated…

- **Limited training data**
  - Especially for real world use (as opposed to writing academic papers), it was very hard to gather test collection queries and relevance judgments that are representative of real user needs and judgments on documents returned
    - This has changed, both in academia and industry

- Poor machine learning techniques

- Insufficient customization to IR problem

- Not enough features for ML to show value

- The real thing that makes L2R hot is click-through based implicit feedback (around 2005; Thorsten Joachim & others)
Types of Learning-to-rank Solutions

Purpose
• Learn a function automatically to rank results (items) effectively

Pointwise approach
• The function is based on features of a single object
• e.g., regress the relevance score, classify docs into R and NR
• Classic retrieval models are also point-wise: score(q, D)

Pairwise
• The function is based on a pair of item
• e.g., given two documents, predict partial ranking

Listwise
• The function is based on a ranked list of items
• e.g., given two ranked list of the same items, which is better?
Logistic Regression for Ranking

• Combine a list of ranking features using a linear model
• Predict binary relevance (R/NR) using a sigmoid function
• Rank results by the predicted probability of being relevant

<table>
<thead>
<tr>
<th>$w_i$</th>
<th>$x_i$</th>
<th>Features (just examples)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>2.6</td>
<td>Mean raw TF of query terms</td>
</tr>
<tr>
<td>0.02</td>
<td>1.3</td>
<td>Mean log TF of query terms</td>
</tr>
<tr>
<td>0.08</td>
<td>1.7</td>
<td>Mean BM25 TF of query terms</td>
</tr>
<tr>
<td>0.04</td>
<td>5.3</td>
<td>log TF x IDF score</td>
</tr>
<tr>
<td>0.12</td>
<td>6.1</td>
<td>BM25 score</td>
</tr>
<tr>
<td>0.32</td>
<td>2</td>
<td># of matched unique query terms</td>
</tr>
<tr>
<td>−0.13</td>
<td>819</td>
<td>Document length</td>
</tr>
</tbody>
</table>

$$z = \left( \sum_{i=1}^{n} w_i x_i \right) + b$$

$$y = \sigma(z) = \frac{1}{1 + e^{-z}}$$
Sigmoid

- Always stays within (0,1) – ideal for modeling probability
- S-shape; sensitive in the middle

\[ y = \sigma(z) = \frac{1}{1 + e^{-z}} \]
Logistic Regression for Ranking

- Classification: usually use 0.5 as the decision boundary
- Ranking: simply rank by $y$ (the predicted chances of relevance)

<table>
<thead>
<tr>
<th>$w_i$</th>
<th>$x_i$</th>
<th>Features (just examples)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>2.6</td>
<td>Mean raw TF of query terms</td>
</tr>
<tr>
<td>0.02</td>
<td>1.3</td>
<td>Mean log TF of query terms</td>
</tr>
<tr>
<td>0.08</td>
<td>1.7</td>
<td>Mean BM25 TF of query terms</td>
</tr>
<tr>
<td>0.04</td>
<td>5.3</td>
<td>log TF x IDF score</td>
</tr>
<tr>
<td>0.12</td>
<td>6.1</td>
<td>BM25 score</td>
</tr>
<tr>
<td>0.32</td>
<td>2</td>
<td># of matched unique query terms</td>
</tr>
<tr>
<td>−0.13</td>
<td>819</td>
<td>Document length</td>
</tr>
</tbody>
</table>

$$z = \left( \sum_{i=1}^{n} w_i x_i \right) + b$$

$$y = \sigma(z) = \frac{1}{1 + e^{-z}}$$
Logistic Regression: Loss Function

• We hope to set weights and the bias term automatically.
• Learning from data!
• Define a function that characterizes classification errors on the training dataset.
  • This involves the comparison of true and predicted labels.
• Try to minimize the error term automatically.
• Let $y$ be the true label (a document is judged as R/NR)
• Let $\hat{y}$ be the predicted probability of relevance (based on the current parameters of the model)
• The cross-entropy loss is defined as follows:

$$L_{CE}(\hat{y}, y) = - \left[ y \log \hat{y} + (1 - y) \log (1 - \hat{y}) \right]$$
Logistic Regression: Loss Function

• The cross-entropy loss is defined as follows:

\[ L_{CE}(\hat{y}, y) = -\left[ y \log \hat{y} + (1 - y) \log (1 - \hat{y}) \right] \]

• \( L_{CE} \) captures classification errors
  • greater value means more errors

• When the true label is R (y=1)
  • \( L_{CE} \) increases while predicted y decreases

\[ L_{CE}(\hat{y}, y) = -\log \hat{y} \]

• When the true label is NR (y=0)
  • \( L_{CE} \) increases while predicted y increases

\[ L_{CE}(\hat{y}, y) = -\log (1 - \hat{y}) \]
Logistic Regression: Loss Function

• Note that  \( \hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}} \)  
  \[ z = \left( \sum_{i=1}^{n} w_i x_i \right) + b \]

• Thus, we can rewrite \( L_{CE} \) as a function of model parameters (\( w \) and \( b \)) as follows:

\[
L_{CE}(w, b) = -[y \log \sigma(w \cdot x + b) + (1 - y) \log (1 - \sigma(w \cdot x + b))]
\]

• The loss function of the whole training set is simply the average of those over each individual data instance

\[
Cost(w, b) = \frac{1}{m} \sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)})
\]

\[
= -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log \sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)}) \log \left(1 - \sigma(w \cdot x^{(i)} + b)\right)
\]
Logistic Regression: Gradient Descent

- How to automatically find the minimum value of a function?
- Follow the slope!
- Analytically: solve the equation slope = 0
- Algorithmically: gradient descent  
  \[ w^{t+1} = w^t - \eta \frac{d}{dw} f(x; w) \]
Logistic Regression: Gradient Descent

• How to automatically minimize the loss function?
• Follow the gradient!
• Gradient is the aggregation of "slopes" for all variables

\[ \nabla_{\theta} L(f(x; \theta), y) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x; \theta), y) \\ \frac{\partial}{\partial w_2} L(f(x; \theta), y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x; \theta), y) \end{bmatrix} \]

\[ \theta_{t+1} = \theta_t - \eta \nabla L(f(x; \theta), y) \]
Logistic Regression: Gradient Descent

• Recall the loss function $L_{CE}$

$$Cost(w, b) = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log \sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)}) \log \left(1 - \sigma(w \cdot x^{(i)} + b)\right)$$

• We can derive the gradient for each $w$

$$\frac{\partial Cost(w, b)}{\partial w_j} = \sum_{i=1}^{m} \left[\sigma(w \cdot x^{(i)} + b) - y^{(i)}\right] x_j^{(i)}$$
Logistic Regression: Regularization

• If a feature appears dominantly in R or NR, logistic regression will tend to learn very high weight for that feature.
  • The model will not correctly learn the importance of other features
  • But this may not generalize to new dataset …

• Solution: add a regularization term in the loss function to penalize large parameter values

\[
\hat{w} = \arg\max_w \sum_{i=1}^m \log P(y^{(i)}|x^{(i)}) - \alpha R(w)
\]
Logistic Regression: Regularization

• L2 norm (most common)

\[ R(W) = \|W\|_2^2 = \sum_{j=1}^{N} w_j^2 \]

\[ \hat{w} = \text{argmax}_w \left[ \sum_{1=i}^{m} \log P(y^{(i)} | x^{(i)}) \right] - \alpha \sum_{j=1}^{n} w_j^2 \]

• L1 norm

\[ R(W) = \|W\|_1 = \sum_{i=1}^{N} |w_i| \]

\[ \hat{w} = \text{argmax}_w \left[ \sum_{1=i}^{m} \log P(y^{(i)} | x^{(i)}) \right] - \alpha \sum_{j=1}^{n} |w_j| \]
Multinomial Logistic Regression

• Apply logistic regression to multiple class classification
• Ranking? Learn a model for each relevance levels
  • 2: highly relevant
  • 1: relevant
  • 0: not relevant
• Some datasets have five different relevant levels
  • It is not ideal to aggregate all positive labels as R

\[
\text{softmax}(z) = \left[ \frac{e^{z_1}}{\sum_{i=1}^{k} e^{z_i}}, \frac{e^{z_2}}{\sum_{i=1}^{k} e^{z_i}}, \ldots, \frac{e^{z_k}}{\sum_{i=1}^{k} e^{z_i}} \right]
\]

\[
p(y = c|x) = \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}
\]

Nallapati (2004)
• A typical example of point-wise learning-to-rank
• Background (2004)
  • Language modeling was super hot
  • Learning-to-rank just started
• 6 ranking features (all of them are TF-IDF variants)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 [\sum_{q_i \in Q \cap D} \log(c(q_i, D))]</td>
<td>4 [\sum_{q_i \in Q \cap D} \log\left(\frac{</td>
</tr>
<tr>
<td>2 [\sum_{i=1}^{n} \log(1 + \frac{c(q_i, D)}{</td>
<td>D</td>
</tr>
<tr>
<td>3 [\sum_{q_i \in Q \cap D} \log(idf(q_i))]</td>
<td>6 [\sum_{i=1}^{n} \log\left(1 + \frac{c(q_i, D)}{</td>
</tr>
</tbody>
</table>

Nallapati (2004)
- A typical example of point-wise learning-to-rank
- Background (2004)
  - Language modeling was super hot
  - Learning-to-rank just started
- 6 ranking features (all of them are TF-IDF variants)
- Two ranking models
  - Maximum entropy (ME)
  - Support vector machines (SVM)


Nallapati (2004)
• Training with unbalanced data
• The majority of the results in qrels are non-relevant (NR)
• Models simply trained using all qrels may always “say no”
• Oversampling: repeating R instances
• Under-sampling: sample only a fraction of NR in training


- Not super exciting, but comparable to QL
- Automatically aggregating 80s techs (TF-IDF variants) is as good as the state-of-the-art hand-designed method (QL)

<table>
<thead>
<tr>
<th>Train ↓ Test →</th>
<th>Disks 1-2 (151-200)</th>
<th>Disk 3 (101-150)</th>
<th>Disks 4-5 (401-450)</th>
<th>WT2G (426-450)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disks 1-2 (101-150)</td>
<td>LM ($\mu^* = 1900$)</td>
<td>0.2561 (6.75e-3)</td>
<td>0.1842</td>
<td>0.2377 (0.80)</td>
</tr>
<tr>
<td></td>
<td>SVM</td>
<td>0.2145</td>
<td>0.1877 (0.3)</td>
<td>0.2356</td>
</tr>
<tr>
<td></td>
<td>ME</td>
<td>0.1513</td>
<td>0.1240</td>
<td>0.1803</td>
</tr>
<tr>
<td>Disk 3 (51-100)</td>
<td>LM ($\mu^* = 500$)</td>
<td>0.2605 (1.08e-4)</td>
<td>0.1785 (0.11)</td>
<td>0.2503 (0.21)</td>
</tr>
<tr>
<td></td>
<td>SVM</td>
<td>0.2064</td>
<td>0.1728</td>
<td>0.2432</td>
</tr>
<tr>
<td></td>
<td>ME</td>
<td>0.1599</td>
<td>0.1221</td>
<td>0.1719</td>
</tr>
<tr>
<td>Disks 4-5 (301-350)</td>
<td>LM ($\mu^* = 450$)</td>
<td>0.2592 (1.75e-4)</td>
<td>0.1773 (7.9e-3)</td>
<td>0.2516 (0.036)</td>
</tr>
<tr>
<td></td>
<td>SVM</td>
<td>0.2078</td>
<td>0.1646</td>
<td>0.2355</td>
</tr>
<tr>
<td></td>
<td>ME</td>
<td>0.1413</td>
<td>0.0978</td>
<td>0.1403</td>
</tr>
<tr>
<td>WT2G (401-425)</td>
<td>LM ($\mu^* = 2400$)</td>
<td>0.2524 (4.6e-3)</td>
<td>0.1838 (0.08)</td>
<td>0.2335</td>
</tr>
<tr>
<td></td>
<td>SVM</td>
<td>0.2199</td>
<td>0.1744</td>
<td>0.2487 (0.046)</td>
</tr>
<tr>
<td></td>
<td>ME</td>
<td>0.1353</td>
<td>0.0969</td>
<td>0.1441</td>
</tr>
<tr>
<td>Best TREC runs (Site)</td>
<td>0.4226 (UMass)</td>
<td>N/A</td>
<td>0.3207 (Queen’s College)</td>
<td>N/A</td>
</tr>
</tbody>
</table>


- Great flexibility to incorporate a large set of features
- Anchor: the text that appears in a hypertext link and that can be clicked to open the target web page.

<table>
<thead>
<tr>
<th>SVM Features</th>
<th>MRR</th>
<th>Success %</th>
<th>Failure %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content + Anchor</td>
<td>0.54</td>
<td>73.0</td>
<td>5.2</td>
</tr>
<tr>
<td>Content + Anchor + Title</td>
<td>0.61</td>
<td>85.7</td>
<td>10.2</td>
</tr>
<tr>
<td>Content + Anchor + Title + URL</td>
<td>0.61</td>
<td>85.7</td>
<td>10.2</td>
</tr>
<tr>
<td>Content + Anchor + Title + URL + Link</td>
<td>0.61</td>
<td>85.7</td>
<td>10.2</td>
</tr>
<tr>
<td>Language Model baseline</td>
<td>0.35</td>
<td>52.0</td>
<td>10.0</td>
</tr>
<tr>
<td>SVM baseline</td>
<td>0.33</td>
<td>53.06</td>
<td>12.24</td>
</tr>
</tbody>
</table>

Some Issues of the Classification-based Point-wise Approaches

Class imbalance
- Many more non-relevant than relevant instances
- Classifiers usually do not handle huge imbalance well
- Need to address by over or under sampling

Classification error is biased toward “big” queries
- Query with 1000 rel documents more important than a query with one
- Address by weighting training instances

Optimization criteria
- Classification error not the same as MAP, nDCG, etc.
- min # errors doesn’t guarantee best MAP or nDCG
- Can incorporate heuristics, e.g., weighting errors based on rel labels
- But cannot solve all the issues
Microsoft’s LETOR project

• Version 4.0 released in June 2009
• Documents from .GOV collection
  • 25 million pages
• Queries from TREC million query track
  • 2500 queries from 2007 and 2008
  • Relevance information is partial because of corpus size and track methodologies
• 46 extracted features (query, doc, relevance)
  • TF, IDF, etc in page, title, anchors, URL, …
  • BM25 score, LM score, of page, title, anchors, URL, …
  • PageRank, in/out link count, …

What are the typical features?  
Complete list of LETOR 4.0 features

<table>
<thead>
<tr>
<th>Column in Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TF(Term frequency) of body</td>
</tr>
<tr>
<td>2</td>
<td>TF of anchor</td>
</tr>
<tr>
<td>3</td>
<td>TF of title</td>
</tr>
<tr>
<td>4</td>
<td>TF of URL</td>
</tr>
<tr>
<td>5</td>
<td>TF of whole document</td>
</tr>
<tr>
<td>6</td>
<td>IDF(Inverse document frequency) of body</td>
</tr>
<tr>
<td>7</td>
<td>IDF of anchor</td>
</tr>
<tr>
<td>8</td>
<td>IDF of title</td>
</tr>
<tr>
<td>9</td>
<td>IDF of URL</td>
</tr>
<tr>
<td>10</td>
<td>IDF of whole document</td>
</tr>
<tr>
<td>11</td>
<td>TF*IDF of body</td>
</tr>
<tr>
<td>12</td>
<td>TF*IDF of anchor</td>
</tr>
<tr>
<td>13</td>
<td>TF*IDF of title</td>
</tr>
<tr>
<td>14</td>
<td>TF*IDF of URL</td>
</tr>
<tr>
<td>15</td>
<td>TF*IDF of whole document</td>
</tr>
<tr>
<td>16</td>
<td>DL(Document length) of body</td>
</tr>
<tr>
<td>17</td>
<td>DL of anchor</td>
</tr>
<tr>
<td>18</td>
<td>DL of title</td>
</tr>
<tr>
<td>19</td>
<td>DL of URL</td>
</tr>
<tr>
<td>20</td>
<td>DL of whole document</td>
</tr>
<tr>
<td>21</td>
<td>BM25 of body</td>
</tr>
<tr>
<td>22</td>
<td>LMIR.ABS of body</td>
</tr>
<tr>
<td>23</td>
<td>LMIR.DIR of body</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column in Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>LMIR.JM of body</td>
</tr>
<tr>
<td>25</td>
<td>BM25 of anchor</td>
</tr>
<tr>
<td>26</td>
<td>LMIR.ABS of anchor</td>
</tr>
<tr>
<td>27</td>
<td>LMIR.DIR of anchor</td>
</tr>
<tr>
<td>28</td>
<td>LMIR.JM of anchor</td>
</tr>
<tr>
<td>29</td>
<td>BM25 of title</td>
</tr>
<tr>
<td>30</td>
<td>LMIR.ABS of title</td>
</tr>
<tr>
<td>31</td>
<td>LMIR.DIR of title</td>
</tr>
<tr>
<td>32</td>
<td>LMIR.JM of title</td>
</tr>
<tr>
<td>33</td>
<td>BM25 of URL</td>
</tr>
<tr>
<td>34</td>
<td>LMIR.ABS of URL</td>
</tr>
<tr>
<td>35</td>
<td>LMIR.DIR of URL</td>
</tr>
<tr>
<td>36</td>
<td>LMIR.JM of URL</td>
</tr>
<tr>
<td>37</td>
<td>BM25 of whole document</td>
</tr>
<tr>
<td>38</td>
<td>LMIR.ABS of whole document</td>
</tr>
<tr>
<td>39</td>
<td>LMIR.DIR of whole document</td>
</tr>
<tr>
<td>40</td>
<td>LMIR.JM of whole document</td>
</tr>
<tr>
<td>41</td>
<td>PageRank</td>
</tr>
<tr>
<td>42</td>
<td>Inlink number</td>
</tr>
<tr>
<td>43</td>
<td>Outlink number</td>
</tr>
<tr>
<td>44</td>
<td>Number of slash in URL</td>
</tr>
<tr>
<td>45</td>
<td>Length of URL</td>
</tr>
<tr>
<td>46</td>
<td>Number of child page</td>
</tr>
</tbody>
</table>
Microsoft L2R datasets

- Released in June 2009
- Web queries (10K and 30K sets)
- Judgments from a “retired” Bing labeling set (0..4)
- Data is rows of query-url values:
  - 0 qid:1 1:3 2:0 3:2 4:2 ... 135:0 136:0
  - 2 qid:1 1:3 2:3 3:0 4:0 ... 135:0 136:0
- 136 features, look like LETOR 4.0 set, but more
  - LETOR 4.0 had 46 features
  - L2R adds more aggregated statistics on TF, IDF, vector comparison, Boolean comparison, …
Yahoo! challenge (Mar’10)

• Primary challenge
  • 30K queries (20K train, 3K validation, 7K test)
  • 700K documents (comparable breakdown)

• Queries
  • Randomly sampled from (real) query log
  • Judged on PEGFB scale

• 700 features provided, 415 appear in all sets
  • Normalized to be in 0..1 range

• Privacy
  • Queries, URLs, and features not revealed
    • Microsoft L2R does not provides queries or URLs
  • No idea what semantics of features are
How to use learning-to-rank?

- **Develop feature sets is the most important step!**
  - Usually problem dependent
  - e.g., add contextual features if you hope to address contextual factors in search

- **Choose a good ranking model**
  - LambdaMART seems the best choice so far
  - RankLib implemented LambdaMART and many others

- **Training, validation, and testing**
  - similar to standard machine learning applications
RankLib Tutorial

• RankLib
  • A part of the Lemur/Indri toolkit, now a popular toolkit
  • Implemented almost all the state-of-the-art L2R models
  • All you need to do is to extract features by yourself
  • By Van Dang, Ph.D. UMass (2014), now at Google
  • https://sourceforge.net/projects/lemur/