Ranking and Learning

Adapted by Diana Inkpen, 2015, from Tao Yang, 2014.
Partially based on Manning, Raghavan, and Schütze’s text book.
• Weighted scoring for ranking
• Learning to rank: A simple example
• Learning to ranking as classification
Scoring

- Similarity-based approach
  - Similarity of query features with document features
- Weighted approach: Scoring with weighted features
  - *return in order the documents most likely to be useful to the searcher*
  - Consider each document has subscores in each feature or in each subarea.
Fred's Tropical Fish Shop is the best place to find tropical fish at low, low prices. Whether you're looking for a little fish or a big fish, we've got what you need. We even have fake seaweed for your fish tank (and little surfboards too).
Similarity ranking: example
[ Croft, Metzler, Strohman’s textbook slides]

\[ R(Q, D) = \sum_{i} g_i(Q) f_i(D) \]

- \( f_i \) is a document feature function
- \( g_i \) is a query feature function

Fred's Tropical Fish Shop is the best place to find tropical fish at low, low prices. Whether you’re looking for a little fish or a big fish, we’ve got what you need. We even have fake seaweed for your fishtank (and little surfboards too).

Topical Features
- fish: 9.7
- tropical: 4.2
- tropical fish: 22.1
- seaweed: 8.2
- surfboards: 4.2

Topical Features
- fish: 5.2
- tropical: 3.4
- tropical fish: 9.9
- chichlids: 1.2
- barbs: 0.7

Quality Features
- incoming links: 14
- update count: 3

Quality Features
- incoming links: 1.2
- update count: 0.9

Document Score: 303.01
Weighted scoring with linear combination

• A simple weighted scoring method: use a linear combination of subscores:
  ▪ E.g.,
    
    Score = 0.6*<Title score> + 0.3*<Abstract score> + 0.1*<Body score>

  ▪ The overall score is in [0,1].

Example with binary subscores

Query term appears in title and body only
Document score: (0.6 * 1) + (0.1 * 1) = 0.7.
Example

- On the query “bill rights” suppose that we retrieve the following docs from the various zone indexes:

<table>
<thead>
<tr>
<th>Title</th>
<th>bill</th>
<th>rights</th>
<th>1 → 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>bill</td>
<td>rights</td>
<td>3 → 5 → 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 → 5 → 9</td>
</tr>
<tr>
<td>Body</td>
<td>bill</td>
<td>rights</td>
<td>1 → 2 → 5 → 9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 → 5 → 8 → 9</td>
</tr>
</tbody>
</table>

Compute the score for each doc based on the weightings 0.6, 0.3, 0.1
How to determine weights automatically:

Motivation

• 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 highlighted on page?
    ▪ Span of query words on page
    ▪ # of (out) links on page?
    ▪ PageRank of page?
    ▪ URL length?
    ▪ URL contains “~”?
    ▪ Page edit recency?
    ▪ Page length?

• Major web search engines use “hundreds” of such features – and they keep changing
Machine learning for computing weights

• How do we combine these signals into a good ranker?
  ▪ “machine-learned relevance” or “learning to rank”

• Learning from examples
  ▪ These examples are called training data

_training examples_  →  _Ranking formula_  →  _User query and matched results_  →  _Ranked results_
Learning weights: Methodology

- Given a set of **training examples**, each contains (query q, document d, relevance score r(d,q)).
  - r(d,q) is relevance judgment for d on q
    - Simplest scheme
      - relevant (1) or nonrelevant (0)
    - More sophisticated: graded relevance judgments
      - 1 (Bad), 2 (Fair), 3 (Good), 4 (Excellent), 5 (Perfect)

- Learn weights from these examples, so that the learned scores approximate the relevance judgments in the training examples
Simple example

- Each doc has two zones, Title and Body
- For a chosen $w \in [0,1]$, score for doc $d$ on query $q$

$$score(d, q) = w \cdot s_T(d, q) + (1 - w) s_B(d, q)$$

where:
- $s_T(d, q) \in \{0,1\}$ is a Boolean denoting whether $q$ matches the Title and
- $s_B(d, q) \in \{0,1\}$ is a Boolean denoting whether $q$ matches the Body
### Examples of Training Data

<table>
<thead>
<tr>
<th>Example</th>
<th>DocID</th>
<th>Query</th>
<th>$s_T$</th>
<th>$s_B$</th>
<th>Judgment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Phi_1$</td>
<td>37</td>
<td>linux</td>
<td>1</td>
<td>1</td>
<td>Relevant</td>
</tr>
<tr>
<td>$\Phi_2$</td>
<td>37</td>
<td>penguin</td>
<td>0</td>
<td>1</td>
<td>Non-relevant</td>
</tr>
<tr>
<td>$\Phi_3$</td>
<td>238</td>
<td>system</td>
<td>0</td>
<td>1</td>
<td>Relevant</td>
</tr>
<tr>
<td>$\Phi_4$</td>
<td>238</td>
<td>penguin</td>
<td>0</td>
<td>0</td>
<td>Non-relevant</td>
</tr>
<tr>
<td>$\Phi_5$</td>
<td>1741</td>
<td>kernel</td>
<td>1</td>
<td>1</td>
<td>Relevant</td>
</tr>
<tr>
<td>$\Phi_6$</td>
<td>2094</td>
<td>driver</td>
<td>0</td>
<td>1</td>
<td>Relevant</td>
</tr>
<tr>
<td>$\Phi_7$</td>
<td>3191</td>
<td>driver</td>
<td>1</td>
<td>0</td>
<td>Non-relevant</td>
</tr>
</tbody>
</table>

From these 7 examples, learn the best value of $w$. 
How?

- For each example $\Phi_t$ we can compute the score based on $\text{score}(d_t, q_t) = w \cdot s_T(d_t, q_t) + (1 - w)s_B(d_t, q_t)$.

- We **quantify** Relevant as 1 and Non-relevant as 0.

- Would like the choice of $w$ to be such that the computed scores are as close to these 1/0 judgments as possible.
  - Denote by $r(d_t, q_t)$ the judgment for $\Phi_t$.

- Then minimize total **squared error**

$$\sum_{\Phi_t} (r(d_t, q_t) - \text{score}(d_t, q_t))^2$$
Optimizing $w$

- There are 4 kinds of training examples
- Thus only four possible values for score
  - And only 8 possible values for error
- Let $n_{01r}$ be the number of training examples for which $s_T(d, q)=0, s_B(d, q)=1$, judgment = Relevant.
- Similarly define $n_{00r}, n_{10r}, n_{11r}, n_{00i}, n_{01i}, n_{10i}, n_{11i}$

<table>
<thead>
<tr>
<th>$s_T$</th>
<th>$s_B$</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$1 - w$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$w$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Judgment=1 $\Rightarrow$ Error=$w$
Judgment=0 $\Rightarrow$ Error=$1-w$

Error: $[1-(1-\omega)]^2 n_{01r} + [0-(1-\omega)]^2 n_{01i}$
Total error – then calculus

- Add up contributions from various cases to get total error

\[(n_{01r} + n_{10i})w^2 + (n_{10r} + n_{01i})(1 - w)^2 + n_{00r} + n_{11i}.\]

- Now differentiate with respect to \(w\) to get optimal value of \(w\) as:

\[
\frac{n_{10r} + n_{01i}}{n_{10r} + n_{10i} + n_{01r} + n_{01i}}.
\]
Generalizing this simple example

- More (than 2) features
- Non-Boolean features
  - What if the title contains some but not all query terms …
  - Categorical features (query terms occur in plain, boldface, italics, etc)
- Scores are nonlinear combinations of features
- Multilevel relevance judgments (Perfect, Good, Fair, Bad, etc.)
- Complex error functions
- Not always a unique, easily computable setting of score parameters
Framework of Learning to Rank

How to sample the most appropriate training set?
How to extract the most useful features?
Learning-based Web Search

• Given features $e_1, e_2, \ldots, e_N$ for each document, learn a ranking function $f(e_1, e_2, \ldots, e_N)$ that minimizes the loss function $L$ under a query

$$f^* = \min_{f \in F} L(f(e_1, e_2, \ldots, e_N), \text{GroundTruth})$$

• Some related issues
  - The functional space $F$
    - linear/non-linear? continuous? Derivative?
  - The search strategy
  - The loss function
A richer example

- Collect a training corpus of \((q, d, r)\) triples
  - Relevance \(r\) is still binary for now
  - Document is represented by a feature vector
    - \(\mathbf{x} = (\alpha, \omega)\) \(\alpha\) is cosine similarity, \(\omega\) is minimum query window size
      - \(\omega\) is the shortest text span that includes all query words (Query term proximity in the document)

- Train a machine learning model to predict the class \(r\) of a document-query pair

<table>
<thead>
<tr>
<th>example</th>
<th>docID</th>
<th>query</th>
<th>cosine score</th>
<th>(\omega)</th>
<th>judgment</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\Phi_1)</td>
<td>37</td>
<td>linux operating system</td>
<td>0.032</td>
<td>3</td>
<td>relevant</td>
</tr>
<tr>
<td>(\Phi_2)</td>
<td>37</td>
<td>penguin logo</td>
<td>0.02</td>
<td>4</td>
<td>nonrelevant</td>
</tr>
<tr>
<td>(\Phi_3)</td>
<td>238</td>
<td>operating system</td>
<td>0.043</td>
<td>2</td>
<td>relevant</td>
</tr>
<tr>
<td>(\Phi_4)</td>
<td>238</td>
<td>runtime environment</td>
<td>0.004</td>
<td>2</td>
<td>nonrelevant</td>
</tr>
<tr>
<td>(\Phi_5)</td>
<td>1741</td>
<td>kernel layer</td>
<td>0.022</td>
<td>3</td>
<td>relevant</td>
</tr>
<tr>
<td>(\Phi_6)</td>
<td>2094</td>
<td>device driver</td>
<td>0.03</td>
<td>2</td>
<td>relevant</td>
</tr>
<tr>
<td>(\Phi_7)</td>
<td>3191</td>
<td>device driver</td>
<td>0.027</td>
<td>5</td>
<td>nonrelevant</td>
</tr>
</tbody>
</table>
Using classification for deciding relevance

- A linear score function is
  \[ \text{Score}(d, q) = \text{Score}(\alpha, \omega) = a\alpha + b\omega + c \]
- And the linear classifier is
  Decide relevant if \( \text{Score}(d, q) > \theta \)
  Otherwise irrelevant

- ... just like when we were doing classification
Using classification for deciding relevance

![Graph showing cosine score vs. term proximity with decision surface]
More complex example of using classification for search ranking

[Nallapati SIGIR 2004]

- We can generalize this to classifier functions over more features
- We can use methods we have seen previously for learning the linear classifier weights
An SVM classifier for relevance
[Nallapati SIGIR 2004]

• Let \( g(r|d,q) = w \cdot f(d,q) + b \)
• Derive weights from the training examples:
  - want \( g(r|d,q) \leq -1 \) for nonrelevant documents
  - \( g(r|d,q) \geq 1 \) for relevant documents
• Testing:
  - decide relevant iff \( g(r|d,q) \geq 0 \)
• Train a classifier as the ranking function
Ranking vs. Classification

- **Classification**
  - Well studied over 30 years
  - Bayesian, Neural network, Decision tree, SVM, Boosting, …
  - Training data: points
    - Pos: x1, x2, x3,
    - Neg: x4, x5

- **Ranking**
  - Less studied: only a few works published in recent years
  - Training data: pairs (partial order)
    - Correct order: (x1, x2), (x1, x3), (x1, x4), (x1, x5)
      - (x2, x3), (x2, x4) …
    - Other order is incorrect
Learning to rank: Classification vs. regression

• Classification probably isn’t the right way to think about score learning:
  ▪ Classification problems: Map to an unordered set of classes
  ▪ Regression problems: Map to a real value
  ▪ Ordinal regression problems: Map to an ordered set of classes

• This formulation gives extra power:
  ▪ Relations between relevance levels are modeled
  ▪ Some documents are better than other documents for some queries; not an absolute scale of goodness
• Assume a number of categories $C$ of relevance exist
  ▪ These are totally ordered: $c_1 < c_2 < \ldots < c_j$
  ▪ This is the ordinal regression setup

• Assume training data is available consisting of document-query pairs represented as feature vectors $\psi_i$ and relevance ranking $c_i$
Modified example

- Collect a training corpus of \((q, d, r)\) triples
  - Relevance label \(r\) has 4 values
    - Perfect, Relevant, Weak, Nonrelevant
- Train a machine learning model to predict the class \(r\) of a document-query pair

<table>
<thead>
<tr>
<th>example</th>
<th>docID</th>
<th>query</th>
<th>cosine score</th>
<th>(\omega)</th>
<th>judgment</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\Phi_1)</td>
<td>37</td>
<td>linux operating system</td>
<td>0.032</td>
<td>3</td>
<td>Perfect</td>
</tr>
<tr>
<td>(\Phi_2)</td>
<td>37</td>
<td>penguin logo</td>
<td>0.02</td>
<td>4</td>
<td>Nonrelevant</td>
</tr>
<tr>
<td>(\Phi_3)</td>
<td>238</td>
<td>operating system</td>
<td>0.043</td>
<td>2</td>
<td>Relevant</td>
</tr>
<tr>
<td>(\Phi_4)</td>
<td>238</td>
<td>runtime environment</td>
<td>0.004</td>
<td>2</td>
<td>Weak</td>
</tr>
<tr>
<td>(\Phi_5)</td>
<td>1741</td>
<td>kernel layer</td>
<td>0.022</td>
<td>3</td>
<td>Relevant</td>
</tr>
<tr>
<td>(\Phi_6)</td>
<td>2094</td>
<td>device driver</td>
<td>0.03</td>
<td>2</td>
<td>Perfect</td>
</tr>
<tr>
<td>(\Phi_7)</td>
<td>3191</td>
<td>device driver</td>
<td>0.027</td>
<td>5</td>
<td>Nonrelevant</td>
</tr>
</tbody>
</table>
“Learning to rank”

• **Point-wise learning**
  - Given a query-document pair, predict a score (e.g., relevancy score)

• **Pair-wise learning**
  - the input is a pair of results for a query, and the class is the relevance ordering relationship between them

• **List-wise learning**
  - Directly optimize the ranking metric for each query
Point-wise learning: Example

- Goal is to learn a threshold to separate each rank
The Ranking SVM: Pairwise Learning
[Herbrich et al. 1999, 2000; Joachims et al. KDD 2002]

- Aim is to classify instance pairs as
  - correctly ranked
  - or incorrectly ranked
- This turns an ordinal regression problem back into a binary classification problem
- We want a ranking function $f$ such that $c_i$ is ranked before $c_k$:
  \[ c_i < c_k \text{ iff } f(\psi_i) > f(\psi_k) \]
- Suppose that $f$ is a linear function
  \[ f(\psi_i) = w \cdot \psi_i \]
- Thus
  \[ c_i < c_k \text{ iff } w(\psi_i - \psi_k) > 0 \]
Ranking SVM

• **Training Set**
  - for each query $q$, we have a ranked list of documents totally ordered by a person for relevance to the query.

• **Features**
  - vector of features for each document/query pair
  - feature differences for two documents $d_i$ and $d_j$

    $$\psi_j = \psi(d_j, q)$$

    $$\Phi(d_i, d_j, q) = \psi(d_i, q) - \psi(d_j, q)$$

• **Classification**
  - if $d_i$ is judged more relevant than $d_j$, denoted $d_i < d_j$
  - then assign the vector $\Phi(d_i, d_j, q)$ the class $y_{ijq} = +1$; otherwise $-1$. 
Optimization problem is equivalent to that of a classification SVM on pairwise difference vectors $\Phi(q_k, d_i) - \Phi(q_k, d_j)$.