Ranking and Learning

290N UCSB, Tao Yang, 2013

Partially based on Manning, Raghavan, and Schütze's text book.

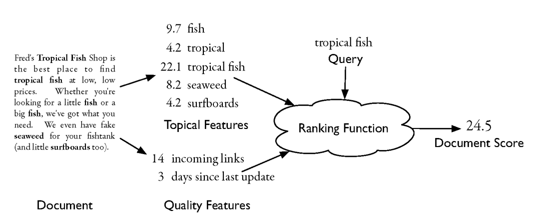
Table of Content

- **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.

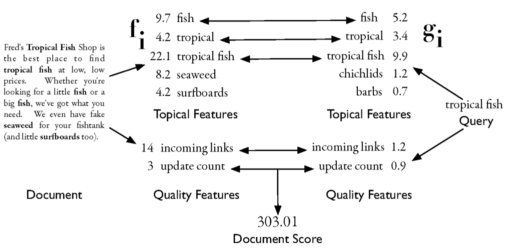
Simple Model of Ranking with Similarity



Similarity ranking: example

$$R(Q,D) = \sum_i g_i(Q) f_i(D)$$

f_i is a document feature function
g_i is a query feature function



Weighted scoring with linear combination

- **A simple weighted scoring method: use a linear combination of subscores:**
  - E.g.,

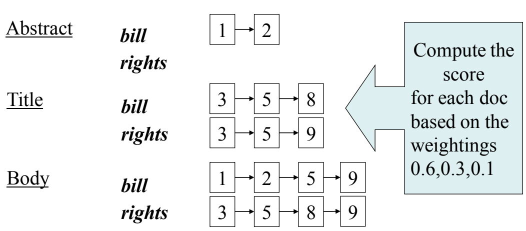
  Score = 0.6*< Title score> + 0.3*<Abstract score> + 0.1*<Body score>
  - The overall score is in [0,1].

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:**

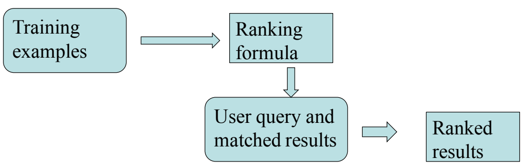


How to determine weights automatically: Motivation

- **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



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

10

Simple example

- **Each doc has two zones, Title and Body**
- **For a chosen *w*∈[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

Learning *w* from training examples

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Non-relevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Non-relevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3191	driver	1	0	Non-relevant

From these 7 examples, learn the best value of *w*.

How?

- For each example Φ_t we can compute the score based on $score(d_t, q_t) = w \cdot s_T(d_t, q_t) + (1-w)s_B(d_t, q_t)$
- 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 Φ_t
- Then minimize total squared error

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

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}

s_T	s_B	Score
0	0	0
0	1	$1-w$
1	0	w
1	1	1

Judgment=1 ⇒ Error=*w*

Judgment=0 ⇒ 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

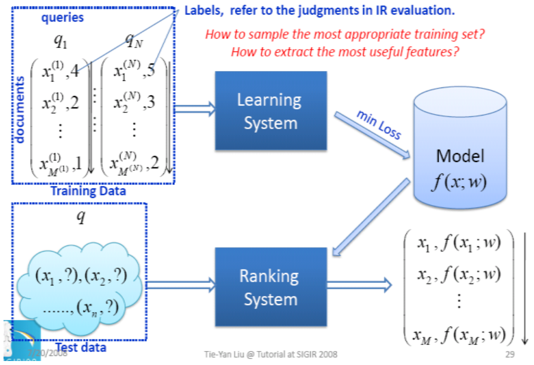
Learning-based Web Search

- **Given a set of features $e_1, e_2, \ldots, e_N$, learn a ranking function $f(e_1, e_2, \ldots, e_N)$ that minimizes the loss function *L*.**

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

- **Some related issues**
  - The functional space *F*
    - linear/non-linear? continuous? Derivative?
  - The search strategy
  - The loss function

Framework of Learning to Rank



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
    - **x** = (α, ω)   α is cosine similarity, ω is minimum query window size
      - ω 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**

example	docID	query	cosine score	ω	judgment
Φ_1	37	linux operating system	0.032	3	*relevant*
Φ_2	37	penguin logo	0.02	4	*nonrelevant*
Φ_3	238	operating system	0.043	2	*relevant*
Φ_4	238	runtime environment	0.004	2	*nonrelevant*
Φ_5	1741	kernel layer	0.022	3	*relevant*
Φ_6	2094	device driver	0.03	2	*relevant*
Φ_7	3191	device driver	0.027	5	*nonrelevant*

Using classification for deciding relevance

- **A linear score function is**

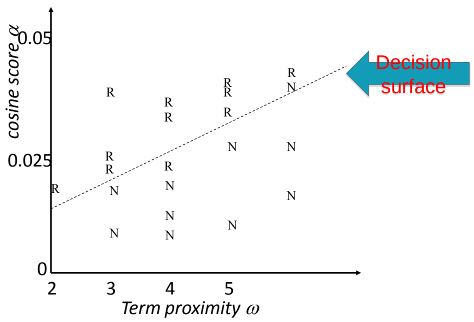
  Score(d, q) = Score(α, ω) = aα + bω + c

- **And the linear classifier is**

  Decide relevant if *Score(d, q) > θ*

- **… just like when we were doing text classification**

Using classification for deciding relevance



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**

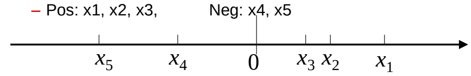
An SVM classifier for relevance

[Nallapati SIGIR 2004]

- Let $g(r|d,q) = \mathbf{w} \bullet f(d,q) + b$
- 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 $g(r|d,q) \geq 0$
- Use SVM classifier

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)
    - (x1, x2), (x1, x3), (x1, x4), (x1, x5)
    - (x2, x3), (x2, x4) …
    - …

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
  - Documents are good versus other documents for query given collection; not an absolute scale of goodness

"Learning to rank"

- **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 ψ_i and relevance ranking c_i**

Modified example

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

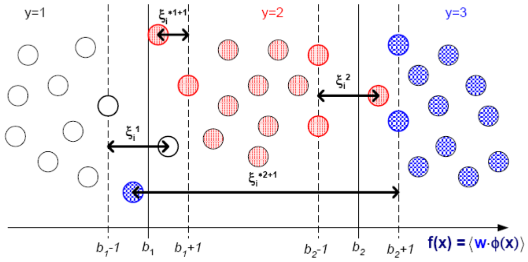
example	docID	query	cosine score	ω	judgment
Φ_1	37	linux operating system	0.032	3	*Perfect*
Φ_2	37	penguin logo	0.02	4	*Nonrelevant*
Φ_3	238	operating system	0.043	2	*Relevant*
Φ_4	238	runtime environment	0.004	2	*Weak*
Φ_5	1741	kernel layer	0.022	3	*Relevant*
Φ_6	2094	device driver	0.03	2	*Perfect*
Φ_7	3191	device driver	0.027	5	*Nonrelevant*

"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**

  $$f(\psi_i) = w \bullet \psi_i$$
- **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

    $$\psi_j = \psi(d_j, q)$$
  - feature differences for two documents d_i and d_i

    $$\Phi(d_i, d_j, q) = \psi(d_i, q) - \psi(d_j, q)$$
- **Classification**
  - if d_i is judged more relevant than d_j, denoted $d_i \prec d_j$
  - then assign the vector $\Phi(d_i, d_j, q)$ the class y_{ijq} =+1; otherwise −1.

Ranking SVM

OPTIMIZATION PROBLEM 1. (RANKING SVM)

minimize: $$V(\vec{w}, \vec{\xi}) = \frac{1}{2}\, \vec{w} \cdot \vec{w} + C \sum \xi_{i,j,k} \quad (12)$$

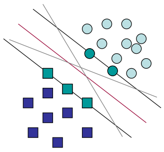
subject to:

$$\forall (d_i, d_j) \in r_1^* : \vec{w}\Phi(q_1, d_i) \geq \vec{w}\Phi(q_1, d_j) + 1 - \xi_{i,j,1}$$

$$\ldots \quad (13)$$

$$\forall (d_i, d_j) \in r_n^* : \vec{w}\Phi(q_n, d_i) \geq \vec{w}\Phi(q_n, d_j) + 1 - \xi_{i,j,n}$$

$$\forall i \forall j \forall k : \xi_{i,j,k} \geq 0 \quad (14)$$



- **optimization problem is equivalent to that of a classification SVM on pairwise difference vectors $\Phi(q_k, d_i)$ - $\Phi(q_k, d_j)$**