Introduction to Information Retrieval

Learning to rank

Overview

1. Learning Boolean Weights
2. Learning Real-Valued Weights
3. Rank Learning as Ordinal Regression

Outline

Main Idea

- The aim of term weights (e.g. TF-IDF) is to measure term salience
  - Summing up term weights for a document allows to measure the *relevance* of a document to a query, hence to *rank* the document
- Think of this as a text classification problem
  - Term weights can be *learned* using training examples that have been judged
- This methodology falls under a general class of approaches known as **machine learned relevance** or **learning to rank**

Learning weights

- Main methodology
  - Given a set of **training examples**, each of which is a tuple of: a query *q*, a document *d*, a relevance judgment for *d* on *q (R(d,q))*
    - Simplest case: *R*(*d*, *q*) is either relevant (1) or non-relevant (0)
    - More sophisticated cases: graded relevance judgments
  - Learn weights from these examples, so that the learned scores approximate the relevance judgments in the training examples.
    - Example: *Weighted zone scoring*

What is weighted zone scoring?

- Given a query and a collection, where documents have three *zones* (a.k.a. *fields*): author, title, body
- Weighted zone scoring requires a separate weight for each zone, e.g. g_1, g_2, g_3
- Not all zones are equally important:
  - Example
    - author < title < body -> $g_1 = 0.2$, $g_2 = 0.3$, $g_3 = 0.5$ (so that they add up to 1)
- Score for a zone = 1 if the query term occurs in that zone, 0 otherwise **(Boolean)**
  - **Example:**
    - Query term appears in title and body only
    - Document score: $(0.3 \cdot 1) + (0.5 \cdot 1) = 0.8$.

Weighted zone scoring: let's generalise

- Given *q* and *d*, weighted zone scoring assigns to the pair (*q*, *d*) a score in the interval [0,1] by computing a **linear combination** of document zone scores, where each zone contributes a value
  - Consider a set of documents, which have *l* zones
  - Let $g_1, \ldots, g_l \in [0, 1]$, such that $\sum_{i=1}^{l} g_i = 1$
- For $1 \leq i \leq l$, let s_i be the Boolean score denoting a match (or non-match) between *q* and the *i*th zone
  - E.g. s_i could be any boolean function that maps the presence of query terms in a zone to {0,1}
- **Weighted zone score a.k.a ranked Boolean retrieval**

$$\sum_{i=1}^{l} g_i s_i$$

Weighted zone scoring and learning weights

- Weighted zone scoring may be viewed as **learning a linear function** of the Boolean match scores contributed by the various zones
- Bad news: labour-intensive assembly of user-generated relevance judgments from which to learn the weights
    - Especially in a dynamic collection (such as the Web)
- Good news: reduce the problem of learning the weights g_i to a simple optimisation problem

Learning weights in weighted zone scoring

- Simple case: let documents have two zones: title, body
- The weighted zone scoring formula we saw before:

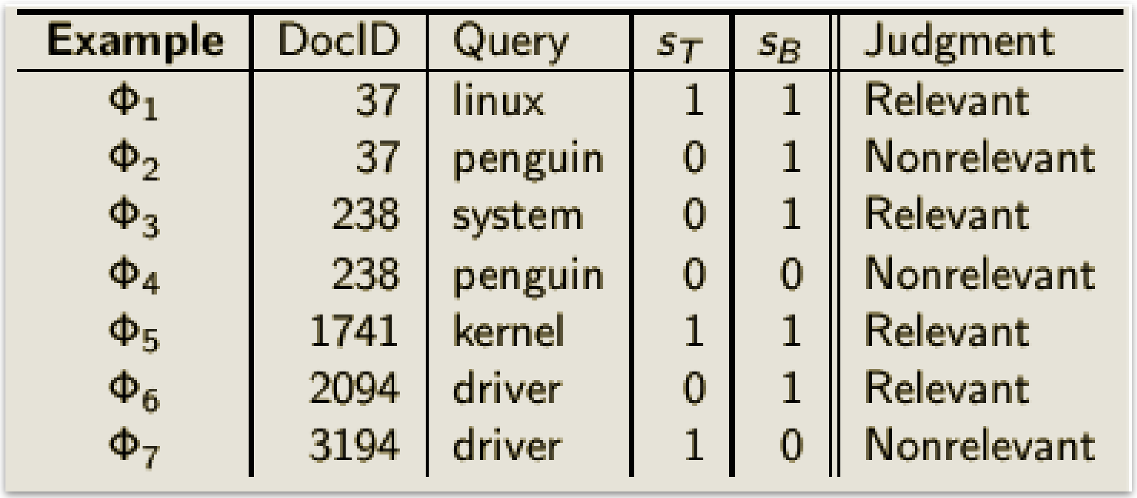
$$\sum_{i=1}^{l} g_i s_i \qquad (2)$$

- Given (q, d), we want to compute $s_T(d, q)$ and $s_B(d, q)$, depending whether the title or body zone of d matches query *q*
- We compute a score between 0 and 1 for each (d, q) pair using $s_T(d, q)$ and $s_B(d, q)$ by using a constant $g \in [0, 1]$:

$$\mathbf{score}(d, q) = g \cdot s_T(d, q) + (1 - g) \cdot s_B(d, q) \qquad (3)$$

Learning weights: determine *g* from training examples

Example



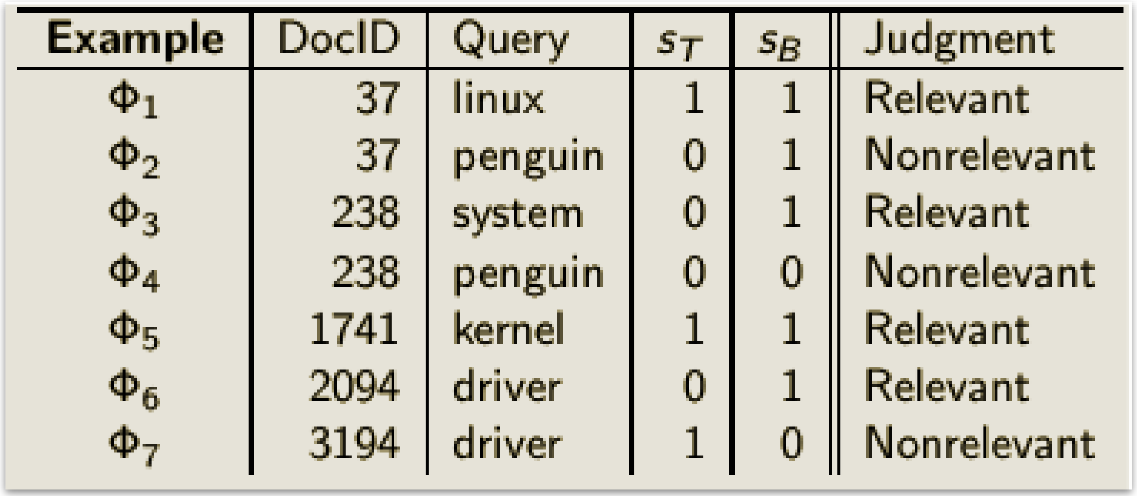
Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Nonrelevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Nonrelevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3194	driver	1	0	Nonrelevant

- Training examples: triples of the form $\Phi_j = (d_j , q_j , r (d_j , q_j))$
- A given training document d_j and a given training query q_j are assessed by a human who decides $r (d_j , q_j)$ (either relevant or non-relevant)

10

Learning weights: determine *g* from training examples

Example



Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Nonrelevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Nonrelevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3194	driver	1	0	Nonrelevant

- For each training example Φ_j we have Boolean values $s_T(d_j, q_j)$ and $s_B(d_j, q_j)$ that we use to compute a score from:

$$score(d_j, q_j) = g \cdot s_T(d_j, q_j) + (1 - g) \cdot s_B(d_j, q_j) \qquad (4)$$

11

Learning weights

- We compare this computed score (score(*dj* , *qj*)) with the human relevance judgment for the same document-query pair (*dj* , *qj*)
    - We quantize each relevant judgment as 1, and each non-relevant judgment as 0
- We define the error of the scoring function with weight g as

$$\epsilon (g,\Phi_\mathrm{j}) = (r(d_j, q_j) - \mathrm{score}(d_j, q_j))^2$$

(5)

Learning weights

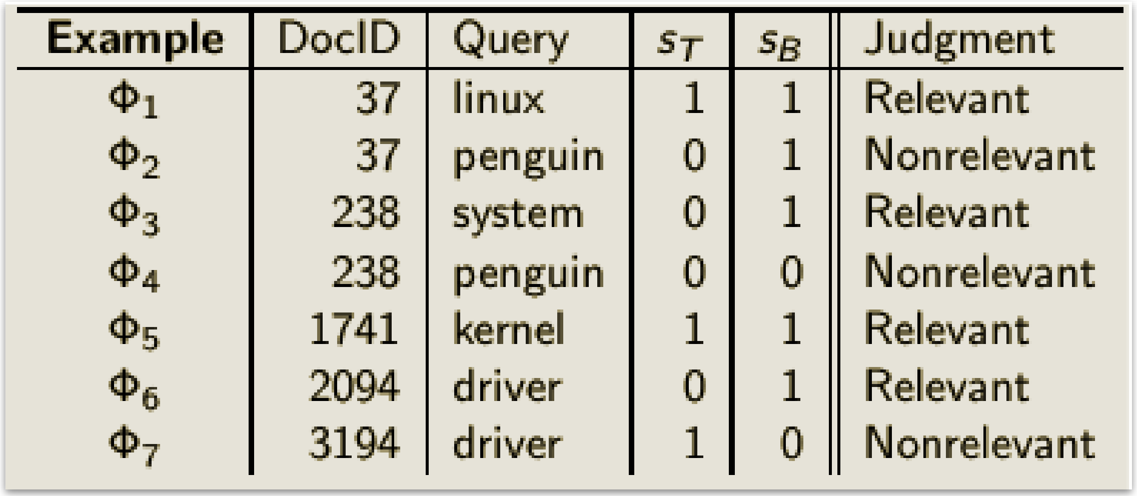
- Then, the total error of a set of training examples is given by

$$\sum_j \epsilon\,(g, \Phi_j) \tag{6}$$

- The problem of learning the constant g from the given training examples then reduces to picking the value of g that minimises the total error

Exercise: Find the value of *g* that minimises total error ϵ

Example



Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Nonrelevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Nonrelevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3194	driver	1	0	Nonrelevant

1 Quantize: relevant as 1, and non-relevant as 0

2 Compute score:

$score(d_j, q_j) = g \cdot s_T(d_j, q_j) + (1 - g) \cdot s_B(d_j, q_j)$

3 Compute total error: $\sum_j \epsilon(g, \Phi_j)$, where

$\epsilon(g, \Phi_j) = (r(d_j, q_j) - \text{score}(d_j, q_j))^2$

4 Pick the value of *g* that minimises the total error

14

Exercise solution

❶ Compute score (d_j, q_j)

$score(d_1, q_1) = g \cdot 1 + (1 - g) \cdot 1 = g + 1 - g = 1$

$score(d_2, q_2) = g \cdot 0 + (1 - g) \cdot 1 = 0 + 1 - g = 1 - g$

$score(d_3, q_3) = g \cdot 0 + (1 - g) \cdot 1 = 0 + 1 - g = 1 - g$

$score(d_4, q_4) = g \cdot 0 + (1 - g) \cdot 0 = 0 + 0 = 0$

$score(d_5, q_5) = g \cdot 1 + (1 - g) \cdot 1 = g + 1 - g = 1$

$score(d_6, q_6) = g \cdot 0 + (1 - g) \cdot 1 = 0 + 1 - g = 1 - g$

$score(d_7, q_7) = g \cdot 1 + (1 - g) \cdot 0 = g + 0 = g$

❷ Compute total error $\sum_j \epsilon(g, \Phi_j)$

$(1 - 1)^2 + (0 - 1 + g)^2 + (1 - 1 + g)^2 + (0 - 0)^2 + (1 - 1)^2 + (1 - 1 + g)^2 + (0 - g)^2$

❸ Pick the value of g that minimises the total error

Solve by ranging g between 0.1 - 0.9 and pick the g value that minimises the error

15

Outline

A simple example of machine learned scoring

- So far, we considered a case where we had to combine Boolean indicators of relevance
- Now consider more general factors that go beyond Boolean functions of query term presence in document zones

A simple example of machine learned scoring

- Setting: the scoring function is a linear combination of two factors:
    - the vector space cosine similarity between query and document (denoted α)
    - the minimum window width within which the query terms lie (denoted ω)
        - query term proximity is often very indicative of topical relevance
        - query term proximity gives an implementation of implicit phrases
- Thus, we have one factor that depends on the statistics of query terms in the document as a bag of words, and another that depends on proximity weighting

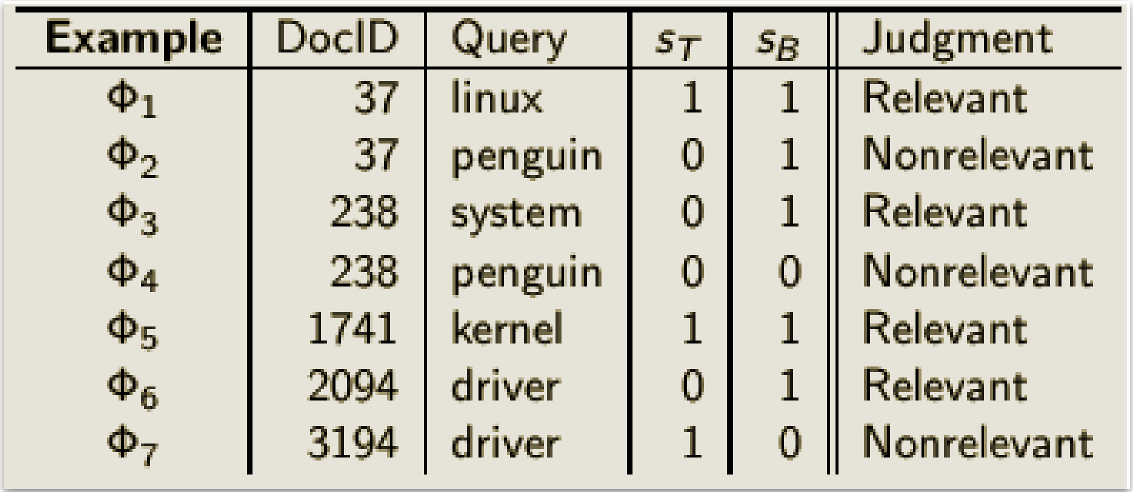
A simple example of machine learned scoring

Given a set of *training examples r* (d_j , q_j). For each example we compute:

- vector space cosine similarity α
- window width ω

The result is a training set, with two real-valued features (α, ω)

Example



Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Nonrelevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Nonrelevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3194	driver	1	0	Nonrelevant

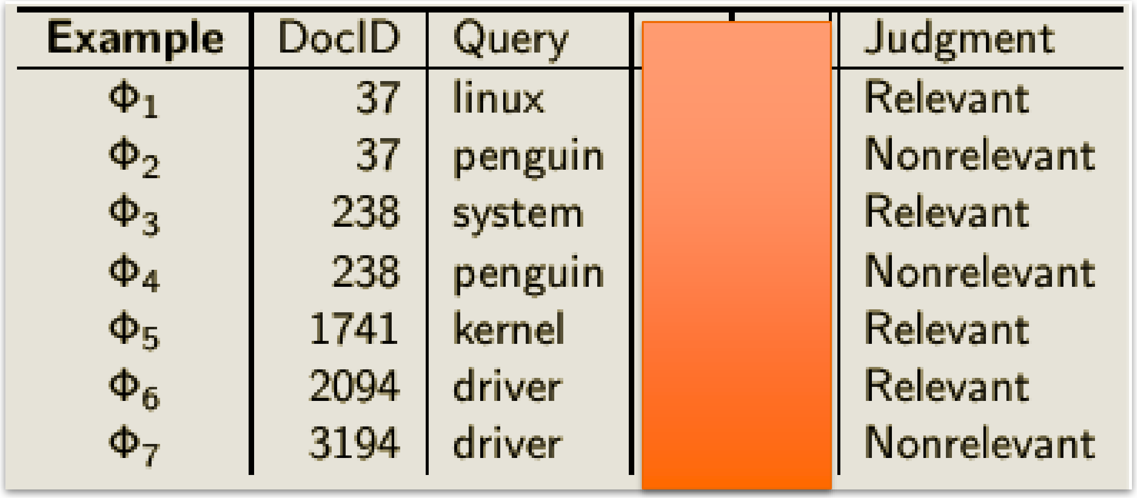
A simple example of machine learned scoring

Given a set of *training examples r* (d_j , q_j). For each example we compute:

- vector space cosine similarity α
- window width ω

The result is a training set, with two real-valued features (α, ω)

Example



Example	DocID	Query	[redacted]	[redacted]	Judgment
Φ_1	37	linux	[redacted]	[redacted]	Relevant
Φ_2	37	penguin	[redacted]	[redacted]	Nonrelevant
Φ_3	238	system	[redacted]	[redacted]	Relevant
Φ_4	238	penguin	[redacted]	[redacted]	Nonrelevant
Φ_5	1741	kernel	[redacted]	[redacted]	Relevant
Φ_6	2094	driver	[redacted]	[redacted]	Relevant
Φ_7	3194	driver	[redacted]	[redacted]	Nonrelevant

20

A simple example of machine learned scoring

- Again, let’s say: relevant = 1 and non-relevant = 0
- We now seek a scoring function that combines the values of the features to generate a value that is (close to) 0 or 1
- We wish this function to be in agreement with our set of training examples as much as possible
- Without loss of generality, a linear classifier will use a linear

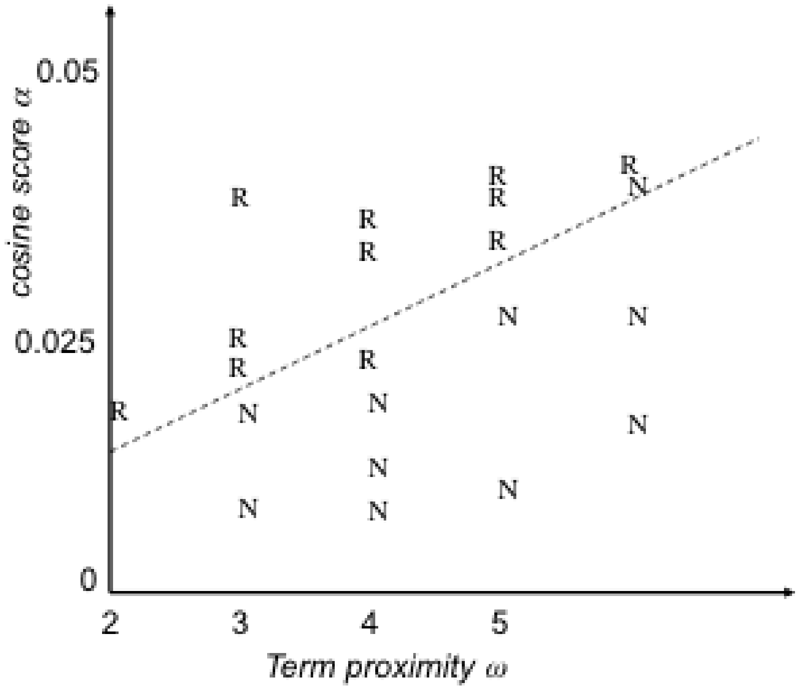
  combination of features of the form:

$$\boldsymbol{Score(d, q) = Score(\alpha, \omega) = a\alpha + b\omega + c,} \qquad \textbf{(7)}$$

with the coefficients *a*, *b*, *c* to be learned from the training data

A simple example of machine learned scoring

- We use *thresholding*: for a query, document pair, we pick a value θ
- if *Score*(*α*, *ω*) > θ, we declare the document *relevant*, otherwise we declare it *non-relevant*
- As we know from SVMs, all points that satisfy *Score*(*α*, *ω*) = θ form a line (dashed here) → linear classifier that separates relevant from non-relevant instances



22

A simple example of machine learned scoring

Thus, the problem of making a binary *relevant/non-relevant* judgment given training examples turns into one of learning the dashed line in the figure separating *relevant* from *non-relevant* training examples

- In the *α-ω* plane, this line can be written as a linear equation involving *α* and *ω*, with two parameters (slope and intercept)
- We have already seen linear classification methods for choosing this line
- Provided we can build a sufficiently rich collection of training samples, we can thus altogether avoid hand-tuning score functions
- Bottleneck: maintaining a suitably representative set of training examples, whose relevance assessments must be made by experts

Result ranking by machine learning

- The above ideas can be readily generalized to functions of **many more than two** variables
- In addition to cosine similarity and query term window, there are lots of other indicators of relevance, e.g. PageRank-style measures, document age, zone contributions, document length, etc.
- If these measures can be calculated for a training document collection with relevance judgments, any number of such measures can be used to train a machine learning classifier

24

134 Features released from Microsoft Research on 16 June 2010

http://research.microsoft.com/en-us/projects/mslr/feature.aspx
Zones: body, anchor, title, url, whole document
Features: query term number, query term ratio, stream length,idf, sum of term frequency, min of term frequency, max of term frequency, mean of term frequency, variance of term frequency, sum of stream length normalized term frequency, min of stream length normalized term frequency, max of stream length normalized term frequency, mean of stream length normalized term frequency, variance of stream length normalized term frequency, sum of tf*idf, min of tf*idf, max of tf*idf, mean of tf*idf, variance of tf*idf, boolean model, vector space model, BM25, LMIR.ABS, LMIR.DIR, LMIR.JM, number of slash in url, length of url, inlink number, outlink number, PageRank, SiteRank, QualityScore, QualityScore2, query-url click count, url click count, url dwell time.

Result ranking by machine learning

- However, approaching IR ranking like this is not necessarily the right way to think about the problem
  - Statisticians normally first divide problems into **classification** problems (where a categorical variable is predicted) versus **regression** problems (where a real number is predicted)
  - In between is the specialised field of **ordinal regression** where a ranking is predicted
  - Machine learning for ad hoc retrieval is most properly thought of as an ordinal regression problem, where the goal is to rank a set of documents for a query, given training data of the same sort

26

Outline

IR ranking as ordinal regression

- Why formulate IR ranking as an ordinal regression problem?
  - because documents can be evaluated relative to other candidate documents for the same query, rather than having to be mapped to a global scale of goodness
  - hence, the problem space weakens, since just a ranking is required rather than an absolute measure of relevance
- Especially germane in web search, where the ranking at the very top of the results list is exceedingly important

The construction of a ranking SVM

- We begin with a set of judged queries
- For each training query q, we have a set of documents returned in response to the query, which have been totally ordered by a person for relevance to the query
- We construct a vector of features $\psi_j = \psi(d_j , q)$ for each document/query pair, using features such as those discussed, and many more
- For two documents d_i and d_j , we then form the vector of feature differences:

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

(8)

29

The construction of a ranking SVM

- By hypothesis, one of d_i and d_j has been judged more relevant
- If d_i is judged more relevant than d_j, denoted $d_i \prec d_j$ (d_i should precede d_j in the results ordering), then we will assign the vector $\Phi(d_i, d_j, q)$ the class $y_{ijq} = +1$; otherwise -1
- The goal then is to build a classifier which will return

$$\vec{w}^{\mathsf{T}} \Phi(d_i, d_j, q) > 0 \text{ iff } d_i \prec d_j \qquad (9)$$

30

Ranking SVM

- This approach has been used to build ranking functions which outperform standard hand-built ranking functions in IR evaluations on standard data sets
- See the references for papers that present such results (page 316)

Recap

- The idea of learning ranking functions has been around for a number of years, but it is only very recently that sufficient machine learning knowledge, training document collections, and computational power have come together to make this method practical and exciting
- While skilled humans can do a very good job at defining ranking functions by hand, hand tuning is difficult, and it has to be done again for each new document collection and class of users