# Information Storage and Retrieval

CSCE 670
Texas A&M University
Department of Computer Science & Engineering
Instructor: Prof. James Caverlee

**Learning to Rank (again)**
**28 February 2017**

# Learning to rank algorithms

Least Square Retrieval Function (TOIS 1989)

Query refinement (WWW 2008)

SVM-MAP (SIGIR 2007)

Nested Ranker (SIGIR 2006)

ListNet (ICML 2007)

Pranking (NIPS 2002)

MPRank (ICML 2007)

LambdaRank (NIPS 2006)

Frank (SIGIR 2007)

Learning to retrieval info (SCC 1995)

MHR (SIGIR 2007)

RankBoost (JMLR 2003)

LDM (SIGIR 2005)

Large margin ranker (NIPS 2002)

IRSVM (SIGIR 2006)

Ranking SVM (ICANN 1999)

RankNet (ICML 2005)

Discriminative model for IR (SIGIR 2004)

SVM Structure (JMLR 2005)

OAP-BPM (ICML 2003)

Subset Ranking (COLT 2006)

GPRank (LR4IR 2007)

QBRank (NIPS 2007)

GBRank (SIGIR 2007)

Constraint Ordinal Regression (ICML 2005)

McRank (NIPS 2007)

SoftRank (LR4IR 2007)

AdaRank (SIGIR 2007)

CCA (SIGIR 2007)

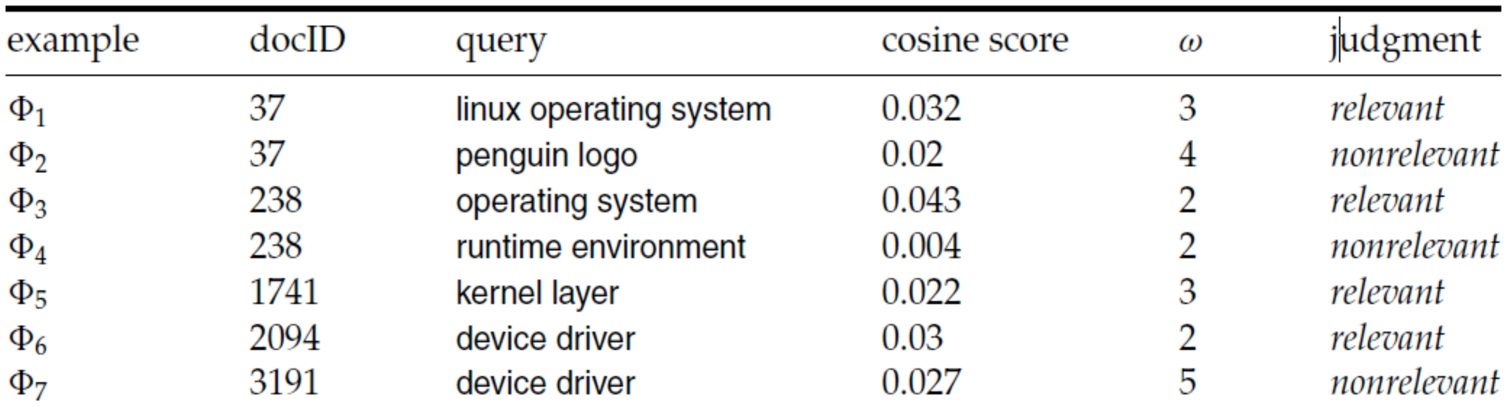
ListMLE (ICML 2008)

RankCosine (IP&M 2007)

Supervised Rank Aggregation (WWW 2007)

# Simple example: Using classification for ad hoc IR

- Collect a training corpus of (q,d,r) triples
  - Relevance r is binary
  - Document is represented by a feature vector
  - **x**=(α, ω) : α is cosine similarity; ω is minimum query window size
    - Query term proximity is a very important new weighting factor

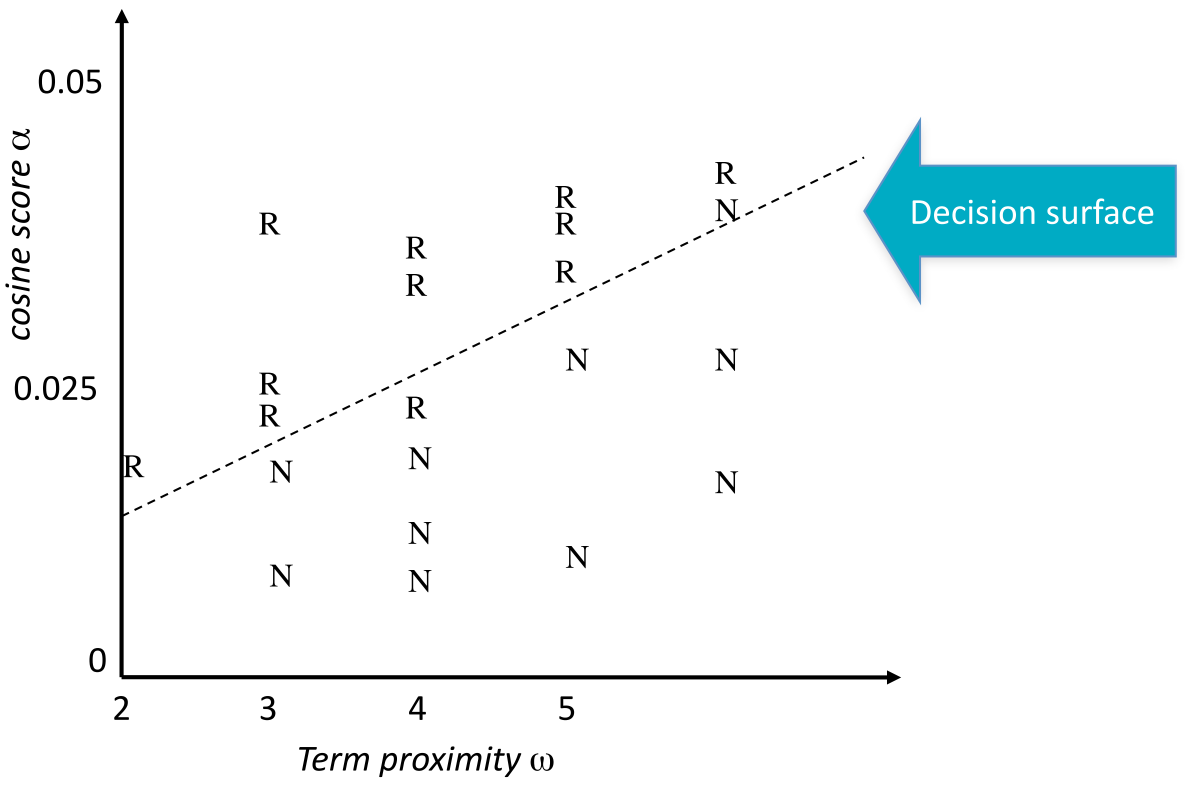


| example | docID | query | cosine score | $\omega$ | judgment |
|---|---|---|---|---|---|
| $\Phi_1$ | 37 | linux operating system | 0.032 | 3 | *relevant* |
| $\Phi_2$ | 37 | penguin logo | 0.02 | 4 | *nonrelevant* |
| $\Phi_3$ | 238 | operating system | 0.043 | 2 | *relevant* |
| $\Phi_4$ | 238 | runtime environment | 0.004 | 2 | *nonrelevant* |
| $\Phi_5$ | 1741 | kernel layer | 0.022 | 3 | *relevant* |
| $\Phi_6$ | 2094 | device driver | 0.03 | 2 | *relevant* |
| $\Phi_7$ | 3191 | device driver | 0.027 | 5 | *nonrelevant* |

# Simple example: Using classification for ad hoc IR

- A linear score function is then:
  - Score(d,q) = Score($\alpha$, $\omega$) = a $\alpha$ + b $\omega$ + c
- And the linear classifier is:
  - Decide relevant if Score(q,d) > $\theta$
- ... this is exactly like text classification

# Simple example: Using classification for ad hoc IR



## Extending the model

- We can generalize this to classifier functions over more features

## Machine learning for IR ranking

- This “good idea” has been actively researched and actively deployed at major web search engines in the last 5 years
- Why didn’t it happen earlier?
  - Modern supervised ML has been around for about 15 years
  - Naive Bayes has been around for about 45 years!

# Machine learning for IR ranking

- There's some truth to the fact that the IR community wasn't very connected to the ML community
- But there were a whole bunch of precursors:
  - Wong, S.K. et al. 1988. Linear structure in information retrieval. SIGIR 1988.
  - Fuhr, N. 1992. Probabilistic methods in information retrieval. Computer Journal.
  - Gey, F. C. 1994. Inferring probability of relevance using the method of logistic regression. SIGIR 1994.
  - Herbrich, R. et al. 2000. Large Margin Rank Boundaries for Ordinal Regression. Advances in Large Margin Classifiers.

## 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

- Microsoft LETOR

# Why wasn't ML much needed?

- Traditional ranking functions in IR used a very small number of features, e.g.,
  - Term frequency
  - Inverse document frequency
  - Document length
- It was easy to tune weighting coefficients by hand
  - And people did

## Why is ML needed now

- 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 edit recency?
  - Page length?
- The New York Times (2008-06-03) quoted Amit Singhal as saying Google was using **over 200 such features.**

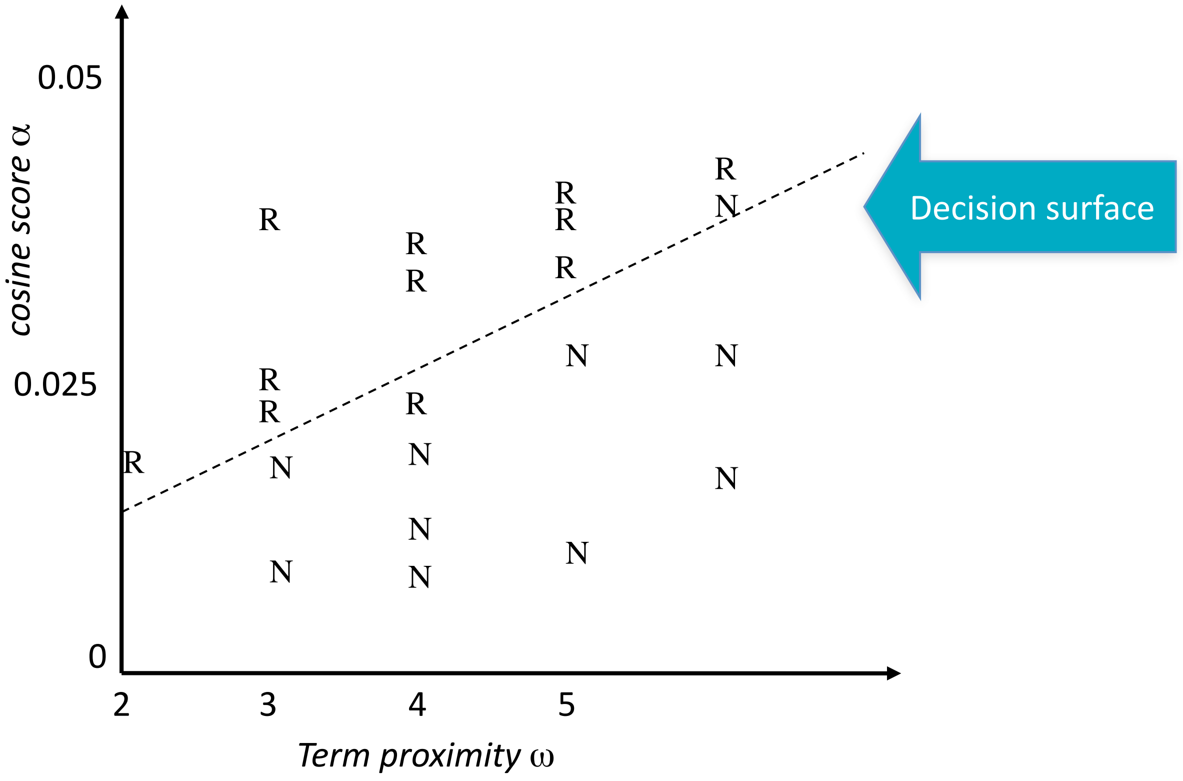
# 134 Features released from Microsoft Research on 16 June 2010

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

# Simple example: Using classification for ad hoc IR



# "Learning to rank"

- Classification probably isn't the right way to think about approaching ad hoc IR:
  - Classification problems: Map to a unordered set of classes
  - Regression problems: Map to a real value
  - Ordinal regression problems: Map to an ordered set of classes
    - A fairly obscure sub-branch of statistics, but what we want here
- 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: c1 < c2 < … < cJ
  - This is the ordinal regression setup
- Assume training data is available consisting of document-query pairs represented as feature vectors ψi and relevance ranking ci
- We could do **point-wise learning**, where we try to map items of a certain relevance rank to a subinterval (e.g, Crammer et al. 2002 PRank)
- But most work does **pair-wise learning**, where the input is a pair of results for a query, and the class is the relevance ordering relationship between them

## 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

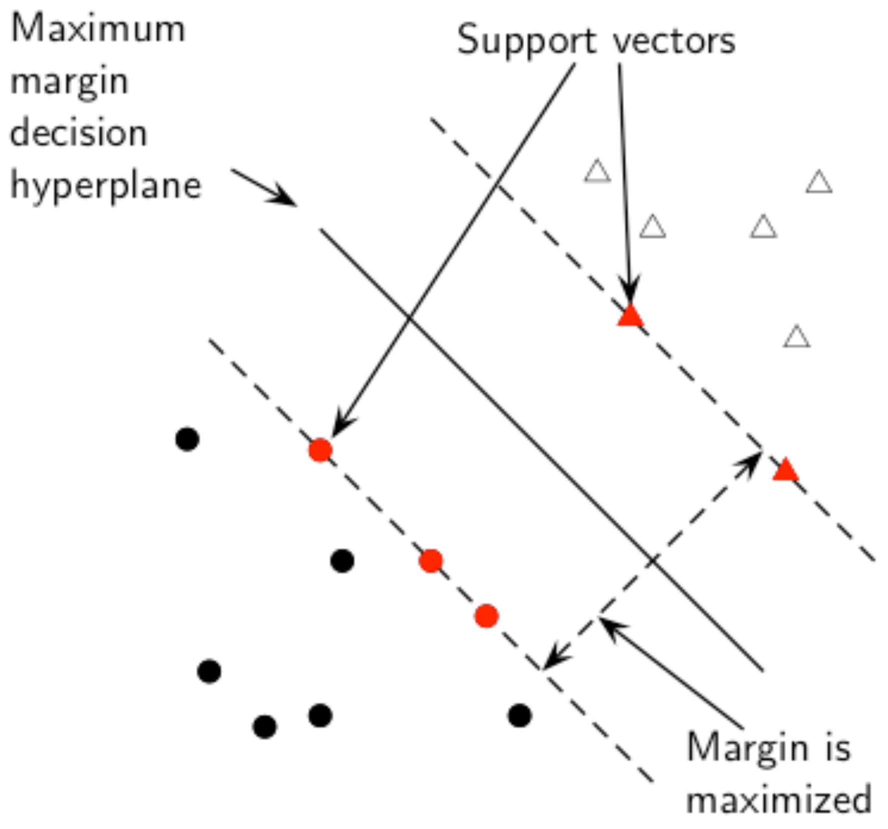
Especially germane in web search, where the ranking at the very top of the results list is exceedingly important

**Structural SVM for IR ranking**

Such work has been pursued using the structural SVM framework, where the class being predicted is a ranking of results for a query

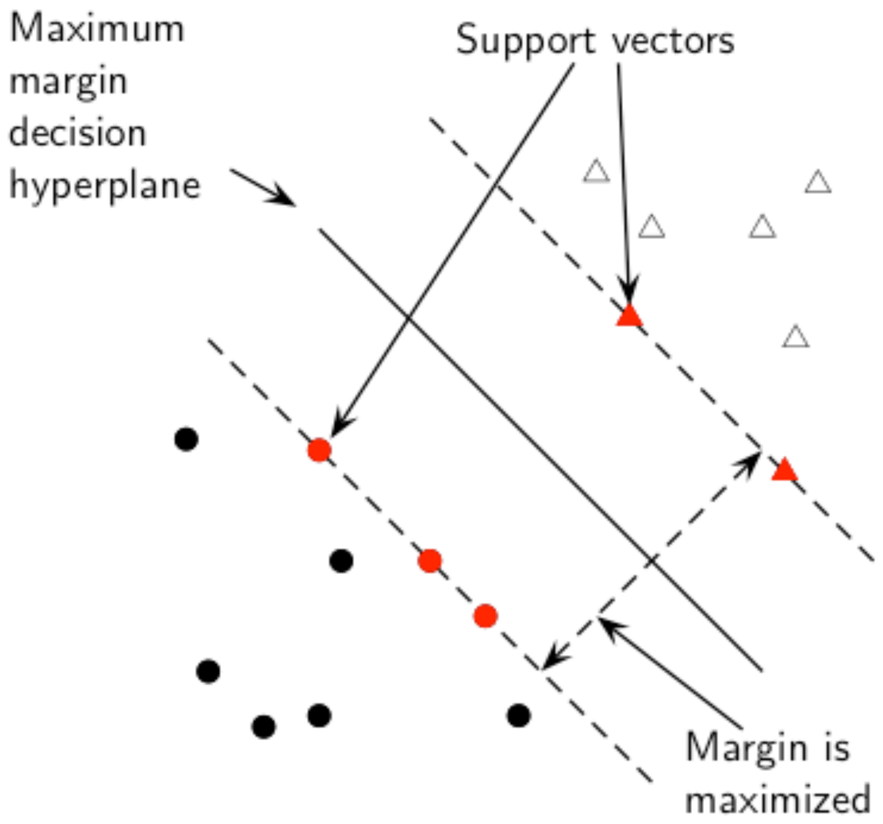
## Support Vector Machines

- 2-class training data
- decision boundary
  - → **linear separator**
- criterion: being maximally far away from any data point
  - → determines classifier **margin**
- linear separator position defined by **support vectors**



## Why maximise the margin?

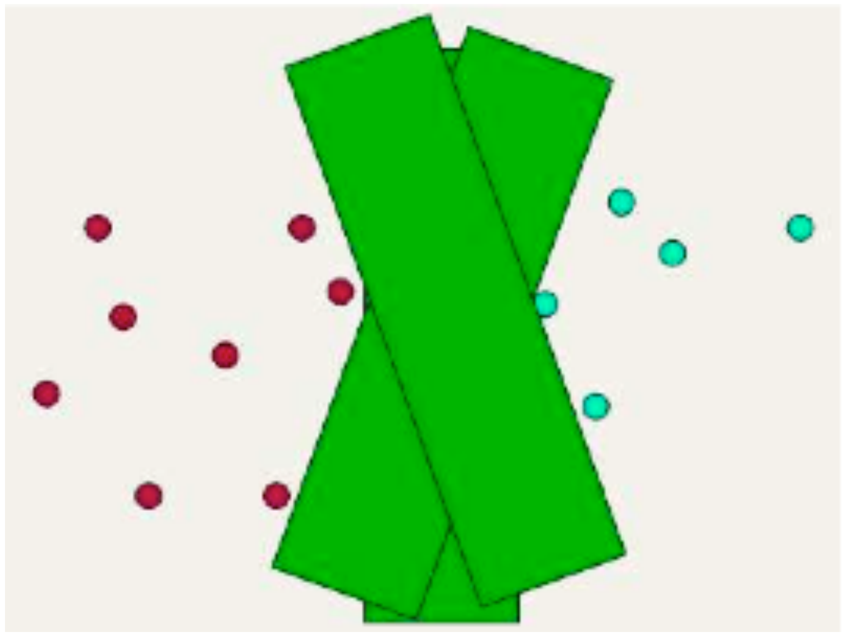
Points near decision surface → uncertain classification decisions (50% either way).
A classifier with a large margin makes no low certainty classification decisions.
Gives classification safety margin w.r.t slight errors in measurement or doc. variation



## Why maximise the margin?

SVM classifier: large margin around decision boundary

- compare to decision hyperplane: place fat separator between classes
  - fewer choices of where it can be put
- decreased memory capacity
- increased ability to correctly generalize to test data



# 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:

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

## The construction of a ranking SVM

- By hypothesis, one of $d_i$ and $d_\mathrm{j}$ has been judged more relevant
- If $d_i$ is judged more relevant than *dj* , denoted $d_\mathrm{i} < \mathrm{d}_j$ ($\mathrm{d}_i$ 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

$$w^\mathrm{T} \Phi(d_i , d_j , q) > 0 \text{ iff } d_i < d_j$$

# 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

## The ranking SVM fails to model the IR problem well…

- Correctly ordering the most relevant documents is crucial to the success of an IR system, while misordering less relevant results matters little
  - The ranking SVM considers all ordering violations as the same
- Some queries have many (somewhat) relevant documents, and other queries few. If we treat all pairs of results for a query equally, queries with many results will dominate the learning
  - But actually queries with few relevant results are at least as important to do well on

## Limitations

- Everything that we have looked at (and most work in this area) produces linear models of features by weighting different base features
- This contrasts with most of the clever ideas of traditional IR, which are nonlinear scalings and combinations of basic measurements
  - log term frequency, idf, pivoted length normalization
- At present, ML is good at weighting features, but not at coming up with nonlinear scalings
- Designing the basic features that give good signals for ranking remains the domain of human creativity

# Summary

- The idea of learning ranking functions has been around for about 20 years
- But only recently have ML knowledge, availability of training datasets, a rich space of features, and massive computation come together to make this a hot research area
- It’s too early to give a definitive statement on what methods are best in this area … it’s still advancing rapidly
- But machine learned ranking over many features now easily beats traditional hand-designed ranking functions in comparative evaluations  [in part by using the hand-designed functions as features!]
- And there is every reason to think that the importance of machine learning in IR will only increase in the future.