# CSCE 670 - Information Storage and Retrieval

## Lecture 9: Learning to Rank

Yu Zhang

yuzhang@tamu.edu

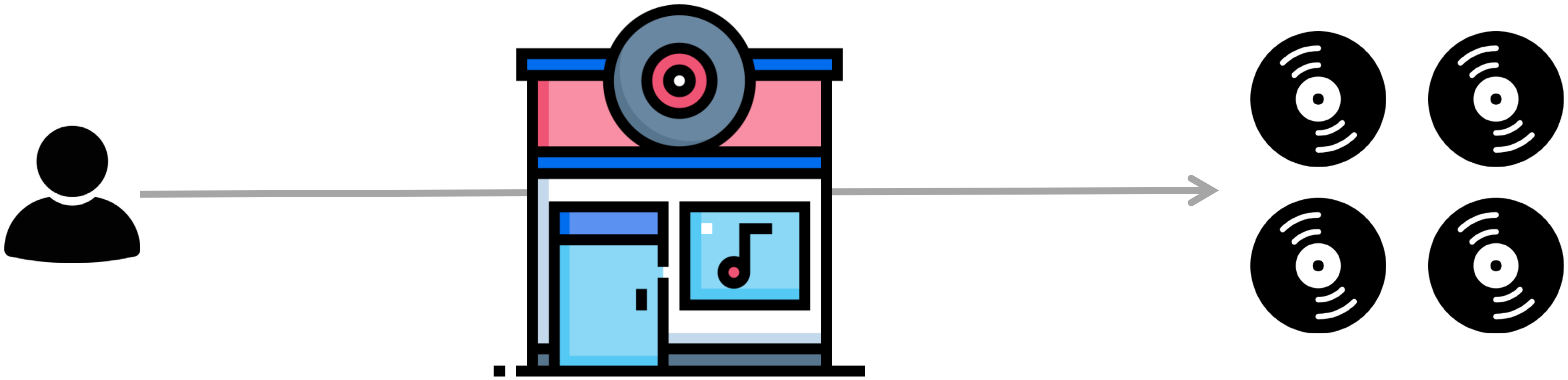
September 23, 2025

Course Website: https://yuzhang-teaching.github.io/CSCE670-F25.html

Adapted from the slides by Prof. James Caverlee

# Ranking should consider multiple factors

- YouTube videos: view, subscribers, video length, user profile factors (e.g., age, location), title relevance, video quality, recency, …
- LinkedIn job postings: posting popularity, company popularity, number of openings, skill match with the user, nearness, recency, salary, …
- Our record store: record popularity, singer popularity, language, keyword match, …



# Hand-tuning a Ranking Function

- Score$(q, d)$ =

  $a_1 \times$ TF-IDF$(q, d)$ +

  $a_2 \times$ BM25$(q, d)$ +

  $a_3 \times$ # views in the last day$(d)$ +

  $a_4 \times$ # views in the last week$(d)$ +

  $a_5 \times$ recency$(d)$ +

  $a_6 \times$ PageRank$(d)$ +

  $\dots$

- After checking some examples, you set $a_1$ as 0.5, $a_2$ as 0.8, $\dots$
- Problems with this strategy?

# Instead, let's learn a good ranker!

- Rough Idea (not 100% accurately framed): Learn the value of $a_1, a_2, \dots$ from data (e.g., relevant query-document pairs according to user clickthrough history)
  - A very natural idea (especially these days)

- But it took a while for ML and IR to be good friends
  - Wong et al., *Linear structure in information retrieval.* SIGIR 1988.
  - Fuhr, *Probabilistic methods in information retrieval.* Computer Journal 1992.
  - Gey, *Inferring probability of relevance using the method of logistic regression.* SIGIR 1994.
  - Herbrich et al., *Large margin rank boundaries for ordinal regression.* Advances in Large Margin Classifiers 2000.

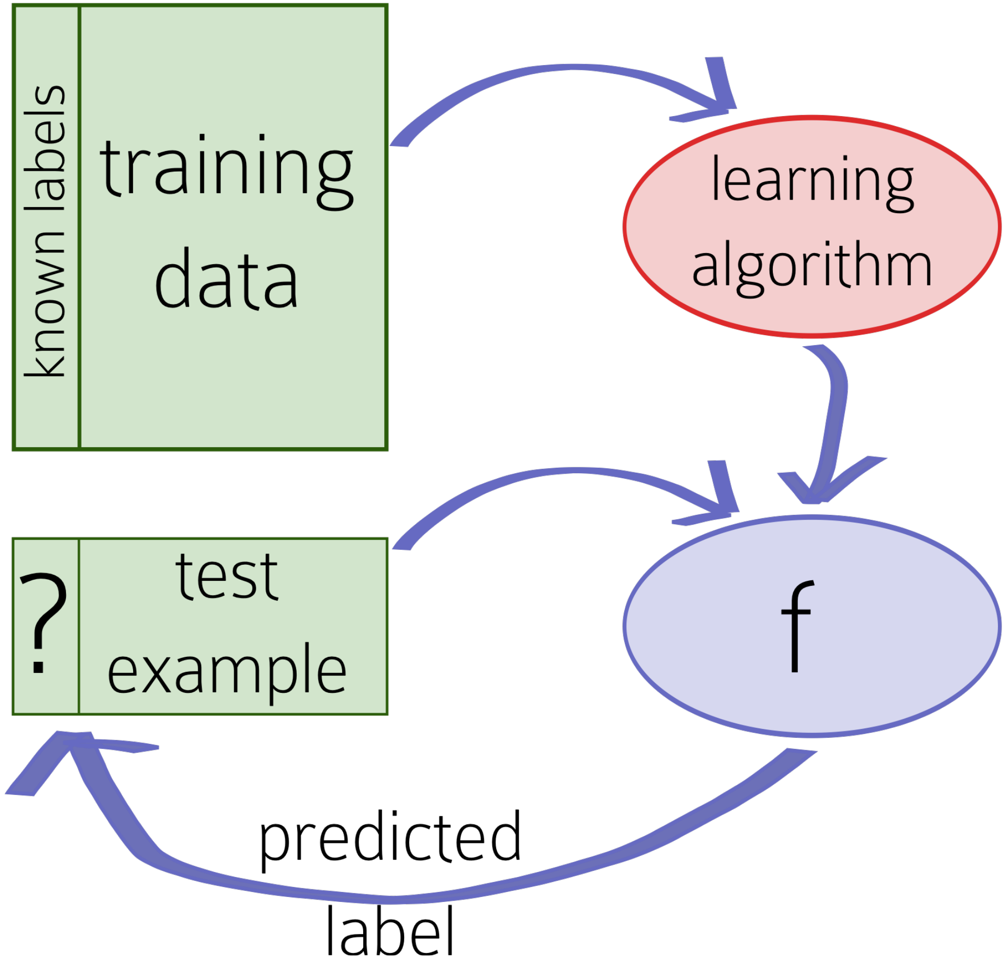
# Background: Text Classification

- Given:
  - A document space $\mathcal{X}$
  - A fixed set of classes $\mathcal{C} = \{c_1, c_2, \dots\}$
  - A training set of labeled documents:
    - E.g., $d_1 \to c_1, \quad d_2 \to c_1, \quad d_3 \to c_2, \quad \dots$
- Use a learning algorithm to learn a classifier $\mathrm{f}$ that maps documents to classes $\mathrm{f}: \mathcal{X} \to \mathcal{C}$
- Examples
  - Paper Topic Classification: $\mathcal{X}$ = academic papers, $\mathcal{C}$ = {math, physics, chemistry, …}
  - Review Sentiment Analysis: $\mathcal{X}$ = food reviews, $\mathcal{C}$ = {1-star, 2-star, 3-star, 4-star, 5-star}
  - Songwriter Prediction: $\mathcal{X}$ = lyrics, $\mathcal{C}$ = songwriters

# Background: Text Classification

- Training: Use a learning algorithm to learn a classifier $\mathrm{f}$ that maps documents to classes $\mathrm{f}: \mathcal{X} \rightarrow \mathcal{C}$

- Testing/Inference: Given an unseen document $d_{\text{test}}$
  - Apply our classifier function $\mathrm{f}(d_{\text{test}})$ to determine the most appropriate class in $\mathcal{C}$



# A Couple of Simple Text Classifiers: Rocchio

- Training: "Learn" class centers for each class by finding the centroid of all the training examples from each class

- Testing/Inference: Assign a new example to the class of the nearest class center

- Example:
  - 2-class classification (*chemistry* paper vs. *history* paper)
  - Training samples
    - *chemistry*: $d_1 = (1.0, 0.9), d_2 = (0.9, 1.0)$
    - *history*: $d_3 = (0.2, 0.3), d_4 = (0.3, 0.2)$

# A Couple of Simple Text Classifiers: Rocchio

- Example:
  - 2-class classification (*chemistry* paper vs. *history* paper)
  - Training samples
    - *chemistry*: $d_1 = (1.0, 0.9), d_2 = (0.9, 1.0)$
    - *history*: $d_3 = (0.2, 0.3), d_4 = (0.3, 0.2)$

- Step 1: Compute class centroids
  - *chemistry*: $c_{\text{chemistry}} = \frac{d_1+d_2}{2} = (0.95, 0.95)$
  - *history*: $c_{\text{history}} = \frac{d_3+d_4}{2} = (0.25, 0.25)$

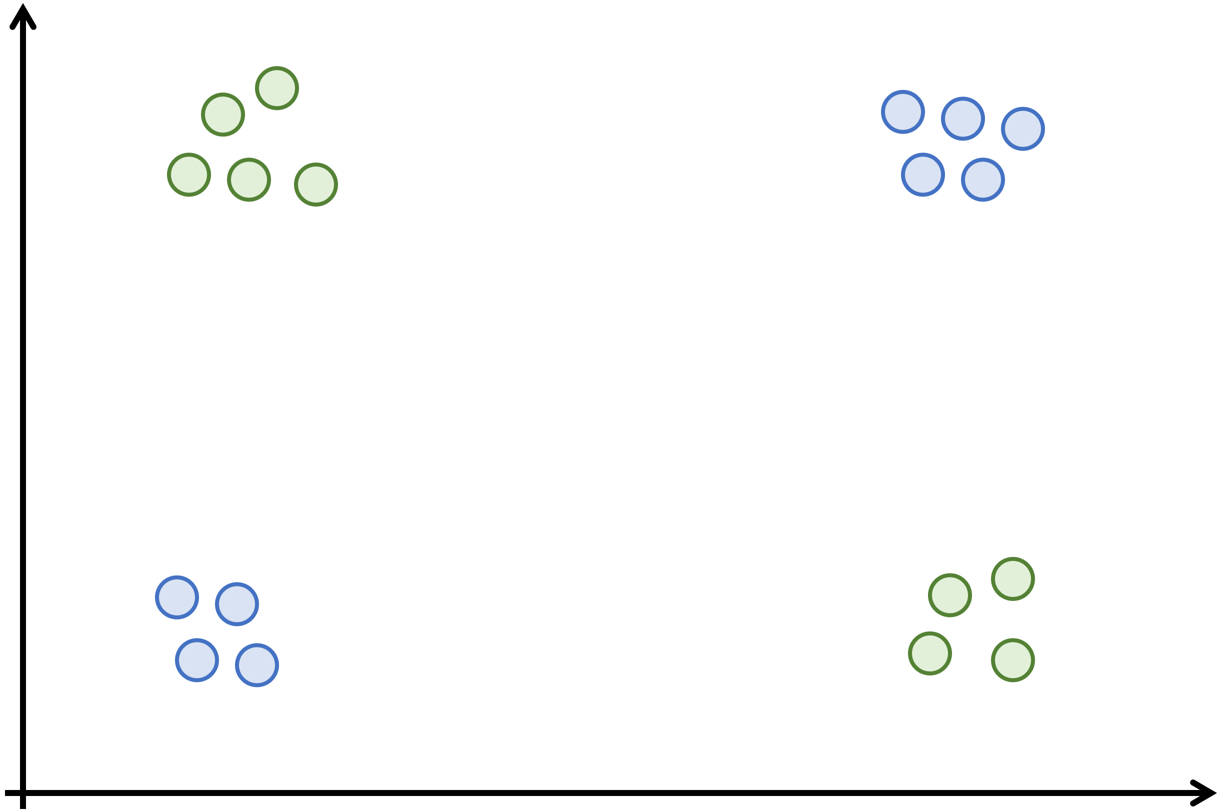
# A Couple of Simple Text Classifiers: Rocchio

- Step 1: Compute class centroids
  - *chemistry*: $c_{\text{chemistry}} = \frac{d_1+d_2}{2} = (0.95, 0.95)$
  - *history*: $c_{\text{history}} = \frac{d_3+d_4}{2} = (0.25, 0.25)$

- Step 2: Classify a new document
  - New document: $d_5 = (0.8, 0.85)$
  - Compute Euclidean distance:
    - To *chemistry*: $\text{dist}\left(d_5, c_{\text{chemistry}}\right) = \sqrt{(0.8-0.95)^2+(0.85-0.95)^2} \approx 0.1803$
    - To *history*: $\text{dist}\left(d_5, c_{\text{history}}\right) = \sqrt{(0.8-0.25)^2+(0.85-0.25)^2} \approx 0.8124$
  - $d_5$ is closer to *chemistry*, so we classify it as a *chemistry* paper.

# A Couple of Simple Text Classifiers: Rocchio

- Can you raise an example where Rocchio does NOT work?



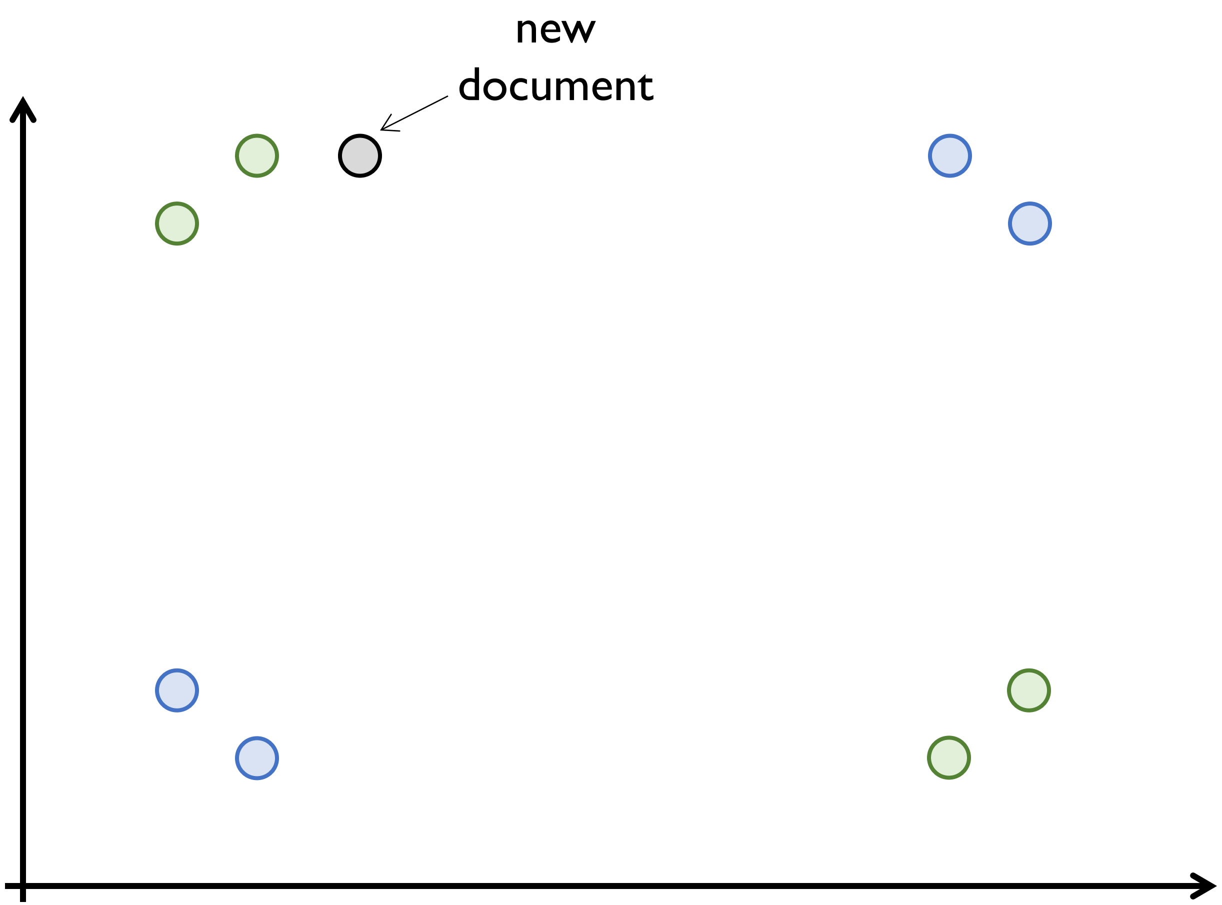
# A Couple of Simple Text Classifiers: $k$ Nearest Neighbors ($k$NN)

- Training: None
- Testing/Inference: Assign a new example to the majority class of the $k$-nearest training examples
- Example:
  - 2-class classification (*chemistry* paper vs. *history* paper)
  - Training samples
    - *chemistry*: $d_1 = (1.0, 0.9), d_2 = (0.9, 1.0), d_3 = (0.2, 0.3), d_4 = (0.3, 0.2)$
    - *history*: $d_5 = (1.0, 0.3), d_6 = (0.9, 0.2), d_7 = (0.3, 1.0), d_8 = (0.2, 0.9)$
  - New document: $d_9 = (0.4, 1.0)$
  - $k = 3$

# A Couple of Simple Text Classifiers: $k$ Nearest Neighbors ($k$NN)

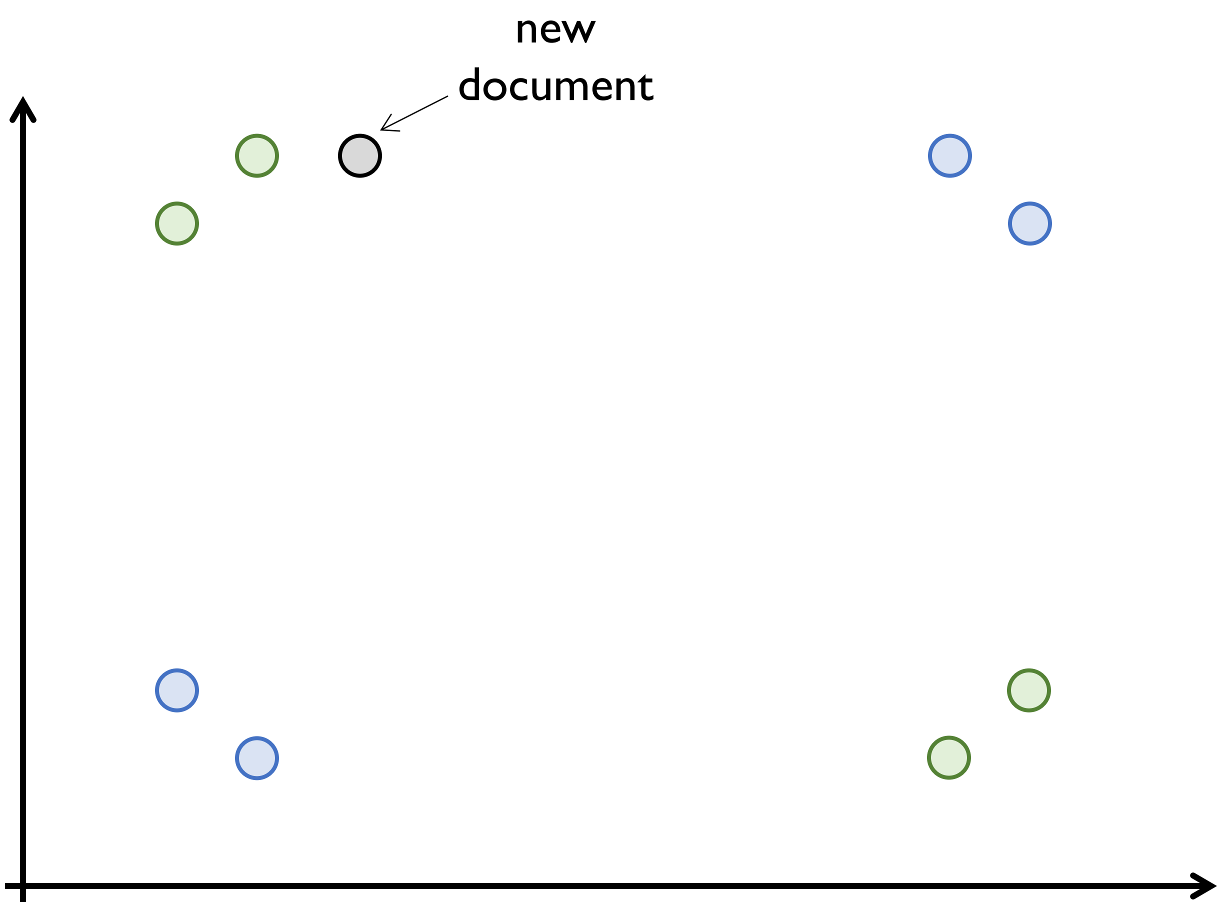
- Nearest neighbor: *history*
- 2nd nearest neighbor: *history*
- 3rd nearest neighbor: *chemistry*

- Majority voting: the new document has more *history* neighbors, so we classify it as a *history* paper.



# A Couple of Simple Text Classifiers: $k$ Nearest Neighbors ($k$NN)

- Can you raise an example where $k$NN does NOT work?
- How to determine $k$? What if $k = 5$?
- Nearest neighbor: *history*
- 2nd nearest neighbor: *history*
- 3rd nearest neighbor: *chemistry*
- 4th nearest neighbor: *chemistry*
- 5th nearest neighbor: *chemistry*



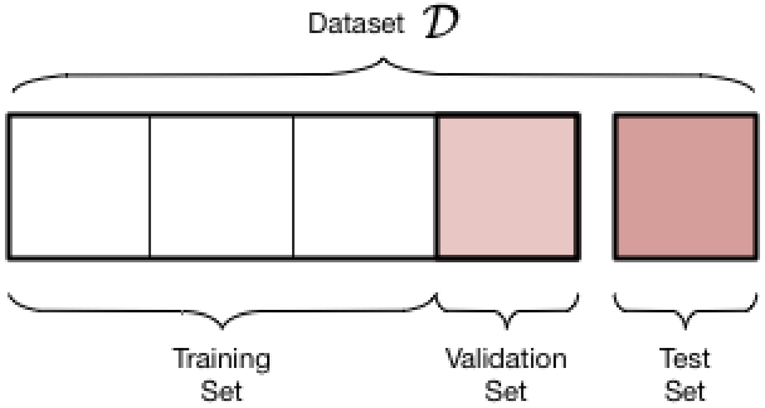
# In practice: Which features?

- Very important to select good features to represent our documents
- Features we know about:
  - TF-IDF score of each word (one feature per word)
  - PageRank/Hub/Authority score of the document
  - Popularity, # of clicks, freshness, …

# In practice: Which classifier?

- Many, many ways to learn a good classifier
  - Rocchio
  - $k$NN
  - Support Vector Machine
  - Naive Bayes
  - Decision Tree
  - Random Forest
  - Gradient-Boosted Decision Tree
  - …

# In practice: How to evaluate?



- Need a way to evaluate how well we do
- Classification accuracy is one way
  - For a held-out test set (for which we know the correct labels), calculate how many labels our classifier correctly predicts
- Many others (some we may talk about later)

- Keep part of the labeled data separate as a validation set
- Train a model over the training data and "test" over the validation set
- Train another model over the training data and "test" over the validation set (and so on and so on)
- Choose model that minimizes error on the validation set

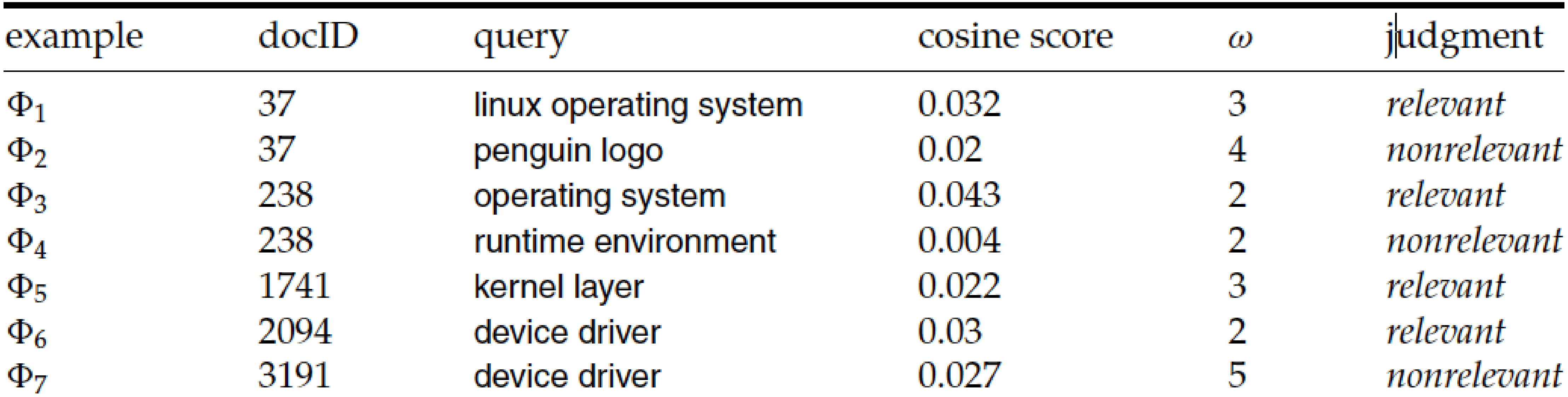
# Back to Ranking

- Assume we have a test collection:
  - A benchmark document collection
  - A benchmark suite of queries
  - A binary assessment of either Relevant or Non-relevant for each query and each document

- Sounds like classification!
  - Classification Training: Given a training set of (*query*, *document* → *relevance*) triples, learn a model f that outputs Relevant or Non-relevant
  - Classification Testing: Given unseen (*query*, *document*), apply f(*query*, *document*) and output Relevant or Non-relevant
  - NOTE: Now our input is not just a *document* but both a *document* and a *query*!

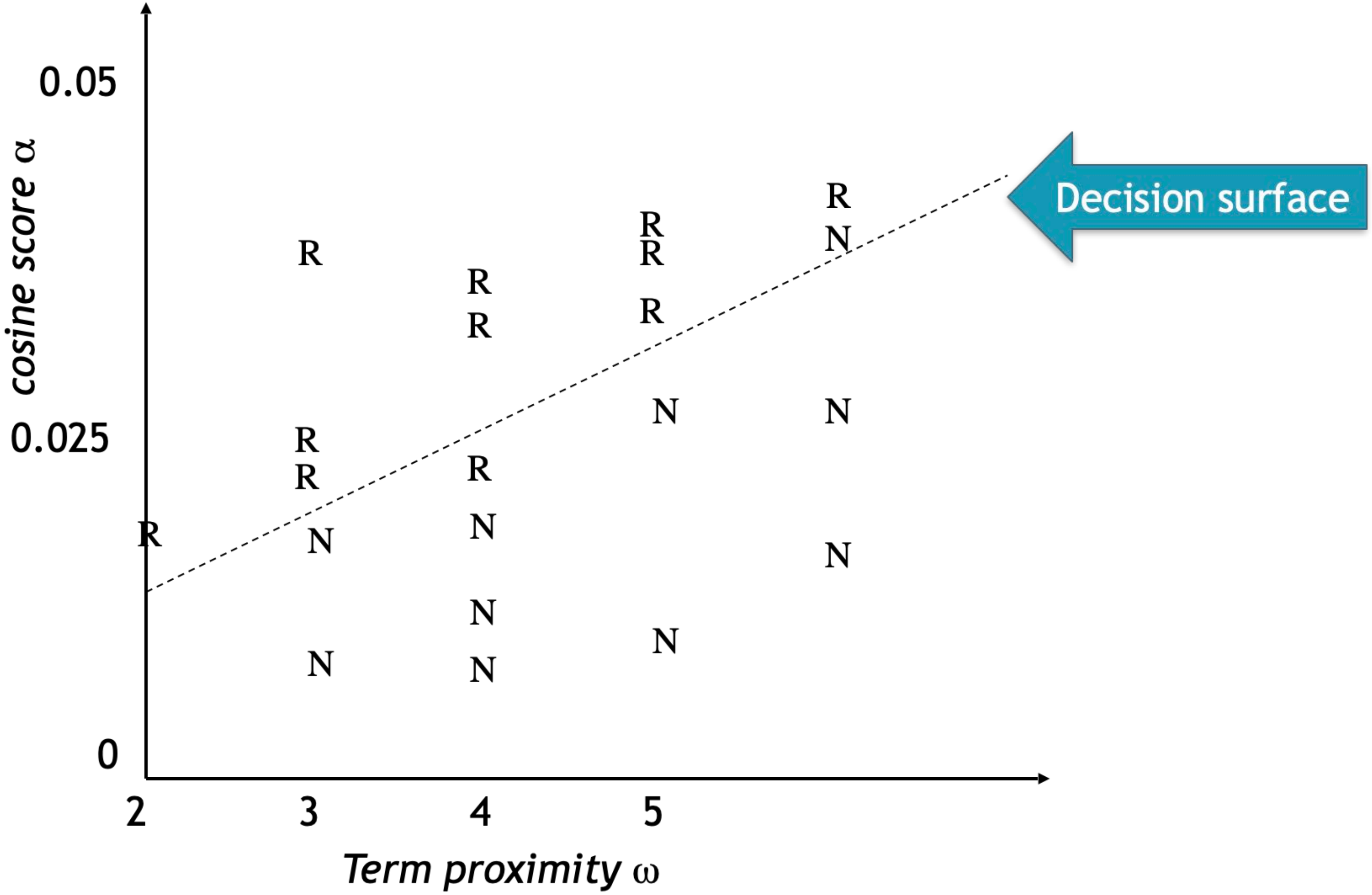
# Relevance Classification: Example

term proximity

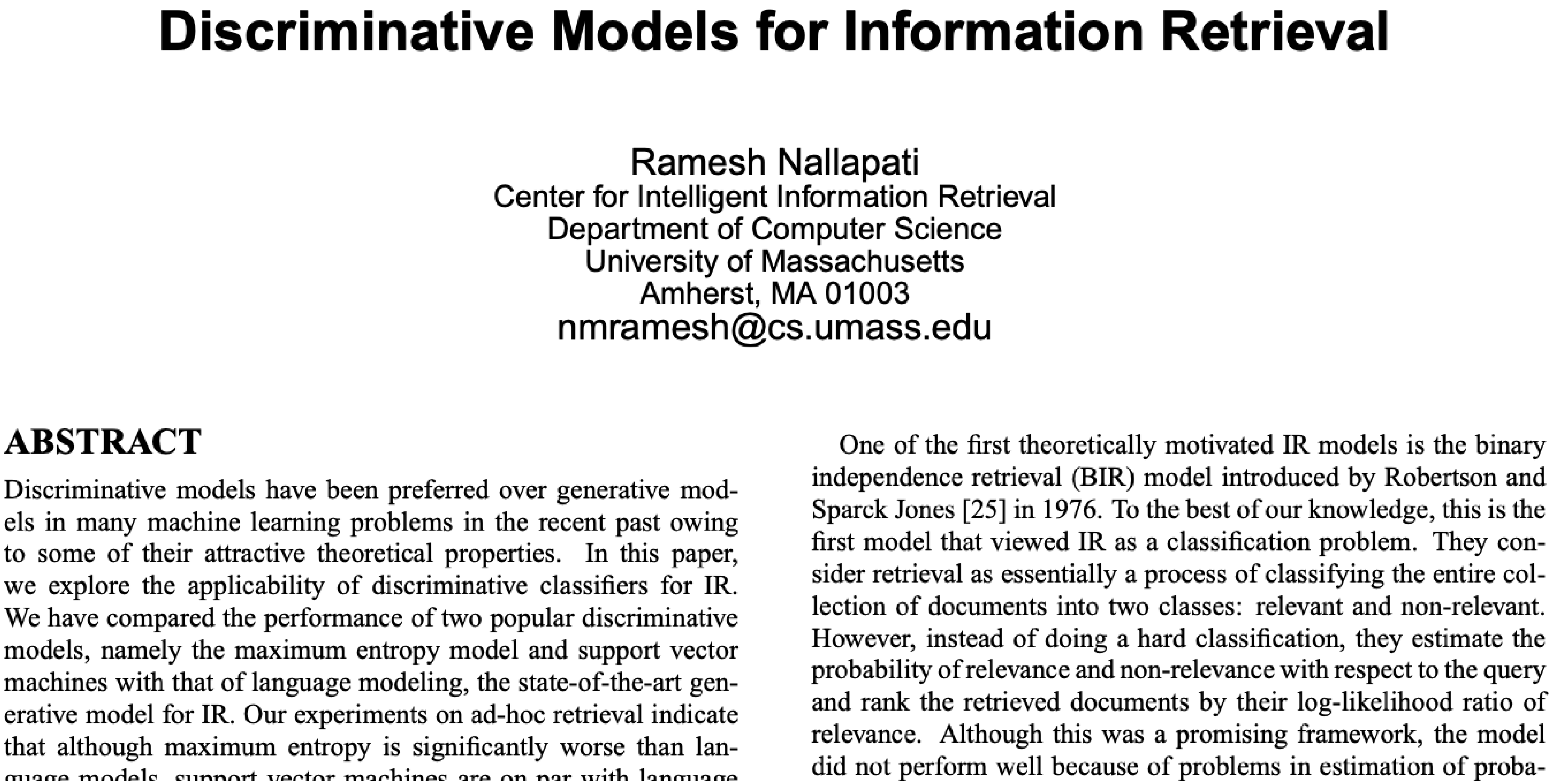


| example | docID | query | cosine score | $\omega$ | judgment |
|---|---|---|---|---|---|
| $\Phi_1$ | 37 | linux operating system | 0.032 | 3 | *relevant* |
| $\Phi_2$ | 37 | penguin logo | 0.02 | 4 | *nonrelevant* |
| $\Phi_3$ | 238 | operating system | 0.043 | 2 | *relevant* |
| $\Phi_4$ | 238 | 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* |

# Relevance Classification: Example



# Nallapati [SIGIR 2004]



## Discriminative Models for Information Retrieval

Ramesh Nallapati
Center for Intelligent Information Retrieval
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
nmramesh@cs.umass.edu

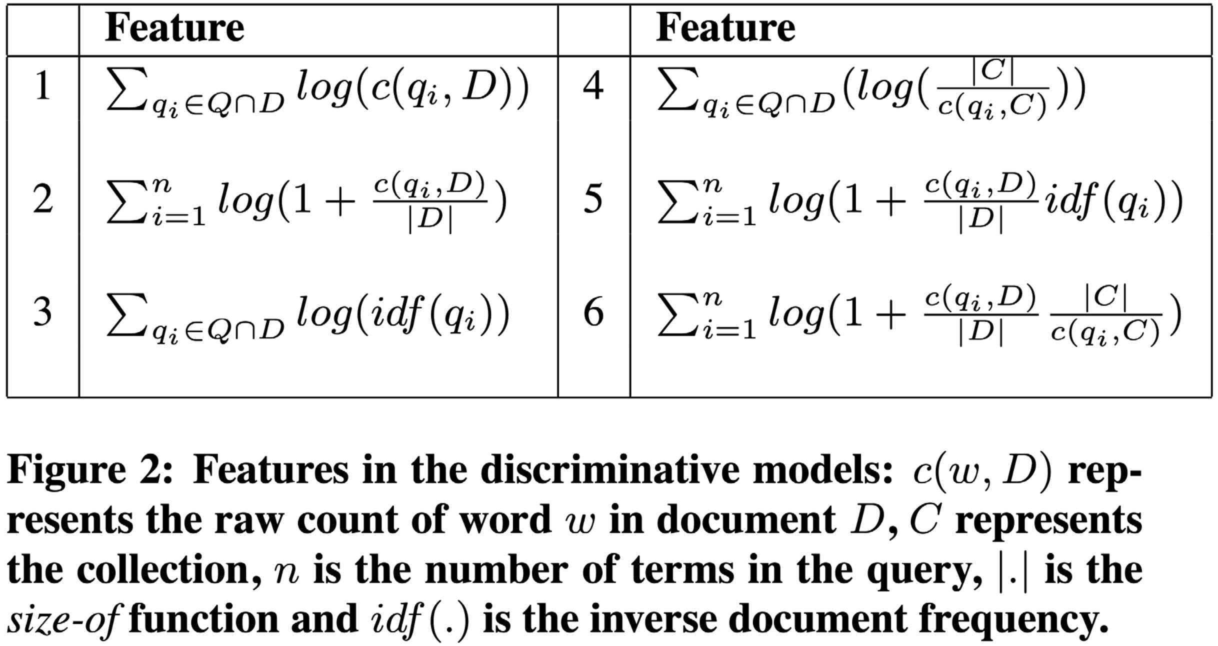
### ABSTRACT

Discriminative models have been preferred over generative models in many machine learning problems in the recent past owing to some of their attractive theoretical properties. In this paper, we explore the applicability of discriminative classifiers for IR. We have compared the performance of two popular discriminative models, namely the maximum entropy model and support vector machines with that of language modeling, the state-of-the-art generative model for IR. Our experiments on ad-hoc retrieval indicate that although maximum entropy is significantly worse than language models, support vector machines are on par with language

One of the first theoretically motivated IR models is the binary independence retrieval (BIR) model introduced by Robertson and Sparck Jones [25] in 1976. To the best of our knowledge, this is the first model that viewed IR as a classification problem. They consider retrieval as essentially a process of classifying the entire collection of documents into two classes: relevant and non-relevant. However, instead of doing a hard classification, they estimate the probability of relevance and non-relevance with respect to the query and rank the retrieved documents by their log-likelihood ratio of relevance. Although this was a promising framework, the model did not perform well because of problems in estimation of proba-

# Nallapati [SIGIR 2004]

- Experiments:
  - Comparisons with Lemur (LM), a state-of-the-art open-source IR engine
  - Which classifier? SVM with linear kernel
  - What features? 6 features, all variants of TF, IDF, and TF-IDF scores

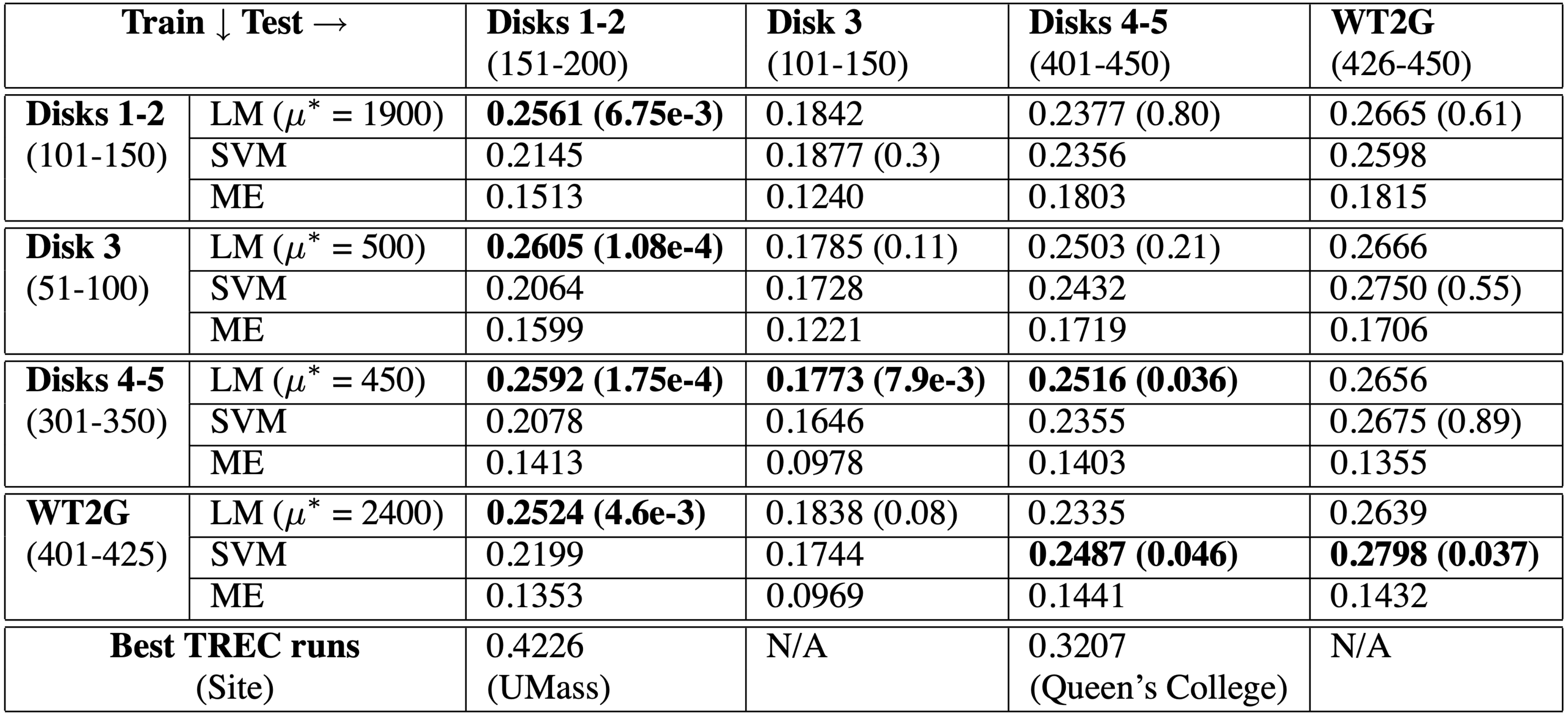


|   | **Feature** |   | **Feature** |
|---|---|---|---|
| 1 | $\sum_{q_i \in Q \cap D} log(c(q_i, D))$ | 4 | $\sum_{q_i \in Q \cap D} (log(\frac{\|C\|}{c(q_i, C)}))$ |
| 2 | $\sum_{i=1}^{n} log(1 + \frac{c(q_i, D)}{\|D\|})$ | 5 | $\sum_{i=1}^{n} log(1 + \frac{c(q_i, D)}{\|D\|} idf(q_i))$ |
| 3 | $\sum_{q_i \in Q \cap D} log(idf(q_i))$ | 6 | $\sum_{i=1}^{n} log(1 + \frac{c(q_i, D)}{\|D\|} \frac{\|C\|}{c(q_i, C)})$ |

**Figure 2: Features in the discriminative models: $c(w, D)$ represents the raw count of word $w$ in document $D$, $C$ represents the collection, $n$ is the number of terms in the query, $|.|$ is the** *size-of* **function and $idf(.)$ is the inverse document frequency.**

# Experiments on 4 TREC Datasets

- Metric: Mean Average Precision (MAP)



| **Train ↓ Test →** | | **Disks 1-2**<br>(151-200) | **Disk 3**<br>(101-150) | **Disks 4-5**<br>(401-450) | **WT2G**<br>(426-450) |
|---|---|---|---|---|---|
| **Disks 1-2**<br>(101-150) | LM ($\mu^* = 1900$) | **0.2561 (6.75e-3)** | 0.1842 | 0.2377 (0.80) | 0.2665 (0.61) |
| | SVM | 0.2145 | 0.1877 (0.3) | 0.2356 | 0.2598 |
| | ME | 0.1513 | 0.1240 | 0.1803 | 0.1815 |
| **Disk 3**<br>(51-100) | LM ($\mu^* = 500$) | **0.2605 (1.08e-4)** | 0.1785 (0.11) | 0.2503 (0.21) | 0.2666 |
| | SVM | 0.2064 | 0.1728 | 0.2432 | 0.2750 (0.55) |
| | ME | 0.1599 | 0.1221 | 0.1719 | 0.1706 |
| **Disks 4-5**<br>(301-350) | LM ($\mu^* = 450$) | **0.2592 (1.75e-4)** | **0.1773 (7.9e-3)** | **0.2516 (0.036)** | 0.2656 |
| | SVM | 0.2078 | 0.1646 | 0.2355 | 0.2675 (0.89) |
| | ME | 0.1413 | 0.0978 | 0.1403 | 0.1355 |
| **WT2G**<br>(401-425) | LM ($\mu^* = 2400$) | **0.2524 (4.6e-3)** | 0.1838 (0.08) | 0.2335 | 0.2639 |
| | SVM | 0.2199 | 0.1744 | **0.2487 (0.046)** | **0.2798 (0.037)** |
| | ME | 0.1353 | 0.0969 | 0.1441 | 0.1432 |
| **Best TREC runs**<br>(Site) | | 0.4226<br>(UMass) | N/A | 0.3207<br>(Queen's College) | N/A |

# Experiments on 4 TREC Datasets

- At best the results are about equal to Lemur
  - Actually a little bit below

- Paper's advertisement: Easy to add more features
- This is illustrated on a homepage finding task on WT10G:

| | Success@10 |
|---|---|
| Lemur | 0.52 |
| SVM with text features only | 0.58 |
| SVM with URL-depth and in-link features | 0.78 |

# Questions?

# But Boolean ≠ Ranking!

- Assigning (*query*, *document*) to
  - Relevant
  - or
  - Not Relevant
- Is not really what we want when we think about ranking

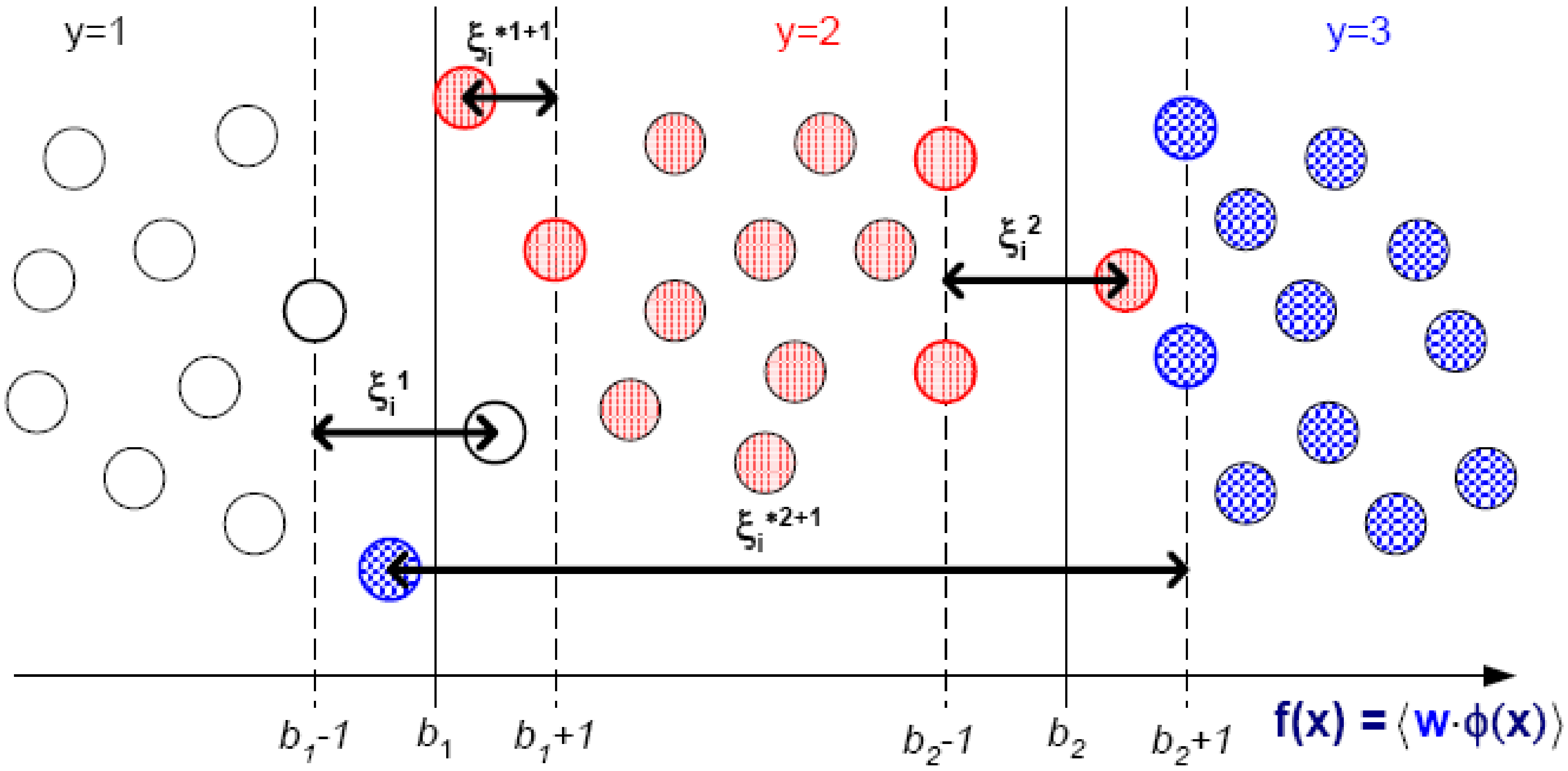
# Pointwise Learning

- Assume we have training data like (*query*, *document*, *score*)
- Here the *score* could be a relevance score like
  - 4 Perfect match
  - 3 Very relevant
  - 2 Relevant
  - 1 Somewhat relevant
  - 0 Not relevant at all

- Our goal is to output a score
  - This is regression (if we can output any value)
  - Or ordinal regression (If we can only output 0, 1, 2, 3, or 4)

# Pointwise Learning

- What could be a difference between classification and ordinal regression?



# Pointwise Learning

- Regression Training: Given a training set of (*query*, *document* → *score*) triples, learn a model f
- Regression Testing: Given unseen (*query*, *document*), apply f(*query*, *document*) and output the *score*

- Challenges?
  - Expensive to collect labels
  - Focuses on scores, not relative ordering (or relationship to other documents)
  - Bias towards frequent queries
  - …

# Pairwise Learning

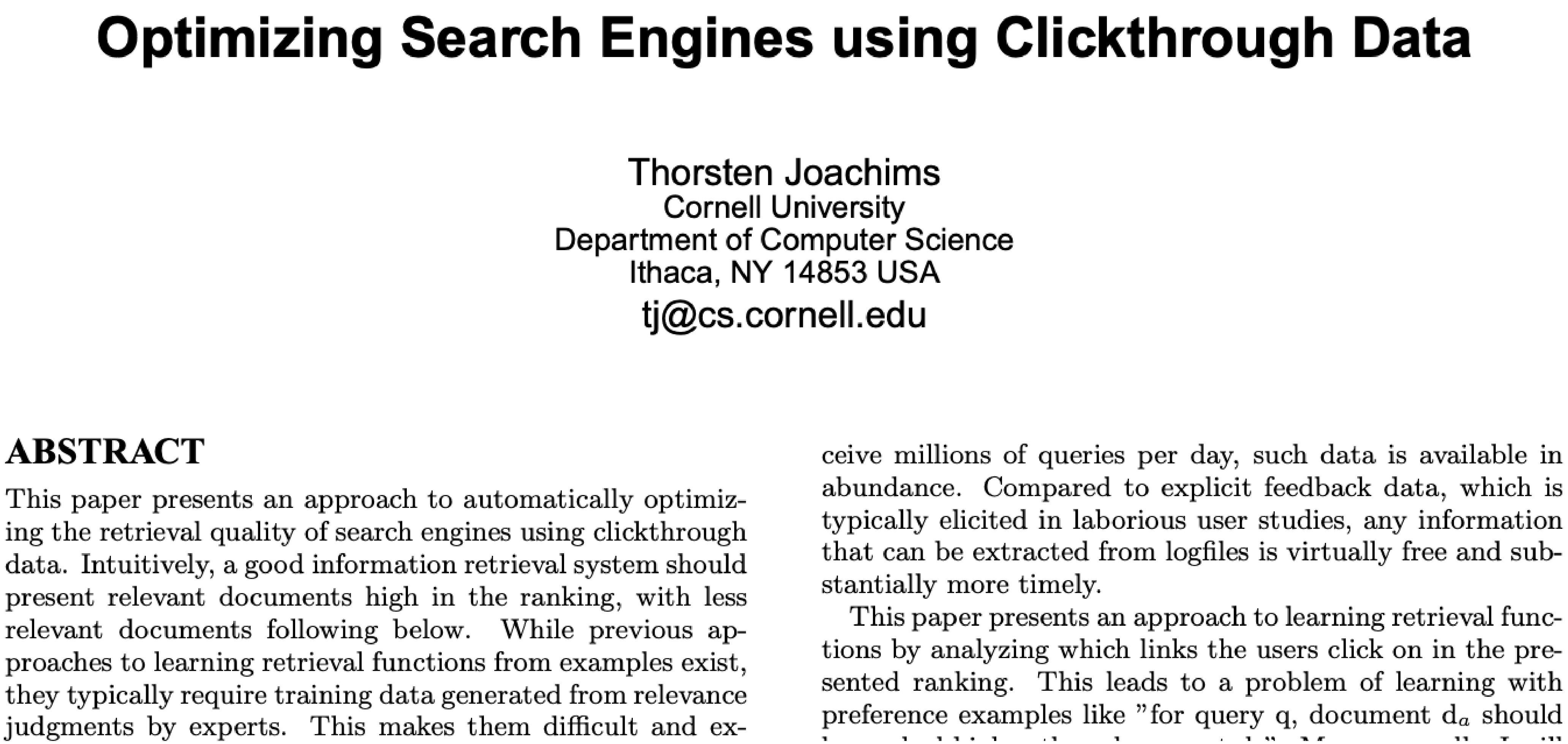
- Aim is to classify instance pairs as correctly ranked or incorrectly ranked
    - Given the query $q$ and two candidates $(c_i, c_k)$, predict if $c_i$ should be ranked higher than $c_k$ (denoted as $c_i > c_k$)

- This turns an ordinal regression problem back into a binary classification problem in an expanded space
    - We only need lots of $(c_i, c_k)$, where we already know $c_i > c_k$, for training

- Formally, we want a ranking function f such that
    - $c_i > c_k \iff \mathrm{f}(\psi_i) > \mathrm{f}(\psi_k)$
    - $\psi_i$ is the feature vector of $c_i$ given the query $q$ (e.g., one entry can be TF-IDF($q$, $c_i$))

- To simplify our discussion, let's suppose that f is a linear function: $\mathrm{f}(\psi_i) = w^T \psi_i$

# Joachims [KDD 2002]



## Optimizing Search Engines using Clickthrough Data

Thorsten Joachims
Cornell University
Department of Computer Science
Ithaca, NY 14853 USA
tj@cs.cornell.edu

### ABSTRACT

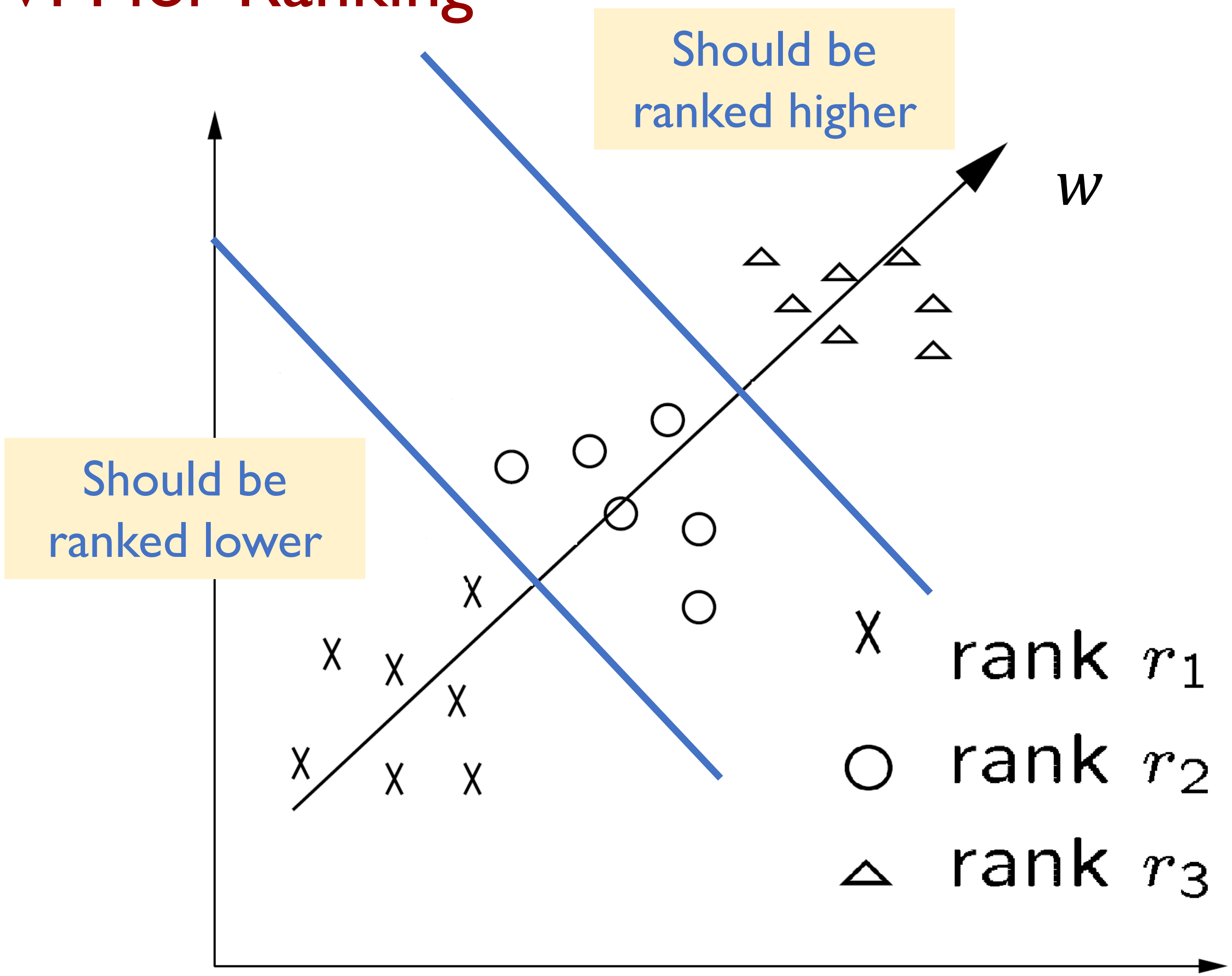
This paper presents an approach to automatically optimizing the retrieval quality of search engines using clickthrough data. Intuitively, a good information retrieval system should present relevant documents high in the ranking, with less relevant documents following below. While previous approaches to learning retrieval functions from examples exist, they typically require training data generated from relevance judgments by experts. This makes them difficult and expensive to apply. The goal of this paper is to develop a method that utilizes clickthrough data for training, namely the query-log of the search engine in connection with the log of links the users clicked on in the presented ranking. Such clickthrough data is available in abundance and can be recorded at very low cost. Taking a Support Vector Machine (SVM) approach, this paper presents a method for learning retrieval functions. From a theoretical perspective, this method is shown to be well-founded in a risk minimization framework. Furthermore, it is shown to be feasible even for large sets of queries and features. The theoretical results are verified in a controlled experiment. It shows that the method can effectively adapt the retrieval function of a meta-search engine to a particular group of users, outperforming Google in terms of retrieval quality after only a couple of hundred training examples.

ceive millions of queries per day, such data is available in abundance. Compared to explicit feedback data, which is typically elicited in laborious user studies, any information that can be extracted from logfiles is virtually free and substantially more timely.

This paper presents an approach to learning retrieval functions by analyzing which links the users click on in the presented ranking. This leads to a problem of learning with preference examples like "for query q, document $d_a$ should

# Training a Linear SVM for Ranking

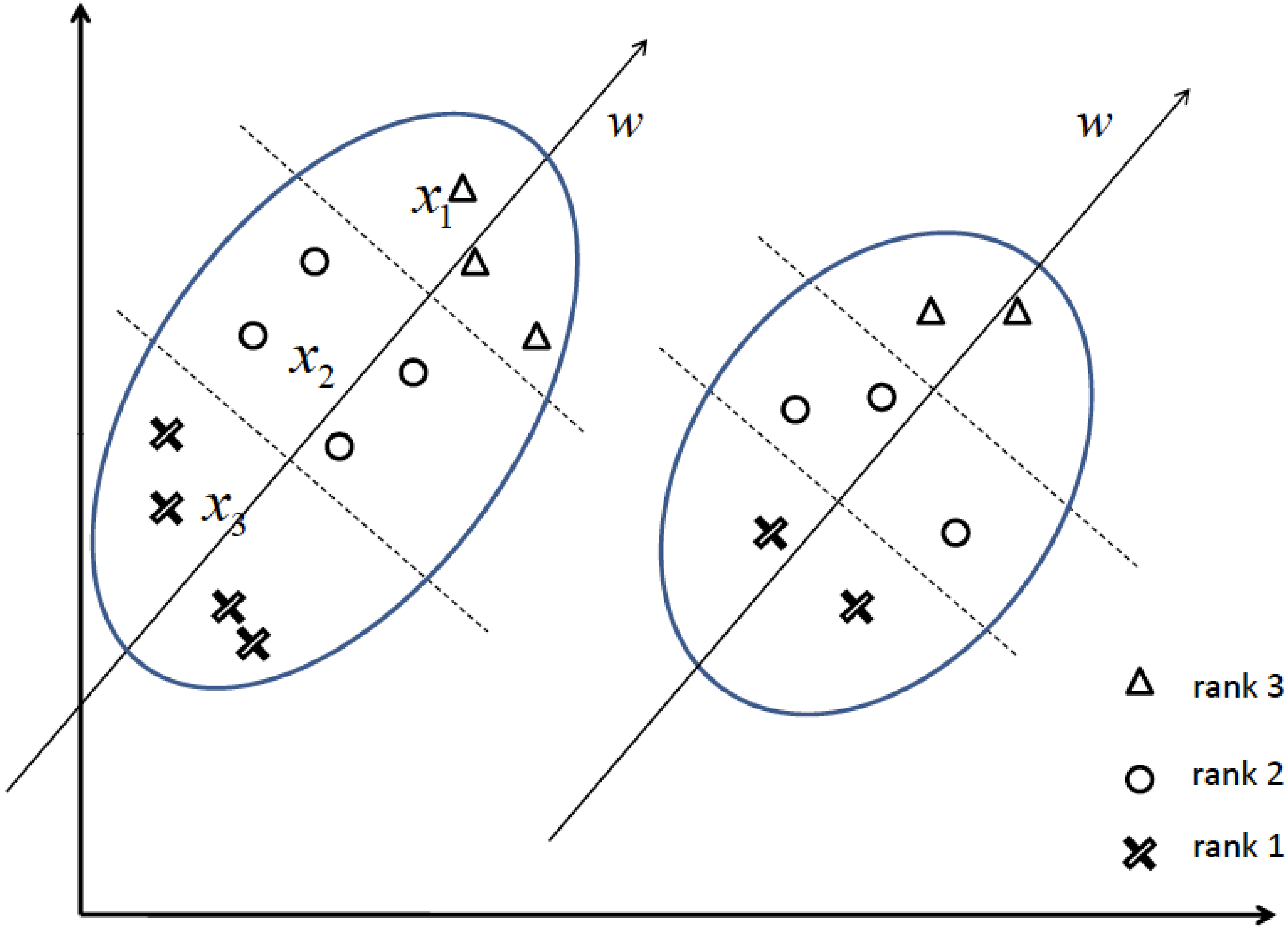
Ranking function:
$f(\psi_i) = w^T \psi_i$



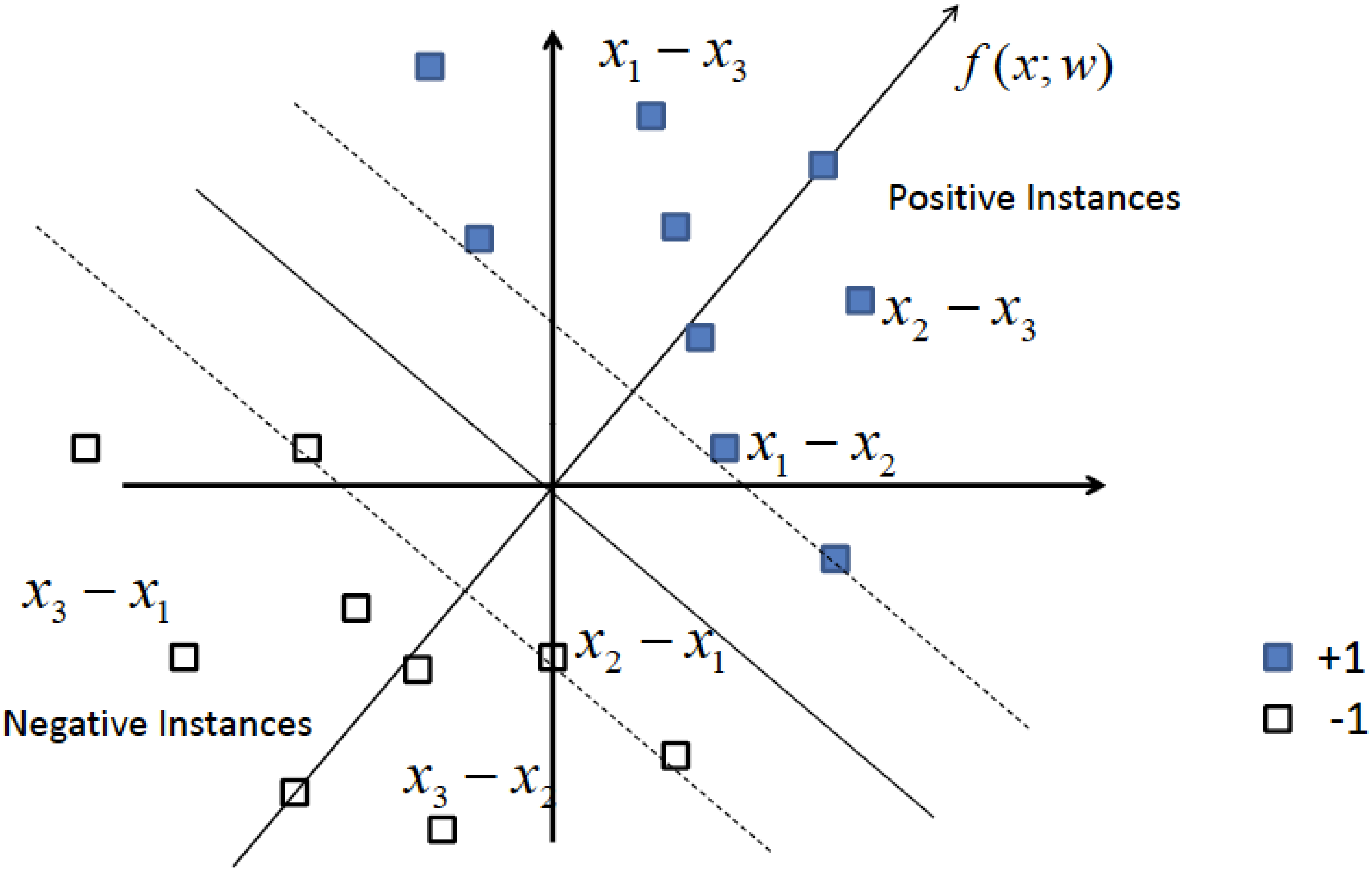
# Training a Linear SVM for Ranking

- But we don't have pointwise training data!
  - Remember we only have lots of $(c_i, c_k)$, where we already know $\mathrm{f}(\psi_i) > \mathrm{f}(\psi_k)$
  - We don't know the value of $\mathrm{f}(\psi_i)$ or $\mathrm{f}(\psi_k)$
- Idea: Create a new instance space from pairwise learning
  - We have $c_i > c_k \iff \mathrm{f}(\psi_i) > \mathrm{f}(\psi_k)$
  - We also have $\mathrm{f}(\psi_i) = w^T\psi_i$ and $\mathrm{f}(\psi_k) = w^T\psi_k$
  - So $c_i > c_k \iff w^T\psi_i > w^T\psi_k \iff w^T(\psi_i - \psi_k) > 0$
  - Let's create a new instance $\phi_u = \psi_i - \psi_k$
  - And $z_u = +1, 0, -1$ as $c_i >, =, < c_k$
  - From training data $\mathcal{S} = \{\phi_u\}$, we train an SVM

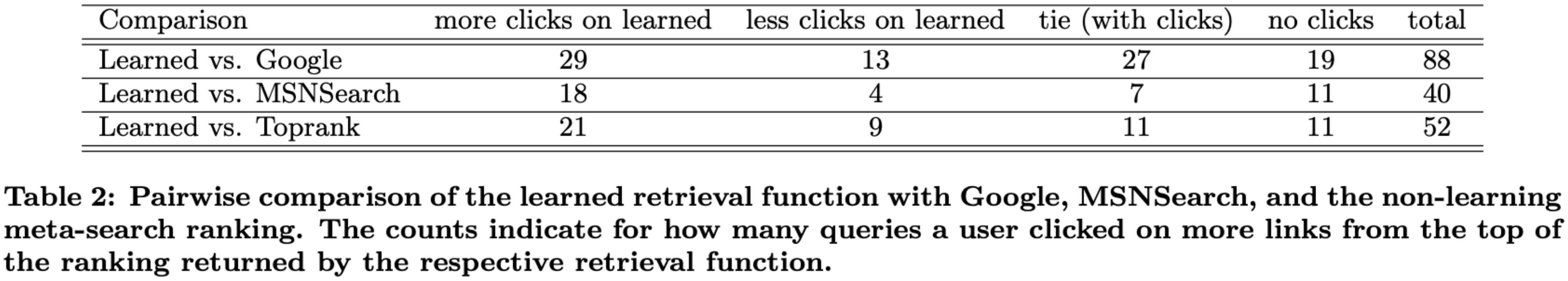
# Two Queries in the Original Space



# Two Queries in the Pairwise Space

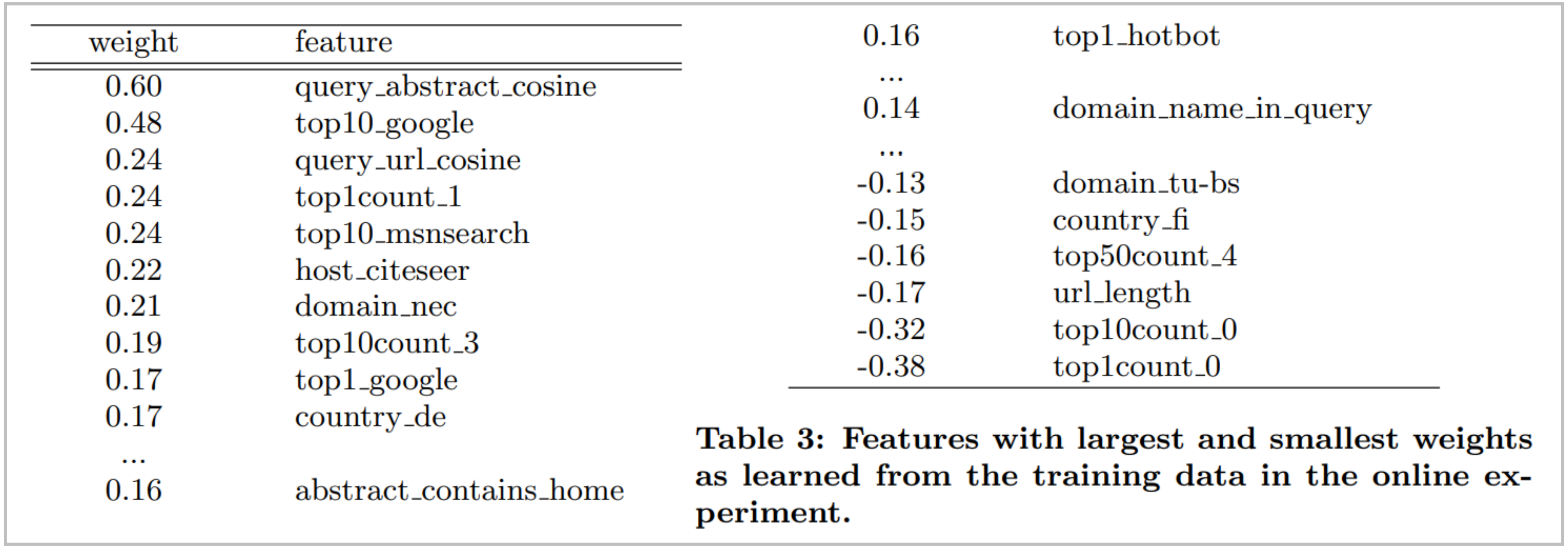


# Performance of Ranking SVM



| Comparison | more clicks on learned | less clicks on learned | tie (with clicks) | no clicks | total |
|---|---|---|---|---|---|
| Learned vs. Google | 29 | 13 | 27 | 19 | 88 |
| Learned vs. MSNSearch | 18 | 4 | 7 | 11 | 40 |
| Learned vs. Toprank | 21 | 9 | 11 | 11 | 52 |

**Table 2: Pairwise comparison of the learned retrieval function with Google, MSNSearch, and the non-learning meta-search ranking. The counts indicate for how many queries a user clicked on more links from the top of the ranking returned by the respective retrieval function.**

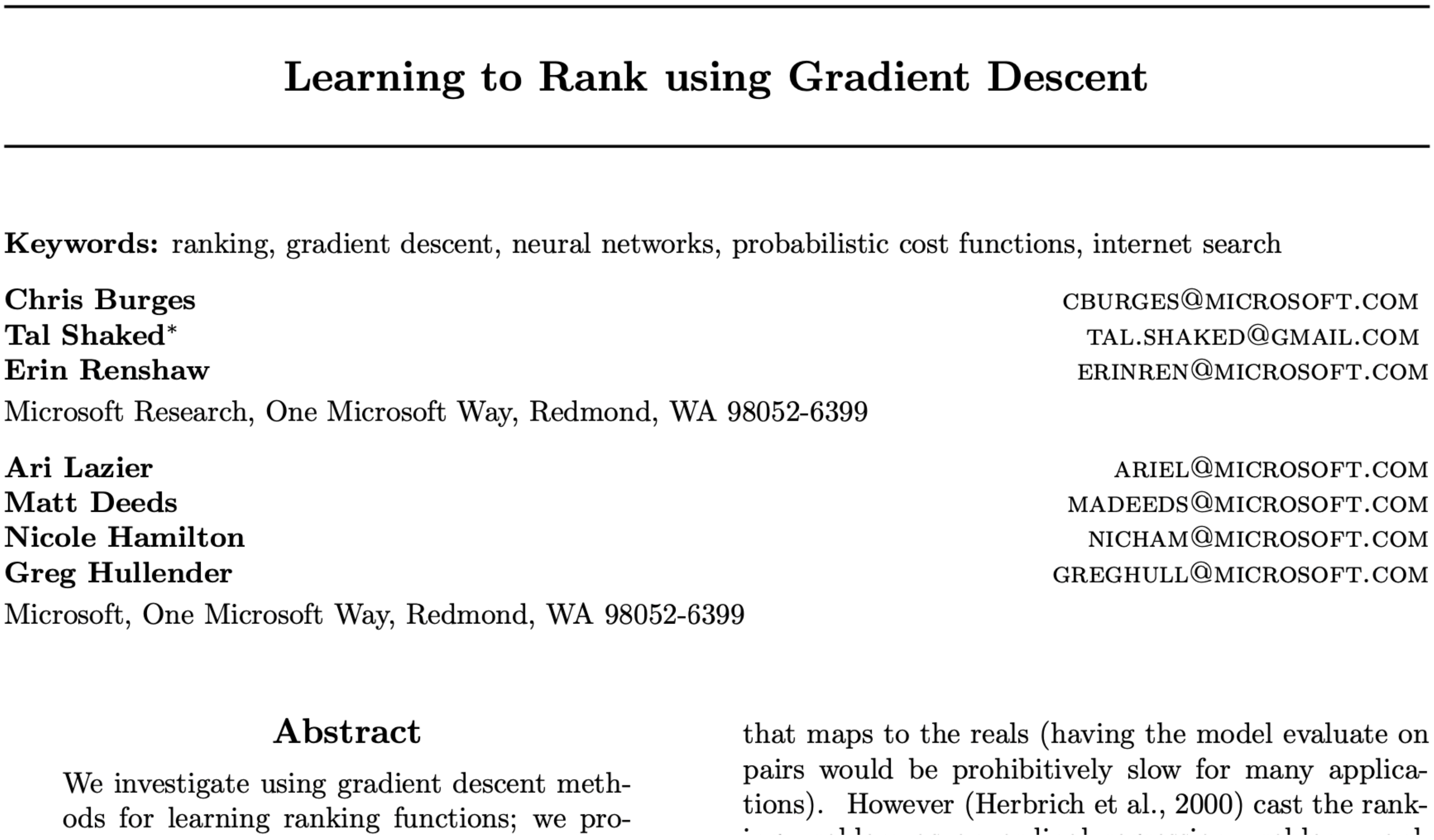


| weight | feature |
|---|---|
| 0.60 | query_abstract_cosine |
| 0.48 | top10_google |
| 0.24 | query_url_cosine |
| 0.24 | top1count_1 |
| 0.24 | top10_msnsearch |
| 0.22 | host_citeseer |
| 0.21 | domain_nec |
| 0.19 | top10count_3 |
| 0.17 | top1_google |
| 0.17 | country_de |
| ... | |
| 0.16 | abstract_contains_home |
| 0.16 | top1_hotbot |
| ... | |
| 0.14 | domain_name_in_query |
| ... | |
| -0.13 | domain_tu-bs |
| -0.15 | country_fi |
| -0.16 | top50count_4 |
| -0.17 | url_length |
| -0.32 | top10count_0 |
| -0.38 | top1count_0 |

**Table 3: Features with largest and smallest weights as learned from the training data in the online experiment.**

Questions?

# Burges et al. [ICML 2025] (RankNet)



## Learning to Rank using Gradient Descent

**Keywords:** ranking, gradient descent, neural networks, probabilistic cost functions, internet search

**Chris Burges** CBURGES@MICROSOFT.COM
**Tal Shaked***  TAL.SHAKED@GMAIL.COM
**Erin Renshaw** ERINREN@MICROSOFT.COM
Microsoft Research, One Microsoft Way, Redmond, WA 98052-6399

**Ari Lazier** ARIEL@MICROSOFT.COM
**Matt Deeds** MADEEDS@MICROSOFT.COM
**Nicole Hamilton** NICHAM@MICROSOFT.COM
**Greg Hullender** GREGHULL@MICROSOFT.COM
Microsoft, One Microsoft Way, Redmond, WA 98052-6399

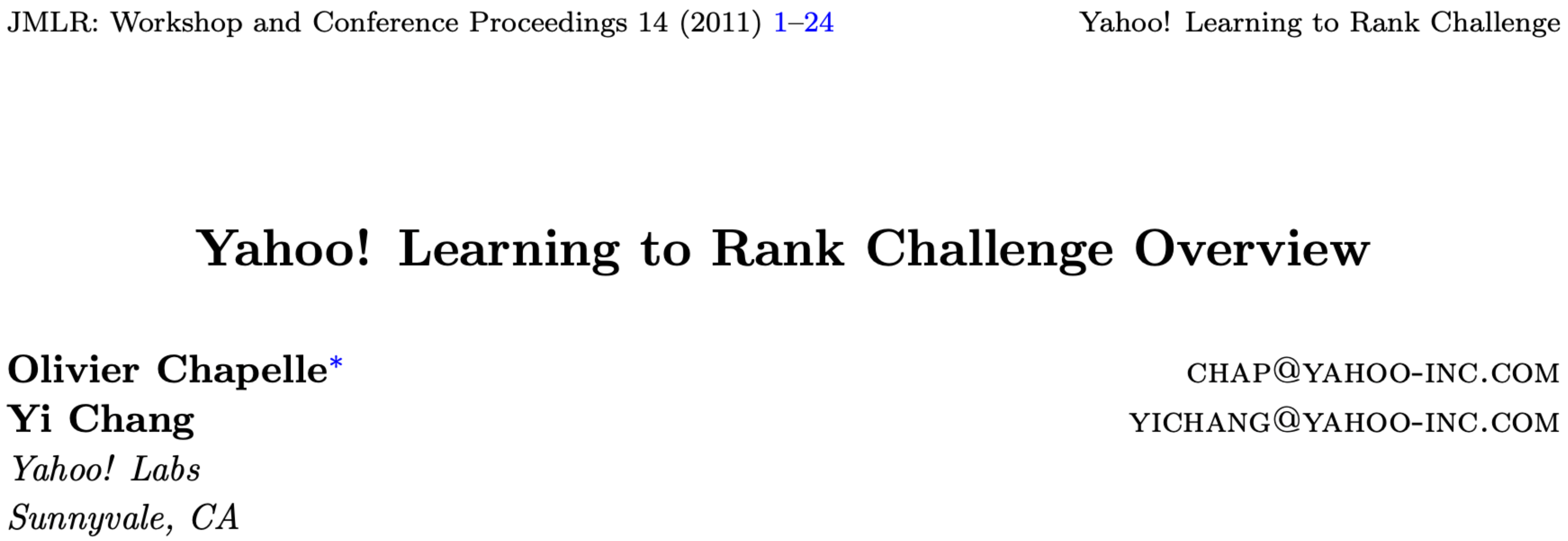
### Abstract

We investigate using gradient descent methods for learning ranking functions; we pro-

that maps to the reals (having the model evaluate on pairs would be prohibitively slow for many applications). However (Herbrich et al., 2000) cast the rank-

# Burges et al. [ICML 2025] (RankNet)

- Led to popular and successful variants:
  - LambdaRank
  - LambdaMART: top performer at the 2010 Yahoo Learning to Rank Challenge



JMLR: Workshop and Conference Proceedings 14 (2011) 1–24 Yahoo! Learning to Rank Challenge

## Yahoo! Learning to Rank Challenge Overview

**Olivier Chapelle*** CHAP@YAHOO-INC.COM
**Yi Chang** YICHANG@YAHOO-INC.COM
*Yahoo! Labs*
*Sunnyvale, CA*

# Existing Public Datasets

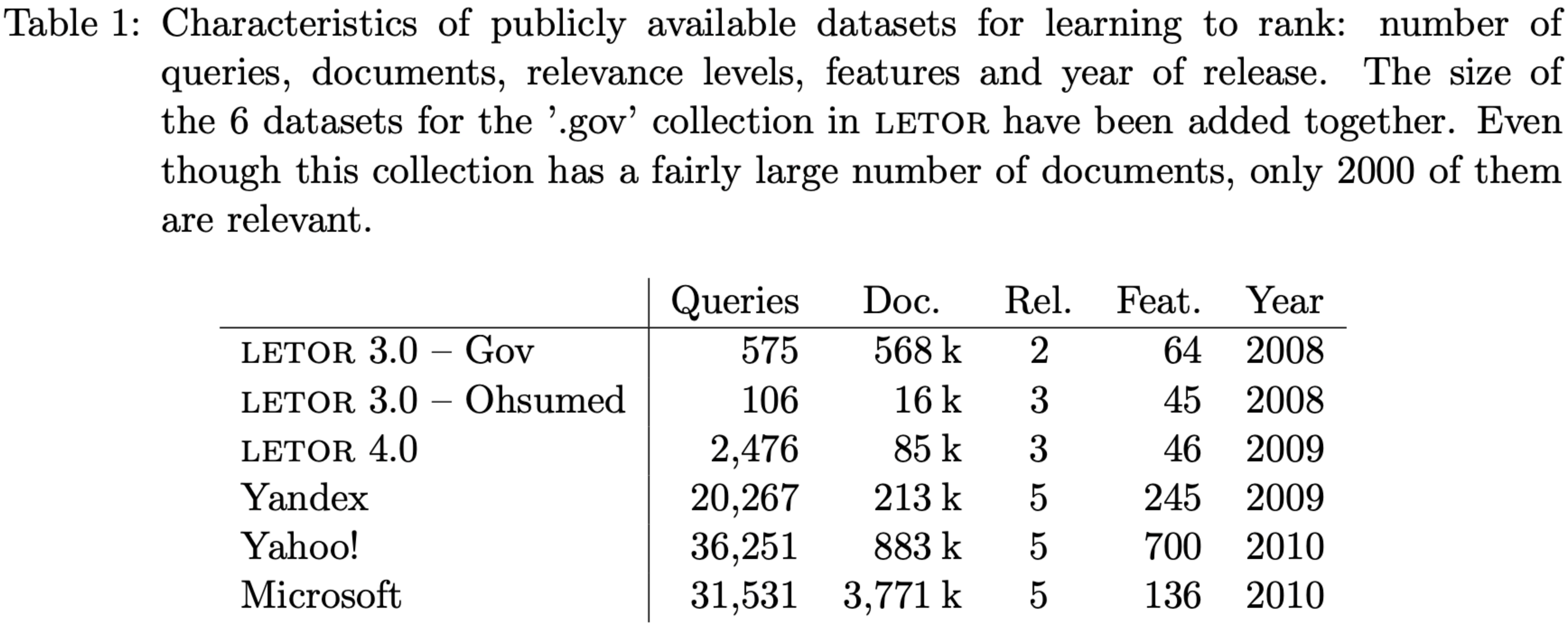
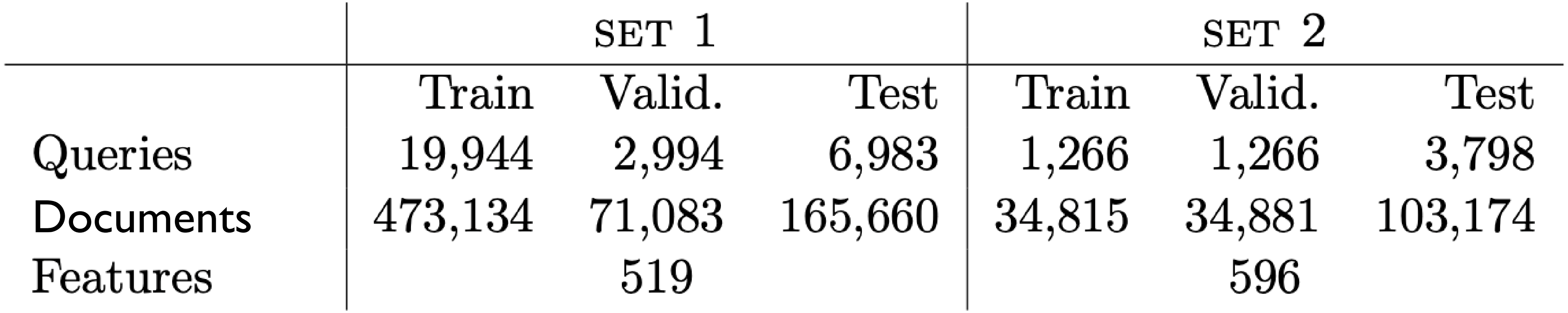


Table 1: Characteristics of publicly available datasets for learning to rank: number of queries, documents, relevance levels, features and year of release. The size of the 6 datasets for the '.gov' collection in LETOR have been added together. Even though this collection has a fairly large number of documents, only 2000 of them are relevant.

| | Queries | Doc. | Rel. | Feat. | Year |
|---|---|---|---|---|---|
| LETOR 3.0 – Gov | 575 | 568 k | 2 | 64 | 2008 |
| LETOR 3.0 – Ohsumed | 106 | 16 k | 3 | 45 | 2008 |
| LETOR 4.0 | 2,476 | 85 k | 3 | 46 | 2009 |
| Yandex | 20,267 | 213 k | 5 | 245 | 2009 |
| Yahoo! | 36,251 | 883 k | 5 | 700 | 2010 |
| Microsoft | 31,531 | 3,771 k | 5 | 136 | 2010 |

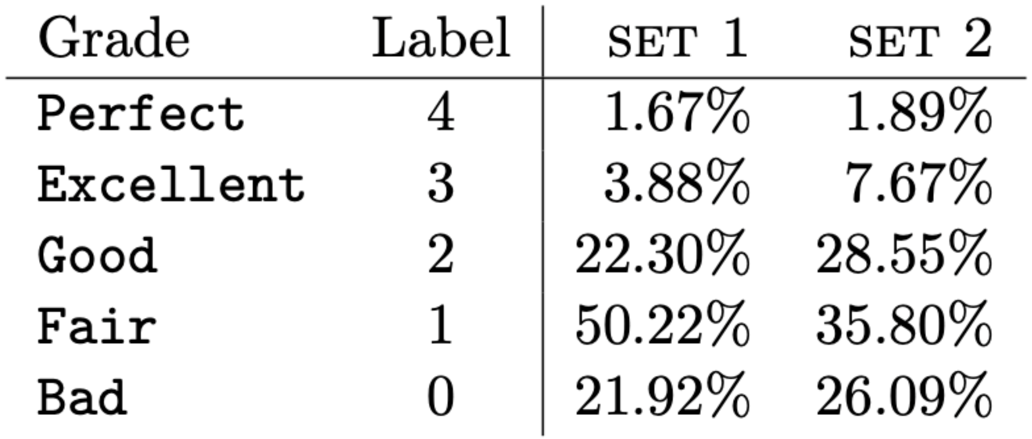
# Datasets for the Challenge

Table 2: Statistics of the two datasets released for the challenge.



| | SET 1 | | | SET 2 | | |
|---|---|---|---|---|---|---|
| | Train | Valid. | Test | Train | Valid. | Test |
| Queries | 19,944 | 2,994 | 6,983 | 1,266 | 1,266 | 3,798 |
| Documents | 473,134 | 71,083 | 165,660 | 34,815 | 34,881 | 103,174 |
| Features | 519 | | | 596 | | |

Table 3: Distribution of relevance labels.



| Grade | Label | SET 1 | SET 2 |
|---|---|---|---|
| Perfect | 4 | 1.67% | 1.89% |
| Excellent | 3 | 3.88% | 7.67% |
| Good | 2 | 22.30% | 28.55% |
| Fair | 1 | 50.22% | 35.80% |
| Bad | 0 | 21.92% | 26.09% |

# Features

- Web graph: in-links, out-links, PageRank, …
- Doc statistics: # of words in title, # of words in body, number of slashes in URL, …
- Doc classifier: spam, topic, language, …
- Query: # of terms, frequency of query and its terms, …
- Text match: BM25, counts, …
- Clicks: probability of a click, dwell time, …
- External references: tags
- Time: age of doc, age of in-links, …

# Baselines

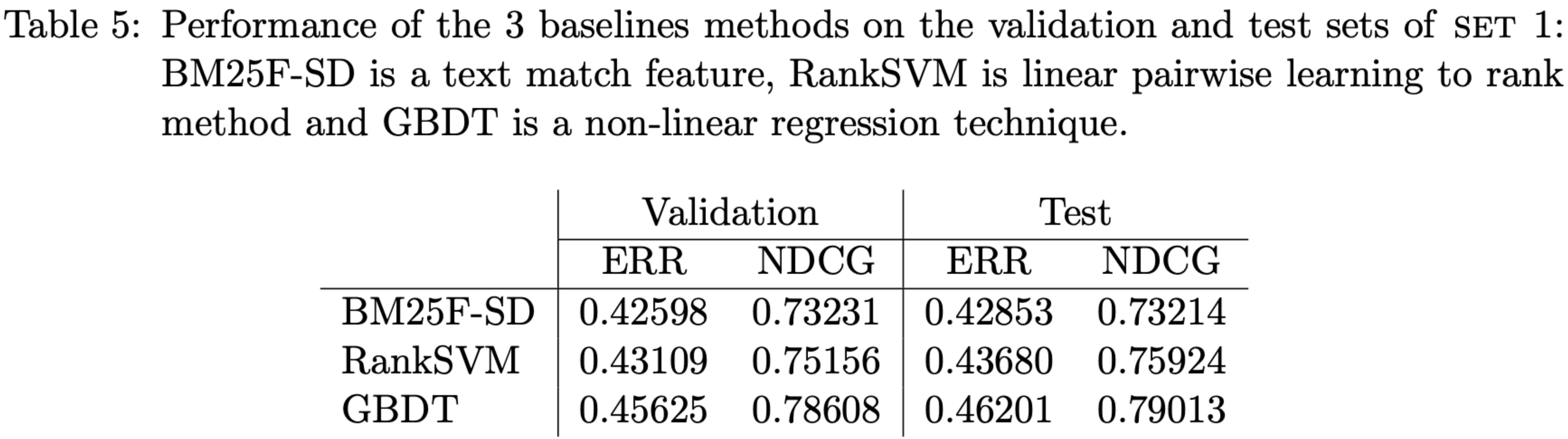
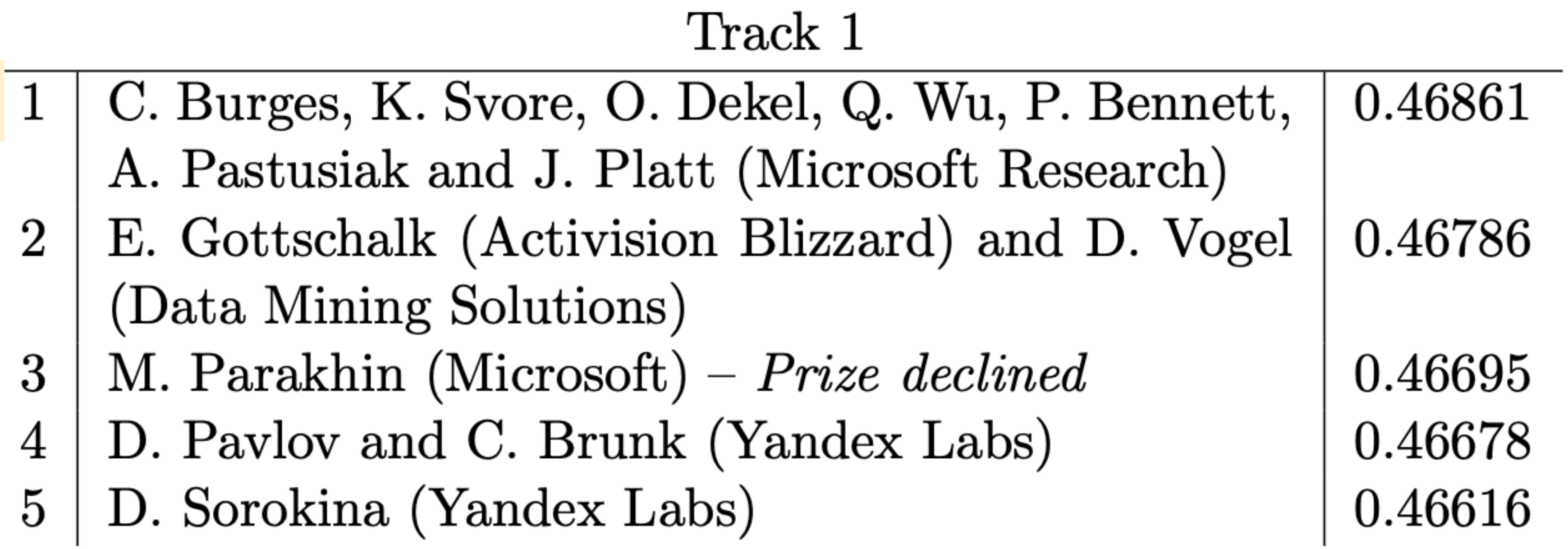


Table 5: Performance of the 3 baselines methods on the validation and test sets of SET 1: BM25F-SD is a text match feature, RankSVM is linear pairwise learning to rank method and GBDT is a non-linear regression technique.

| | Validation | | Test | |
|---|---|---|---|---|
| | ERR | NDCG | ERR | NDCG |
| BM25F-SD | 0.42598 | 0.73231 | 0.42853 | 0.73214 |
| RankSVM | 0.43109 | 0.75156 | 0.43680 | 0.75924 |
| GBDT | 0.45625 | 0.78608 | 0.46201 | 0.79013 |

# The Winners



Track 1

| | | |
|---|---|---|
| 1 | C. Burges, K. Svore, O. Dekel, Q. Wu, P. Bennett, A. Pastusiak and J. Platt (Microsoft Research) | 0.46861 |
| 2 | E. Gottschalk (Activision Blizzard) and D. Vogel (Data Mining Solutions) | 0.46786 |
| 3 | M. Parakhin (Microsoft) – *Prize declined* | 0.46695 |
| 4 | D. Pavlov and C. Brunk (Yandex Labs) | 0.46678 |
| 5 | D. Sorokina (Yandex Labs) | 0.46616 |

RankNet

- What is RankNet? Next lecture!

# Thank You!

Course Website: https://yuzhang-teaching.github.io/CSCE670-F25.html