Introduction to

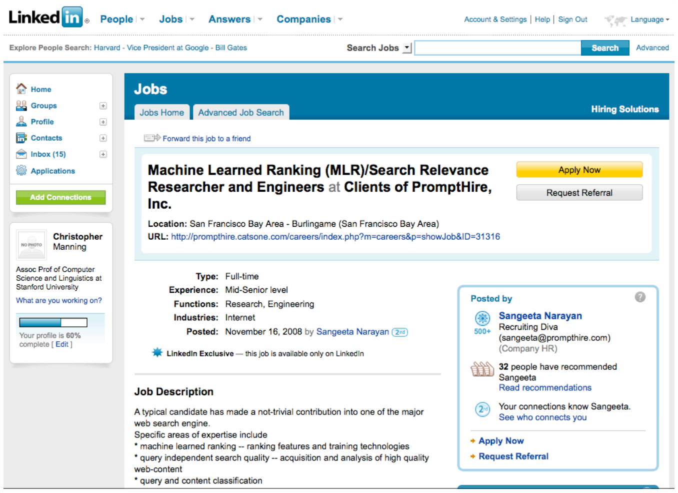
Information Retrieval

CS276: Information Retrieval and Web Search
Christopher Manning and Pandu Nayak

Lecture 15: Learning to Rank

Machine learning for IR ranking?

- We've looked at methods for ranking documents in IR
  - Cosine similarity, inverse document frequency, proximity, pivoted document length normalization, Pagerank, …
- We've looked at methods for classifying documents using supervised machine learning classifiers
  - Naïve Bayes, Rocchio, kNN, SVMs
- Surely we can also use machine learning to rank the documents displayed in search



Machine Learned Ranking (MLR)/Search Relevance Researcher and Engineers at **Clients of PromptHire, Inc.**

Location: San Francisco Bay Area - Burlingame (San Francisco Bay Area)
URL: http://prompthire.catsone.com/careers/index.php?m=careers&p=showJob&ID=31316

Type: Full-time
Experience: Mid-Senior level
Functions: Research, Engineering
Industries: Internet
Posted: November 16, 2008 by Sangeeta Narayan 2nd

LinkedIn Exclusive — this job is available only on LinkedIn

Job Description

A typical candidate has made a not-trivial contribution into one of the major web search engine.
Specific areas of expertise include
* machine learned ranking -- ranking features and training technologies
* query independent search quality -- acquisition and analysis of high quality web-content
* query and content classification

Posted by
Sangeeta Narayan
500+
Recruiting Diva
(sangeeta@prompthire.com)
(Company HR)

32 people have recommended Sangeeta
Read recommendations

Your connections know Sangeeta.
See who connects you

Apply Now
Request Referral

Christopher Manning
NO PHOTO
Assoc Prof of Computer Science and Linguistics at Stanford University
What are you working on?
Your profile is **60%** complete [Edit]

Machine learning for IR ranking

- This "good idea" has been actively researched – and actively deployed by major web search engines – in the last 7 or so years
- Why didn't it happen earlier?
  - Modern supervised ML has been around for about 20 years…
  - Naïve Bayes has been around for about 50 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

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

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.

Simple example: Using classification for ad hoc IR

- Collect a training corpus of (q, d, r) triples
  - Relevance r is here binary (but may be multiclass, with 3–7 values)
  - Document is represented by a feature vector
    - **x** = (α, ω) α is cosine similarity, ω is minimum query window size
      - ω is the the shortest text span that includes all query words

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*

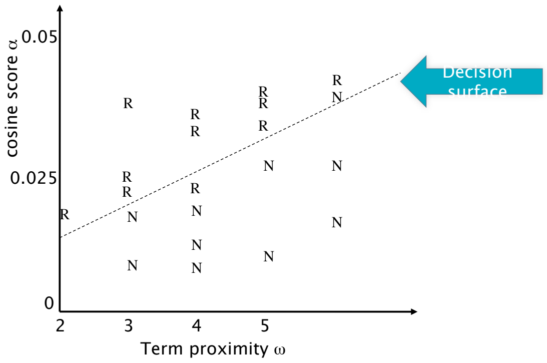
Simple example: Using classification for ad hoc IR

- A linear score function is then

  $\mathrm{Score}(d, q) = \mathrm{Score}(\alpha, \omega) = a\alpha + b\omega + c$
- And the linear classifier is

  Decide relevant if $\mathrm{Score}(d, q) > \theta$
- … just like when we were doing text classification

Simple example: Using classification for ad hoc IR



More complex example of using classification for search ranking [Nallapati 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 information retrieval [Nallapati 2004]

- Let g(r|d,q) = **w**•f(d,q) + b
- SVM training: want g(r|d,q) ≤ −1 for nonrelevant documents and g(r|d,q) ≥ 1 for relevant documents
- SVM testing: decide relevant iff g(r|d,q) ≥ 0

- Features are not word presence features (how would you deal with query words not in your training data?) but scores like the summed (log) tf of all query terms
- Unbalanced data (which can result in trivial always-say-nonrelevant classifiers) is dealt with by undersampling nonrelevant documents

An SVM classifier for information retrieval [Nallapati 2004]

- Experiments:
  - 4 TREC data sets
  - Comparisons with Lemur, a state-of-the-art open source IR engine (Language Model (LM)-based – see IIR ch. 12)
  - Linear kernel normally best or almost as good as quadratic kernel, and so used in reported results
  - 6 features, all variants of tf, idf, and tf.idf scores

An SVM classifier for information retrieval [Nallapati 2004]

Train \		Disk 3	Disk 4-5	WT10G
Disk 3	LM	**0.1785**	**0.2503**	0.2666
	SVM	0.1728	0.2432	**0.2750**
Disk 4-5	LM	**0.1773**	**0.2516**	0.2656
	SVM	0.1646	0.2355	**0.2675**

- At best the results are about equal to LM
  - Actually a little bit below
- Paper's advertisement: Easy to add more features
  - This is illustrated on a homepage finding task on WT10G:
  - Baseline LM 52% success@10, baseline SVM 58%

"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

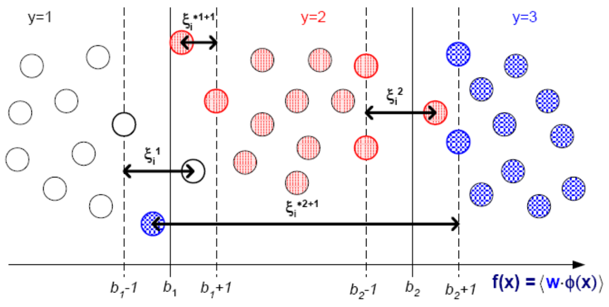
"Learning to rank"

- Assume a number of categories **C** of relevance exist
  - These are totally ordered: $c_1 < c_2 < \dots < c_J$
  - 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

- 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

Point-wise learning

- Goal is to learn a threshold to separate each rank



Pairwise learning: The Ranking SVM

- 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

$$c_i > c_k \text{ iff } f(\psi_i) > f(\psi_k)$$

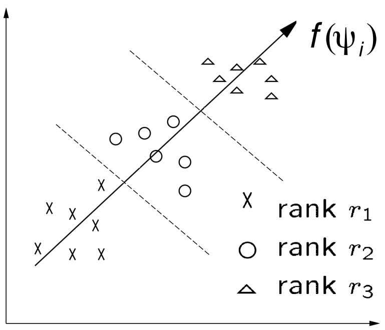
- … or at least one that tries to do this with minimal error
- Suppose that f is a linear function

$$f(\psi_i) = \mathbf{w}\bullet\psi_i$$

The Ranking SVM

[Herbrich et al. 1999, 2000; Joachims et al. 2002]

- Ranking Model: $f(\psi_i)$



The Ranking SVM

[Herbrich et al. 1999, 2000; Joachims et al. 2002]

- Then (combining the two equations on the last slide):

$$c_i > c_k \text{ iff } \mathbf{w}\bullet(\psi_i - \psi_k) > 0$$

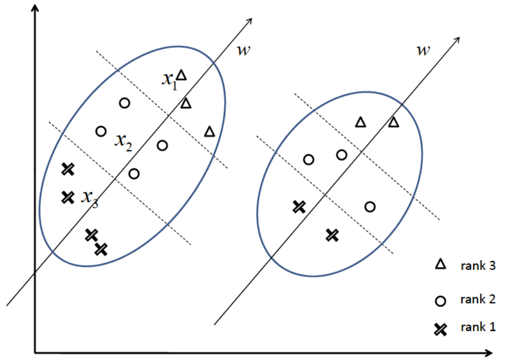
- Let us then create a new instance space from such pairs:

$$\Phi_u = \Phi(d_i, d_j, q) = \psi_i - \psi_k$$

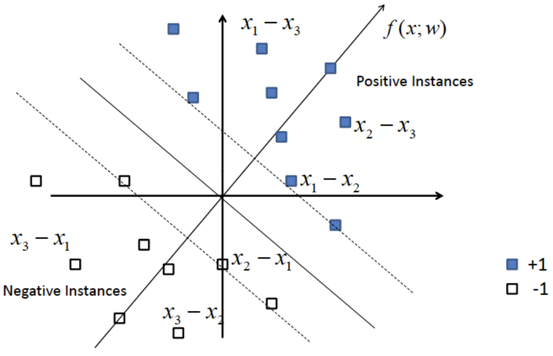
$$z_u = +1, 0, -1 \text{ as } c_i >,=,< c_k$$

- We can build model over just cases for which $z_u = -1$
- From training data $S = \{\Phi_u\}$, we train an SVM

Two queries in the original space



Two queries in the pairwise space



The Ranking SVM

[Herbrich et al. 1999, 2000; Joachims et al. 2002]

- The SVM learning task is then like other examples that we saw before
- Find **w** and $\xi_u \geq 0$ such that
  - $\frac{1}{2}\mathbf{w}^T\mathbf{w} + C\,\Sigma\,\xi_u$ is minimized, and
  - for all Φ_u such that $z_u < 0$, $\mathbf{w}\bullet\Phi_u \geq 1 - \xi_u$
- We can just do the negative z_u, as ordering is antisymmetric
- You can again use SVMlight (or other good SVM libraries) to train your model (SVMrank

Aside: The SVM loss function

- The minimization

$$\min_{\mathbf{w}} \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\,\Sigma\,\xi_u$$

and for all Φ_u such that $z_u < 0$, $\mathbf{w}\bullet\Phi_u \geq 1 - \xi_u$

- can be rewritten as

$$\min_{\mathbf{w}} (1/2C)\mathbf{w}^T\mathbf{w} + \Sigma\,\xi_u$$

and for all Φ_u such that $z_u < 0$, $\xi_u \geq 1 - (\mathbf{w}\bullet\Phi_u)$

- Now, taking $\lambda = 1/2C$, we can reformulate this as

$$\min_{\mathbf{w}} \Sigma\,[1 - (\mathbf{w}\bullet\Phi_u)]_+ + \lambda\mathbf{w}^T\mathbf{w}$$

Aside: The SVM loss function

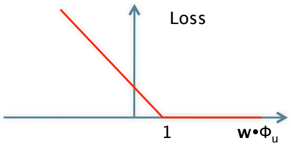
- The reformulatio

Hinge loss

Regularizer of||w||

$$\min_{\mathbf{w}} \underbrace{\Sigma\,[1 - (\mathbf{w}\bullet\Phi_u)]_+}_{} + \lambda\mathbf{w}^T\mathbf{w}$$

- shows that an SVM can be thought of as having an empirical **“hinge” loss** combined with a **weight regularizer**



Adapting the Ranking SVM for (successful) Information Retrieval

[Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, Hsiao-Wuen Hon SIGIR 2006]

- A Ranking SVM model already works well
    - Using things like vector space model scores as features
    - As we shall see, it outperforms them in evaluations
- But it does not model important aspects of practical IR well
- This paper addresses two customizations of the Ranking SVM to fit an IR utility model

The ranking SVM fails to model the IR problem well …

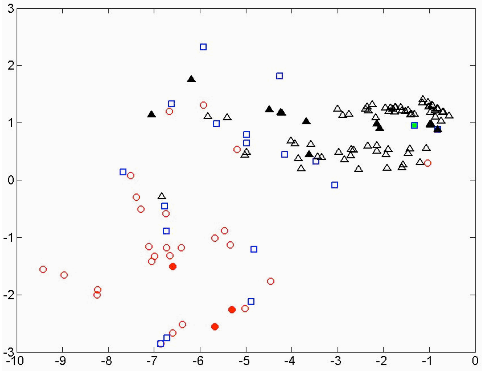
1. Correctly ordering the most relevant documents is crucial to the success of an IR system, while misordering less relevant results matters little
    - The ranking SVM considers all ordering violations as the same
2. Some queries have many (somewhat) relevant documents, and other queries few. If we 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

Based on the LETOR test collection

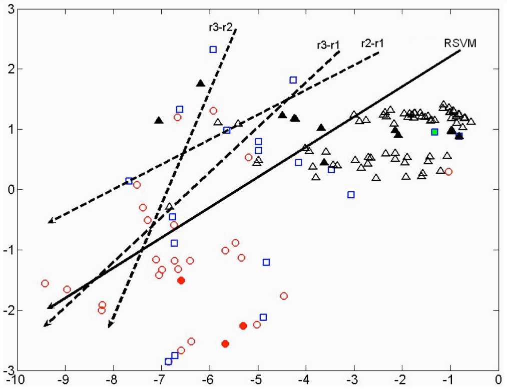
- From Microsoft Research Asia
- An openly available standard test collection with pregenerated features, baselines, and research results for learning to rank
- It’s availability has really driven research in this area
- OHSUMED, MEDLINE subcollection for IR
    - 350,000 articles
    - 106 queries
    - 16,140 query-document pairs
    - 3 class judgments: Definitely relevant (DR), Partially Relevant (PR), Non-Relevant (NR)
- TREC GOV collection (predecessor of GOV2, cf. IIR p. 142)
    - 1 million web pages

Principal components projection of 2 queries

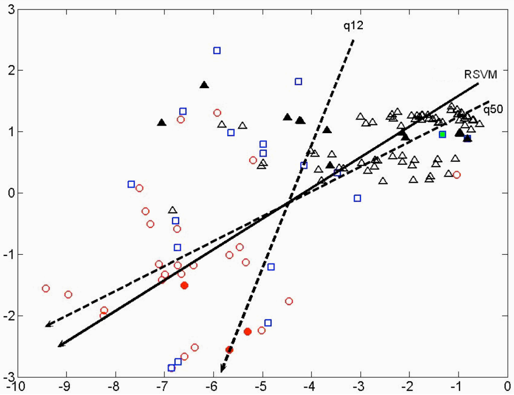
[solid = q12, open = q50; circle = DR, square = PR,



Ranking scale importance discrepancy



Number of training documents per query discrepancy [solid = q12, open =



Recap: Two Problems with Direct Application of the Ranking SVM

- Cost sensitiveness: negative effects of making errors on top ranked documents

  d: *definitely relevant*, p: *partially relevant*, n: *not relevant*

  ranking 1: p d p n n n n
  ranking 2: d p n p n n n

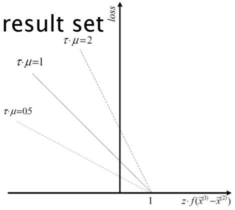
- Query normalization: number of instance pairs varies according to query

  q1: d p p n n n n
  q2: d d p p p n n n n n
  q1 pairs: 2*(d, p) + 4*(d, n) + 8*(p, n) = 14
  q2 pairs: 6*(d, p) + 10*(d, n) + 15*(p, n) = 31

These problems are solved with a new Loss function

$$\min_{\vec{w}} L(\vec{w}) = \tau_{k(i)}\mu_{q(i)}\ 1 - z_i\ \vec{w}, \vec{x}_i^{(1)} - \vec{x}_i^{(2)}$$

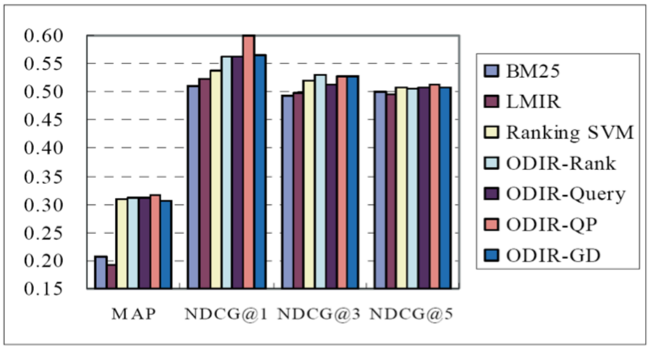
- τ weights for type of rank difference
  - Estimated empirically from effect on NDCG
- μ weights for size of ranked result set
  - Linearly scaled versus biggest result set



Experiments

- OHSUMED (from LETOR)
- Features:
  - 6 that represent versions of tf, idf, and tf.idf factors
  - BM25 score (IIR sec. 11.4.3)
    - A scoring function derived from a probabilistic approach to IR, which has traditionally done well in TREC evaluations, etc.

Experimental Results (OHSUMED)

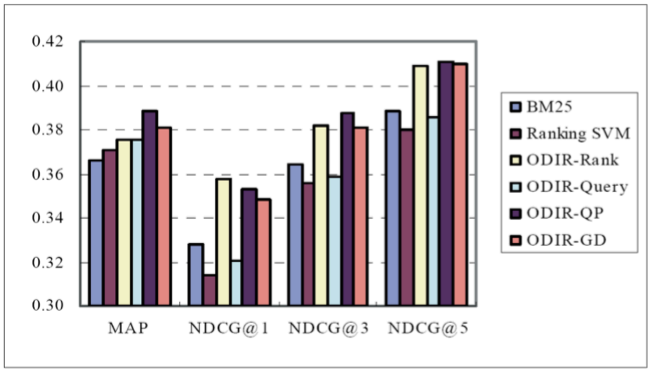


MSN Search [now Bing]

- Second experiment with MSN search
- Collection of 2198 queries
- 6 relevance levels rated:

Definitive	8990
Excellent	4403
Good	3735
Fair	20463
Bad	36375
Detrimental	310

Experimental Results (MSN search)



Alternative: Optimizing Rank-Based Measures

[Yue et al. SIGIR 2007]

- If we think that NDCG is a good approximation of the user's utility function from a result ranking
- Then, let's directly optimize this measure
  - As opposed to some proxy (weighted pairwise prefs)
- But, there are problems …
  - Objective function no longer decomposes
    - Pairwise prefs decomposed into each pair
  - Objective function is flat or discontinuous

Discontinuity Example

- NDCG computed using rank positions
- Ranking via retrieval scores
- Slight changes to model parameters
  - Slight changes to retrieval scores
  - No change to ranking
  - No change to NDCG

NDCG = 0.63

	d_1	d_2	d_3
Retrieval Score	0.	0.	0.
Rank	1	2	3
Relevance	0	1	0

NDCG discontinuous w.r.t model parameters!

Structural SVMs [Tsochantaridis et al., 2007]

- Structural SVMs are a generalization of SVMs where the output classification space is not binary or one of a set of classes, but some complex object (such as a sequence or a parse tree)
- Here, it is a complete (weak) ranking of documents for a query
- The Structural SVM attempts to predict the complete ranking for the input query and document set
- The **true labeling** is a ranking where the relevant documents are all ranked in the front, e.g.,

y= 🟩🟩🟩🟥🟥🟥

- An **incorrect labeling** would be any other ranking, e.g.,

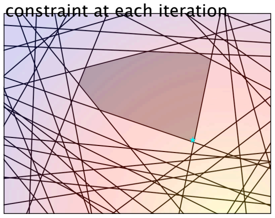
y= 🟩🟥🟩🟩🟥🟥

- **There are an intractable number of rankings, thus an**

Structural SVM training

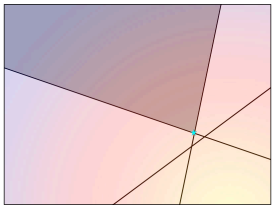
[Tsochantaridis et al., 2007]

Structural SVM training proceeds incrementally by starting with a working set of constraints, and adding in the most violated constraint at each iteration



Original SVM Problem

- Exponential constraints
- Most are dominated by a small set of "important" constraints



Structural SVM Approach

- Repeatedly finds the next most violated constraint…
- …until a set of constraints which is a good approximation is found

Other machine learning methods for learning to rank

- Of course!
- I've only presented the use of SVMs for machine learned relevance, but other machine learning methods have also been used successfully
  - Boosting: RankBoost
  - Ordinal Regression loglinear models
  - Neural Nets: RankNet
  - (Gradient-boosted) Decisision Trees

The Limitation of Machine Learning

- 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

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

Resources

- IIR secs 6.1.2-3 and 15.4
- LETOR benchmark datasets
  - Website with data, links to papers, benchmarks, etc.
  - http://research.microsoft.com/users/LETOR/
  - Everything you need to start research in this area!
- Nallapati, R. Discriminative models for information retrieval. SIGIR 2004.
- Cao, Y., Xu, J. Liu, T.-Y., Li, H., Huang, Y. and Hon, H.-W. Adapting Ranking SVM to Document Retrieval, SIGIR 2006.
- Y. Yue, T. Finley, F. Radlinski, T. Joachims. A Support Vector Method for Optimizing Average Precision. SIGIR 2007.