Fall 2016
CS646: *Information Retrieval*

Lecture 16: Learning-to-rank

Jiepu Jiang

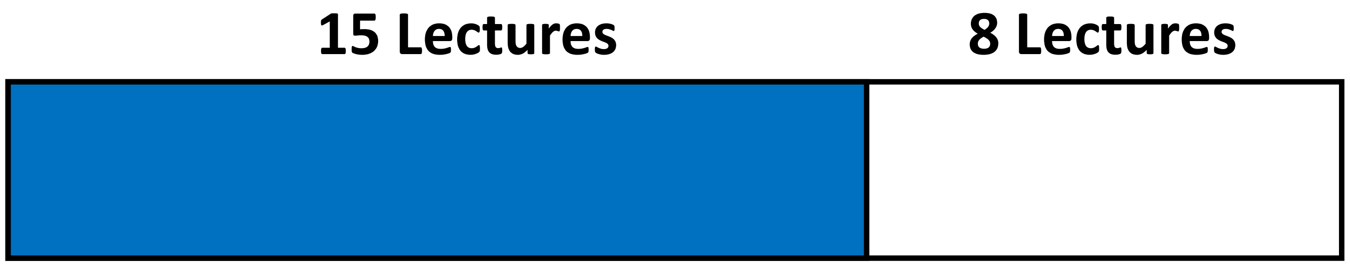
University of Massachusetts Amherst

2016/11/2

Credit: some materials are from Christopher D. Manning, James Allan, and Honglin Wang's course slides

Congratulations!

- **You’ve already finished the “core” part of this course**
  - IR system core issues, e.g., inverted index, index search
  - “Classic” retrieval models, e.g., VSM, BM25, LM
    - In contrast to the learning-to-rank methods today
  - Evaluation techniques, especially the test-collection based ones
  - These are topics that everyone finished an IR course should know
  - Also, almost all IR courses include these topics
    - except user-oriented evaluation (but this is necessary because about 1/3 of the IR community works on user-related topics & most textbooks do not include these topics ….)



We still have 1/3 (8 lectures)

- **The follow-up lectures**
  - Topics are very diverse and interdisciplinary! ML & DL, HCI, NLP
  - Welcome to the real world!
    - Few search engines are using the classic retrieval models alone
    - Most are using learning-to-rank (combining classic with others)
    - There are topics beyond ranking & evaluation
      - e.g., query suggestion, Q&A
  - Technical details are optional
    - The purpose is to inform you there are such issues/techniques/ideas
    - We will not ask you technical details in your final exam
      - All questions in your final are open ended, e.g., give you a scenario, discuss the solutions
    - You can always come back to check details if you need to use them …
  - But you need to know the high-level ideas

Outline

- ***Learning-to-rank***
- RankLib Tutorial
- Midterm Q & A

Classic Retrieval Models

- **We’ve discussed many different retrieval models so far**
  - VSM
  - BIM, BM25
  - LM, KL-divergence
  - There are other popular models, just we don’t have time to introduce all
    - e.g., divergence from randomness
- **But they all implemented many similar ideas**
  - TF
  - IDF
  - Penalizing repeated occurrences of the same term
  - Penalizing TF for longer documents

Classic Retrieval Models

- **Classic Retrieval Models**
  - Input: *q*, *d*
    - Or more specifically, inputs are: TF, IDF, |D|, P(t|Corpus) etc.
  - Output: some score indicating *relevance*(*q*, *d*)
  - Classical retrieval models: manually design a function

$$\text{VSM cosine: } \cos\left(\vec{q},\vec{d}\right) \propto \frac{\sum_{i=1}^{k} q_i d_i}{\sqrt{\sum_{i=1}^{k} d_i^2}} = \frac{\sum_{t\in q} w(t,q)\,w(t,d)}{\sqrt{\sum_{i=1}^{k} d_i^2}} = \frac{\sum_{t\in q} qtf \cdot tf \cdot idf^2}{\sqrt{\sum_{i=1}^{k} d_i^2}}$$

$$\text{BM25: } \quad w_i = \frac{(k_3+1)\cdot qtf}{k_3 + qtf} \cdot \frac{(k_1+1)\cdot tf}{k_1\left((1-b)+b\cdot\frac{dl}{avdl}\right)+tf} \cdot \log\frac{N-n+0.5}{n+0.5}$$

$$\text{LM Dirichlet: } \quad \sum_{t\in q} \log \frac{tf + \mu \cdot P(t\,|\,Corpus)}{dl + \mu}$$

Classic models vs. Learning-to-rank

- **Classic Retrieval Models**
  - Features (factors): only a few, e.g., TF, IDF, |D|, P(t|Corpus) etc.
  - Structure: optimized for the a few particular features
  - Parameter & training
    - Often 1-2; not every factor has a parameter controlling its influence
    - Hand-tuning or data-based; can exhaustive since just 1-2 parameters
- **Learning-to-rank**
  - Features: can include up to hundreds, thousands, or even more
  - Define the basic structure of a model
    - Quite generic: such as a weighted linear combination of all features
  - Parameters & training
    - Many; control the influence of each feature and their combinations
    - Impossible to tune by hand; Must be data-driven

Why Learning-to-rank?

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

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?

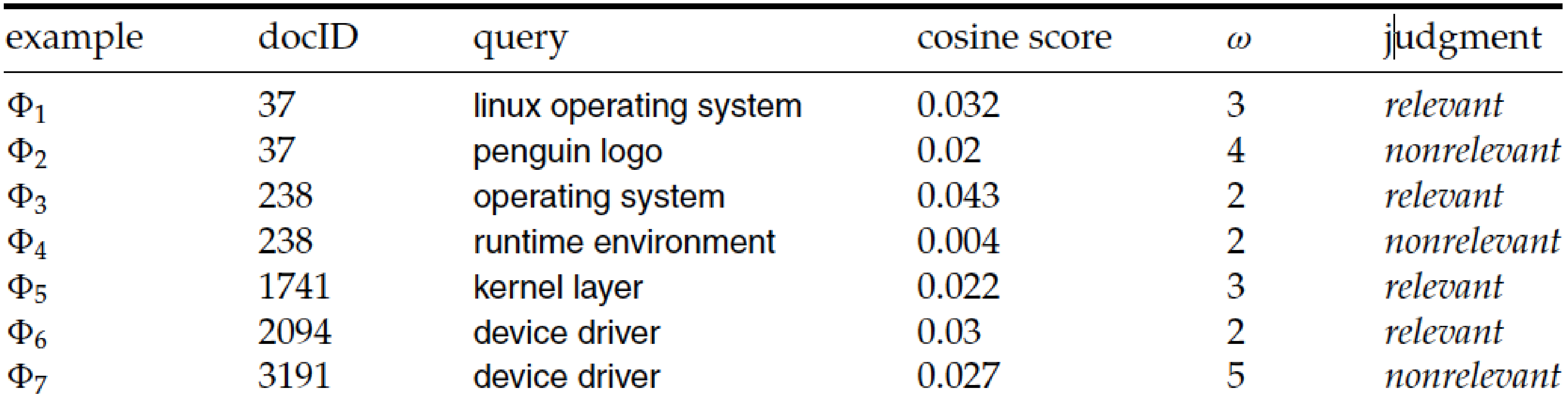
- **Traditional ranking functions in IR used a very small number of features, e.g.,**
  - Term frequency
  - Inverse document frequency
  - Document length
- **It was easy to tune weight**
  - And people did
  - You guys did in HW2 and HW3
- **Classic retrieval model hasn’t yet reached its upperbound**
  - But no substantial improvements since BM25 and LM

So, What is Learning-to-rank?

- **Purpose**
  - Learn a function automatically to rank results (items) effectively
- **Point-wise approach**
  - The function is based on features of a single object
  - e.g., regress the relevance score, classify docs into R and NR
  - Classic retrieval models are also point-wise: score(q, D)
- **Pair-wise**
  - The function is based on a pair of item
  - e.g., given two documents, predict partial ranking
- **List-wise**
  - The function is based on a ranked list of items
  - e.g., given two ranked list of the same items, which is better?

Simple example of point-wise approach: 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 shortest text span that includes all query words
      - Query term proximity is a **very important** new weighting factor
  - 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*

Simple example of point-wise approach: Using classification for ad hoc IR

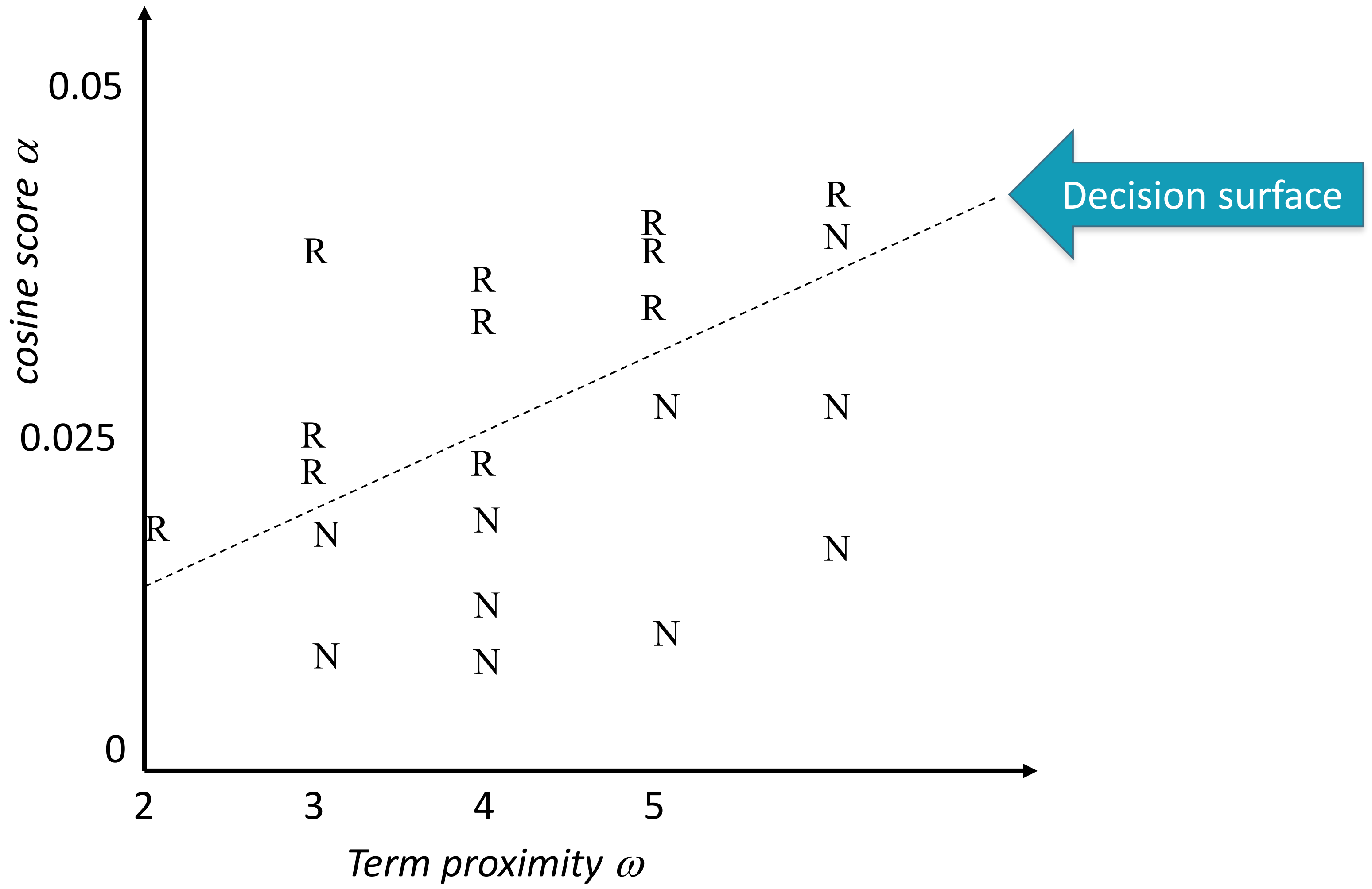
- A linear score function is then

$$Score(d, q) = Score(\alpha, \omega) = a\alpha + b\omega + c$$

- And the linear classifier is

Decide relevant if $Score(d, q) > \theta$

Simple example of point-wise approach: 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
- This is a typical point-wise approach, because the learnt function operates on a single document’s features

An SVM classifier for information retrieval

[Nallapati 2004]

- Let $g(r|d,q) = \mathbf{w} \bullet f(d,q) + b$
- SVM training: want $g(r|d,q) \leq -1$ for nonrelevant documents and $g(r|d,q) \geq 1$ for relevant documents
- SVM testing: decide relevant iff $g(r|d,q) \geq 0$

- 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 during training (just take some at random) [there are other ways of doing this – cf. Cao et al. later]

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 \ Test		Disk 3	Disk 4-5	WT10G (web)
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%
    - SVM with URL-depth, and in-link features: 78% S@10

Some Issues of this Classification-based Point-wise Approach

- **Class imbalance**
  - Many more non-relevant than relevant instances
  - Classifiers usually do not handle huge imbalance well
  - Need to address by over or under sampling
- **Classification error is biased toward "big" queries**
  - Query with 1000 rel documents more important than a query with one
  - Address by weighting training instances
- **Optimization criteria**
  - Classification error not the same as MAP, nDCG, etc.
  - min # errors doesn't guarantee best MAP or nDCG
  - Can incorporate heuristics, e.g., weighting errors based on rel labels
  - But cannot solve all the issues

An Example of the Pairwise approach: The Ranking SVM

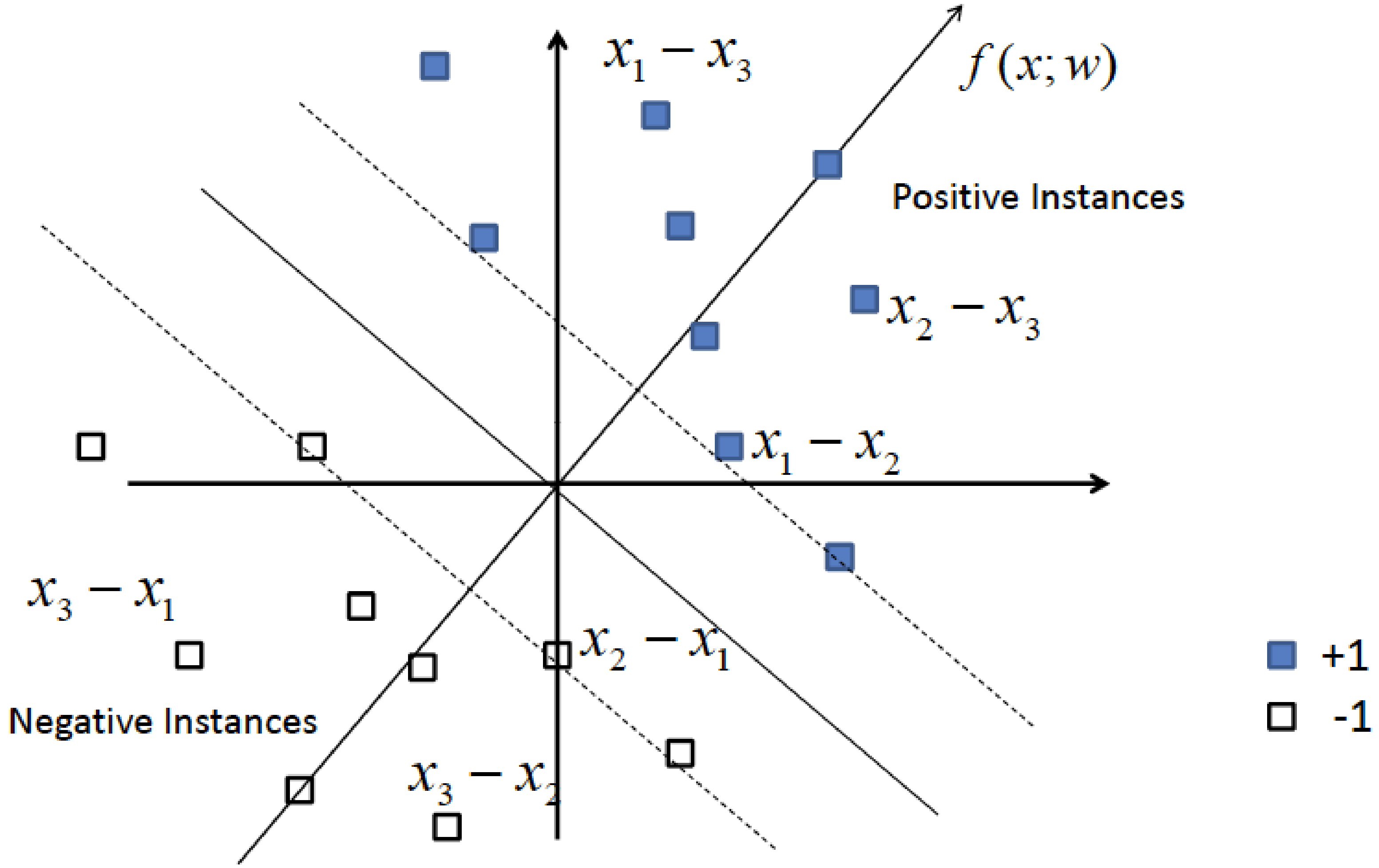
[Herbrich et al. 1999, 2000; Joachims et al. 2002]

- Input:
  - a pair of documents Di and Dj
  - Characterized by feature vectors ψ_i *and* ψ_k
- Classify:

$$\textit{rank } i \textit{ before } k \text{ iff } \mathbf{w} \bullet (\psi_i - \psi_k) > 0$$

An Example of the Pairwise approach: The Ranking SVM

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



An Example of the Pairwise approach: The Ranking SVM

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

- A generalization of SVM that supports ranking
- Trying to classify
    - Which document of two should be ranked at a higher position?
- Optimize based on
    - Margin between decision hyperplane and instances
    - Errors
    - Weighted based on some hyper-parameter C

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

$$\textit{subject to:}$$

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

$$\ldots$$

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

$$\forall i \forall j \forall k : \xi_{i,j,k} \geq 0$$

Pair-wise Approaches

- **More popular than list-wise and point-wise ones**
  - Ranking SVM is only one of them (and not the best one)
  - Many others, e.g., RankBoost, Ranknet, GBRank, IR SVM, LambdaRank, LambdaMART
  - The best so far (in terms of effectiveness): LambdaMART
- **Several issues of ranking SVM**
  - Still, it does not directly optimize an evaluation metric
  - But pairwise ranking error often better correlations with evaluation metrics than the loss/objective functions in point-wise approaches
    - Why: evaluation measures only care about rankings!
    - e.g., groundtruth rel(D1) = 2 rel(D2) = 1
      - Regression model 1: pred.rel(D1) = 1 pred.rel(D2) = 2
      - Regression model 2: pred.rel(D1) = 0 pred.rel(D2) = -1
      - Model 1 is better than model 2 by criterion of evaluation regression (the prediction error), but model 2 yields a correct ranking of documents

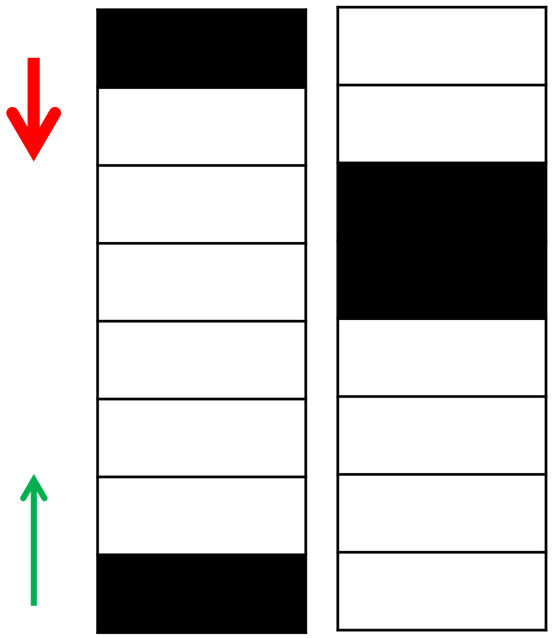
From RankNet to LambdaRank to LambdaMART: An Overview Christopher J.C. Burges, 2010

- Mis-ordered pairs are not equally important
- Depends on how much they contribute to the changes in the target evaluation measure

Mis-ordered pairs: 6

AP: $\frac{5}{8}$

DCG: 1.333



Mis-ordered pairs: 4

AP: $\frac{5}{12}$

DCG: 0.931

From RankNet to LambdaRank to LambdaMART: An Overview Christopher J.C. Burges, 2010

- Weight the mis-ordered pairs?
  - Some pairs are more important to be placed in the right order
  - Inject into object function
    - $\sum_{y_i > y_j} \Omega(d_i, d_j) \exp\left(-w\Delta\Phi\left(q_n, d_i, d_j\right)\right)$
  - Inject into gradient
    - $\lambda_{ij} = \frac{\partial O_{appro}}{\partial w} \Delta O \longleftarrow$ Change in original object, e.g., NDCG, if we switch the documents i and j, leaving the other documents unchanged

Gradient with respect to approximated object, i.e., exponential loss on mis-ordered pairs

Pair-wise Approaches

- **Optimizing for an evaluation metric**
  - The general idea is to weight loss/objective function or gradient with pairwise changes in evaluation measure.
  - e.g., in LambdaMART: lambda gradient

- **Can we optimize all measures?**
  - Not necessarily
  - For some measures, pairwise change do not only relate to the two documents themselves, but also others …
  - Position-based measures do not have the issues (pairwise change only depends on the two documents)
  - Cascade measures may have issues

Experimental Comparisons

- Experiments
  - 1.2k queries, 45.5K documents with 1890 features
  - 800 queries for training, 400 queries for testing

	MAP	P@1	ERR	MRR	NDCG@5
ListNET	*0.2863*	*0.2074*	*0.1661*	*0.3714*	*0.2949*
LambdaMART	**0.4644**	**0.4630**	**0.2654**	**0.6105**	**0.5236**
RankNET	0.3005	0.2222	0.1873	0.3816	0.3386
RankBoost	0.4548	0.4370	0.2463	0.5829	0.4866
RankingSVM	0.3507	0.2370	0.1895	0.4154	0.3585
AdaRank	0.4321	0.4111	0.2307	0.5482	0.4421
pLogistic	0.4519	0.3926	0.2489	0.5535	0.4945
Logistic	0.4348	0.3778	0.2410	0.5526	0.4762

From Hongning Wang's slides

What about list-based learning?

- Need a loss function on a list of documents

- Challenge is scale
    - Huge number of potential lists
- Can develop tricks
    - Consider only possible re-rankings of top N retrieved by some fixed method
- Still expensive

- No clear benefits over pairwise ones (so far)

Microsoft’s LETOR project

- Version 4.0 released in June 2009
- Documents from .GOV collection
  - 25 million pages
- Queries from TREC million query track
  - 2500 queries from 2007 and 2008
  - Relevance information is partial because of corpus size and track methodologies
- 46 extracted features (query, doc, relevance)
  - TF, IDF, etc in page, title, anchors, URL, …
  - BM25 score, LM score, of page, title, anchors, URL, …
  - PageRank, in/out link count, …

[http://research.microsoft.com/en-us/um/beijing/projects/letor//default.aspx]

Microsoft L2R datasets

- Released in June 2009

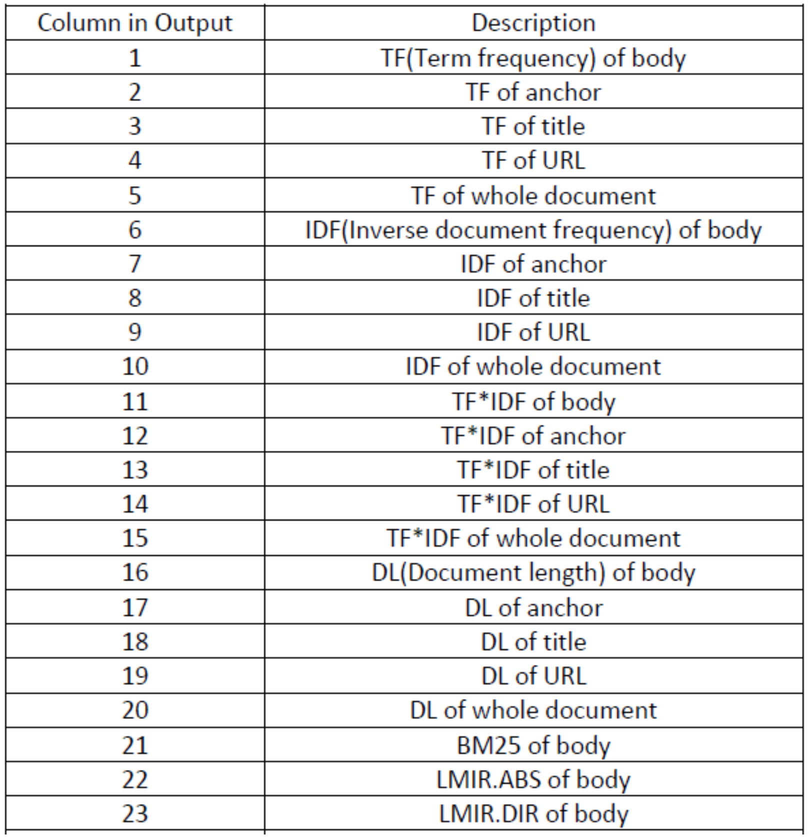
- Web queries (10K and 30K sets)
- Judgments from a “retired” Bing labeling set (0..4)
- Data is rows of query-url values:
  - 0 qid:1 1:3 2:0 3:2 4:2 ... 135:0 136:0
  - 2 qid:1 1:3 2:3 3:0 4:0 ... 135:0 136:0
- 136 features, look like LETOR 4.0 set, but more
  - LETOR 4.0 had 46 features
  - L2R adds more aggregated statistics on TF, IDF, vector comparison, Boolean comparison, …

[http://research.microsoft.com/en-us/projects/mslr/default.aspx]

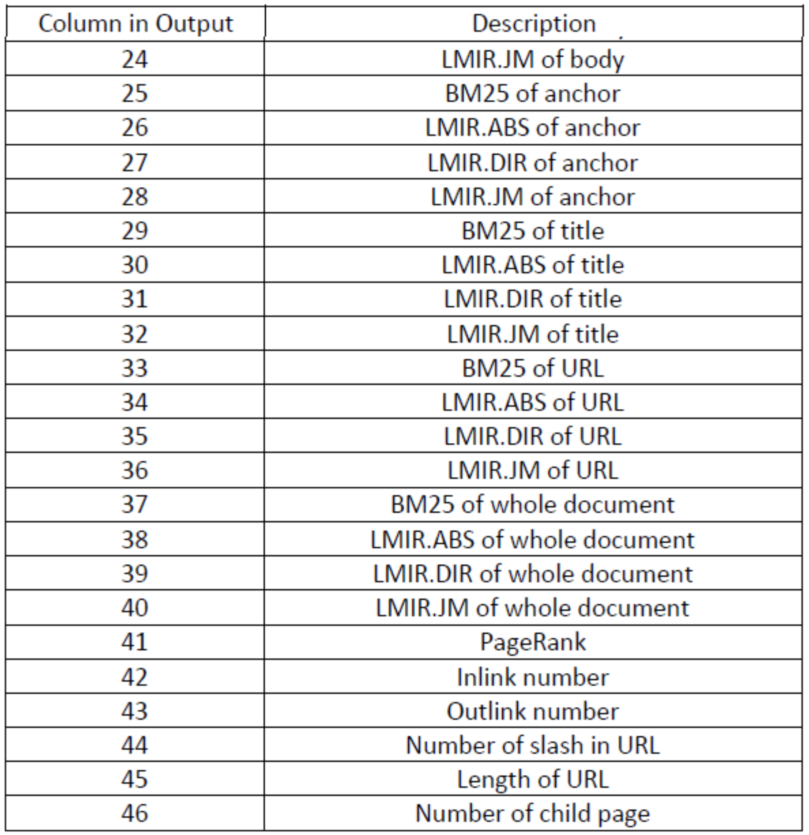
Yahoo! challenge (Mar'10)

- Primary challenge
  - 30K queries (20K train, 3K validation, 7K test)
  - 700K documents (comparable breakdown)
- Queries
  - Randomly sampled from (real) query log
  - Judged on PEGFB scale
- 700 features provided, 415 appear in all sets
  - Normalized to be in 0..1 range
- Privacy
  - Queries, URLs, and features not revealed
    - Microsoft L2R does not provides queries or URLs
  - No idea what semantics of features are

What are the typical features?
Complete list of LETOR 4.0 features



Column in Output	Description
1	TF(Term frequency) of body
2	TF of anchor
3	TF of title
4	TF of URL
5	TF of whole document
6	IDF(Inverse document frequency) of body
7	IDF of anchor
8	IDF of title
9	IDF of URL
10	IDF of whole document
11	TF*IDF of body
12	TF*IDF of anchor
13	TF*IDF of title
14	TF*IDF of URL
15	TF*IDF of whole document
16	DL(Document length) of body
17	DL of anchor
18	DL of title
19	DL of URL
20	DL of whole document
21	BM25 of body
22	LMIR.ABS of body
23	LMIR.DIR of body



Column in Output	Description
24	LMIR.JM of body
25	BM25 of anchor
26	LMIR.ABS of anchor
27	LMIR.DIR of anchor
28	LMIR.JM of anchor
29	BM25 of title
30	LMIR.ABS of title
31	LMIR.DIR of title
32	LMIR.JM of title
33	BM25 of URL
34	LMIR.ABS of URL
35	LMIR.DIR of URL
36	LMIR.JM of URL
37	BM25 of whole document
38	LMIR.ABS of whole document
39	LMIR.DIR of whole document
40	LMIR.JM of whole document
41	PageRank
42	Inlink number
43	Outlink number
44	Number of slash in URL
45	Length of URL
46	Number of child page

How to use?

- **Develop feature sets is the most important step!**
  - Usually problem dependent
  - e.g., add contextual features if you hope to address contextual factors in search
- **Choose a good ranking model**
  - LambdaMART seems the best choice so far
  - RankLib implemented LambdaMART and many others
- **Training, validation, and testing**
  - similar to standard machine learning applications

Outline

- ~~Learning-to-rank~~
- ***<u>RankLib Tutorial</u>***
  - https://sourceforge.net/projects/lemur/
- Midterm Q & A

Outline

- ~~Learning-to-rank~~
- ~~RankLib Tutorial~~
- ***Midterm Q & A***