Machine Learning for IR

CISC489/689-010, Lecture #22

Wednesday, May 6th

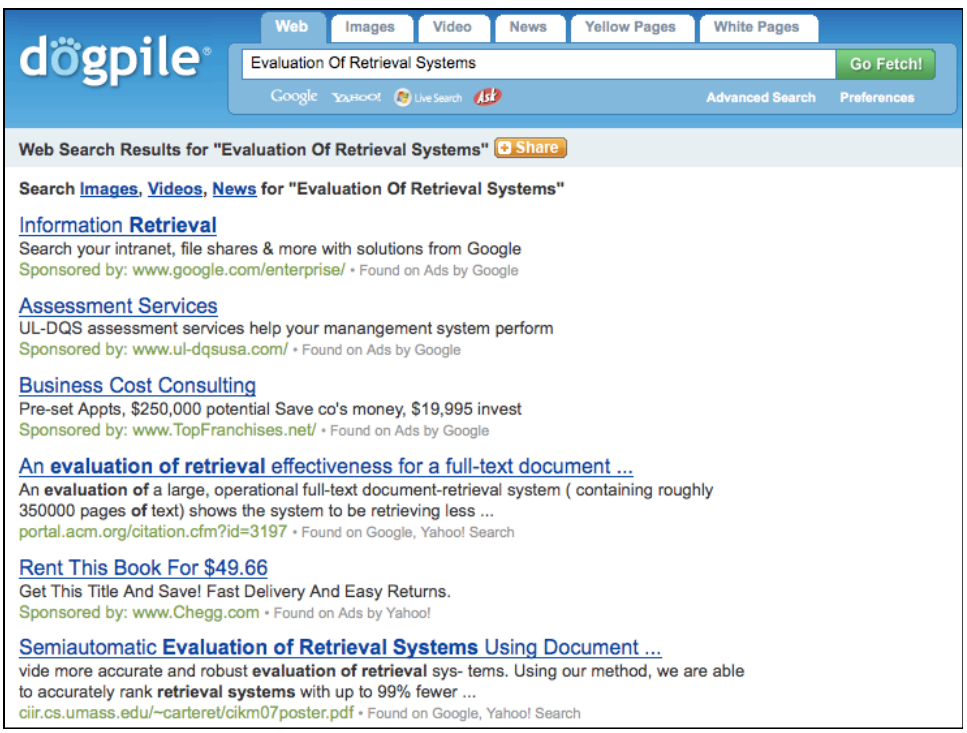
Ben Carterette

Learning to Rank

- Monday:
  - Machine learning for classification
  - Generative vs discriminative models
  - SVMs for classification
- Today:
  - Machine learning for ranking
  - RankSVM, RankNet, RankBoost
  - But first, a bit of metasearch

Metasearch

- Different search engines have different strengths
- Some may find relevant documents that others miss
- Idea: merge results from multiple engines into a single final ranking



Score Combination

- Each system provides a score for each document
- We can combine the scores to obtain a single score for each document
  - If many systems are giving a document a high score, then maybe that document is much more likely to be relevant
  - If many systems are giving a document a low score, maybe that document is much less likely to be relevant
  - What about some systems giving high scores and some giving low scores?

Score Combination Methods

- There are many different ways to combine scores
  - CombMIN: minimum of document scores
  - CombMAX: maximum of document scores
  - CombMED: median of document scores
  - CombSUM: sum of document scores
  - CombANZ: CombSUM / (# scores not zero)
  - CombMNZ: CombSUM * (# scores not zero)
- "Analysis of Multiple Evidence Combination", Lee

Voting Algorithms

- In voting combination, each system is considered a voter providing a "ballot" of relevant document candidates
- The ballots need to be tallied to produce a final ranking of candidates
- Two primary methods:
  - Borda count
  - Condorcet method

Borda Count

- Each voter provides a ranked list of candidates
- Assign each rank a certain number of points
  - Highest rank gets maximum points, lowest rank minimum
- The Borda count of a candidate is the sum of its assigned points over all the voters
- Rank candidates in decreasing order of Borda count

Borda Counts

- Typically, if there are N candidates, the top-ranked candidate will get N points.
  - Second-ranked gets N-1
  - Third-ranked gets N-2
  - Etc
- A document ranked first by all m systems will have a Borda count of mN
- A document ranked last by just one system will have a Borda count of 1

Condorcet Method

- In the Condorcet method, N candidates compete in pairwise preference elections
  - Voter 1 gives a preference on candidate A versus B
  - Voter 2 gives a preference on candidate A versus B
  - etc
  - Then the voters give a preference on A versus C, and so on
- $O(mN^2)$ total preferences

Condorcet Method

- After getting all voter preferences, we add up the number of times each candidate won
- The candidates are then ranked in decreasing order of the number of preferences they won
- In IR, we have a ranking of documents (candidates)
- Decompose ranking into pairwise preferences, then add up preferences over systems

Borda versus Condorcet Example

- Engine 1: A, B, C, D
- Engine 2: A, B, C, E
- Engine 3: A, B, C, F
- Engine 4: B, C, A, D
- Engine 5: B, C, A, F

- Borda counts:
  - A: 6+6+6+4+4 = 26
  - B: 5+5+5+6+6 = 27
  - C: 4+4+4+5+5 = 22
  - D: 3+1.5+1.5+3+1.5 = 10.5
  - E: 1.5+3+1.5+1.5+1.5 = 9
  - F: 1.5+1.5+3+1.5+3 = 10.5
- Condorcet counts:
  - A: 21 wins
  - B: 22 wins
  - C: 17 wins
  - D: 4 wins
  - E: 2 wins
  - F: 4 wins

Metasearch vs Learning to Rank

- Metasearch is not really "learning"
  - It is trusting the input systems to do a good job
- Learning uses some queries and documents along with human labels to learn a general ranking function
- Currently learning approaches are a bit like metasearch with training data
  - Learn how to combine features in order to rerank a provided set of documents

Learning to Rank

- Three approaches:
  - Classification-based
    - Classify documents as relevant or not relevant
    - Rank in decreasing order of classification prediction
  - Preference-based
    - Similar to Condorcet voting algorithm
    - Decompose ranking into preferences
    - Learn preference functions on pairs
  - List-based
    - Full-ranking based
    - Very complicated and highly mathematical

Classification-Based

- Use SVM to classify documents as relevant or not relevant
  - Recall that the SVM provides feature weights **w**
  - Classification function is $f(x) = \text{sign}(\mathbf{w}'\mathbf{x} + b)$
- To turn this into a ranker, just drop the sign function
  - $S(Q, D) = f(\mathbf{x}) = \mathbf{w}'\mathbf{x} + b$
  - (**x** is the feature vector for document D)
- First we have to train a classifier
- What are the features?

Features for Discriminative Models

- Recall SVM is a discriminative classifier
- All the probabilistic models we previously discussed were generative
- With generative models we could just use terms as features
- With discriminative models we cannot
  - Why not?
  - Terms that are related to relevance for one query are not necessarily related to relevance for another

SVM Features

- Instead, use features derived from term features
- LM score, BM25 score, tf-idf score, …
- This is pretty much like score-combination metasearch
  - Only differences:
  - There is training data
  - We use SVM to learn averaging weights instead of just doing a straight average/max/min/etc

RankSVM

- RankSVM idea: learn from preferences between documents
  - Like Condorcet method, but with training data
- Training data: pairs of documents d_i, d_j with a preference relation y_{ijq} for query q
  - E.g. doc A preferred to doc B for query q: d_i = A, d_j = B, y_{ijq} = 1

RankSVM

- Standard SVM optimization problem:

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\mathbf{w}'\mathbf{w} + C\sum \zeta_i$$
$$\text{s.t. } y_i(\mathbf{w}'x_i + b) \geq 1 - \zeta_i$$

- RankSVM optimization problem:

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\mathbf{w}'\mathbf{w} + C\sum_{i,j,q} \zeta_{ijq}$$
$$\text{s.t. } y_{ijq}(\mathbf{w}'(d_i - d_j) + b) \geq 1 - \zeta_{ijq}$$

RankSVM Training Data

- Where do the preference relations come from?
  - Relevance judgments:
    - if A is relevant and B is not, then A is preferred to B
    - If A is highly relevant and B is moderately relevant, then A is preferred to B
  - Clicks:
    - If users consistently click on the document at rank 3 instead of documents at ranks 1 and 2, infer that the document at rank 3 is preferred to those at ranks 1 and 2

RankNet

- Like RankSVM, use preferences between documents
- Unlike RankSVM, use the *magnitude* of the preference
  - If A is highly relevant, B is moderately relevant, and C is only slightly relevant, then A is preferred to B and C, and B is preferred to C
  - But the magnitude of the preference of A over C is greater than the magnitude of the preference of A over B

RankNet

- Instead of becoming a classification problem like RankSVM, ranking becomes a regression problem
  - y_{ijq} is a real number
- We can apply standard regression models
- Neural net (nonlinear regression) is an obvious choice and can be trained using gradient descent

RankBoost

- Boosting-based preference learner
- First learn a weak ranker, weighting all pairs equally
- Then find the pairs that ranker put in correct order and decrease their weights; find the pairs the ranker put in the wrong order and increase their weights
- Iterate until convergence (or T times)
- The final ranker combines all T weak rankers into a single ranking

Comparing L2R Methods

- Data: LETOR (LEarning TO Rank) assembled by Microsoft Research Asia
- Two subsets:
  - OHSUMED (biomedical abstracts)
    - 350,000 abstracts from 270 medical journals from 1981-1991
    - 106 queries
    - 16,000 judgments on three-level scale: definitely, partially, and not relevant
  - .GOV (web pages in .gov domain)
    - 1 million web pages with 11 million links
    - 125 queries
    - Relevance judgments include all relevant documents plus top 1000 top-BM25 scoring documents

LETOR Features

Feature	Formulations	Descriptions
L1	$\sum_{q_i \in q \cap d} c(q_i, d)$	Term frequency (*tf*)
L2	$\sum_{q_i \in q \cap d} \log(c(q_i, d) + 1)$	SIGIR feature
L3	$\sum_{q_i \in q \cap d} \frac{c(q_i,d)}{\lvert d \rvert}$	Normalized *tf*
L4	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i,d)}{\lvert d \rvert} + 1\right)$	SIGIR feature
L5	$\sum_{q_i \in q \cap d} \log\left(\frac{\lvert C \rvert}{df(q_i)}\right)$	Inverse doc frequency (*idf*)
L6	$\sum_{q_i \in q \cap d} \log\left(\log\left(\frac{\lvert C \rvert}{df(q_i)}\right)\right)$	SIGIR feature
L7	$\sum_{q_i \in q \cap d} \log\left(\frac{\lvert C \rvert}{c(q_i,C)} + 1\right)$	SIGIR feature
L8	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i,d)}{\lvert d \rvert} \log\left(\frac{\lvert C \rvert}{df(q_i)}\right) + 1\right)$	SIGIR feature
L9	$\sum_{q_i \in q \cap d} c(q_i, d) \log\left(\frac{\lvert C \rvert}{df(q_i)}\right)$	*tf* idf*
L10	$\sum_{q_i \in q \cap d} \log\left(\frac{c(q_i,d)}{\lvert d \rvert} \frac{\lvert C \rvert}{c(q_i,C)} + 1\right)$	SIGIR feature

Feature	Descriptions
H1	BM25 score
H2	log(BM25 score)
H3	LMIR with DIR smoothing
H4	LMIR with JM smoothing
H5	LMIR with ABS smoothing

- OHSUMED features:
  - 10 "low-level" features based on tf, idf, collection size, etc
  - 5 "high-level" features that are standard IR scoring functions
  - 15 features for each of title and abstract, for 30 total features for each OHSEUMED doc

LETOR Features

Feature	Descriptions
1	BM25
2	document length (dl) of body
3	dl of anchor
4	dl of title
5	dl of URL
6	HITS authority
7	HITS hub
8	HostRank (SIGIR feature)
9	Inverse document frequency (idf) of body
10	idf of anchor
11	idf of title
12	idf of URL
13	Sitemap based score propagation (SIGIR feature)
14	PageRank
15	LMIR.ABS of anchor
16	BM25 of anchor
17	LMIR.DIR of anchor
18	LMIR.JM of anchor
19	LMIR.ABS of extracted title (SIGIR feature)
20	BM25 of extracted title (SIGIR feature)
21	LMIR.DIR of extracted title (SIGIR feature)
22	LMIR.JM of extracted title (SIGIR feature)
23	LMIR.ABS of title
24	BM25 of title
25	LMIR.DIR of title
26	LMIR.JM of title
27	Sitemap based feature propagation (SIGIR feature)

- .GOV features

28	tf of body
29	tf of anchor
30	tf of title
31	tf of URL
32	tf*idf of body
33	tf*idf of anchor
34	tf*idf of title
35	tf*idf of URL
36	Topical PageRank (SIGIR feature)
37	Topical HITS authority (SIGIR feature)
38	Topical HITS hub (SIGIR feature)
39	Hyperlink base score propagation: weighted in-link (SIGIR feature)
40	Hyperlink base score propagation: weighted out-link (SIGIR feature)
41	Hyperlink base score propagation: uniform out-link (SIGIR feature)
42	Hyperlink base feature propagation: weighted in-link (SIGIR feature)
43	Hyperlink base feature propagation: weighted out-link (SIGIR feature)
44	Hyperlink base feature propagation: uniform out-link (SIGIR feature)

Three Sub-Collections

- OHSUMED with 106 queries
- .GOV/TD2003 with 50 queries
- .GOV/TD2004 with 75 queries
- Each collection broken out into 5 folds for cross-validation

Folds	Training set	Validation set	Test set
Fold1	{S1, S2, S3}	S4	S5
Fold2	{S2, S3, S4}	S5	S1
Fold3	{S3, S4, S5}	S1	S2
Fold4	{S4, S5, S1}	S2	S3
Fold5	{S5, S1, S2}	S3	S4

Evaluation Measures

- Precision at rank n
- Mean average precision
- NDCG (normalized DCG)

- Comparing RankSVM and RankBoost (RankNet results not included)

Results: OHSUMED

(a) Precision at position *n*

Algorithms	P@1	P@2	P@3	P@4	P@5
RankBoost	0.605	0.595	0.586	0.562	0.545
Ranking SVM	0.634	0.619	0.592	0.579	0.577

Algorithms	P@6	P@7	P@8	P@9	P@10
RankBoost	0.525	0.516	0.505	0.494	0.495
Ranking SVM	0.558	0.536	0.525	0.517	0.507

(b) Mean average precision

Algorithms	MAP
RankBoost	0.440
Ranking SVM	0.447

(c) NDCG at position *n*

Algorithms	NDCG@1	NDCG@2	NDCG@3	NDCG@4	NDCG@5
RankBoost	0.498	0.483	0.473	0.461	0.450
Ranking SVM	0.495	0.476	0.465	0.459	0.458

Algorithms	NDCG@6	NDCG@7	NDCG@8	NDCG@9	NDCG@10
RankBoost	0.442	0.439	0.436	0.433	0.436
Ranking SVM	0.455	0.447	0.445	0.443	0.441

- BM25 alone gives:
  - P@1 = 0.519
  - NDCG@1 = 0.399
  - MAP = 0.425

Results: TD2003 and TD2004

(a) Precision at position *n*

Algorithms	P@1	P@2	P@3	P@4	P@5
RankBoost	0.260	0.270	0.240	0.230	0.220
Ranking SVM	0.420	0.350	0.340	0.300	0.264

Algorithms	P@6	P@7	P@8	P@9	P@10
RankBoost	0.210	0.211	0.193	0.182	0.178
Ranking SVM	0.243	0.234	0.233	0.218	0.206

(b) Mean average precision

Algorithms	MAP
RankBoost	0.212
Ranking SVM	0.256

(c) NDCG at position *n*

Algorithms	NDCG@1	NDCG@2	NDCG@3	NDCG@4	NDCG@5
RankBoost	0.260	0.280	0.270	0.272	0.279
Ranking SVM	0.420	0.370	0.379	0.363	0.347

Algorithms	NDCG@6	NDCG@7	NDCG@8	NDCG@9	NDCG@10
RankBoost	0.280	0.287	0.282	0.282	0.285
Ranking SVM	0.341	0.340	0.345	0.342	0.341

(a) Precision at position *n*

Algorithms	P@1	P@2	P@3	P@4	P@5
RankBoost	0.480	0.447	0.404	0.347	0.323
Ranking SVM	0.440	0.407	0.351	0.327	0.291

Algorithms	P@6	P@7	P@8	P@9	P@10
RankBoost	0.304	0.293	0.277	0.262	0.253
Ranking SVM	0.273	0.261	0.247	0.236	0.225

(b) Mean average precision

Algorithms	MAP
RankBoost	0.383514
Ranking SVM	0.350459

(c) NDCG at position *n*

Algorithms	NDCG@1	NDCG@2	NDCG@3	NDCG@4	NDCG@5
RankBoost	0.480	0.473	0.464	0.439	0.437
Ranking SVM	0.440	0.433	0.409	0.406	0.393

Algorithms	NDCG@6	NDCG@7	NDCG@8	NDCG@9	NDCG@10
RankBoost	0.448	0.457	0.461	0.464	0.472
Ranking SVM	0.397	0.406	0.410	0.414	0.420

TD2004: L2R vs TREC

- TREC used the TD2004 subcollection for the Web track in 2004
- Comparing results of TREC runs to L2R methods:
  - MAP: RankBoost 0.384; best TREC system 0.179
  - P10: RankBoost 0.253; best TREC system 0.347
- TREC systems had to rank the entire collection; L2R methods only ranked a small subset