

Introduction to Information Retrieval

CS276: Information Retrieval and Web Search
Pandu Nayak and Prabhakar Raghavan

Lecture 19: Learning to Rank

Introduction to Information Retrieval

Final exam

- Tue Jun 7th from 12:15-3:15pm; Gates B01, B03
 - Alternate final Friday Jun 3rd 12:15-3:15pm; Gates 260
- Open book/notes
- Devices ok provided they're not connected to other devices, or the internet
 - Calculator recommended

Introduction to Information Retrieval Sec. 15.4

Machine learning for IR ranking?

- We've looked at methods for ranking documents in IR
 - Cosine similarity, inverse document frequency, pivoted document length normalization, Pagerank, ...
 - How do we combine these signals into a good ranker?
- 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 figure out how to combine scoring/ranking signals?
 - A.k.a. "machine-learned relevance" or "learning to rank"

Introduction to Information Retrieval

Machine learning for IR ranking

- This "good idea" has been actively researched – and actively deployed by the 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...
 - Naïve Bayes has been around for about 45 years...

Introduction to Information Retrieval

Why weren't early attempts very successful/influential?

- 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
- The Web provided impetus with constantly evolving spam

Introduction to Information Retrieval

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

Introduction to Information Retrieval Sec. 6.1.2

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?
- Major web search engines publicly state that they use “hundreds” of such features – and they keep changing

Introduction to Information Retrieval Sec. 6.1.2

Simple example

- Consider the presence of query terms in the Title (T) and the Body (B) of a document
 - Boolean indicator (0/1) of whether the query term occurs in the Title (s_T) or Body (s_B)
- We'll compute a score in $[0,1]$ for each doc d and for each query q using a linear combination of s_T and s_B

$$\text{score}(d, q) = g \cdot s_T(d, q) + (1 - g) s_B(d, q).$$

Thus our scores are all 0, g , $1-g$ or 1.

- g is a parameter to be learned from examples

Introduction to Information Retrieval Sec. 6.1.2

We are given examples

- Created by human judges

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Non-relevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Non-relevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3191	driver	1	0	Non-relevant

- We *quantize* the human relevance judgments to be 1 or 0 respectively, for Relevant and Non-relevant
 - The scores we compute will be 0, g , $1-g$ or 1 – how do we tell how good our scoring function is?

Introduction to Information Retrieval Sec. 6.1.2

Least square errors

- For each human-judged example, we compute its squared error:

$$\text{score}(d_j, q_j) = g \cdot s_T(d_j, q_j) + (1 - g) s_B(d_j, q_j)$$
- Then we can compute a total error of

$$\varepsilon(g, \Phi_j) = (r(d_j, q_j) - \text{score}(d_j, q_j))^2$$
- We will pick g to minimize this total error.

Introduction to Information Retrieval Sec. 6.1.2

Choosing g

- In our simple setting, all that matters is the *number* of examples* of each equivalence class
- Define:
 - n_{01r} = # examples with $s_T=0, s_B=1$, judgment = Rel
 - n_{01n} = # examples with $s_T=0, s_B=1$, judgment = NonRel
 - n_{10r} = # examples with $s_T=1, s_B=0$, judgment = Rel
 - n_{10n} = # examples with $s_T=1, s_B=0$, judgment = NonRel
 - (and similarly $n_{00r}, n_{00n}, n_{11r}$, and n_{11n} corresponding to the 4 other equivalence classes)

* this may not hold for other sets of features, e.g., the # characters in the query

Introduction to Information Retrieval Sec. 6.1.2

Choosing g

- The n_{01r} examples with $s_T=0, s_B=1$ combined contribute a total least-squared error of

$$[1 - (1 - g)]^2 n_{01r} + [0 - (1 - g)]^2 n_{01n}$$
- Similarly, add up the error contributions of the other 3 combinations of s_T and s_B for a total error of

$$(n_{01r} + n_{10n})g^2 + (n_{10r} + n_{01n})(1 - g)^2 + n_{00r} + n_{11n}$$

Introduction to Information Retrieval

Choosing g is now elementary calculus

- Differentiating the total error wrt g we get the optimal value for g to be

$$\frac{n_{10r} + n_{01n}}{n_{10r} + n_{10n} + n_{01r} + n_{01n}}$$

Introduction to Information Retrieval

Generalizing this simple example

- More (than 2) features
- Non-Boolean features
 - What if the title contains some but not all query terms ...
 - Categorical features (query terms occur in plain, boldface, italics, etc)
- Scores are nonlinear combinations of features
- Multilevel relevance judgments (Perfect, Good, Fair, Bad, etc)
- Complex error functions
- Not always a unique, easily computable setting of score parameters

Introduction to Information Retrieval Sec. 15.4.1

A richer example

- Collect a training corpus of (q, d, r) triples
 - Relevance r is still binary for now
 - Document is represented by a feature vector
 - $\mathbf{x} = (\alpha, \omega)$ α is cosine similarity, ω is minimum query window size
 - ω is the the shortest text span that includes all query words
 - Query term proximity
- Train a machine learning model to predict the class r

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

Introduction to Information Retrieval Sec. 15.4.1

Using classification for deciding relevance

- A linear score function is

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

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

Introduction to Information Retrieval Sec. 15.4.1

Using classification for deciding relevance

Introduction to Information Retrieval

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 relevance [Nallapati 2004]

- Let $g(r|d,q) = \mathbf{w} \cdot \mathbf{f}(d,q) + b$
- 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 $g(r|d,q) \geq 0$

An SVM classifier for relevance

- 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)

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 (LM)
 - 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

"Learning to rank"

- Classification probably isn't the right way to think about score learning:
 - 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: $c_1 < c_2 < \dots < c_j$
 - 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

Introduction to Information Retrieval Sec. 15.4.1

Modified example

- Collect a training corpus of (q, d, r) triples
 - Relevance r is here 4 values
 - Perfect, Relevant, Weak, Nonrelevant
- 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	Perfect
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	Weak
Φ_5	1741	kernel layer	0.022	3	Relevant
Φ_6	2094	device driver	0.03	2	Perfect
Φ_7	3191	device driver	0.027	5	Nonrelevant

Introduction to Information Retrieval

“Learning to rank”

- 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

Introduction to Information Retrieval Sec. 15.4.2

Point-wise learning

- Goal is to learn a threshold to separate each rank

Introduction to Information Retrieval Sec. 15.4.2

The Ranking SVM

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

- 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

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

Introduction to Information Retrieval Sec. 15.4.2

The Ranking SVM

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

- Ranking Model: $f(\psi_i)$

Introduction to Information Retrieval

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

Introduction to Information Retrieval

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 at least as important to do well on

Introduction to Information Retrieval

IR Evaluation Measures

- Some evaluation measures strongly weight doing well in highest ranked results:
 - MAP (Mean Average Precision)
 - NDCG (Normalized Discounted Cumulative Gain)
- NDCG has been especially popular in machine learned relevance research
 - It handles multiple levels of relevance (MAP doesn't)
 - It seems to have the right kinds of properties in how it scores system rankings

Introduction to Information Retrieval Sec 8.4

Normalized Discounted Cumulative Gain (NDCG) evaluation measure

- Query: q_i
- DCG at position m : $DCG_m = \sum_{j=1}^m (2^{r_j} - 1) / \log(1 + j)$
- NDCG at position m : average over queries
- Example
 - (3, 3, 2, 2, 1, 1, 1) r Quantized relevance by position
 - (7, 7, 3, 3, 1, 1, 1) Gain by position $2^{r_j} - 1$
 - (1, 0.63, 0.5, 0.43, 0.39, 0.36, 0.33) discount $1 / \log(1 + j)$
 - (7, 11.41, 12.91, 14.2, 14.59, 14.95, 15.28) $\sum_{j=1}^m (2^{r_j} - 1) / \log(1 + j)$
- Z_i normalizes against best possible result for query, the above, versus lower scores for other rankings
 - Necessarily: High ranking number is good (more relevant)

Introduction to Information Retrieval

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

Introduction to Information Retrieval

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

Introduction to Information Retrieval

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	0.9	0.6	0.3
Rank	1	2	3
Relevance	0	1	0

NDCG discontinuous w.r.t model parameters!

Introduction to Information Retrieval

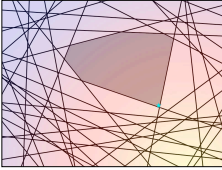
Structural SVMs [Tsochantaridis et al., 2007]

- 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.,
 $y = \begin{matrix} \text{green} & \text{green} & \text{red} & \text{red} & \text{red} & \text{red} \end{matrix}$
- An **incorrect labeling** would be any other ranking, e.g.,
 $y = \begin{matrix} \text{red} & \text{green} & \text{green} & \text{red} & \text{red} & \text{red} \end{matrix}$
- There are an **intractable number of rankings**, thus an **intractable number of constraints!**

Introduction to Information Retrieval

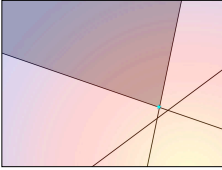
Structural SVM training [Tsochantaridis et al., 2007]

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

Introduction to Information Retrieval

Other machine learning methods for learning to rank

- We'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

Introduction to Information Retrieval

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 nonlinear scalings
 - Designing the basic features that give good signals for ranking remains the domain of human creativity

Introduction to Information Retrieval

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.

Introduction to Information Retrieval

Final exam

- Tue Jun 7th from 12:15-3:15pm; Gates B01, B03
 - Alternate final Friday Jun 3rd 12:15-3:15pm; Gates 260
- Open book/notes
- Devices ok provided they're not connected to other devices, or the internet
 - Calculator recommended

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*.