Search Engine Architecture

10. Learning to Rank

Machine learning for IR ranking?

- We've looked at methods for ranking documents in IR
  - Cosine similarity, inverse document frequency, Pagerank, …
- We've looked at methods for classifying documents using supervised machine learning classifiers
  - Logistic regression, ANNs, etc.
- Surely we can also use *machine learning* to rank the documents displayed in search results?
  - Sounds like a good idea
  - A.k.a. "machine-learned relevance" or "learning to rank"

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…

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

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 Boundaries for Ordinal Regression. *Advances in Large Margin Classifiers.*

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

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

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

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
  - (You guys did in assignment 2)

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

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.

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

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
      - 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

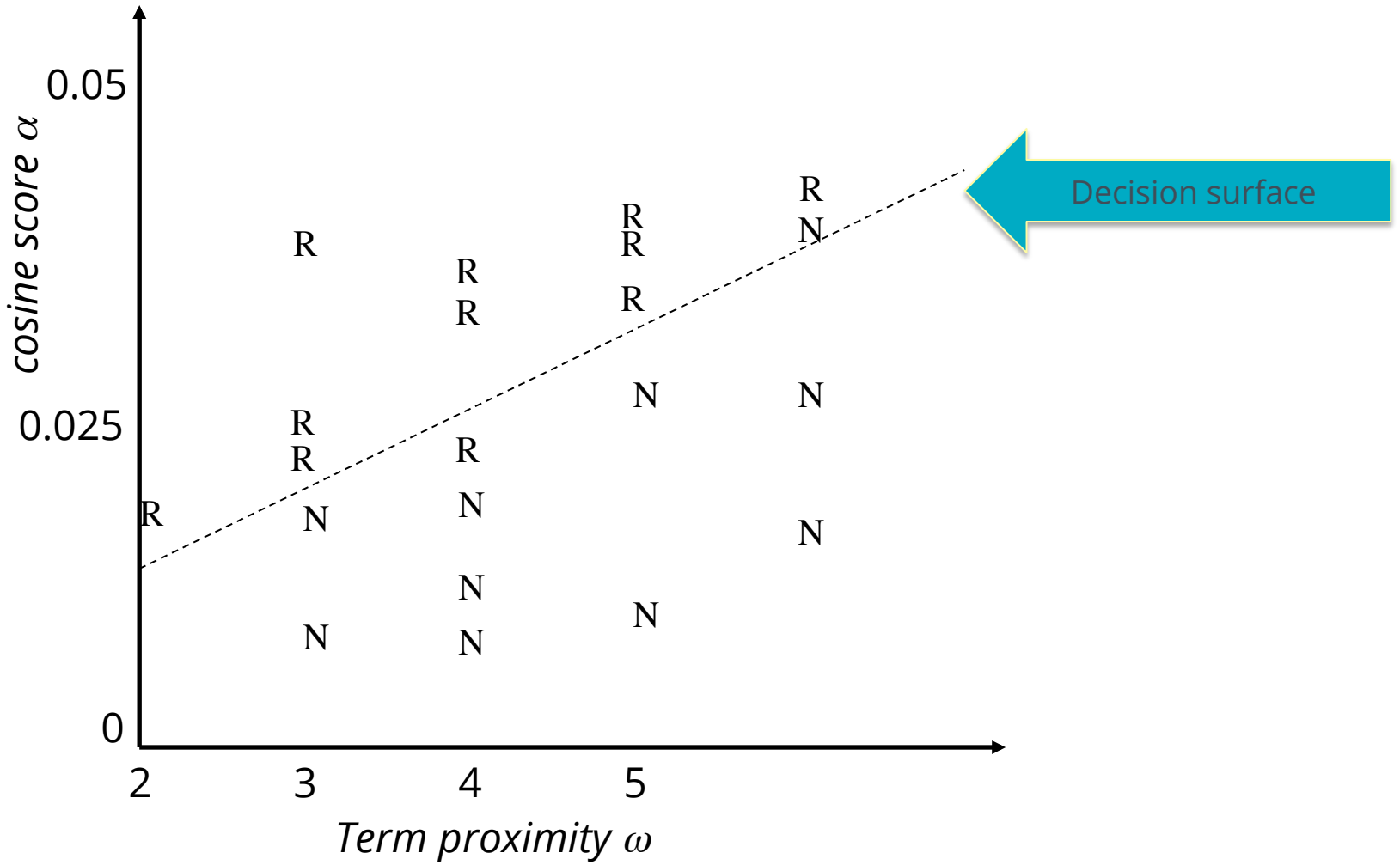
example	docID	query	cosine score	ω	judgment
Φ_1	37	linux operating system	0.032	3	*relevant*
Φ_2	37	penguin logo	0.02	4	*nonrelevant*
Φ_3	238	operating system	0.043	2	*relevant*
Φ_4	238	runtime environment	0.004	2	*nonrelevant*
Φ_5	1741	kernel layer	0.022	3	*relevant*
Φ_6	2094	device driver	0.03	2	*relevant*
Φ_7	3191	device driver	0.027	5	*nonrelevant*

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

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 $Score(d, q) > \theta$
- … just like text classification

Using Classification for Ad Hoc IR



"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 goodness

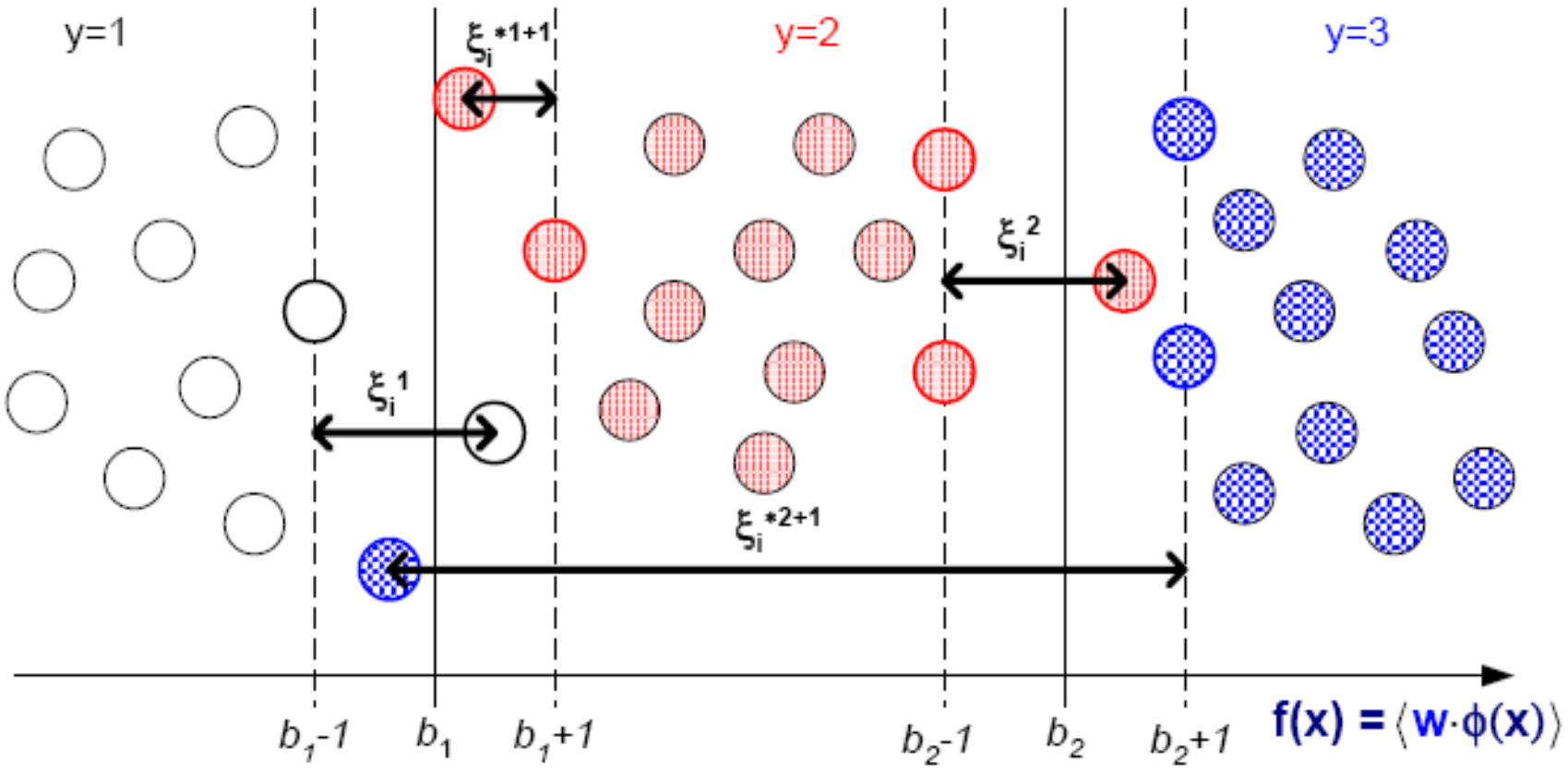
"Learning to Rank"

- Assume a number of categories ***C*** of relevance exist
  - These are totally ordered: $c_1 < c_2 < \ldots < c_J$
  - 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 the class is the relevance ordering relationship between them

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

Point-wise Learning

- Goal is to learn a threshold to separate each rank



Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

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

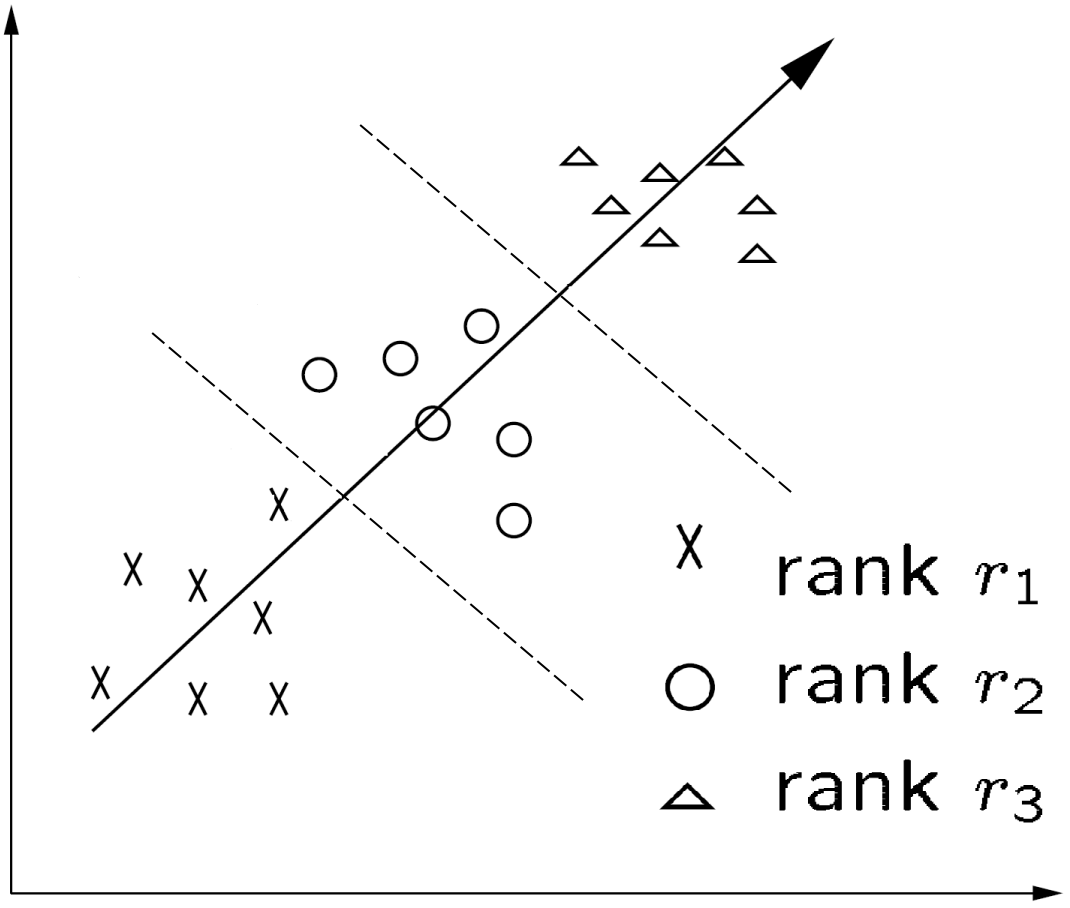
$$f(\psi_i) = \mathbf{w} \bullet \psi_i$$

Source: Herbrich et al. 1999, 2000; Joachims et al. 2002 via Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

Sec. 15.4.2

The Ranking SVM

- Ranking Model: $f(\psi_i)$



Source: Herbrich et al. 1999, 2000; Joachims et al. 2002 via Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

The Ranking SVM

- Then:

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

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

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

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

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

Source: Herbrich et al. 1999, 2000; Joachims et al. 2002 via Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

Modeling Problems

- 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
- 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

Source: Cao et al. SIGIR 2006 via Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

Two Problems with Ranking SVM

Cao et al. SIGIR 2006

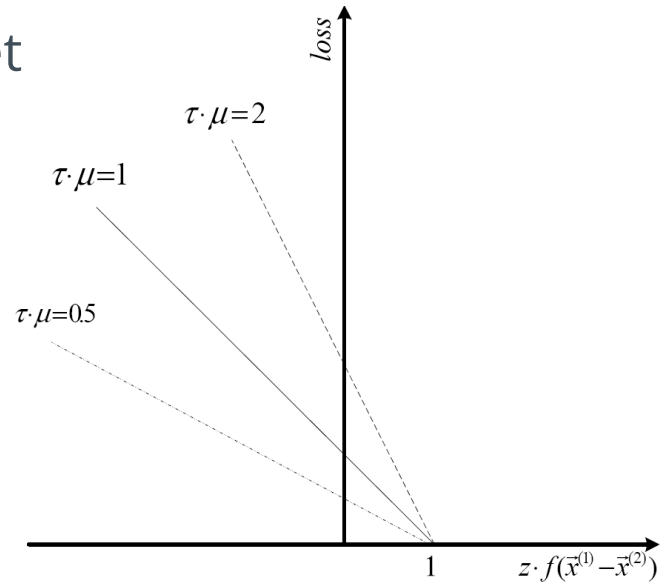
- Cost sensitivity: negative effects of making errors on top ranked documents
  - d: *definitely relevant*, p: *partially relevant*, n: *not relevant*
  - ranking 1: p d p n n n n
  - ranking 2: d p n p n n n
- Query normalization: number of instance pairs varies according to query
  - q1: d p p n n 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

Source: Cao et al. SIGIR 2006 via Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

New Loss Function

$$\min_{\vec{w}} L(\vec{w}) = \sum_{i=1}^{l} \tau_{k(i)} \mu_{q(i)} \left[1 - z_i \left\langle \vec{w}, \vec{x}_i^{(1)} - \vec{x}_i^{(2)} \right\rangle \right]_+ + \lambda \left\| \vec{w} \right\|^2$$

- 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



Source: Cao et al. SIGIR 2006 via Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

Optimizing Rank-Based Measures

- 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

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

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

NDCG discontinuous w.r.t model parameters!

	d_1	d_2	d_3
Retrieval Score	0.9	0.6	0.3
Rank	1	2	3
Relevance	0	1	0

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

RankNet to LambdaRank

- RankNet
  - ANN two-class classifier
- LambdaRank
  - Key observation – only need gradient, not cost
  - Gradient descent speedup – instead of computing gradient on every pairwise sample, gradient can be efficiently summed across an entire query before updating weights
  - Better objective function – scale gradients by change in NDCG due to inverting pair

Source: Burges. (2010) "From RankNet to LambdaRank to LambdaMART: An Overview." *Microsoft Research Technical Report.*

LambdaMART

- Multiple Additive Regression Trees
- Instead of updating weights (as in ANN), add a new tree where target variables are “pseudo residuals”
- Pseudo-residual defined as gradient of cost function
  - You can see where this is going…
- Lots of advantages for modeling
  - Easily handles both discrete and continuous features

Source: Burges. (2010) “From RankNet to LambdaRank to LambdaMART: An Overview.” *Microsoft Research Technical Report.*

Limitations 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

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
- 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!)

Source: Manning et al. CS276 - Information Retrieval and Web Search, Stanford Spring 2016

Questions?