Introduction to

Information Retrieval

Learning to Rank

Slides borrowed from Stanford

Machine learning for IR ranking?

- We've looked at methods for ranking documents in IR
  - Cosine similarity, inverse document frequency, BM25, proximity, pivoted document length normalization, (will look at) Pagerank, …
- We've looked at methods for classifying documents using supervised machine learning classifiers
  - Rocchio, kNN, 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 10 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 Boundaries for Ordinal Regression. *Advances in Large Margin Classifiers.*

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~~ possible 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 loading speed
- The *New York Times* in 2008-06-03 quoted Amit Singhal as saying Google was using over 200 such features ("signals")

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
      - Query term proximity is an **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: Using classification for ad hoc IR

- A linear score function is then

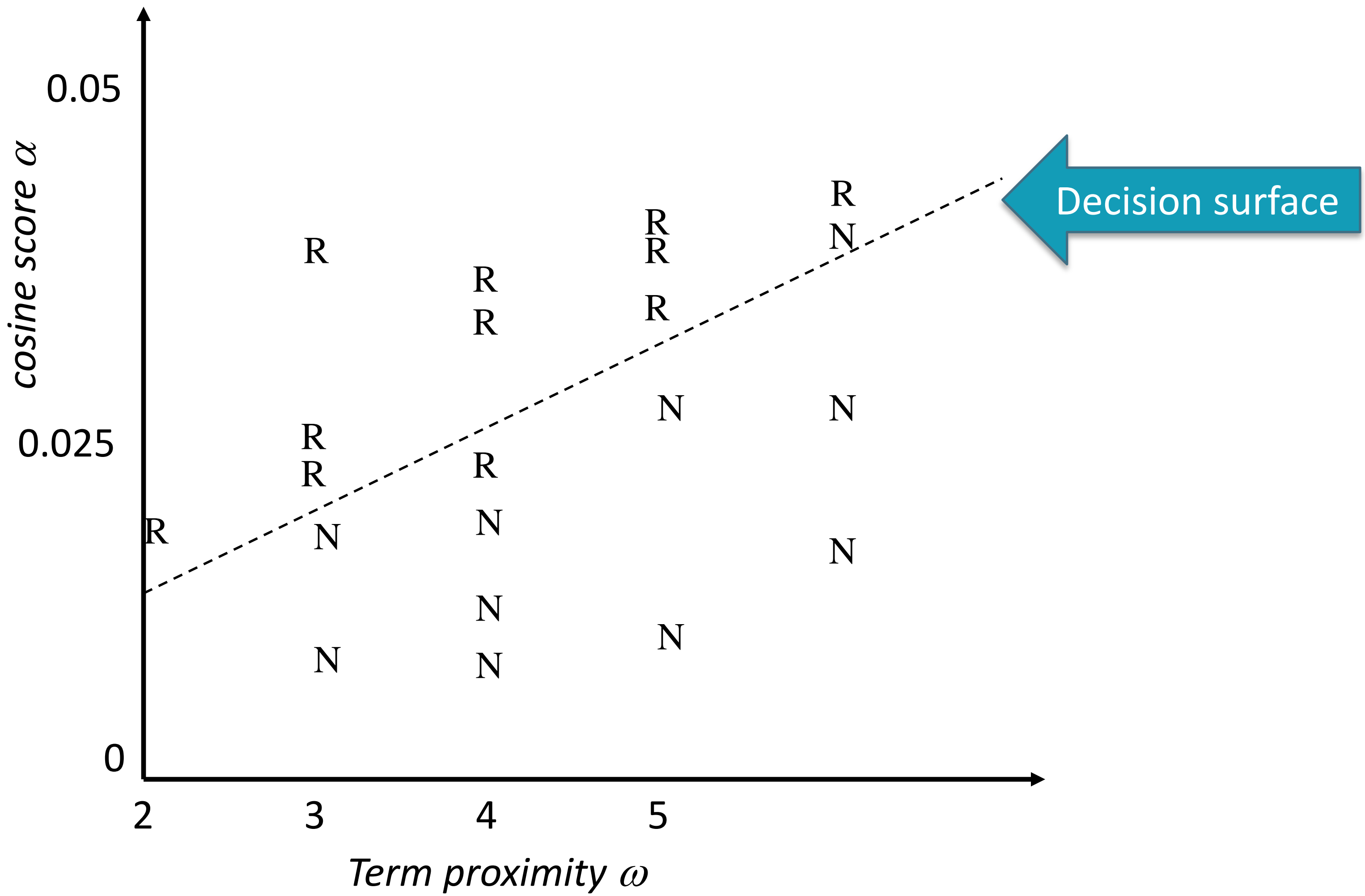
$$Score(d, q) = Score(\alpha, \omega) = a\alpha + b\omega + c$$

- And the linear classifier is

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

- … just like 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 learning methods to learn the linear classifier weights

An SVM classifier for information retrieval

[Nallapati 2004]

- Let relevance score $g(r|d,q) = \mathbf{w} \bullet f(d,q) + b$
- 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)

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)
TREC Disk 3	Lemur	**0.1785**	**0.2503**	0.2666
	SVM	0.1728	0.2432	**0.2750**
Disk 4-5	Lemur	**0.1773**	**0.2516**	0.2656
	SVM	0.1646	0.2355	**0.2675**

- At best the results are about equal to Lemur
  - Actually a little bit below
- Paper's advertisement: Easy to add more features
  - This is illustrated on a homepage finding task on WT10G:
    - Baseline Lemur 52% success@10, baseline SVM 58%
    - SVM with URL-depth, and in-link features: 78% success@10

“Learning to rank”

- Classification probably isn’t the right way to think about approaching ad hoc IR:
  - Classification problems: Map to an 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

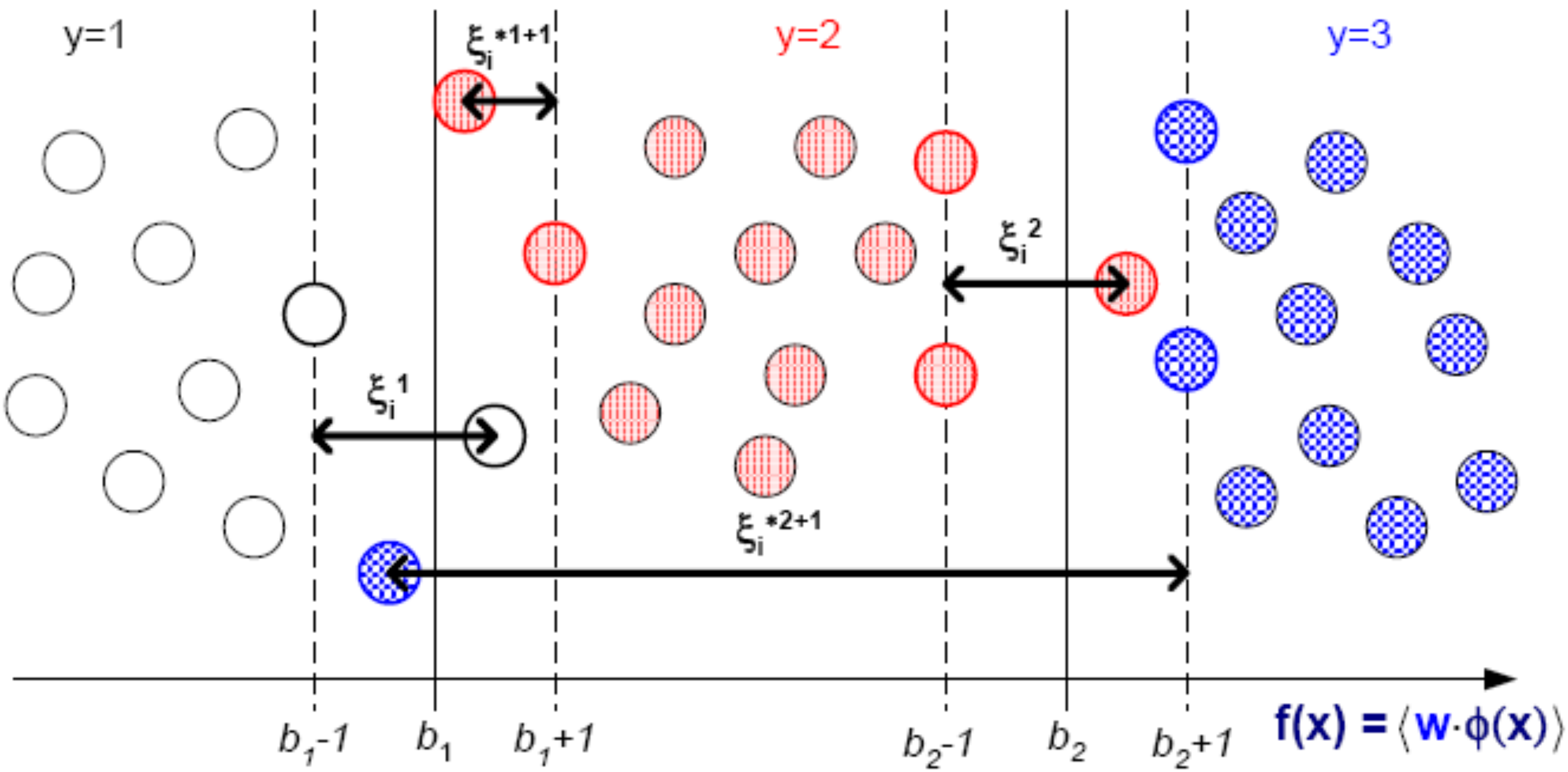
"Learning to rank"

- 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

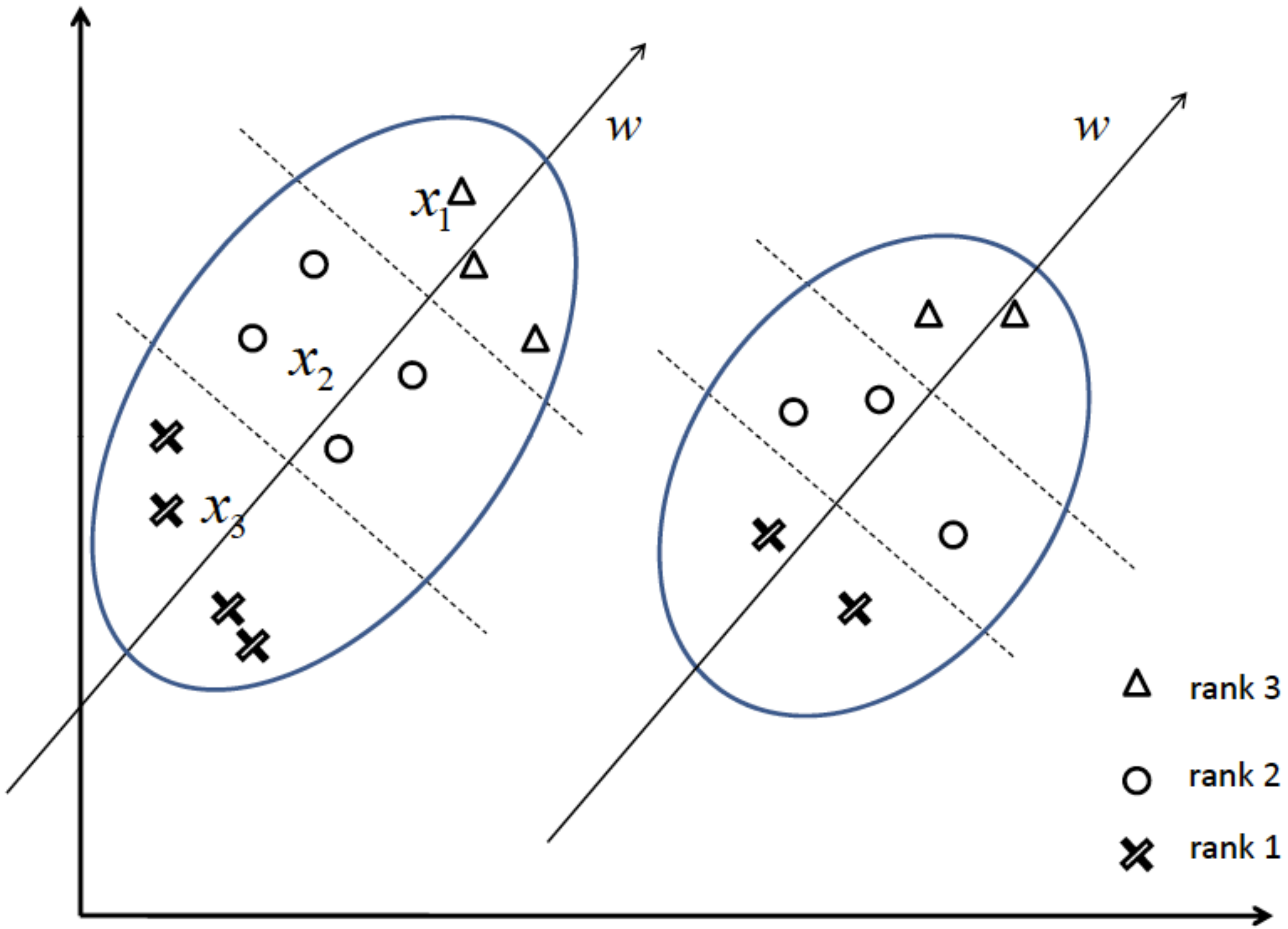
- We could do ***point-wise* learning**, where we try to map items of a certain relevance rank to a subinterval (e.g, Crammer et al. 2002 PRank)
- But most work does ***pair-wise* learning**, where the input is a pair of results for a query, and the class is the relevance ordering relationship between them

Point-wise learning

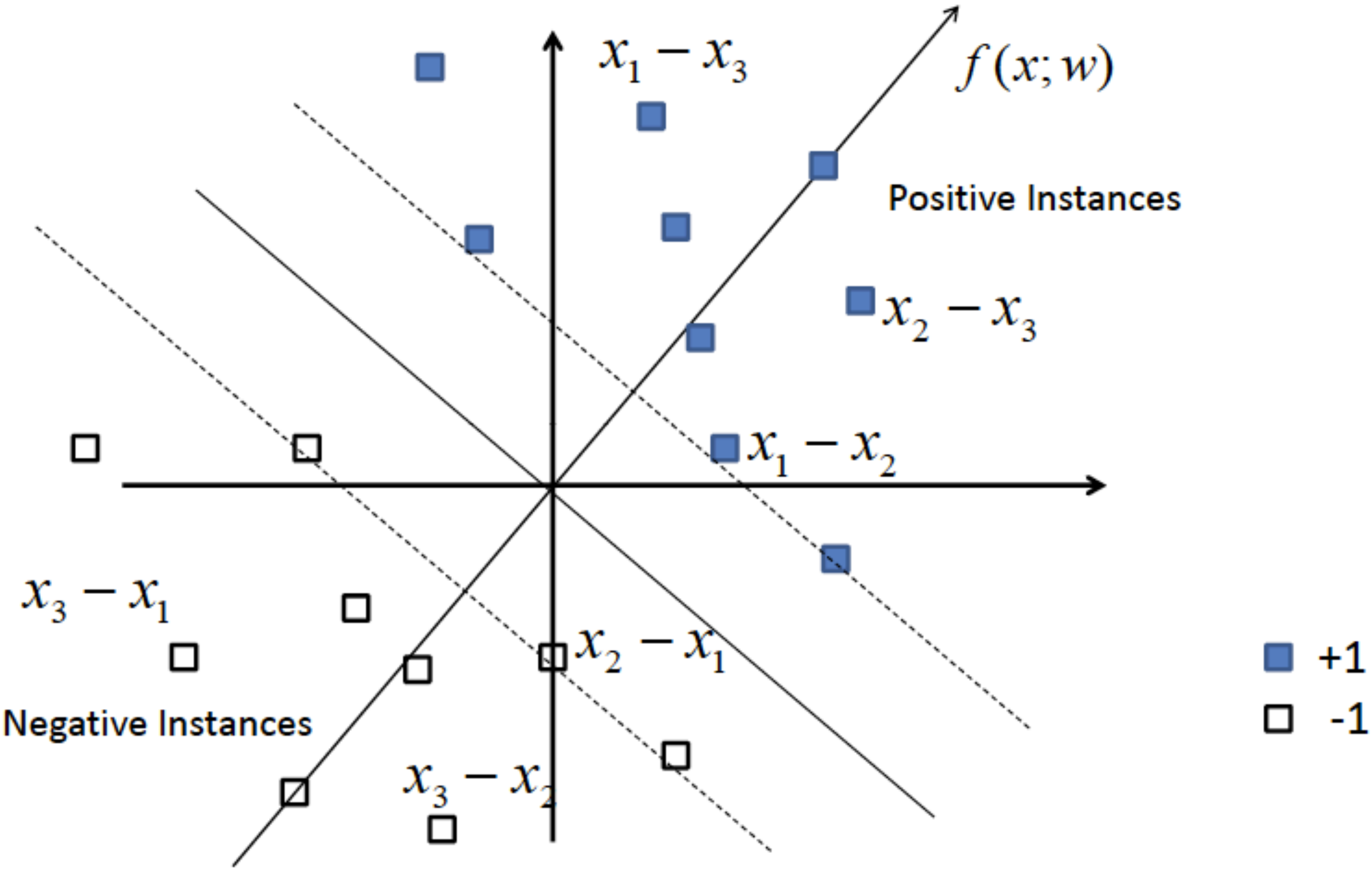
- Goal is to learn a threshold to separate each rank



Two queries in the original space



Two queries in the pairwise space



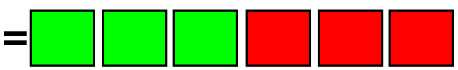
Listwise Learning to Rank

- Can we directly optimize the ranking?
  - f → **order** → metric

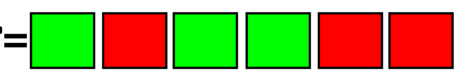
The slide was borrowed from Hongning Wang

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 intractable number of constraints**!

Summary

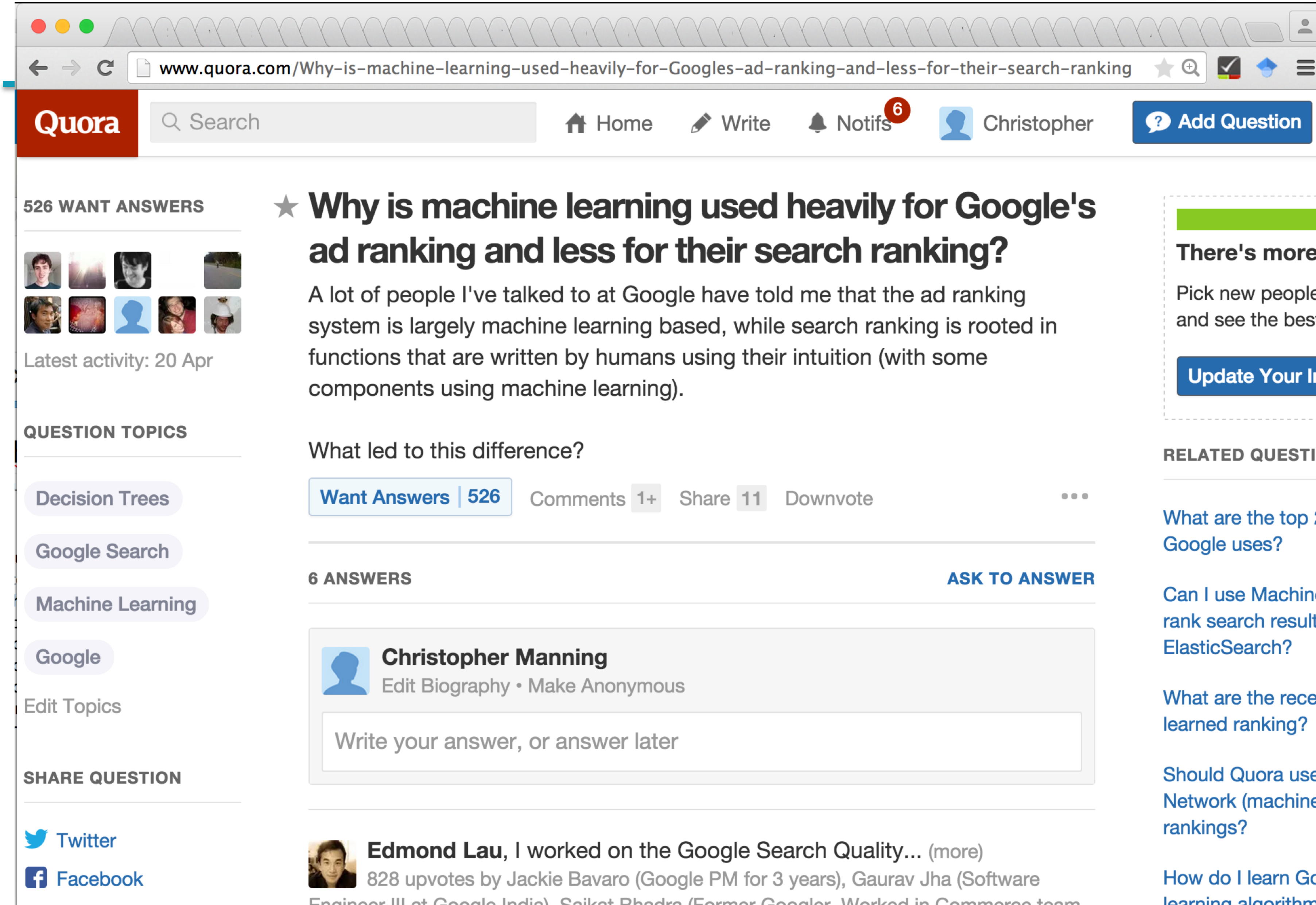
- Learning to rank
  - Automatic combination of ranking features for optimizing IR evaluation metrics
- Approaches
  - Pointwise
    - Fit the relevance labels individually
  - Pairwise
    - Fit the relative orders
  - Listwise
    - Fit the whole order

The slide was borrowed from Hongning Wang

The Limitations of Machine Learning

- Everything that we have looked at (and most work in this area) produces *linear* models over features
- This contrasts with most of the clever ideas of traditional IR, which are *nonlinear* scalings and combinations (products, etc.) of basic measurements
  - log term frequency, idf, tf.idf, pivoted length normalization
- At present, ML is good at weighting features, but not as good at coming up with nonlinear scalings
  - Designing the basic features that give good signals for ranking remains the domain of human creativity
  - Or maybe we can do it with deep learning ☺

http://www.quora.com/Why-is-machine-learning-used-heavily-for-Googles-ad-ranking-and-less-for-their-search-ranking



Summary

- The idea of learning ranking functions has been around for about 20 years
- But only more 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!]
- There is every reason to think that the importance of machine learning in IR will grow in the future.

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*.