Introduction to

Information Retrieval

Lecture 14: Learning to Rank

Machine learning for IR ranking?

- Many heuristic or unsupervised methods for ranking documents in IR
  - Cosine similarity, inverse document frequency, proximity, pivoted document length normalization, Pagerank, …
- Supervised machine learning classifiers can be used to classify documents using
  - Naïve Bayes, Rocchio, kNN, SVMs
- *What about using 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”

Practical Usage

Practical usage by search engines [edit]

Commercial web search engines began using machine learned ranking systems since the 2000s (decade). One of the first search engines to start using it was AltaVista (later its technology was acquired by Overture, and then Yahoo), which launched a gradient boosting-trained ranking function in April 2003.[31][32]

Bing's search is said to be powered by RankNet algorithm,[33][*when?*] which was invented at Microsoft Research in 2005.

In November 2009 a Russian search engine Yandex announced[34] that it had significantly increased its search quality due to deployment of a new proprietary MatrixNet algorithm, a variant of gradient boosting method which uses oblivious decision trees.[35] Recently they have also sponsored a machine-learned ranking competition "Internet Mathematics 2009"[36] based on their own search engine's production data. Yahoo has announced a similar competition in 2010.[37]

As of 2008, Google's Peter Norvig denied that their search engine exclusively relies on machine-learned ranking.[38] Cuil's CEO, Tom Costello, suggests that they prefer hand-built models because they can outperform machine-learned models when measured against metrics like click-through rate or time on landing page, which is because machine-learned models "learn what people say they like, not what people actually like".[39]

In January 2017 the technology was included in the open source search engine Apache Solr™,[40] thus making machine learned search rank widely accessible also for enterprise search.

https://en.wikipedia.org/wiki/Learning_to_rank#Practical_usage_by_search_engines

Overview of Supervised Learning

- Given a new object o, map it to a feature vector

$$\mathbf{x} = (x_1, x_2, \ldots, x_d)^\top$$

- Predict the output (class label)

$$y \in \mathcal{Y}$$

$$f(\mathbf{x}) : \mathbb{R}^d \mapsto \mathcal{Y}$$

  - Binary classification: $\mathcal{Y} = \{-1, +1\}$

    Sometimes, $\{0, 1\}$

  - Multi-class classification: $\mathcal{Y} = \{1, 2, \ldots, C\}$

- Learn a classification **function**:

- Regression:

$$f(\mathbf{x}) : \mathbb{R}^d \mapsto \mathbb{R}$$

Example

- Text categorization:

Doc: Months of campaigning and weeks of round-the-clock efforts in Iowa all came down to a final push Sunday, …

→ **Topic:** { Politics / Sport }

- Input object: a document = a sequence of words
- Input features : $\mathbf{X}$ = word frequencies
  - freq(democrats)=2, freq(basketball)= 0, …
  - $\mathbf{X}$ = [1, 2, 0, …]^T

← looks familiar?

- Class label: 'Politics' vs 'Sport':

$$y = +1 \qquad y = -1$$

How to Learn?

- How to find f()?
  - Typically simplify it to finding a parameterized function f(x; $\boldsymbol{\theta}$) within a chosen Function Family.
  - e.g., linear functions $f(\mathbf{x};\boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \mathbf{x})$
- Which one to choose
  - The one that performs the "best" on a set of training examples $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$
  - Use a Loss Function to define the 'badness' of a function

    $\mathbf{E}_{\mathbf{x}\in\mathcal{D}_{\text{train}}}[\ell(y,\hat{y})]$, where $\hat{y} = f(\mathbf{x};\boldsymbol{\theta}_*)$

    aka., Risk function
  - It's a function of $\boldsymbol{\theta}$. ➔ find $\boldsymbol{\theta}$ that minimizes its value

Other terms can be added to the loss function (e.g., regularization, prior, etc.)

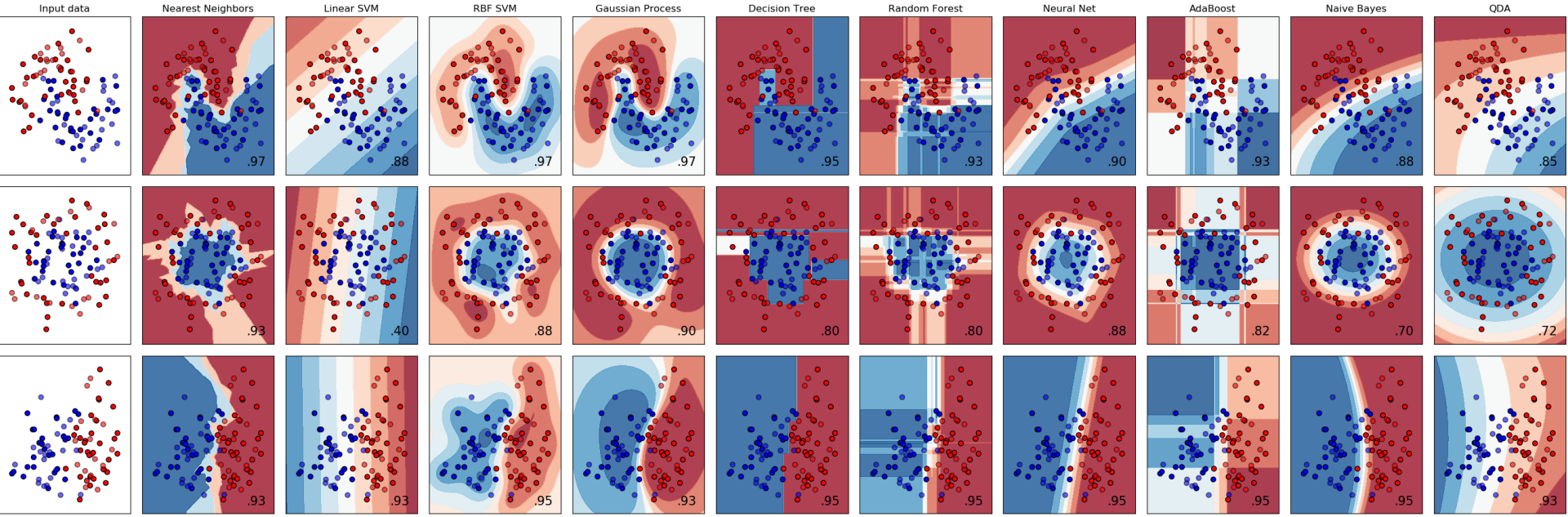
How to Evaluate?

- Evaluate accuracy on another set of test examples
  - $\mathcal{D}_{\text{test}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$

$$\mathbf{1}(y, \hat{y}) = \begin{cases} 0 & , y = \hat{y}; \\ 1 & , \text{else.} \end{cases}$$

  - Testing error: $\mathcal{E} = \mathbf{E}_{\mathbf{x} \in \mathcal{D}_{\text{test}}}[\mathbf{1}(y, \hat{y})]$, where $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta}_*)$
    - i.e., the expectation of the 0-1 loss function on D_{train}.
  - Training error $\varepsilon = \mathbf{E}_{\mathbf{x} \in \mathcal{D}_{\text{train}}}[\mathbf{1}(y, \hat{y})]$, where $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta}_*)$

- PAC learning theory assumps that both training and testing data are drawn i.i.d. (Identically independently distributed)
  - Then, $Pr[\mathcal{E} \leq \varepsilon + t] \geq 1 - \delta$



https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

Machine Learning Terminologies

- Supervised learning has input labelled data
  - #instances x #attributes matrix/table
  - #attributes = #features + 1
    - 1 (usu. the last attribute) is for the class attribute
- Labelled data split into 2 or 3 disjoint subsets
  - Training data
    - Training error = $1.0 - \frac{\#correctly\_classified}{\#training\_instances}$

➔ Build a model

  - Validation/development data

➔ Select/refine the model

  - Testing data
    - Testing/generalization error = $1.0 - \frac{\#correctly\_classified}{\#testing\_instances}$

➔ Evaluate the model

- We mainly discuss binary classification here
  - i.e., #labels = 2

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…

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 to tune weighting coefficients by hand
  - And people did
  - e.g., BM25 with manual setting of the hyper-parameters

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.

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
    - $\mathbf{x} = (\alpha, \omega)^T$, 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*

Simple example: Using classification for ad hoc IR

- A linear score function is then

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

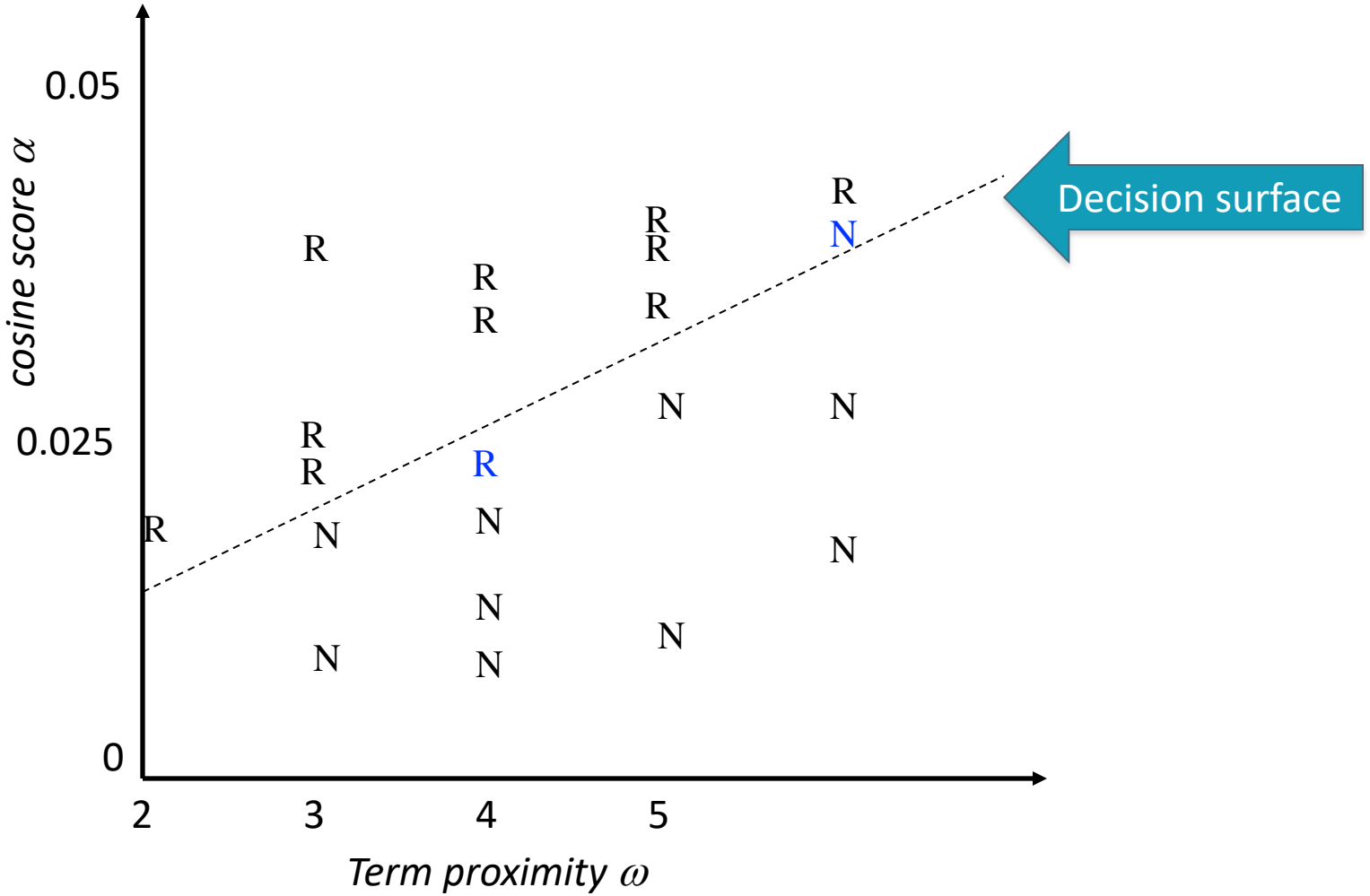
- And the linear classifier is

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

- Denote these parameters collectively as θ

- We can reduce the number of parameters by 1
- θ ➔ w in the later slides

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 methods we have seen previously for learning the linear classifier weights

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

"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

Pairwise learning: 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

$$c_i > c_k \text{ iff } f(\psi_i) > f(\psi_k)$$

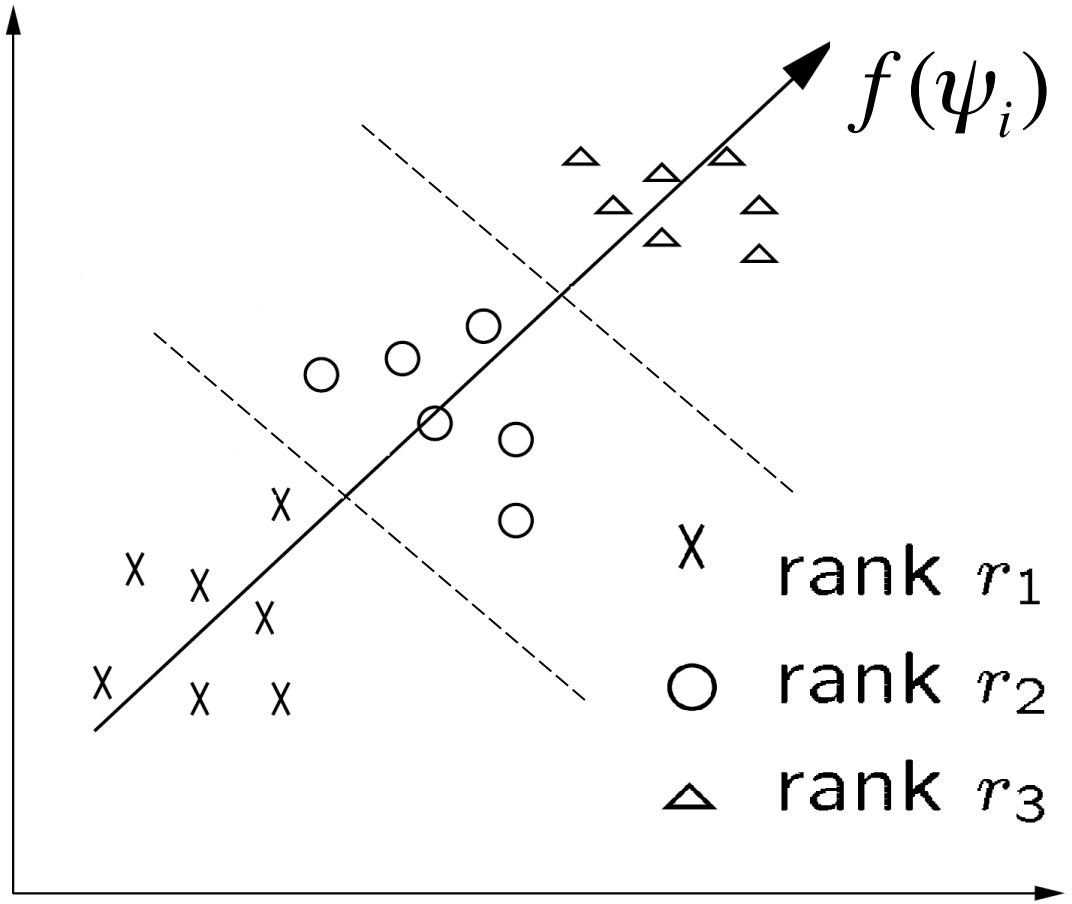
- … or at least one that tries to do this with minimal error
- Suppose that f is a linear function

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

The Ranking SVM

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

- Ranking Model: $f(\psi_i)$



The Ranking SVM

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

- Then (combining the two equations on the last slide):

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

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

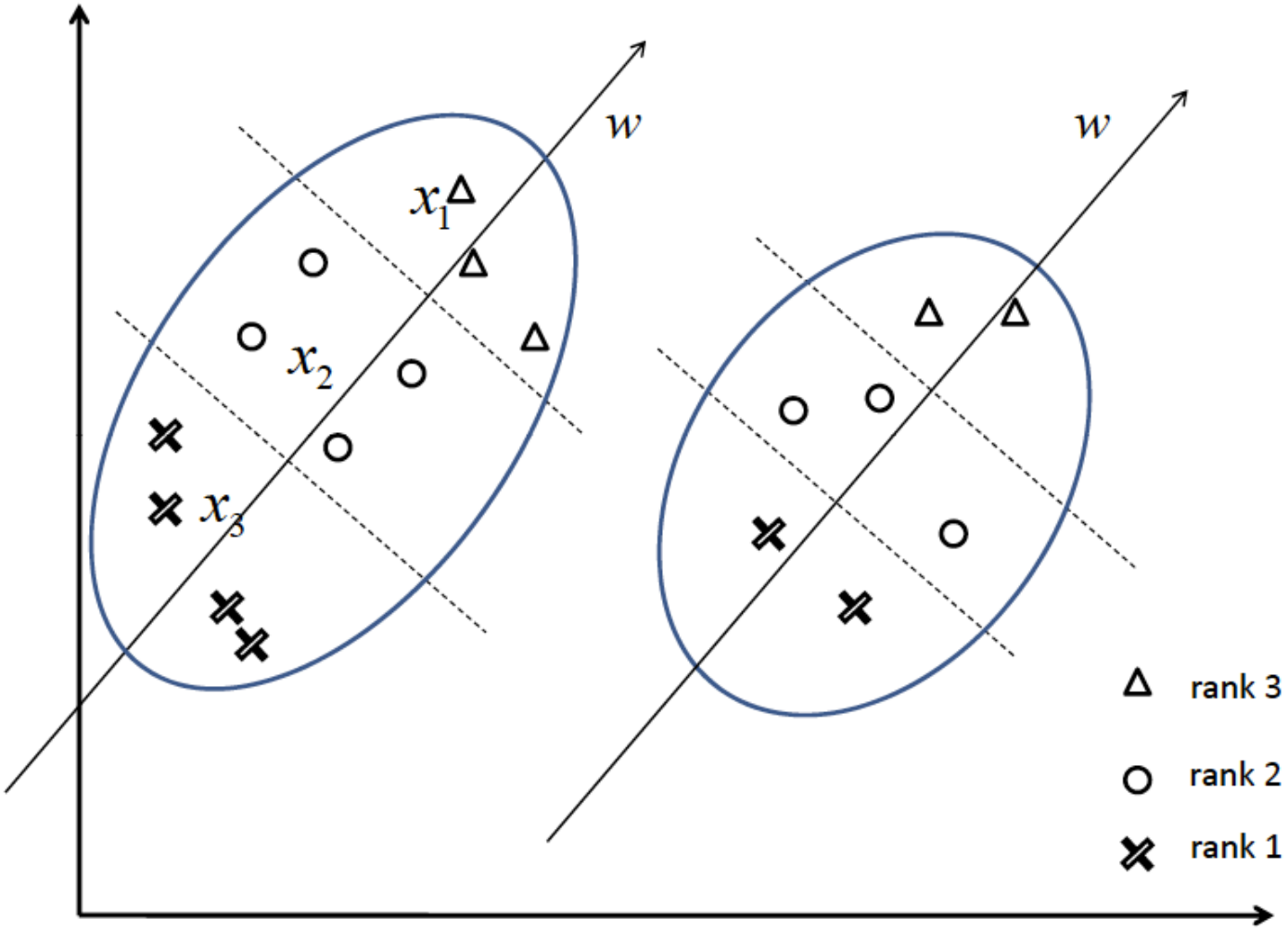
$$\Phi_u = \Phi(d_i, d_j, q) = \psi_i - \psi_k$$

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

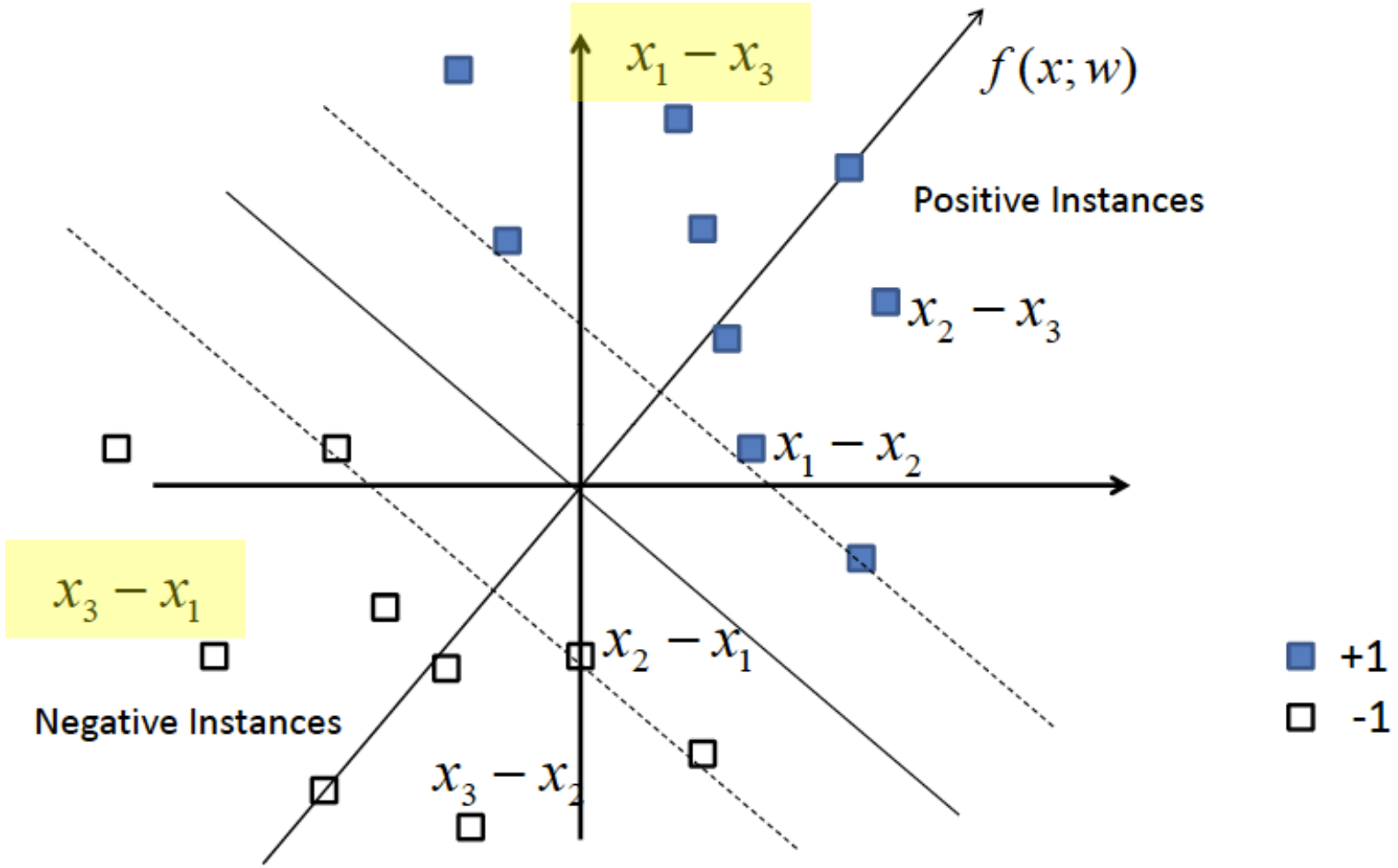
- We can build model over just cases for which $z_u = -1$
- From training data $S = \{\Phi_u\}$, we train an SVM

Q: *How do we obtain the final total order once a model is learned?*

Two queries in the original space



Two queries in the pairwise space



The Ranking SVM

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

- The SVM learning task is then like other examples that we saw before
- Find $\mathbf{w}$ and $\xi_u \geq 0$ such that
  - $\frac{1}{2}\mathbf{w}^T\mathbf{w} + C \Sigma \xi_u$ is minimized, and
  - for all Φ_u such that $z_u < 0$, $\mathbf{w} \bullet \Phi_u \geq 1 - \xi_u$
- We can just do the negative z_u, as ordering is antisymmetric
- You can again use SVMlight (or other good SVM libraries) to train your model

Aside: The SVM loss function

- The minimization

$$\min_{\mathbf{w}} \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C \Sigma \xi_u$$

and for all Φ_u such that $z_u < 0$, $\mathbf{w} \bullet \Phi_u \geq 1 - \xi_u$

- can be rewritten as

$$\min_{\mathbf{w}} (1/2C)\mathbf{w}^T\mathbf{w} + \Sigma \xi_u$$

and for all Φ_u such that $z_u < 0$, $\xi_u \geq 1 - (\mathbf{w} \bullet \Phi_u)$

- Now, taking $\lambda = 1/2C$, we can reformulate this as

$$\min_{\mathbf{w}} \Sigma [1 - (\mathbf{w} \bullet \Phi_u)]_+ + \lambda \mathbf{w}^T\mathbf{w}$$

- Where $[]_+$ is the positive part (0 if a term is negative)

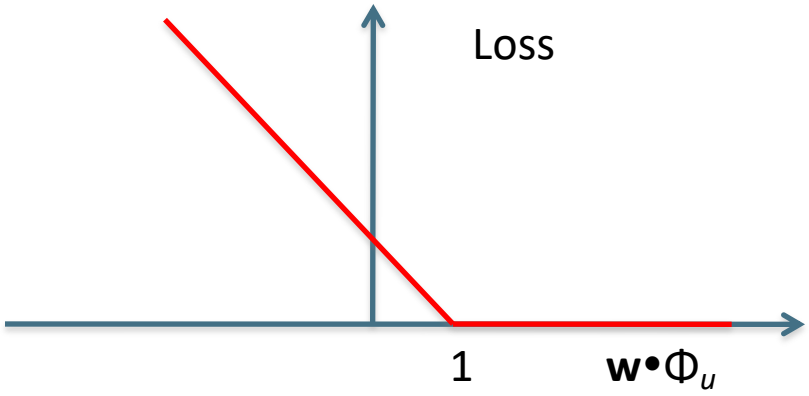
Aside: The SVM loss function

- The reformulation

Hinge loss | Regularizer of $\|\mathbf{w}\|$

$$\min_{\mathbf{w}} \Sigma\, [1 - (\mathbf{w} \bullet \Phi_u)]_+ + \lambda \mathbf{w}^T \mathbf{w}$$

- shows that an SVM can be thought of as having an empirical **"hinge" loss** combined with a **weight regularizer**



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

Problems with the naïve ranking SVM approach

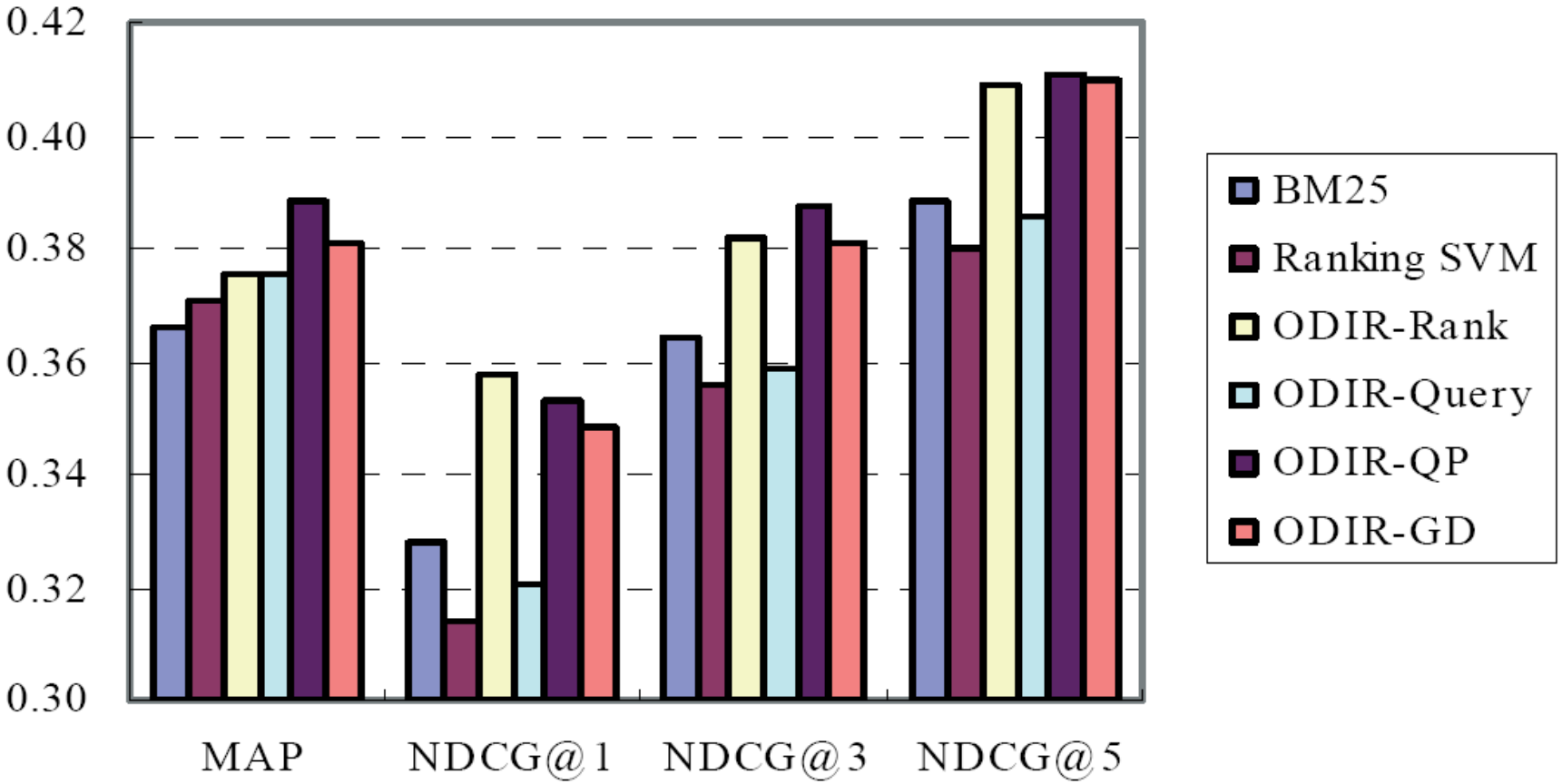
1. Correctly ordering the most relevant documents is crucial to the success of an IR system, while misordering less relevant results matters little
   - The ranking SVM considers all ordering violations as the same
2. Some queries have many (somewhat) relevant documents, and other queries few. If we 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

Both can be solved by weighting the loss function appropriately

MSN Search [now Bing]

- Second experiment with MSN search
- Collection of 2198 queries
- 6 relevance levels rated:
  - Definitive 8990
  - Excellent 4403
  - Good 3735
  - Fair 20463
  - Bad 36375
  - Detrimental 310

Experimental Results (MSN search)



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

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

Other machine learning methods for learning to rank

- Of course!
- I'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
  - (Gradient-boosted) Decisision Trees

The 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
- 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*.