Information Storage and Retrieval

CSCE 670
Texas A&M University
Department of Computer Science & Engineering
Instructor: Prof. James Caverlee

Evaluation + Learning to Rank
30 January / 01 February 2018

Some slides from https://nlp.stanford.edu/IR-book/

Evaluation

Evaluation

- Critical step for understanding if our system/algorithm/improvement actually does anything net positive
  - Or even if it worsens things (!!!)
- Evaluation framework should be targeted to the application scenario:
  - Typically, different metrics and approaches for ranking, clustering, classification, recommender systems
- Today:
  - Focus on evaluating a search engine

IR is an experimental science

- Formulate a research question: the hypothesis
- Design an experiment to answer the question
- Perform the experiment
  - Compare with a baseline "control"
- Does the experiment answer the question?
  - Are the results significant? Or is it just luck?
- Report the results!

Research questions

- Does keyword highlighting help users evaluate document relevance?
  - Experiment: Build two different interfaces, one with highlighting, one without; run a user study
- Is letting users weight search terms a good idea?
  - Experiment: Build two different interfaces, one with term weighting, one without: run a user study

The importance of evaluation

- The ability to measure differences underlies experimental science
  - How well do our systems work?
  - Is A better than B?
  - Really?
  - Under what conditions?
- Evaluation drives what to research
  - Identify techniques that work and that don't

What we look for in evaluations ...

- Insightful
- Affordable
- Repeatable
- Explainable

Evaluating a Search Engine

Measures for a search engine

- How fast does it index
  - Number of documents/hour
  - (Average document size)
- How fast does it search
  - Latency as a function of index size
- Expressiveness of query language
  - Ability to express complex information needs
  - Speed on complex queries

Measures for a search engine

- All of the preceding criteria are *measurable*: we can quantify speed/size; we can make expressiveness precise
- The key measure: user happiness
  - What is this?
  - Speed of response/size of index are factors
  - But blindingly fast, useless answers won't make a user happy
- Need a way of quantifying user happiness

How do you tell if users are happy?

- Search returns products relevant to users
  - How do you assess this at scale?
- Search results get clicked a lot
  - Misleading titles/summaries can cause users to click
- Users buy after using the search engine
  - Or, users spend a lot of $ after using the search engine
- Repeat visitors/buyers
  - Do users leave soon after searching?
  - Do they come back within a week/month/… ?

Happiness: elusive to measure

- Most common proxy: **relevance** of search results
  - But how do you measure relevance?
- Pioneered by Cyril Cleverdon in the Cranfield Experiments

Be careful ...

McNamara's fallacy

The quantifying of success in the war (e.g. in terms of enemy body count) while ignoring other variables.

Beware of measuring what is easy instead of what's important.

McNamara’s fallacy

The first step is to measure whatever can be easily measured. This is OK as far as it goes.

The second step is to disregard that which can't be easily measured or to give it an arbitrary quantitative value. This is artificial and misleading.

The third step is to presume that what can't be measured easily really isn't important. This is blindness.

The fourth step is to say that what can't be easily measured really doesn't exist. This is suicide.

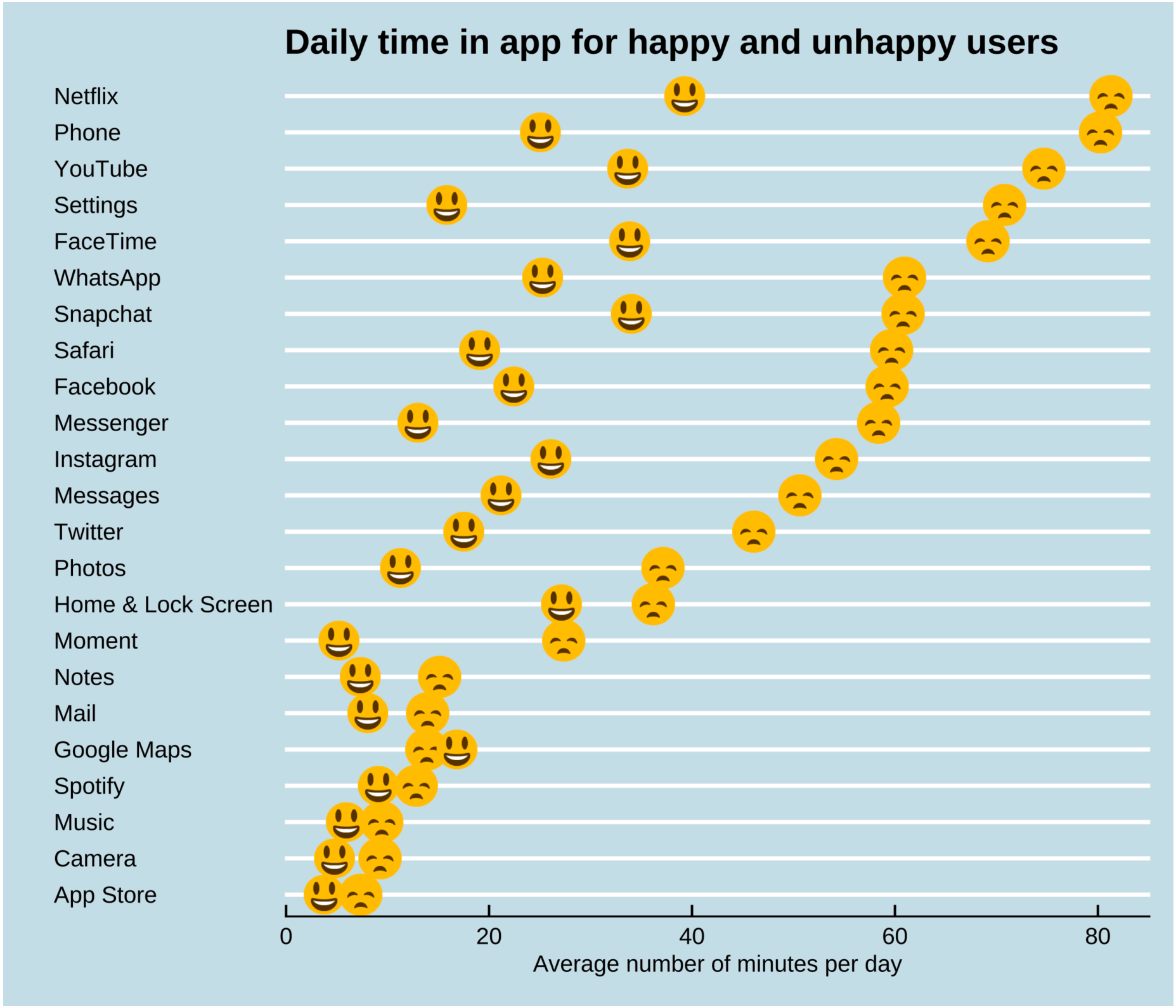
Examples of easy to measure but (possibly) not important

Time spent on website (Objective: MAX)

- Split pages (annoying news sites) = $$$
- Problem: boneheaded data-driven decisions

Time until purchase (Objective: MIN)

- Perhaps miss out on serendipity, exploration of other items (e.g., kill the recommender system so you optimize time to checkout)


Daily time in app for happy and unhappy users
Netflix
Phone
YouTube
Settings
FaceTime
WhatsApp
Snapchat
Safari
Facebook
Messenger
Instagram
Messages
Twitter
Photos
Home & Lock Screen
Moment
Notes
Mail
Google Maps
Spotify
Music
Camera
App Store
0
20
40
60
80
Average number of minutes per day

Happiness: elusive to measure

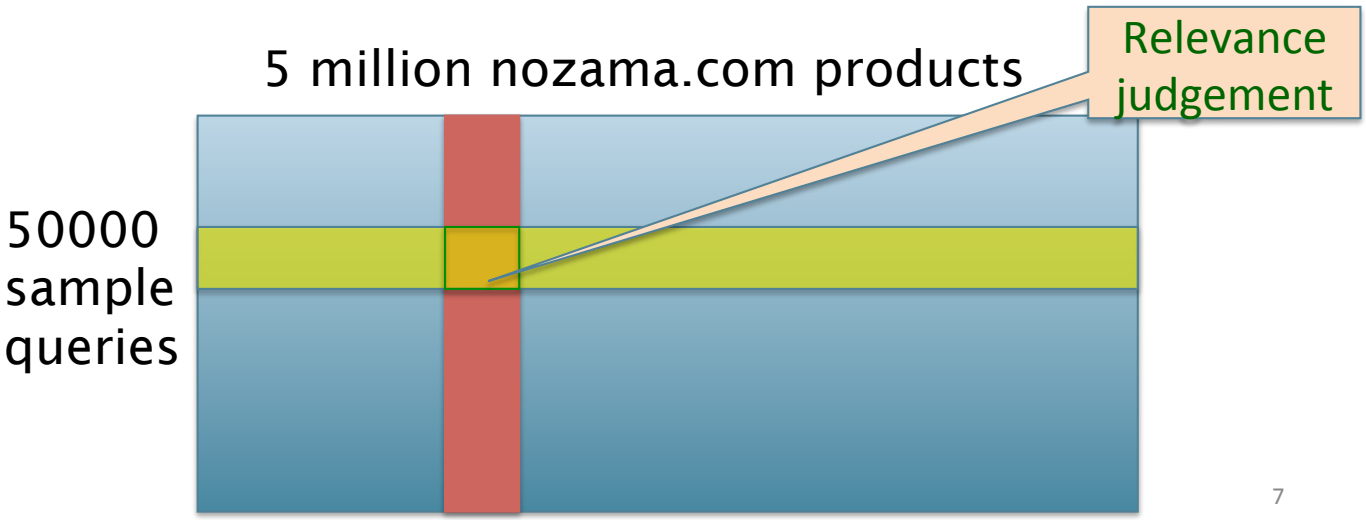
- Most common proxy: **relevance** of search results
  - But how do you measure relevance?
- Pioneered by Cyril Cleverdon in the Cranfield Experiments

Measuring relevance

- Three elements:
  - A benchmark document collection
  - A benchmark suite of queries
  - A binary assessment of either Relevant or Nonrelevant for each query and each document

Let's measure the quality of our new search algorithm

- Benchmark documents – (could be products, web pages, etc.)
- Benchmark query suite – more on this
- Judgments of document relevance for each query



Relevance judgments

- Binary (relevant vs. non-relevant) in the simplest case, more nuanced (0, 1, 2, 3 …) in others
- What are some issues already?
- 5 million times 50K takes us into the range of a quarter trillion judgments
  - If each judgment took a human 2.5 seconds, we'd still need 1011 seconds, or nearly $300 million if you pay people $10 per hour to assess
  - 10K new products per day

Crowdsource relevance judgments?

- Present query-document pairs to low-cost labor on online crowd-sourcing platforms
  - Hope that this is cheaper than hiring qualified assessors
- Lots of literature on using crowd-sourcing for such tasks
- Main takeaway – you get some signal, but the variance in the resulting judgments is very high

What else?

- Still need test queries
  - Must be germane to docs available
  - Must be representative of actual user needs
  - Random query terms from the documents generally not a good idea
  - Sample from query logs if available
- Classically (non-Web)
  - Low query rates – not enough query logs
  - Experts hand-craft "user needs"

Some public test collections

TABLE 4.3 Common Test Corpora

Collection	*NDocs*	*NQrys*	*Size (MB)*	*Term/Doc*	*Q-D RelAss*
ADI	82	35			
AIT	2109	14	2	400	>10,000
CACM	3204	64	2	24.5	
CISI	1460	112	2	46.5	
Cranfield	1400	225	2	53.1	
LISA	5872	35	3		
Medline	1033	30	1		
NPL	11,429	93	3		
OSHMED	34,8566	106	400	250	16,140
Reuters	21,578	672	28	131	
TREC	740,000	200	2000	89-3543	» 100,000

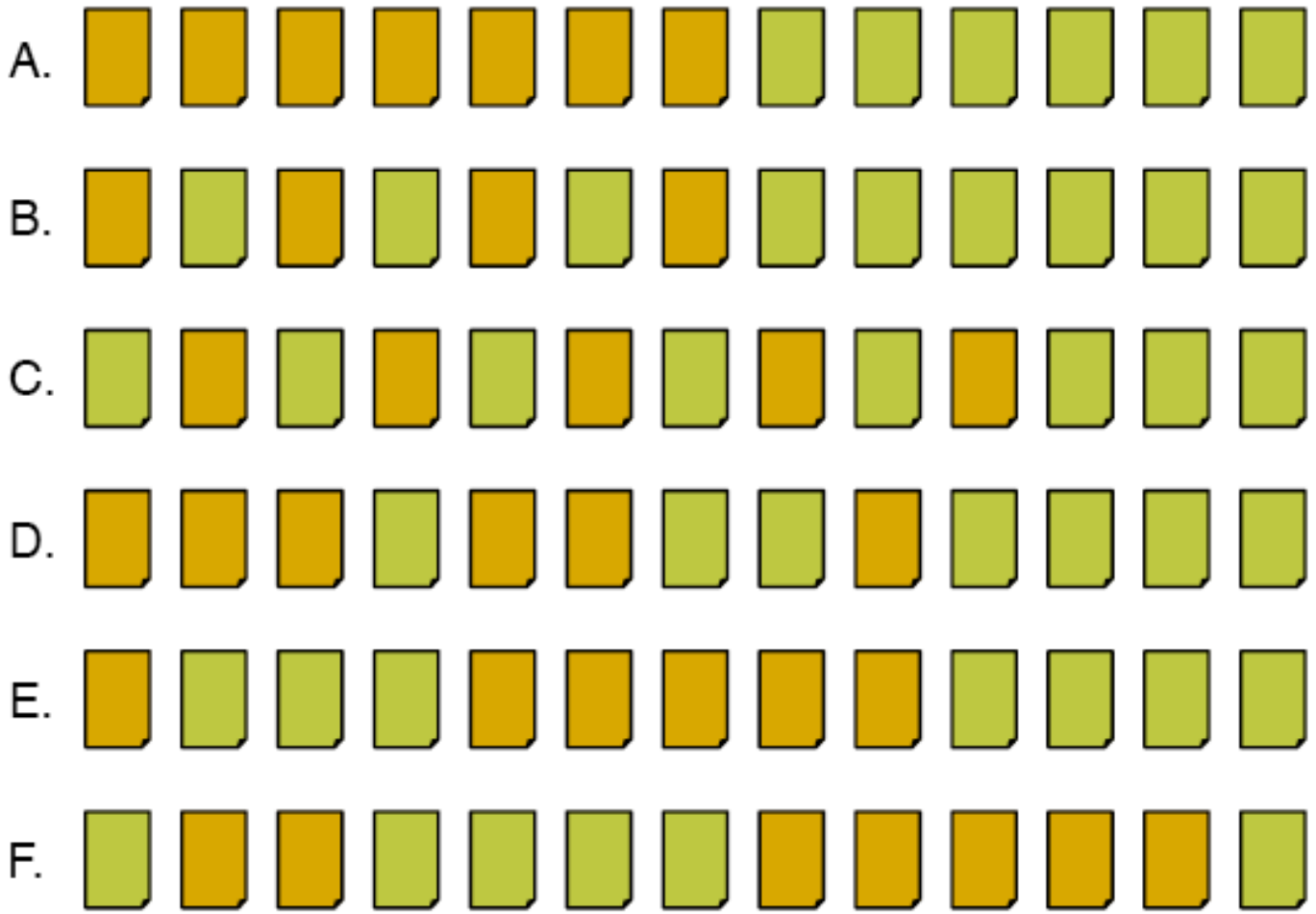
Now we have the basics of a benchmark

- Let's review some evaluation measures
  - Precision
  - Recall
  - F
  - Precision @ k
  - NDCG

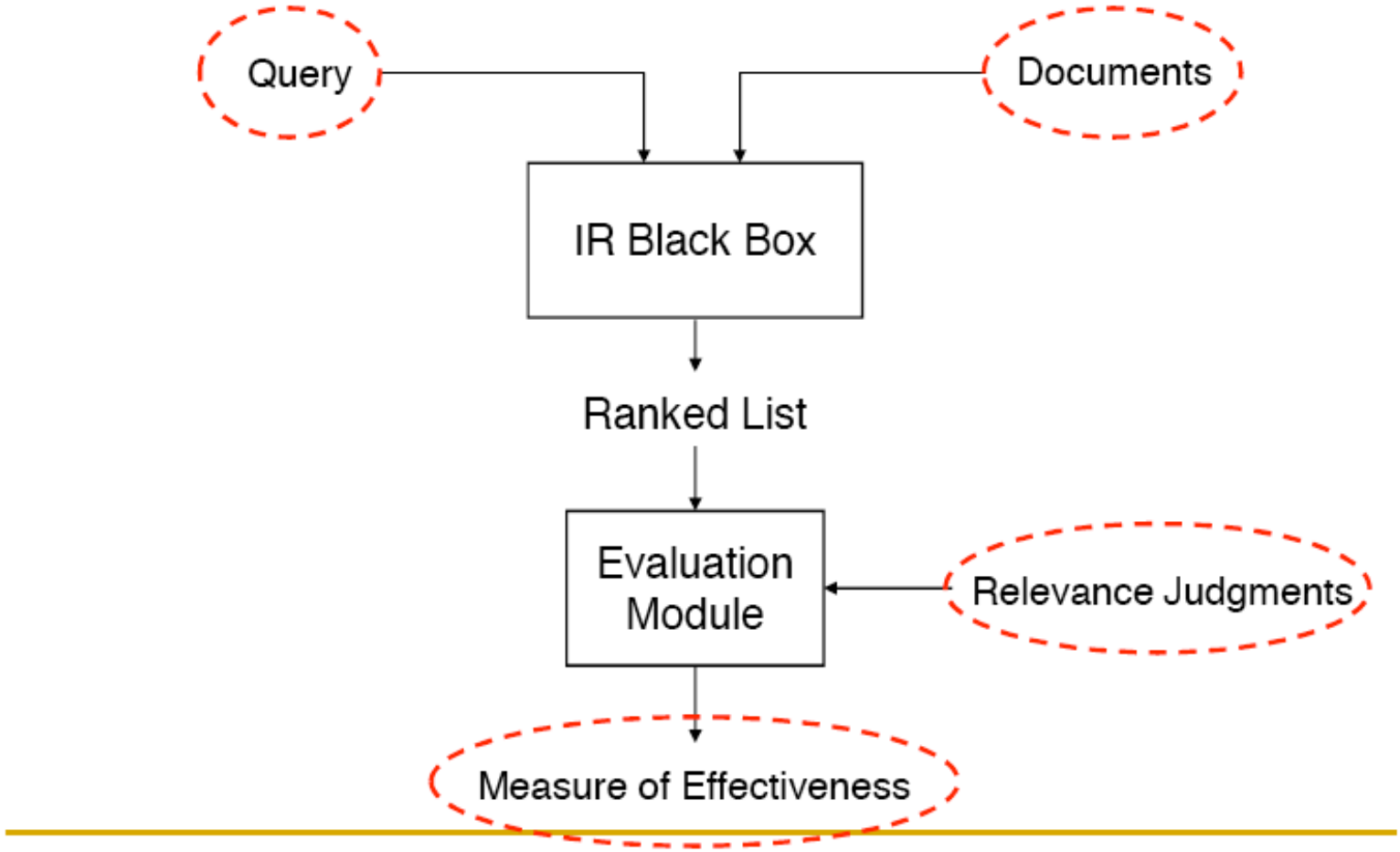
Evaluating an IR system

- Note: the **information need** is translated into a **query**
- Relevance is assessed relative to the **information need** *not* the **query**
- E.g., Information need: *: My swimming pool bottom is becoming black and needs to be cleaned.*
- Query: ***pool cleaner***
- You evaluate whether the doc addresses the information need, not whether it has those words

Which is the best rank order?



Automatic evaluation model



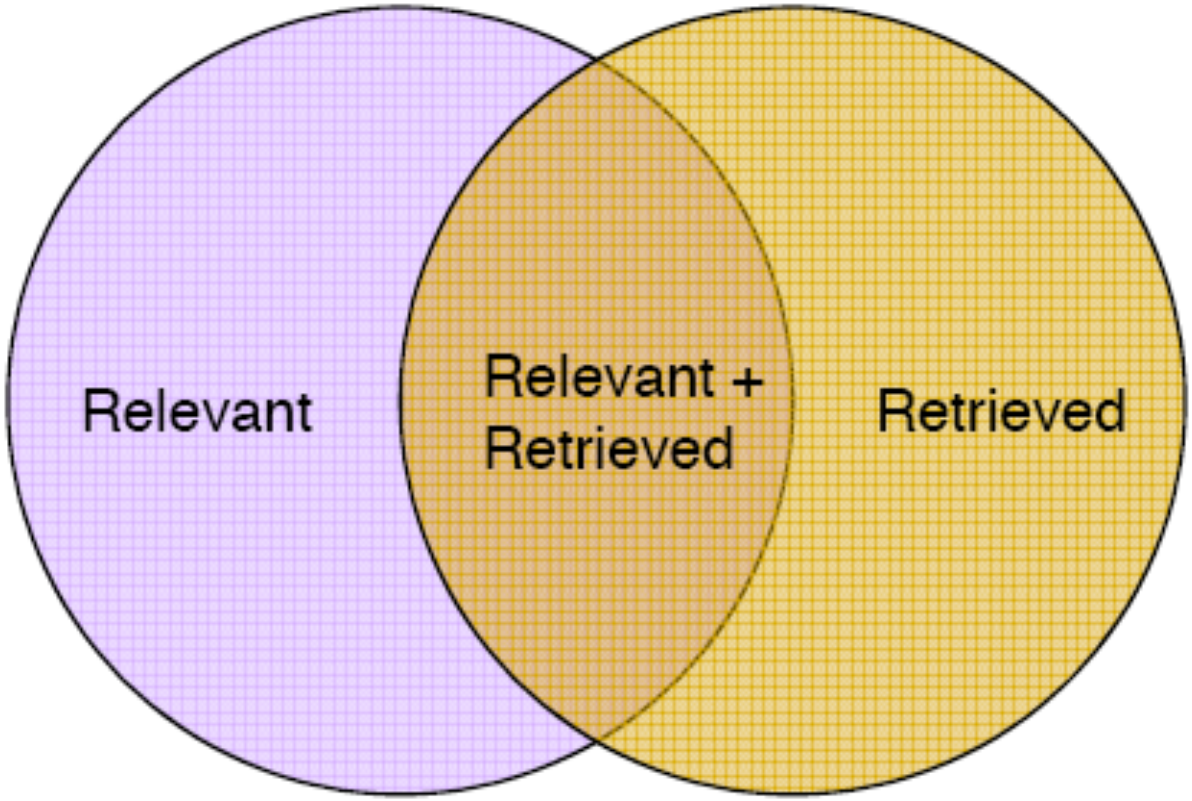
Unranked Evaluation

Unranked retrieval evaluation: Precision and Recall

- **Precision**: fraction of retrieved docs that are relevant = P(relevant|retrieved)
- **Recall**: fraction of relevant docs that are retrieved = P(retrieved|relevant)

	Relevant	Not Relevant
Retrieved	tp	fp
Not Retrieved	fn	tn

- Precision P = tp/(tp + fp)
- Recall R = tp/(tp + fn)


Relevant
Relevant +
Retrieved
Retrieved
Not Relevant + Not Retrieved

Precision/Recall: Things to watch out for

- Should average over large number of queries
  - 100s to 1000s
- Assessments have to be binary
  - more on this later
- Heavily skewed by corpus/authorship
  - Results may not translate from one domain to another

Precision/Recall tradeoff

- You can increase recall by returning more docs.
- Recall is a non-decreasing function of the number of docs retrieved.
- A system that returns all docs has 100% recall!
- The converse is also true (usually): It's easy to get high precision for very low recall.
- Assume the document with the largest score is relevant. How can we maximize precision?

A combined measure: *F*

- Combined measure that assesses this tradeoff is F measure (weighted harmonic mean):

$$F = \frac{1}{\alpha \frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

- People usually use balanced F_1 measure
  - i.e., with $\beta = 1$ or $\alpha = ½$
- Harmonic mean is a conservative average

F-measure details

$\beta^2 = \frac{1-\alpha}{\alpha}$

Harmonic mean: $\frac{1}{F} = \frac{1}{2}(\frac{1}{P} + \frac{1}{R})$

$F_1 = \frac{2PR}{P+R}$

F-measure example

	relevant	not relevant
retrieved	18	2
not retrieved	82	1,000,000,000

Precision?
Recall?
F?

F-measure example

	relevant	not relevant
retrieved	18	2
not retrieved	82	1,000,000,000

- precision = 18/(18+2) = 0.9
- recall = 18/(18+82) = 0.18
- F = 2PR/(P+R) = 2 * 0.9 * 0.18 / (0.9+0.18) = 0.3
- Note: F is a lot lower than AVG(P,Q) = 0.54
- Number of true negatives is not factored in: same F for 1000 true negatives

Ranked evaluation

Precision @ k

- Ranked version of precision
- Only evaluate precision for the top-k results
  - The denominator is always k
- Example: if 3 out of 5 results are relevant, then precision@5 is 0.6

DCG

- Normalized Discounted Cumulative Gain
- Sensitive to the position of the highest rated page
- Log-discounting of results
- Normalized for different lengths lists

DCG

- Popular measure for evaluating web search and related tasks
- Two assumptions:
  - Highly relevant documents are more useful than marginally relevant documents
  - the lower the ranked position of a relevant document, the less useful it is for the user, since it is less likely to be examined

DCG

- Uses graded relevance as a measure of usefulness, or gain, from examining a document
- Gain is accumulated starting at the top of the ranking and may be reduced, or discounted, at lower ranks
- Typical discount is 1/log (rank)
  - With base 2, the discount at rank 4 is 1/2, and at rank 8 it is 1/3

DCG

- What if relevance judgments are in a scale of [0,r]? r>2
- Cumulative Gain (CG) at rank n
  - Let the ratings of the n documents be r1, r2, …rn (in ranked order)
  - CG = r1+r2+…rn
- Discounted Cumulative Gain (DCG) at rank n
  - DCG = r1 + r2/$\log_2 2$ + r3/$\log_2 3$ + … rn/$\log_2 n$
  - We may use any base for the logarithm

DCG

- DCG is the total gain accumulated at a particular rank p:

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i}$$

- Alternative formulation:

$$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{log(1+i)}$$

- used by some web search companies
- emphasis on retrieving highly relevant documents

DCG Example

- 10 ranked documents judged on 0-3 relevance scale:
  - 3, 2, 3, 0, 0, 1, 2, 2, 3, 0
- discounted gain:
  - 3, 2/1, 3/1.59, 0, 0, 1/2.59, 2/2.81, 2/3, 3/3.17, 0
  - = 3, 2, 1.89, 0, 0, 0.39, 0.71, 0.67, 0.95, 0
- DCG:
  - 3, 5, 6.89, 6.89, 6.89, 7.28, 7.99, 8.66, 9.61, 9.61

Summarize a ranking: NDCG

- Normalized Discounted Cumulative Gain (NDCG) at rank n
  - Normalize DCG at rank n by the DCG value at rank n of the ideal ranking
  - The ideal ranking would first return the documents with the highest relevance level, then the next highest relevance level, etc
- Normalization useful for contrasting queries with varying numbers of relevant results
- NDCG is now quite popular in evaluating Web search

NDCG: Example

i	Ground Truth		Ranking Function$_1$		Ranking Function$_2$	
	Document Order	r_i	Document Order	r_i	Document Order	r_i
1	d4	2	d3	2	d3	2
2	d3	2	d4	2	d2	1
3	d2	1	d2	1	d4	2
4	d1	0	d1	0	d1	0
	$NDCG_{GT}$=1.00		$NDCG_{RF1}$=1.00		$NDCG_{RF2}$=0.9203	

$$DCG_{GT} = 2 + \left(\frac{2}{\log_2 2} + \frac{1}{\log_2 3} + \frac{0}{\log_2 4} \right) = 4.6309$$

$$DCG_{RF1} = 2 + \left(\frac{2}{\log_2 2} + \frac{1}{\log_2 3} + \frac{0}{\log_2 4} \right) = 4.6309$$

$$DCG_{RF2} = 2 + \left(\frac{1}{\log_2 2} + \frac{2}{\log_2 3} + \frac{0}{\log_2 4} \right) = 4.2619$$

$$MaxDCG = DCG_{GT} = 4.6309$$

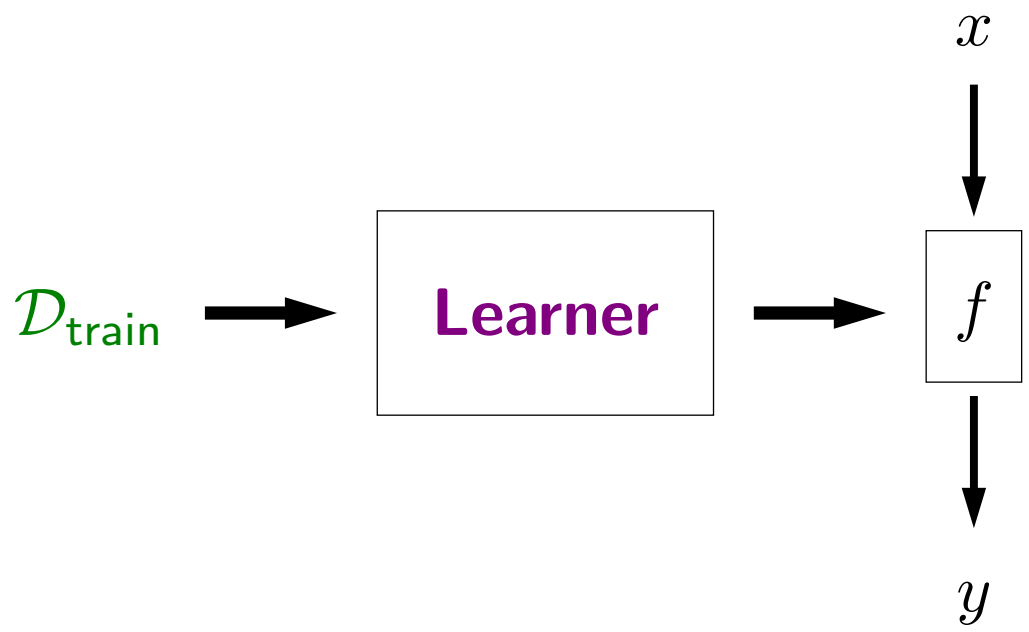
Learning to Rank

Recall: Basics of ML

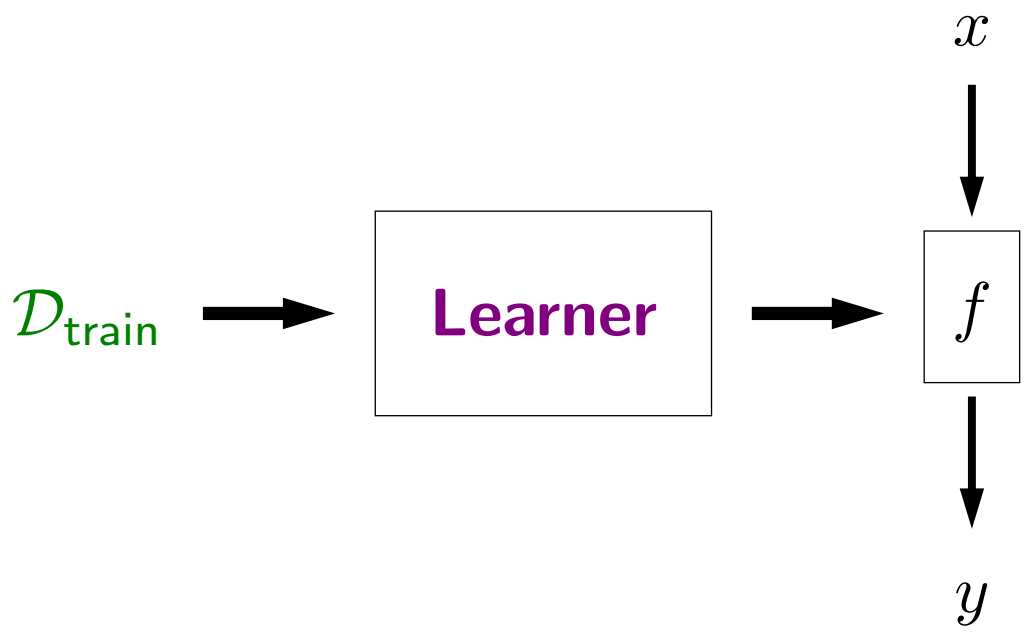
Training a model

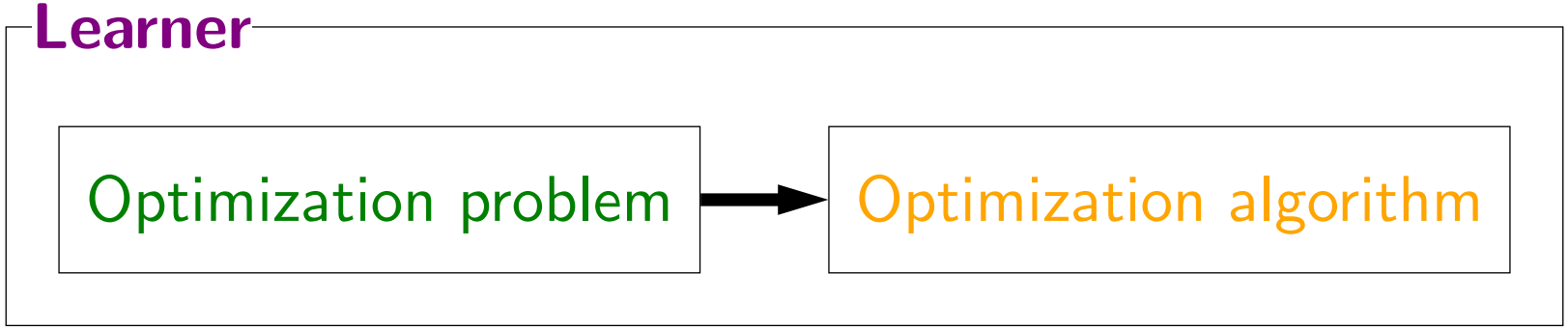
- Overfitting vs underfitting

Framework



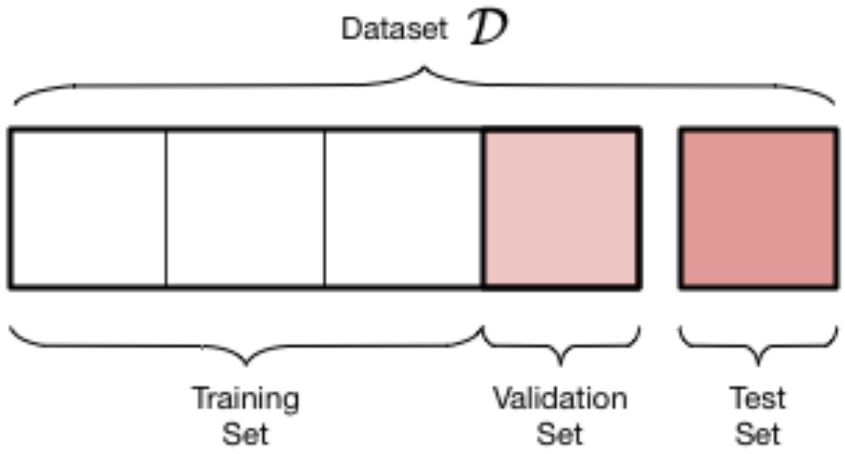
Framework


x
$\mathcal{D}_{\text{train}}$
Learner
f
y


Learner
Optimization problem
Optimization algorithm

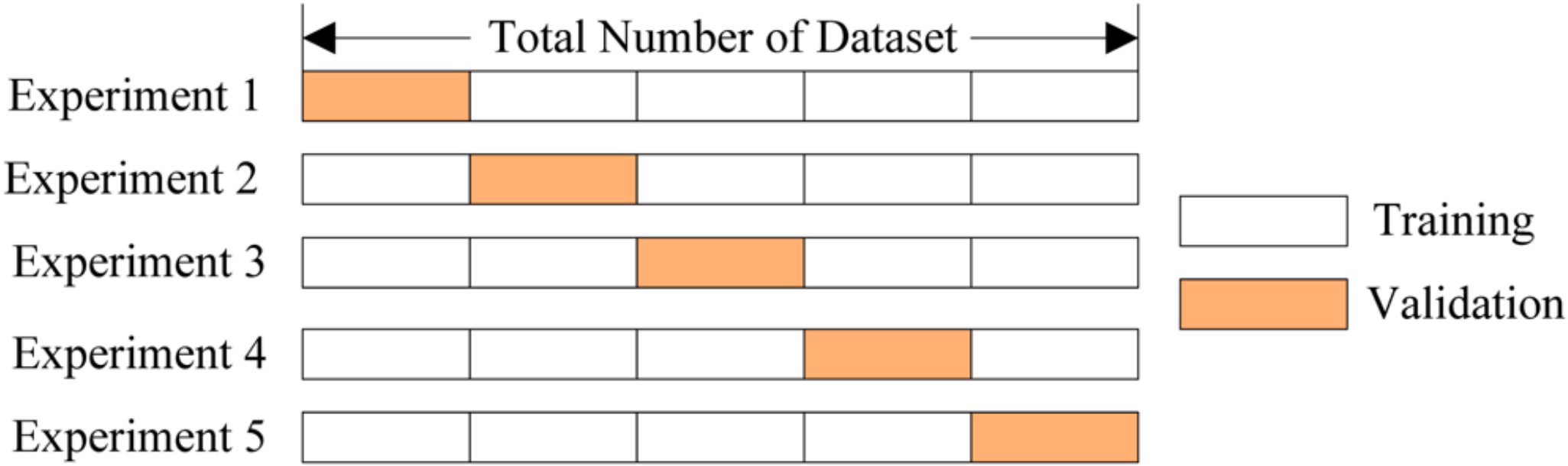
Use a validation set

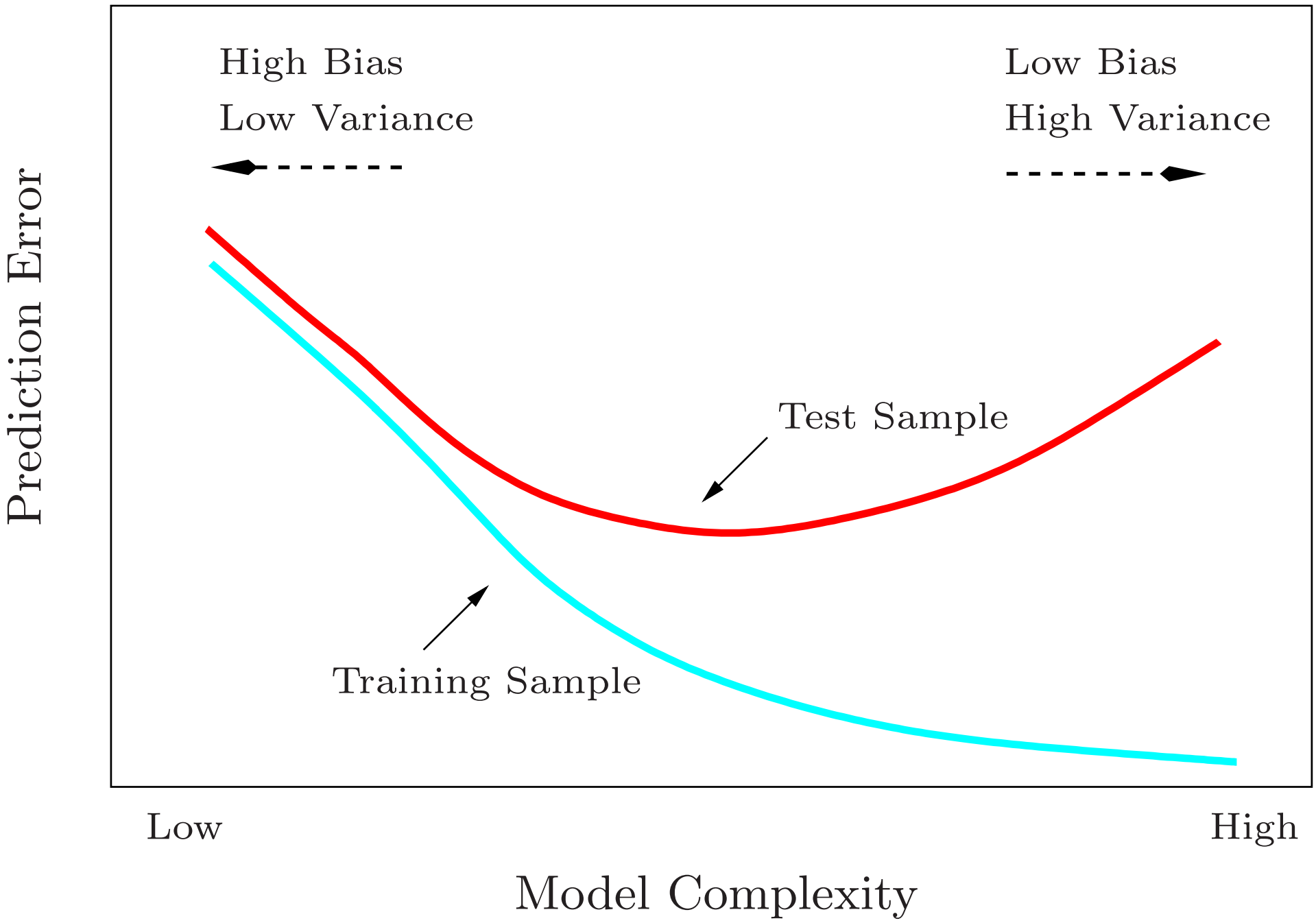
- Keep part of the labeled data **separate** as a validation set
- Train a model over the **training data** and “test” over the validation set
- Train *another model* over the **training data** and “test” over the validation set (and so on and so on)
- Choose model that minimizes error on the validation set



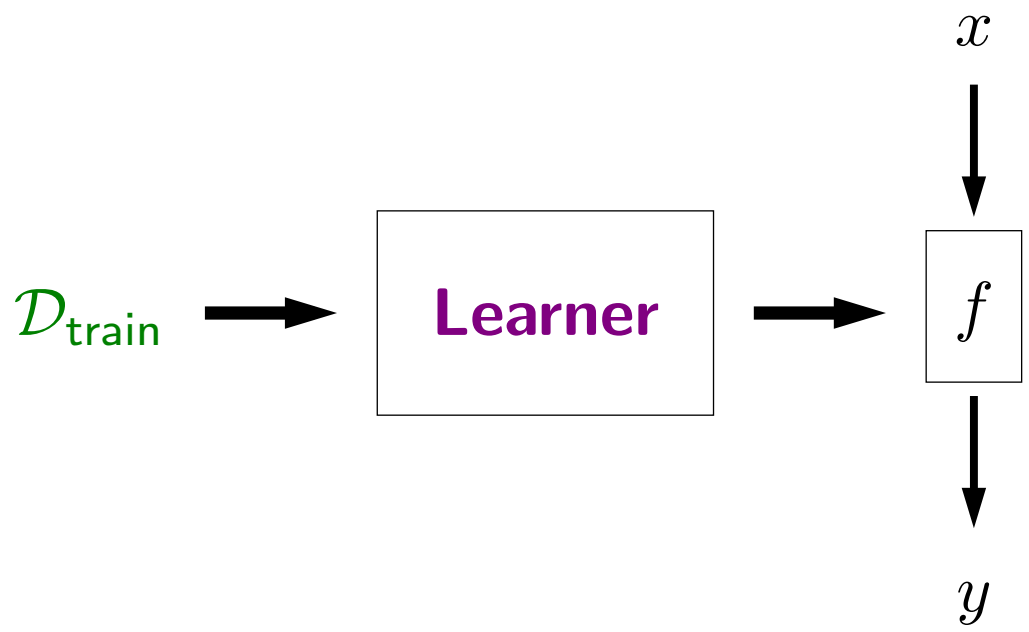
K-fold cross validation

- Split the training data into k subsets
- Train/test k times




High Bias
Low Variance
Low Bias
High Variance
Test Sample
Training Sample
Prediction Error
Low
High
Model Complexity

Framework



Feature extraction

Example task: predict y, whether a string x is an email address

Question: what properties of x **might be** relevant for predicting y?

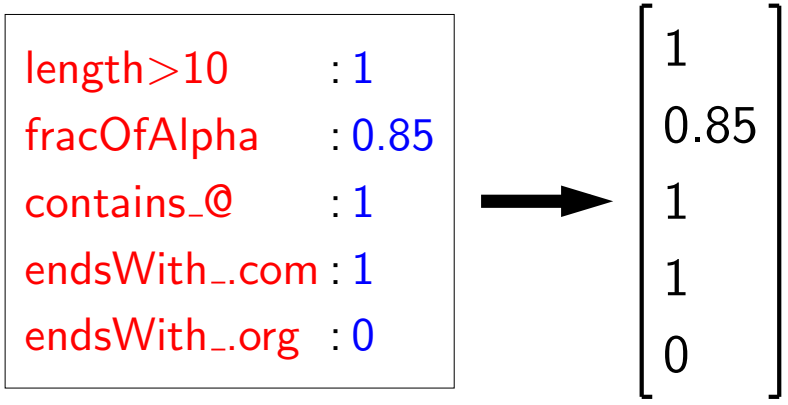
Feature extractor: Given input x, output a set of (feature name, feature value) pairs.

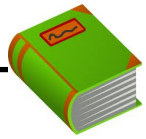
"abc@gmail.com" → **feature extractor** → arbitrary!

length>10	: 1
fracOfAlpha	: 0.85
contains_@	: 1
endsWith_.com	: 1
endsWith_.org	: 0

Feature vector notation

Mathematically, feature vector doesn't need feature names:





Definition: feature vector

For an input x, its feature vector is:

$$\phi(x) = [\phi_1(x), \ldots, \phi_d(x)].$$

Think of $\phi(x) \in \mathbb{R}^d$ as a point in a high-dimensional space.

Weight vector

Weight vector: for each feature j, have real number w_j representing contribution of feature to prediction

length>10	:-1.2
fracOfAlpha	:0.6
contains_@	:3
endsWith_.com	:2.2
endsWith_.org	:1.4
...	

Linear predictors

Weight vector $\mathbf{w} \in \mathbb{R}^d$

length>10	:-1.2
fracOfAlpha	:0.6
contains_@	:3
endsWith_.com	:2.2
endsWith_.org	:1.4

Feature vector $\phi(x) \in \mathbb{R}^d$

length>10	:1
fracOfAlpha	:0.85
contains_@	:1
endsWith_.com	:1
endsWith_.org	:0

Score: weighted combination of features

$$\mathbf{w} \cdot \phi(x) = \sum_{j=1}^{d} w_j \phi(x)_j$$

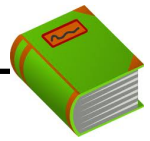
Example: $-1.2(1) + 0.6(0.85) + 3(1) + 2.2(1) + 1.4(0) = 4.51$

Linear predictors

Weight vector $\mathbf{w} \in \mathbb{R}^d$

Feature vector $\phi(x) \in \mathbb{R}^d$

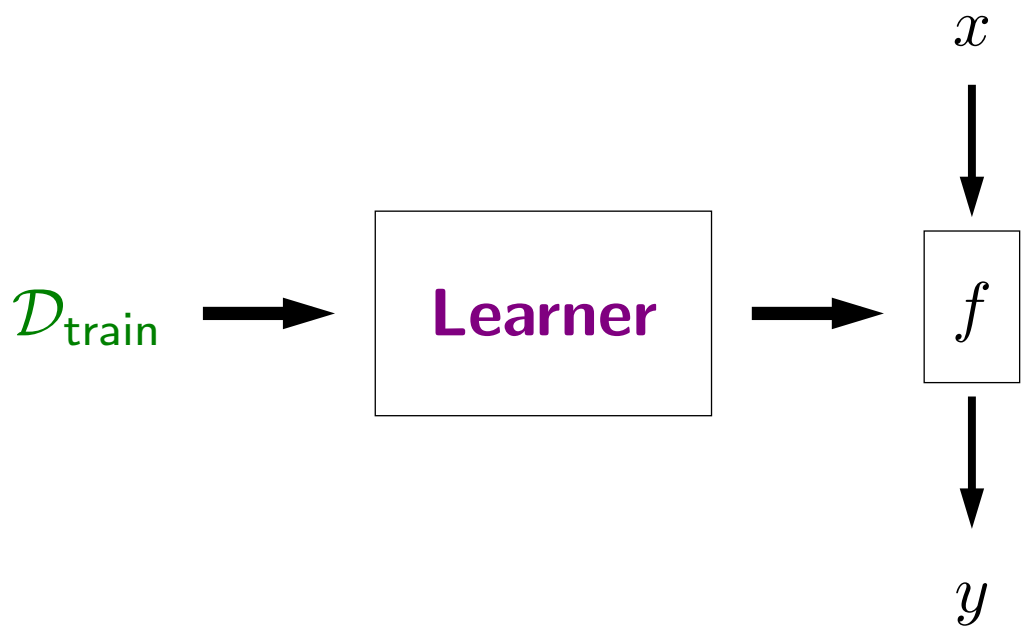
For binary classification:

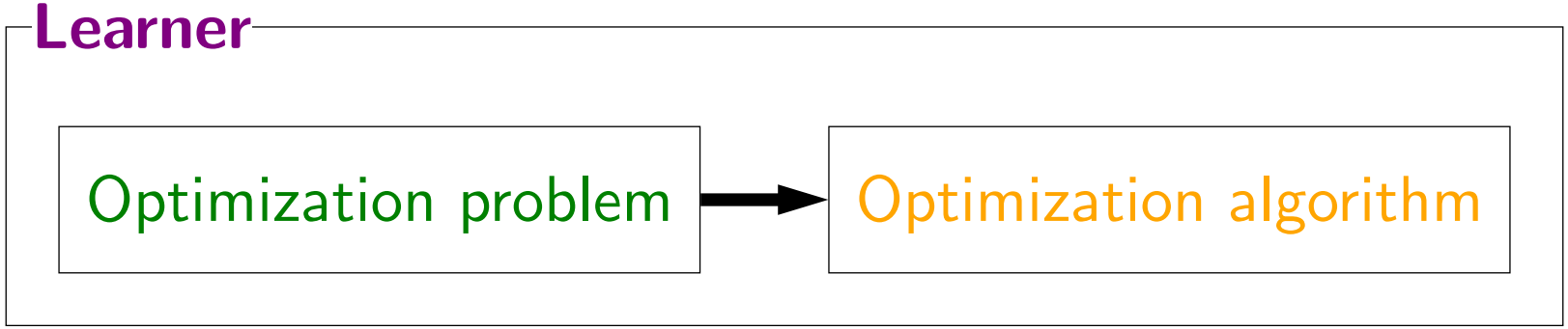


Definition: (binary) linear classifier

$$f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x)) = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \phi(x) > 0 \\ -1 & \text{if } \mathbf{w} \cdot \phi(x) < 0 \\ ? & \text{if } \mathbf{w} \cdot \phi(x) = 0 \end{cases}$$

Framework


x
$\mathcal{D}_{\text{train}}$
Learner
f
y


Learner
Optimization problem
Optimization algorithm

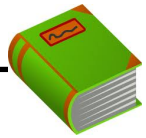
- Learning by optimizing a cost function:

$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\underbrace{\{y(x_i,\mathbf{w}) - t_i\}^2}_{\text{loss function}} + \underbrace{\frac{\lambda}{2}\|\mathbf{w}\|^2}_{\text{regularization}}$$

- In general Minimize with respect to $f \in \mathcal{F}$

$$\sum_{i=1}^{N} l\left(f(x_i), y_i\right) + \lambda R\left(f\right)$$

Loss functions



Definition: loss function

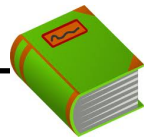
A loss function $\text{Loss}(x, y, \mathbf{w})$ quantifies how unhappy you would be if you used $\mathbf{w}$ to make a prediction on x when the correct output is y. It is the object we want to minimize.

Binary classification

Example: $\mathbf{w} = [2, -1], \phi(x) = [2, 0], y = -1$

Recall the binary classifier:

$$f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))$$

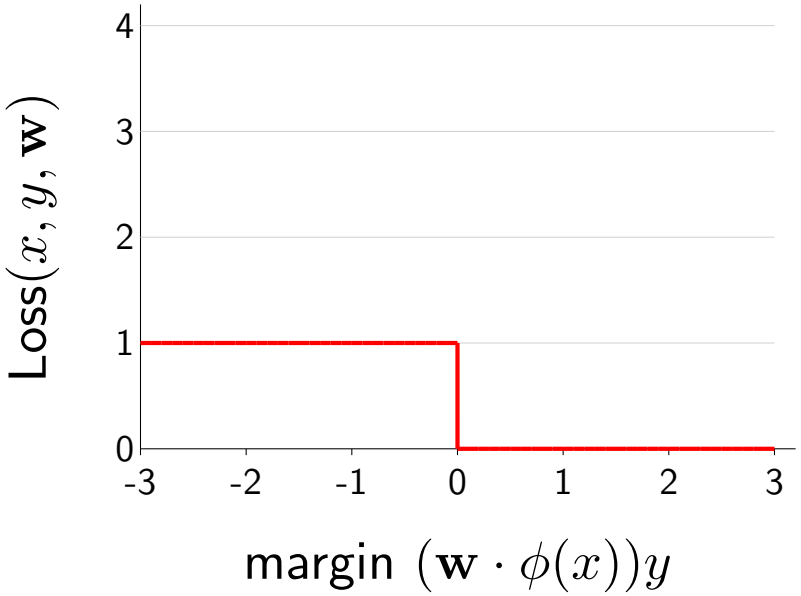


Definition: zero-one loss

$$\text{Loss}_{\text{0-1}}(x, y, \mathbf{w}) = \mathbf{1}[f_{\mathbf{w}}(x) \neq y]$$

$$= \mathbf{1}[\underbrace{(\mathbf{w} \cdot \phi(x))y}_{\text{margin}} \leq 0]$$

Binary classification

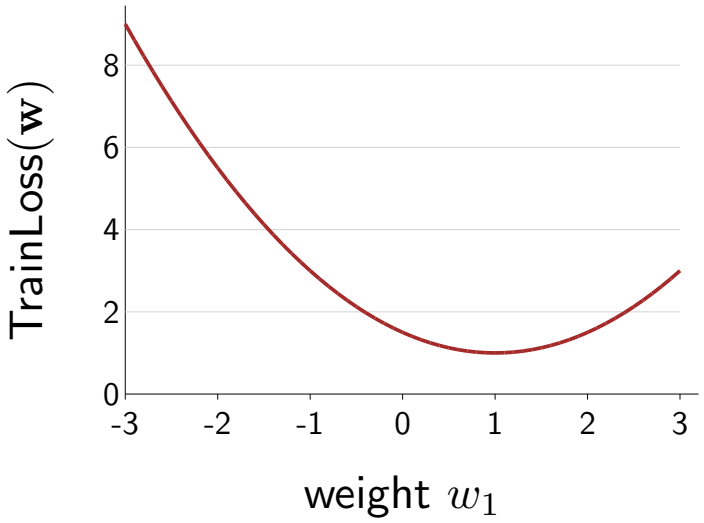


$$\text{Loss}_{0\text{-}1}(x, y, \mathbf{w}) = \mathbf{1}[(\mathbf{w} \cdot \phi(x))y \leq 0]$$

Optimization problem

Objective: $\min_{\mathbf{w} \in \mathbb{R}^d} \text{TrainLoss}(\mathbf{w})$

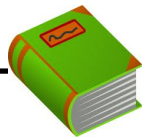
$\mathbf{w} \in \mathbb{R}$



$\mathbf{w} \in \mathbb{R}^2$

[gradient plot]

How to optimize?



Definition: gradient

The gradient $\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$ is the direction that increases the loss the most.

Algorithm: gradient descent

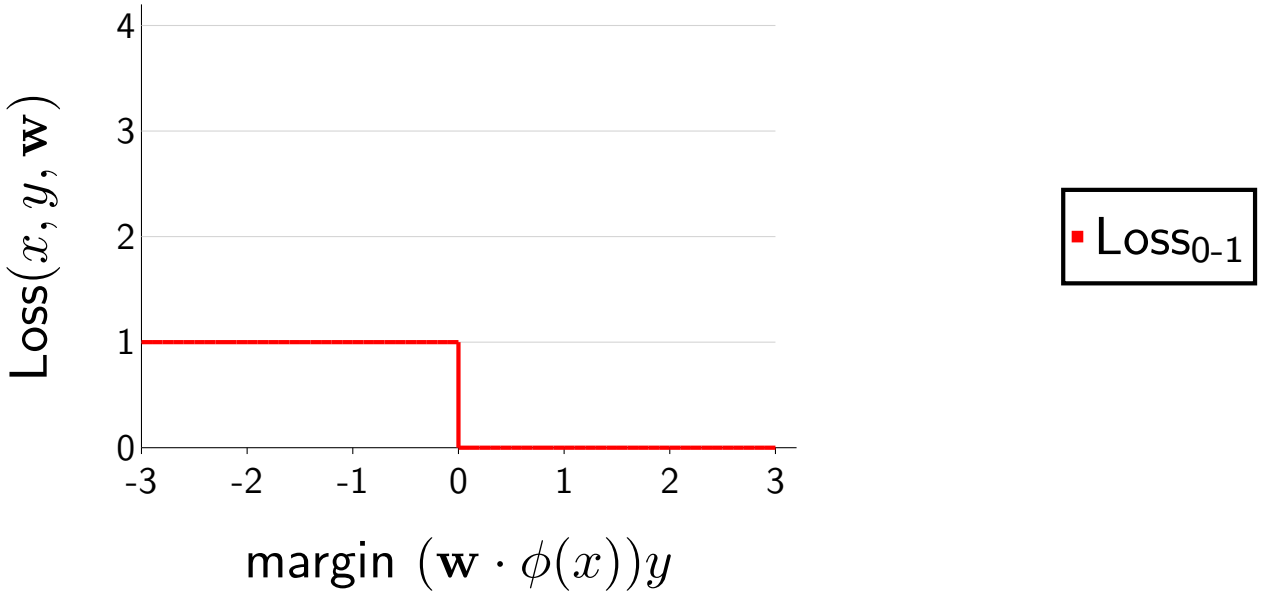
Initialize $\mathbf{w} = [0, \ldots, 0]$

For $t = 1, \ldots, T$:

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \underbrace{\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})}_{\text{gradient}}$$

Zero-one loss

$$\text{Loss}_{0\text{-}1}(x, y, \mathbf{w}) = \mathbf{1}[(\mathbf{w} \cdot \phi(x))y \leq 0]$$

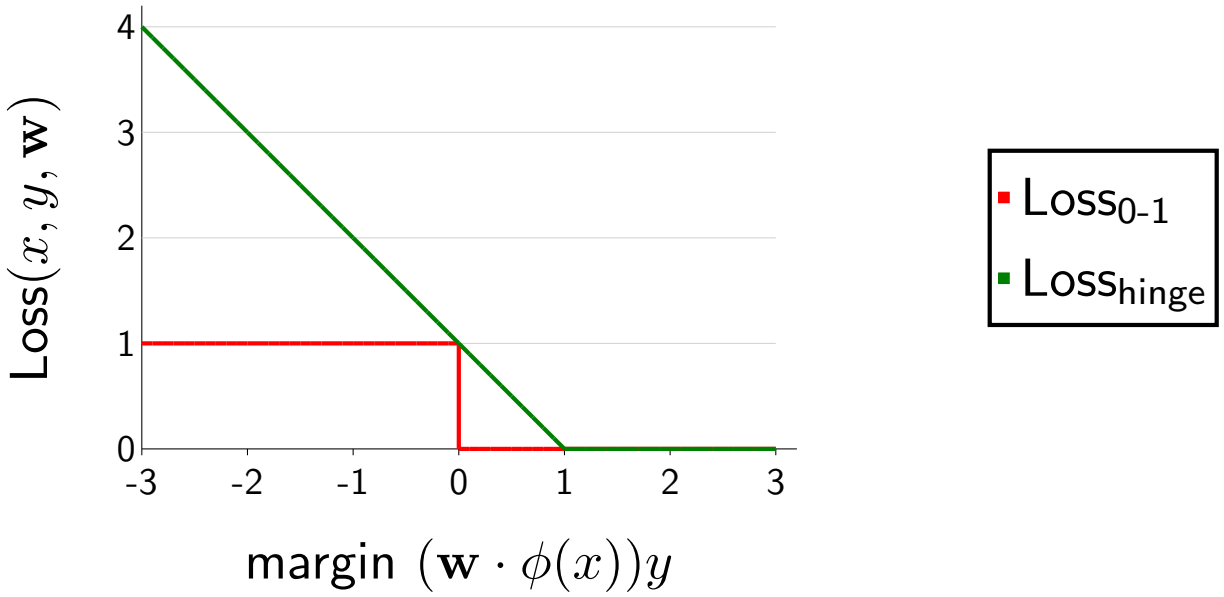


Problems:

- Gradient of Loss$_{0\text{-}1}$ is 0 everywhere, SGD not applicable
- Loss$_{0\text{-}1}$ is insensitive to how badly model messed up

Support vector machines*

$$\text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \max\{1 - (\mathbf{w} \cdot \phi(x))y, 0\}$$



- Intuition: hinge loss upper bounds 0-1 loss, has non-trivial gradient
- Try to increase margin if less than 1

Learning to rank

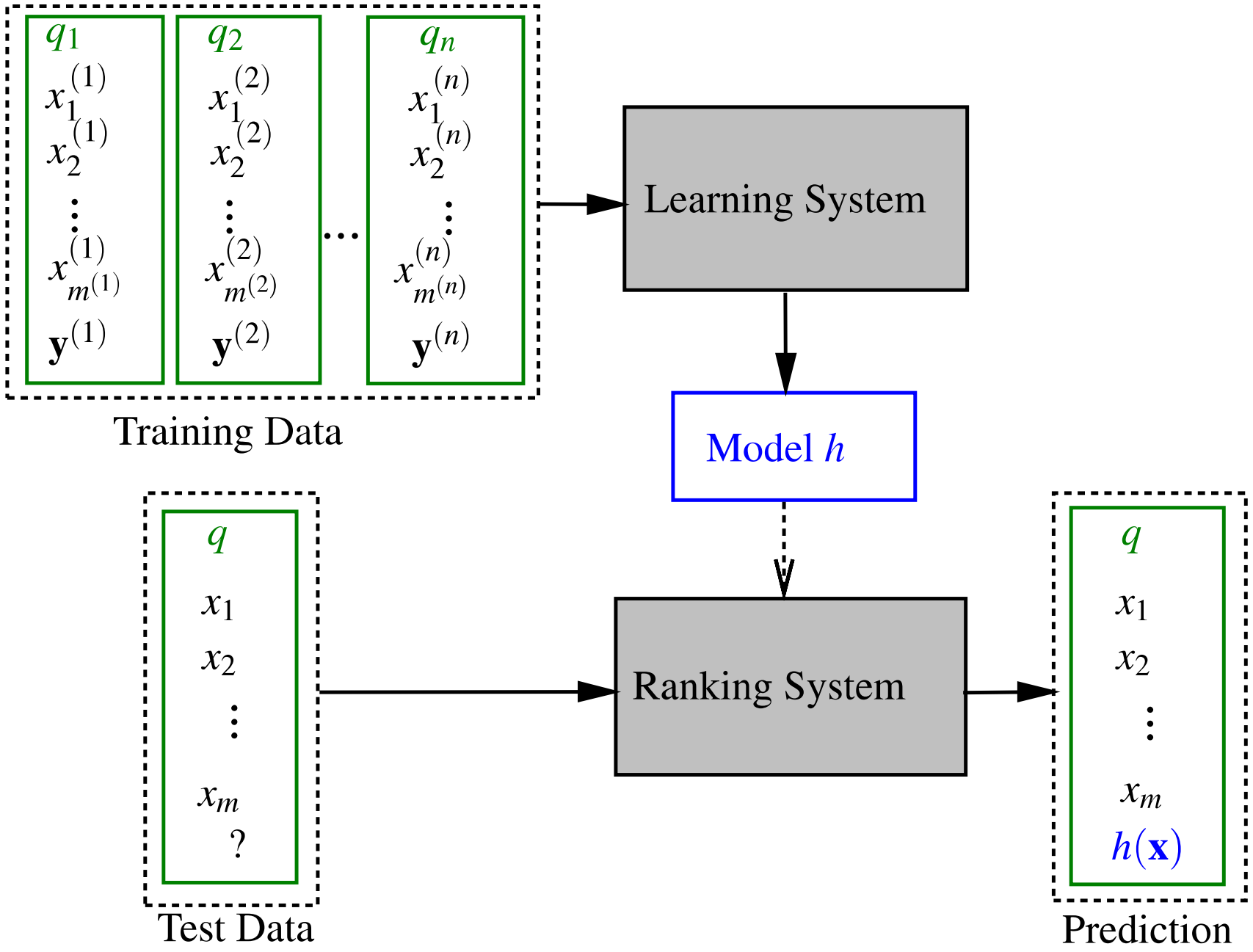


Fig. 1.6 Learning-to-rank framework

Machine learning for IR ranking

- This "good idea" has been actively researched – and actively deployed by major web search engines – in the last 7–10 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")

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 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 $g(r|d,q) \geq 0$

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

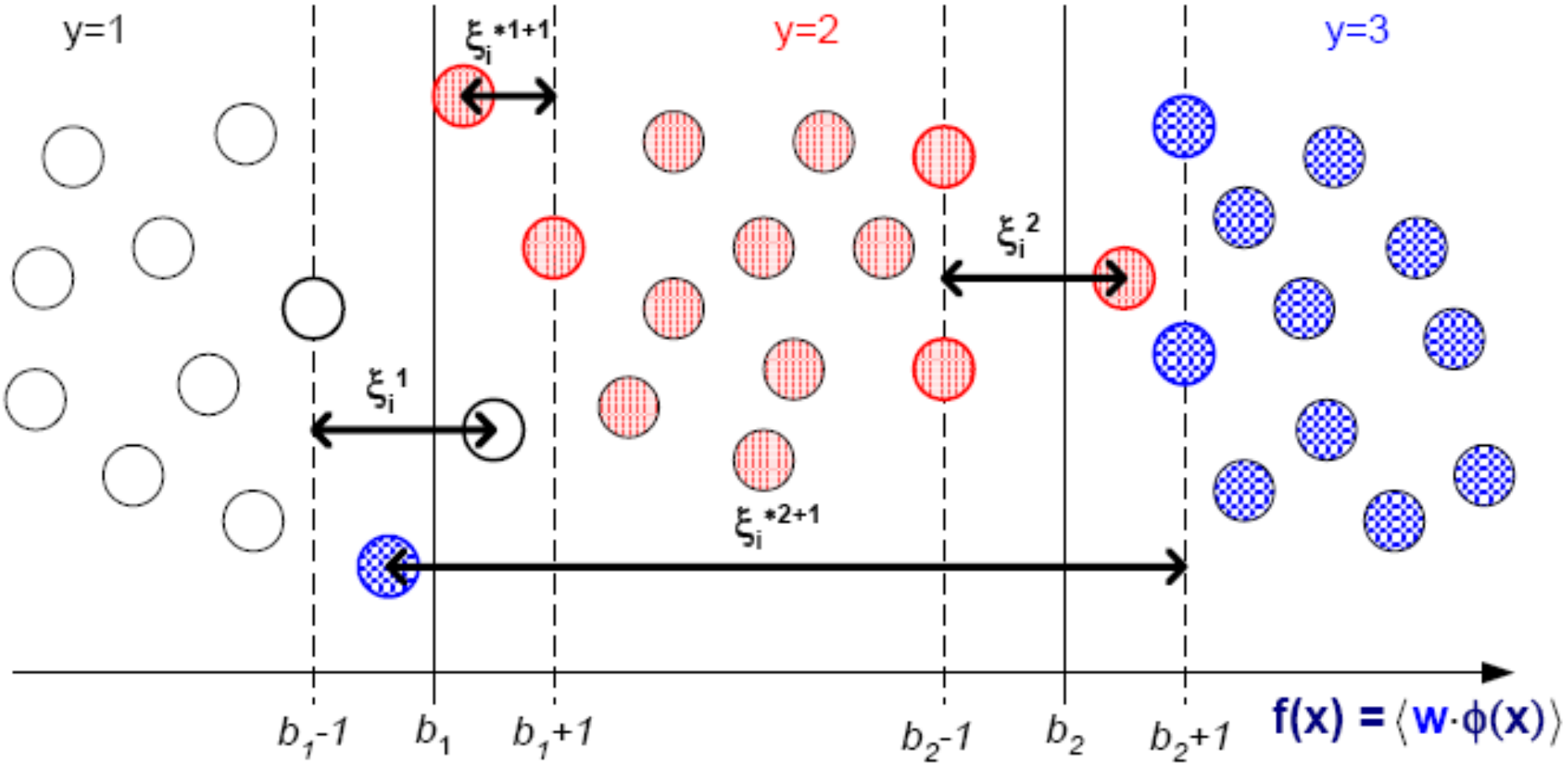
"Learning to rank"

- Assume a number of categories ***C*** of relevance exist
  - These are totally ordered: $c_1 < c_2 < \ldots < c_J$
  - 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

Point-wise learning

- Goal is to learn a threshold to separate each rank



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 in an expanded space
- We want a ranking function f such that

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

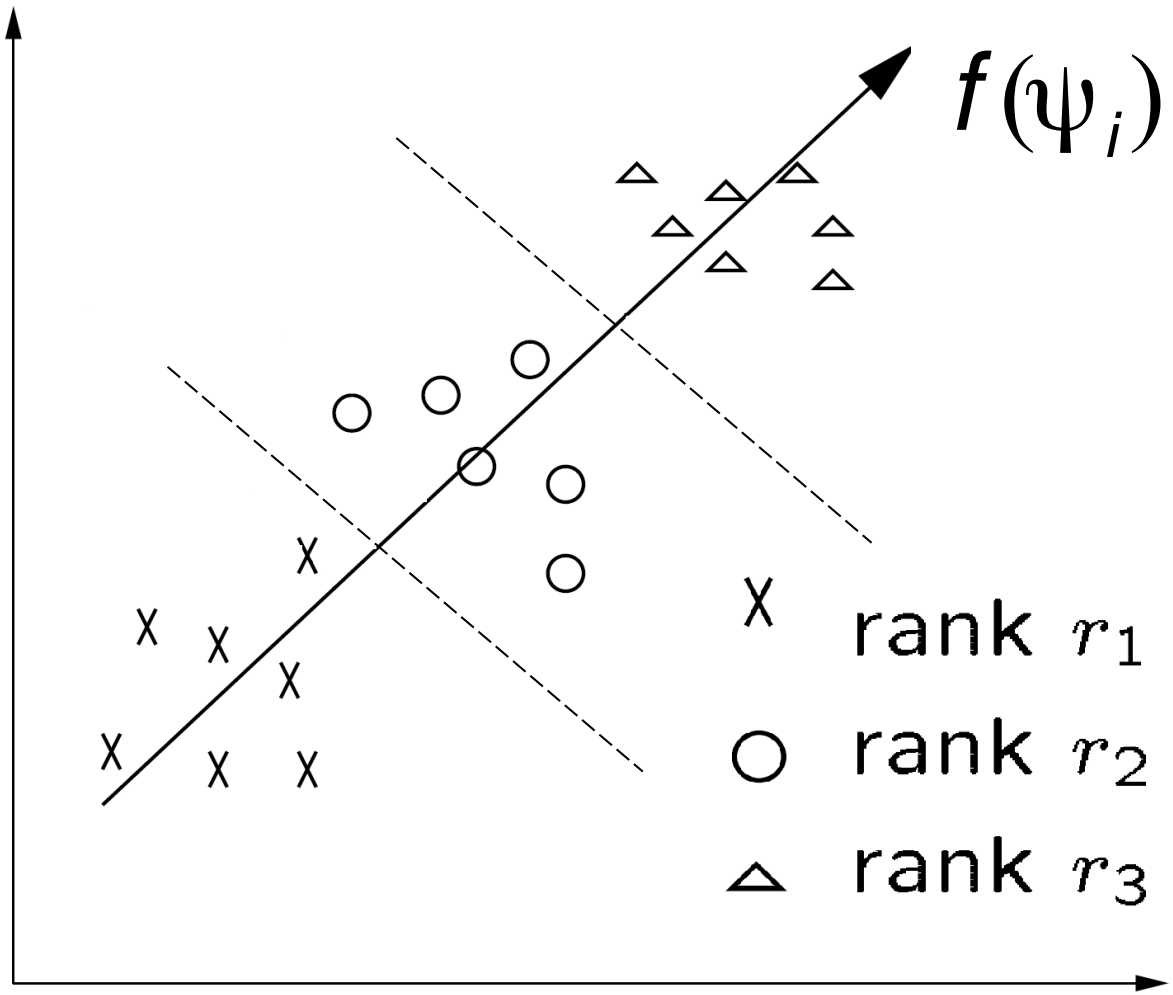
- … or at least one that tries to do this with minimal error
- Suppose that f is a linear function

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

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 $c_i > c_k$ iff $f(\psi_i) > f(\psi_k)$ and $f(\psi_i) = \mathbf{w} \bullet \psi_i$):

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

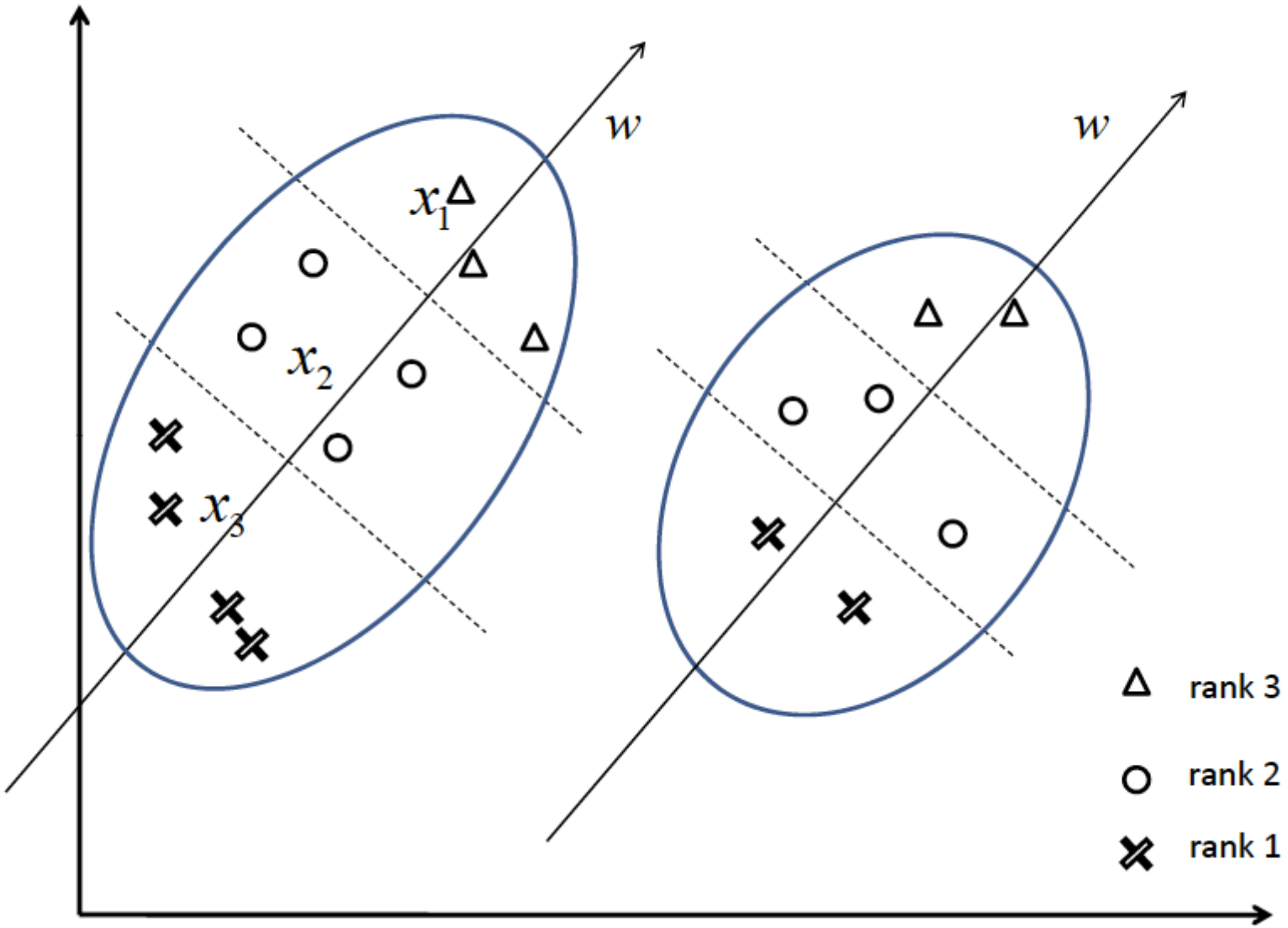
- Let us then create a new instance space from such pairs:

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

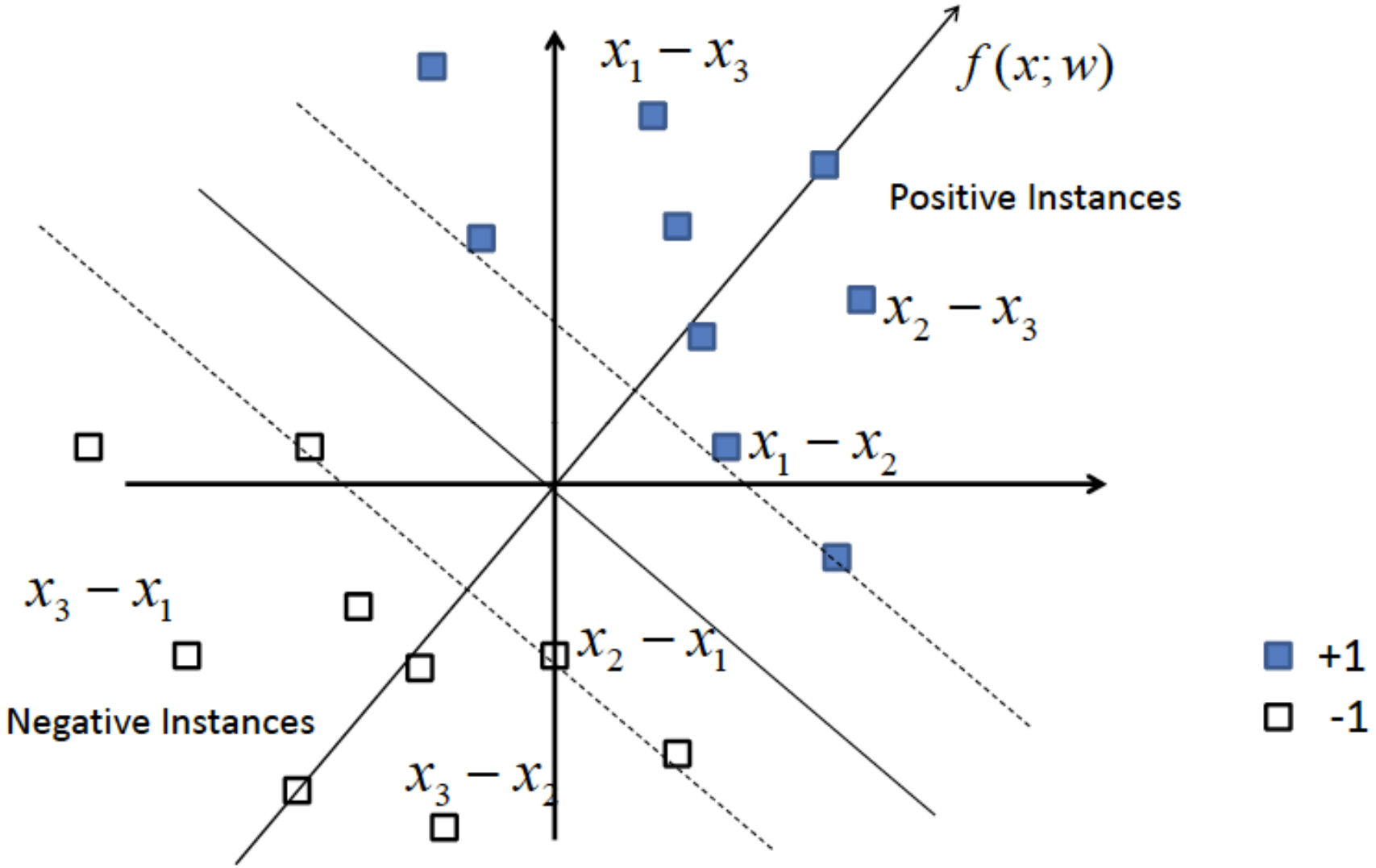
$$z_u = +1, 0, -1 \text{ as } c_i >, =, < c_k$$

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

Two queries in the original space



Two queries in the pairwise space



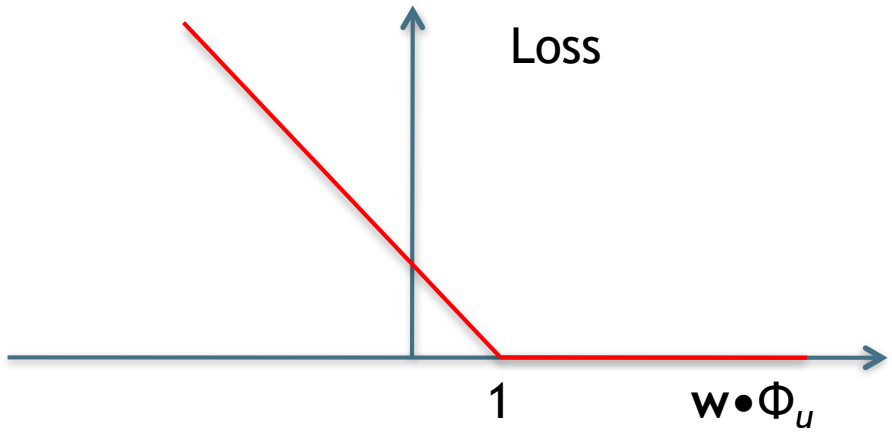
Aside: The SVM loss function

- The reformulation

Hinge loss | Regularizer of ||w||

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

- shows that an SVM can be thought of as having an empirical **"hinge" loss** combined with a **weight regularizer** (smaller weights are preferred)



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 standard IR 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

The ranking SVM fails to model the IR problem well …

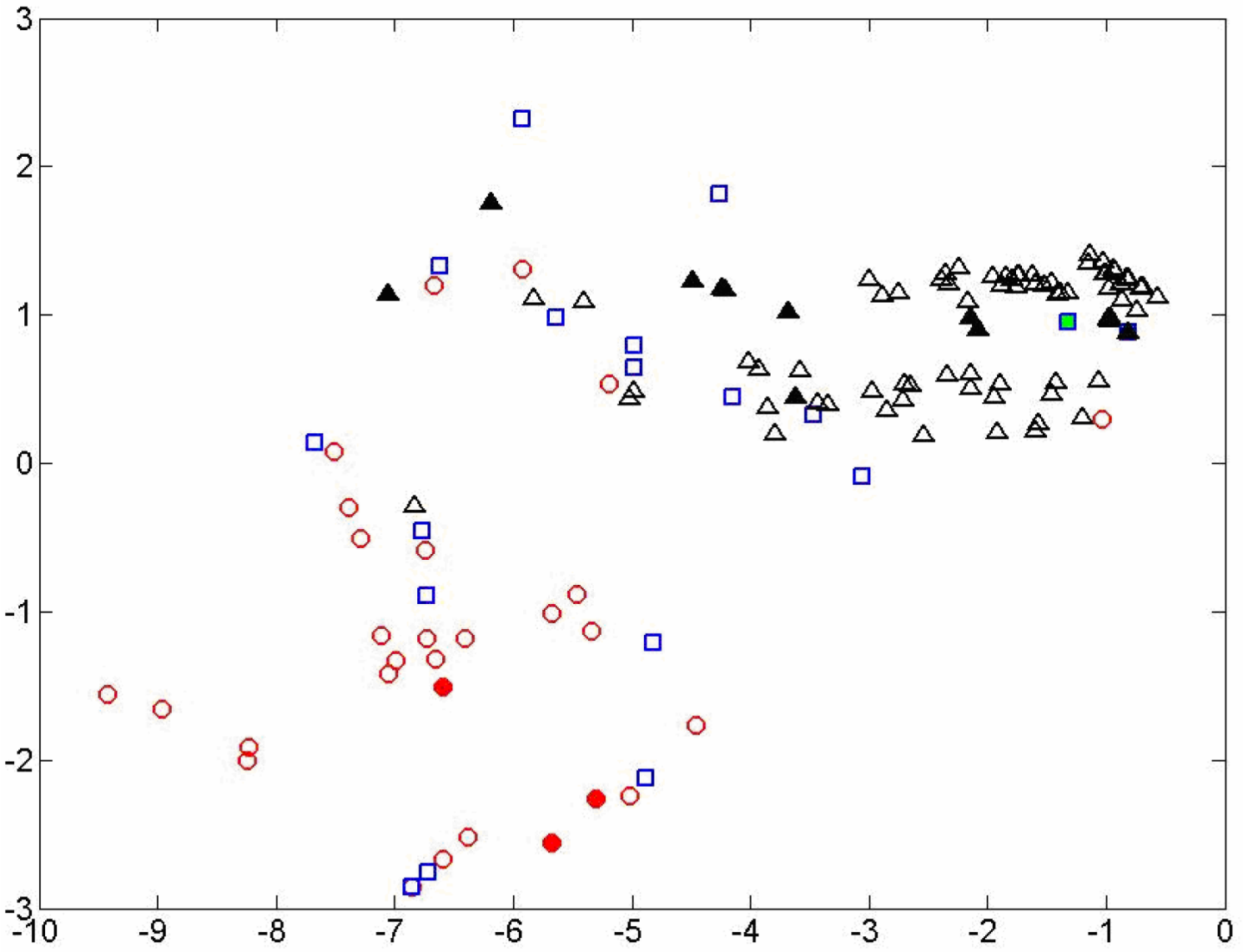
1. Correctly ordering the most relevant documents is crucial to the success of an IR system, while misordering less relevant results matters little
   - The ranking SVM considers all ordering violations as the same
2. Some queries have many (somewhat) relevant documents, and other queries few. If we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

Experiments use LETOR test collection

- https://www.microsoft.com/en-us/research/project/letor-learning-rank-information-retrieval/
  - From Microsoft Research Asia
  - An openly available standard test collection with pregenerated features, baselines, and research results for learning to rank
- It's availability has really driven research in this area
- OHSUMED, MEDLINE subcollection for IR
  - 350,000 articles
  - 106 queries
  - 16,140 query-document pairs
  - 3 class judgments: Definitely relevant (DR), Partially Relevant (PR), Non-Relevant (NR)
- TREC GOV collection (predecessor of GOV2, cf. *IIR* p. 142)
  - 1 million web pages
  - 125 queries

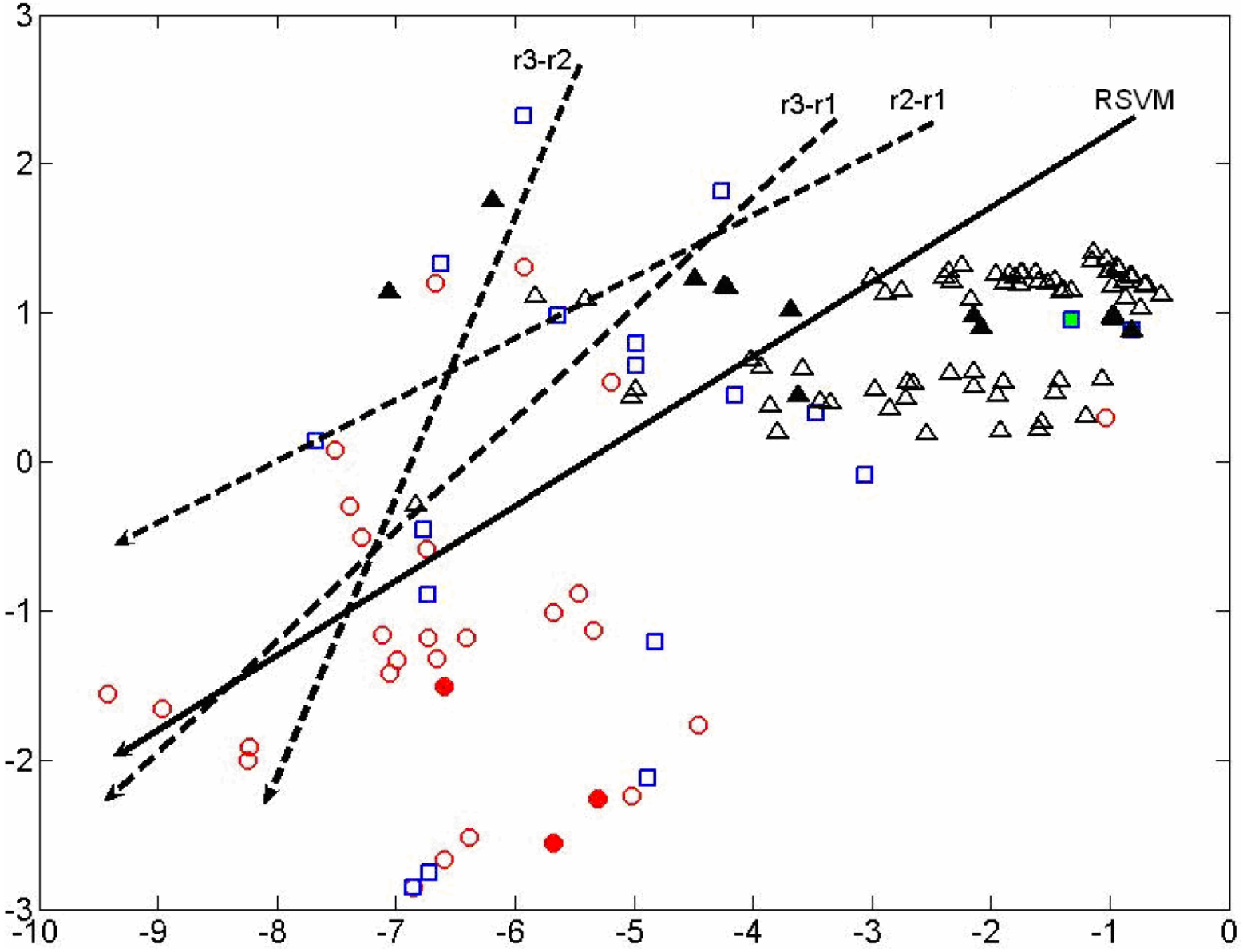
Principal components projection of 2 queries

[solid = q12, open = q50; circle = DR, square = PR, triangle = NR]

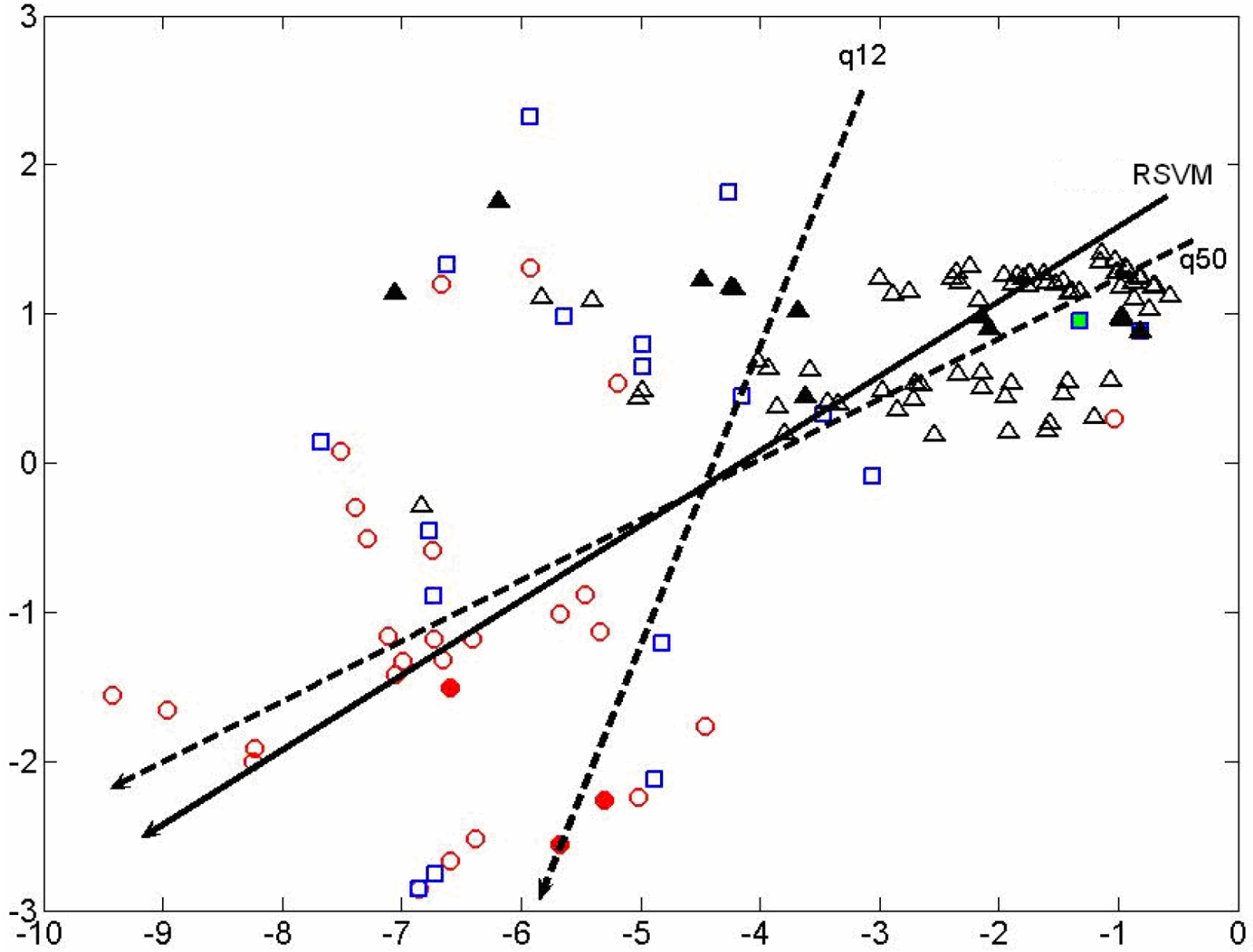


Ranking scale importance discrepancy

[r3 = Definitely Relevant, r2 = Partially Relevant, r1 = Nonrelevant]



Number of training documents per query discrepancy [solid = q12, open = q50]



Recap: Two Problems with Direct Application of the Ranking SVM

- Cost sensitivity: negative effects of making errors on top ranked documents

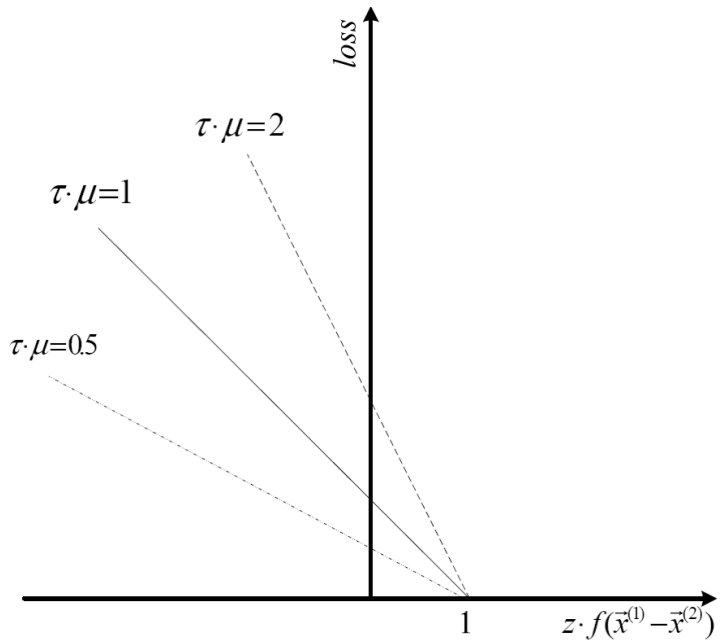
  d: *definitely relevant*, p: *partially relevant*, n: *not relevant*
  ranking 1: p d p n n n n
  ranking 2: d p n p n n n

- 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

These problems are solved with a new Loss function

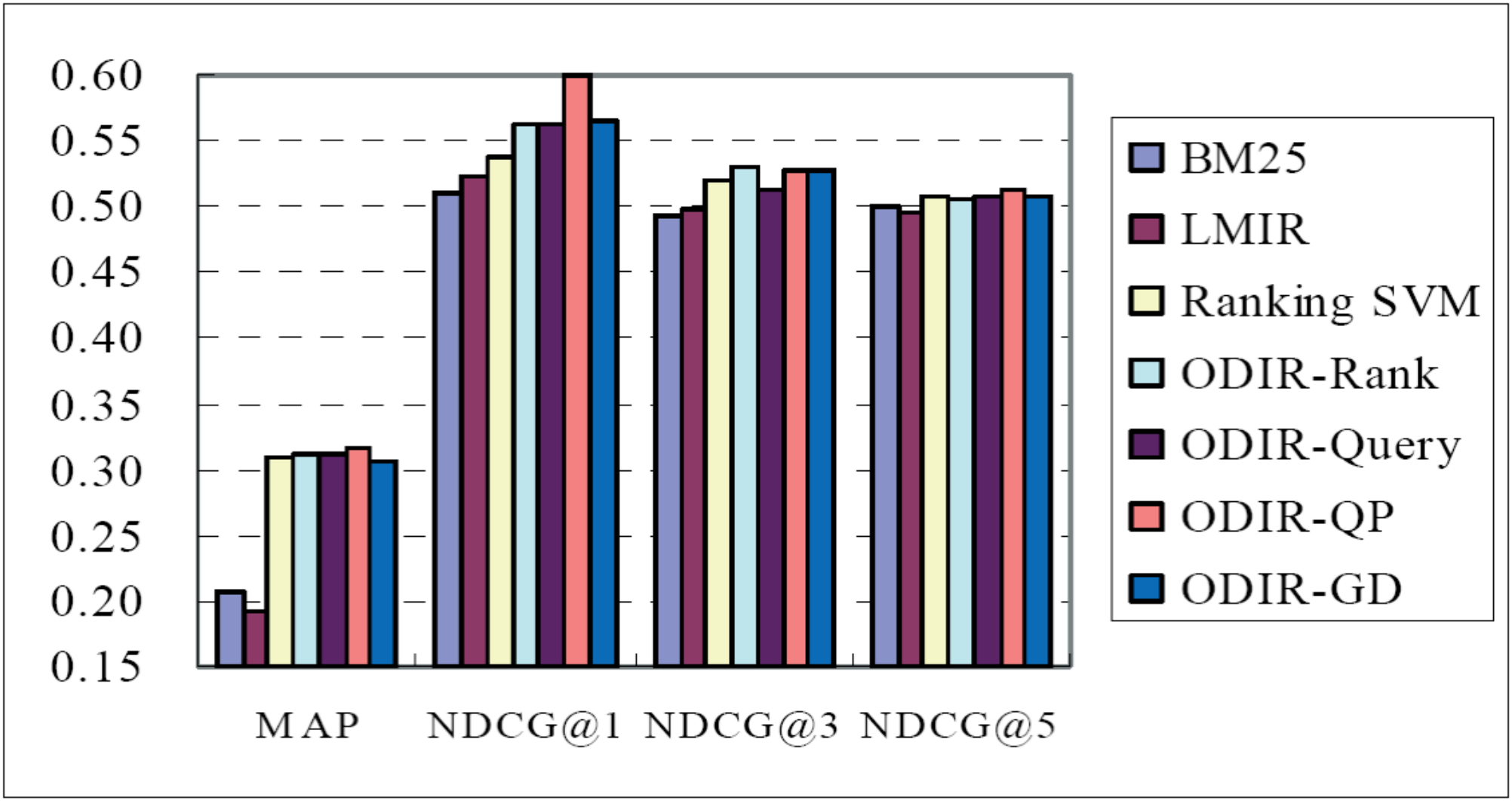
- τ weights for type of rank difference
  - Estimated empirically from effect on NDCG
- μ weights for size of ranked result set
  - Linearly scaled versus biggest result set



Experiments

- OHSUMED (from LETOR)
- Features:
  - 6 that represent versions of tf, idf, and tf.idf factors
  - BM25 score (*IIR* sec. 11.4.3)

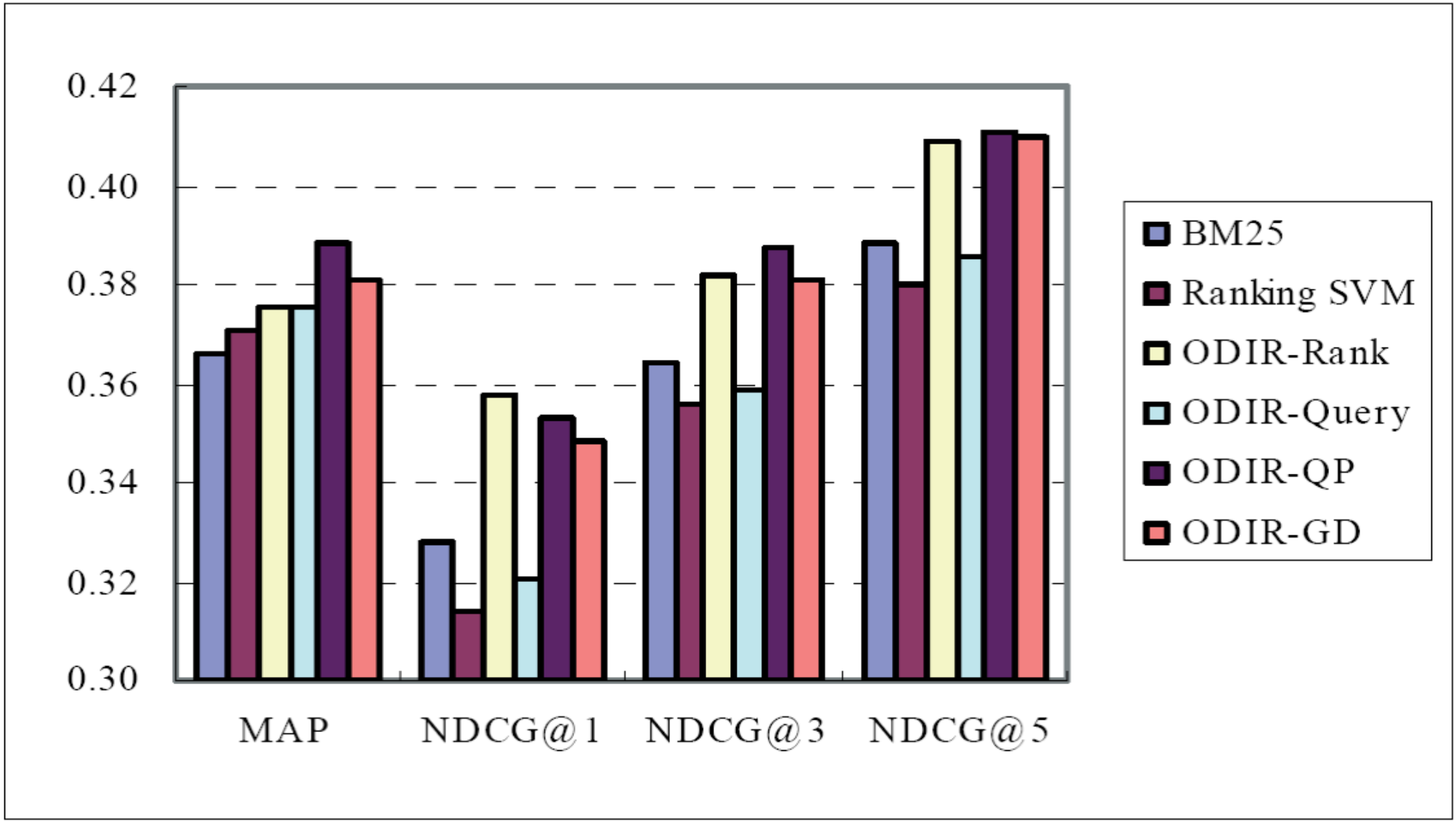
Experimental Results (OHSUMED)



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)



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 ☺

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.