CS6200
Information Retrieval

Jesse Anderton
College of Computer and Information Science
Northeastern University

Machine Learning in IR

- There is a lot of overlap between Machine Learning and Information Retrieval tasks.
- ML focuses on making predictions in the face of uncertainty. If those predictions involve an IR task, you are using ML for IR.
- Common applications include:
  - Ranking: Learning to Rank, etc.
  - Clustering: Grouping similar documents
  - Feature generation: e.g. Classifying documents by type (news, blogs, references, song lyrics, whatever)

but first, some probability…

Probability

Probability | Machine Learning
Learning to Rank | Features for Ranking

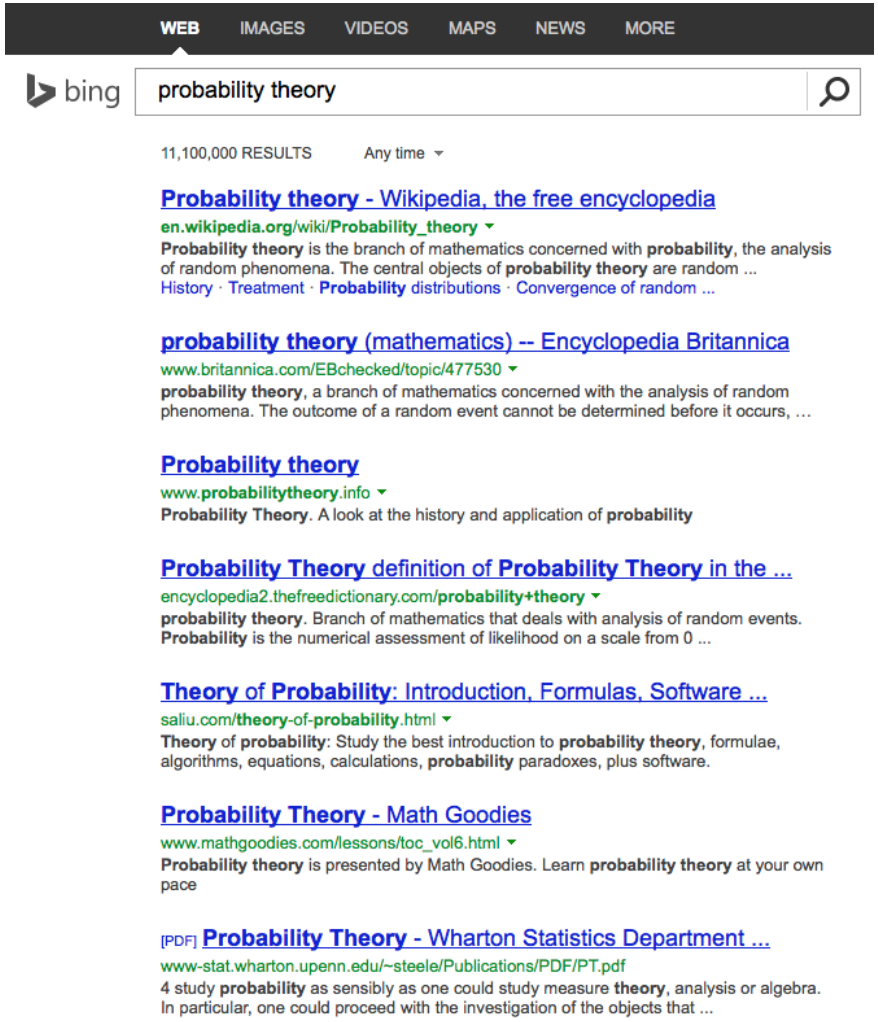
Random Experiments

- A **random experiment** is a process with some fixed set of possible outcomes, and whose outcomes are not deterministic (predictable).
- The set of all possible outcomes of a random experiment is its **sample space**.
- Each possible outcome has a non-negative **probability** of occurring. The sum of all outcomes' probabilities is one.

Swiss archer William Tell teaches son probability theory

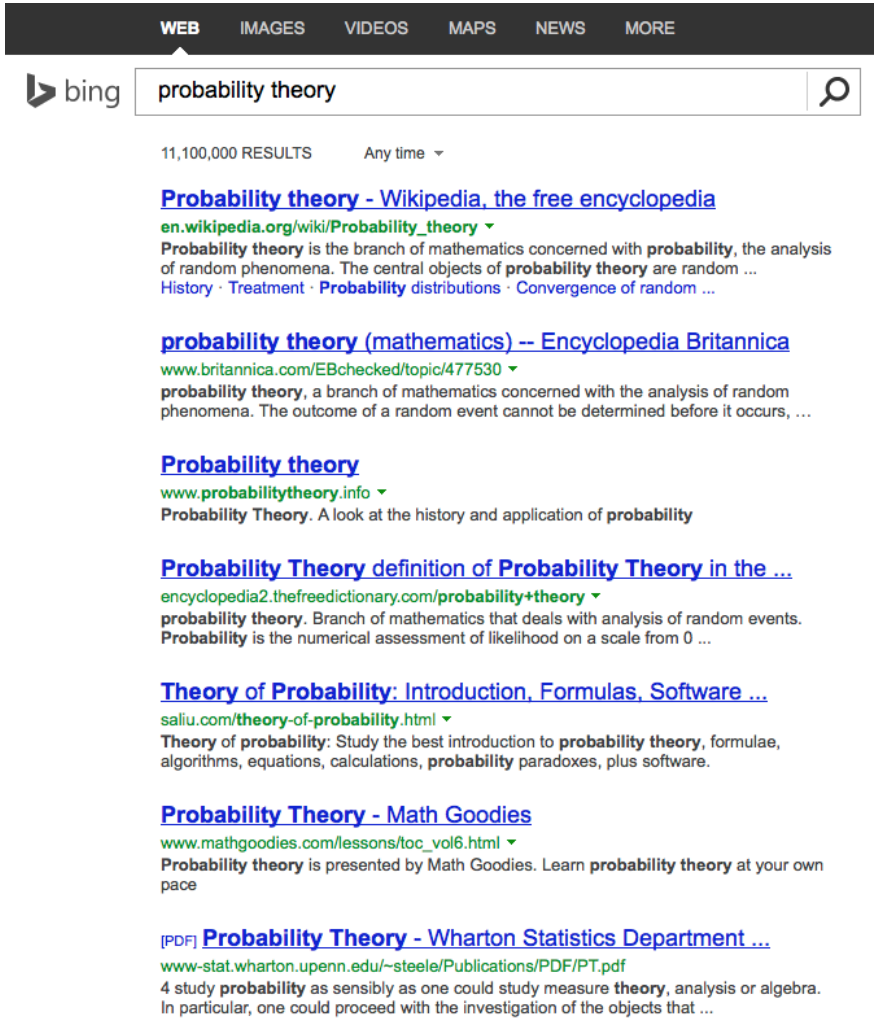
Random Events

- A **random event** is a subset of the sample space. Its probability is the sum of the probabilities of the outcomes it includes.
  - The entire sample space is a random event, with probability one.
  - Any single possible outcome is a random event.
  - “Nothing happening” is a random event, with probability zero.
- Example: If your sample space is the set of all Internet documents a random event might be getting a particular search result.



Random Variables

- A **random variable** is a function from random events to numbers.
- Suppose your random experiment is running a web search.
  - One **discrete random variable** is the total number of pages found.
  - One **continuous random variable** is the MAP of the resulting ranked list.



Expected Values

- Since the variable's value depends on a random event, which has some probability of occurring, any possible value of a random variable has some probability of occurring.
- The **expected value** of a random variable is the weighted sum of its possible values, where the weight is the probability of that value occurring.
- If you repeated the experiment many times and took the mean of the random variable's values, that mean will approach the expected value.

For discrete R.V. X with possible outcomes $\{x_1, x_2, \dots\}$,

$$\mathbf{E}[X] = \sum_i x_i \cdot Pr(X = x_i)$$

For continuous R.V. Y with possible outcomes $\{y_1, y_2, \dots\}$ and density $f(y)$,

$$\mathbf{E}[Y] = \int_{-\infty}^{\infty} y_i \cdot f(y_i) dy$$

Expected Values

- The expected value of a fair die-roll is the mean face value: 3.5

$$\sum_i x_i \cdot Pr(X = x_i) = 1 \cdot 1/6 + 2 \cdot 1/6 + 3 \cdot 1/6 + 4 \cdot 1/6 + 5 \cdot 1/6 + 6 = 21/6$$

- In IR, AP is also the expected value of a random experiment:
  - *Random experiment:* Given a ranked list, select the rank k of a relevant document uniformly at random.
  - *Random event:* The rank k which was selected
  - *Random variable:* The precision at rank k
  - *Expected value:* The P@K value for each relevant document, multiplied by the change 1/R in R@K at that rank.

$$AP(\vec{r}, R) = 1/|R| \cdot \sum_{i:r_i \neq 0} P@k(\vec{r}, i)$$

Rules of Probability

- Random events are sets, and manipulated using set theory:
  - A given B: “I know the outcome is in B; is it in A?”

$$Pr(A|B) = \frac{\sum_{o:o\in A\wedge o\in B} Pr(o)}{\sum_{o:o\in B} Pr(o)}$$

  - A or B: “any outcome which is in either set A or set B”

$$\begin{aligned} Pr(A\vee B) &= \sum_{o:o\in A\vee o\in B} Pr(o) \\ &= Pr(A) + Pr(B) - Pr(A\wedge B) \end{aligned}$$

  - A and B: “any outcome which is in both A and B”

$$\begin{aligned} Pr(A\wedge B) &= \sum_{o:o\in A\wedge o\in B} Pr(o) \\ &= Pr(B) + Pr(A|B) \end{aligned}$$

Bayes' Rule

- **Bayes' Rule** is a key element of probabilistic modeling. It tells you how to update your probability estimate in response to new data.

- This allows you to start with a *prior* belief in A's probability Pr(A) and calculate a *posterior* belief Pr(A|B) based on learning that B occurred.

$$Pr(A|B) = \frac{Pr(B|A)Pr(A)}{Pr(B)}$$

Thomas Bayes

and now, on to…

Machine Learning

Probability | **Machine Learning**
Learning to Rank | Features for Ranking

What is Machine Learning?

Machine Learning is a collection of methods for using *data* to select a *model* which can make *decisions* in the face of uncertainty.

- The **data** could be anything: numbers, categories, time series, text, images, dates…
- The **models** are mathematical functions which can be tuned through parameters. They are often conditional probability distributions.
- The **decisions** are most often either predicting a number or predicting a category.

Data

docid	tf(tropical)	tf(fish)	tf(lincoln)	Rel?
d1	5	10	0	**Yes**
d2	0	3	15	**No**
d3	7	0	0	**Yes**
d4	0	2	3	**No**

$$\boldsymbol{X} = \begin{bmatrix} 5 & 10 & 0 \\ 0 & 3 & 15 \\ 7 & 0 & 0 \\ 0 & 2 & 3 \end{bmatrix} \quad \boldsymbol{Y} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Decisions

$$\text{Is } \begin{bmatrix} 3 & 2 & 7 \end{bmatrix} \text{ relevant?}$$

$$\text{Which documents are similar?}$$

Data

- The data are generally treated as records drawn independently and identically distributed (IID) from a sample space.
  - ➡ You build a training set by drawing many records.
  - ➡ You may also build other collections at this time, e.g. a testing set to test predictions on data you didn't train with.
- The ability to make accurate predictions depends on whether your training data represents future records adequately.
- Choosing better features and increasing the amount of training data often make a bigger difference in prediction quality than improving your learning algorithm.
- What can go wrong with your data?

Data

- Your prediction quality is a direct result of how well the features you choose are correlated with the value you're trying to predict (see Fano's Inequality).
- If your sample is too small, it can't capture all the nuances ("black swan" events).
- If your sample isn't independent, it may overrepresent some type at the expense of some other type.
- If the distribution of feature values changes over time, your training data may work now and not work later.

  ➡ Does the average quality of Wikipedia content change over time? How does this affect the utility of a "page URL" feature?

Models

- A ML **model** is a mathematical function with appropriate domain (a record drawn from the sample space) and range (the type of value you're trying to predict).

- We generally use multivariate functions, where some variables ("parameters", or θ) are chosen by the learning method and others are inputs from the data.

  ➡ A linear model: $f(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_i \theta_i \cdot x_i$

  ➡ A probabilistic model: $p(y|\boldsymbol{x}, \boldsymbol{\theta}) = \frac{1}{1 + e^{\sum_i \theta_i \cdot x_i}}$

Models

- All else being equal, a better model:
  - ➡ Is *flexible* – can adapt to different kinds of data. This is a tradeoff: if you have a lot of data, you can use a simple, flexible model. If you don't, you often have to build more assumptions into the model to compensate.
  - ➡ Is *parsimonious* – uses few parameters. More parameters increase model flexibility. Too-flexible models memorize the training data, and don't work on future data ("overfitting").
  - ➡ Is *efficiently trainable* – you can mathematically prove that optimal parameters can be found, ideally in linear time in the number of training records. It's even better if you can later efficiently update it with new records ("online" or "adaptive" models).
  - ➡ Is *interpretable* – reveals something about the relationship between data and predictions.

Error Functions

- In order to choose the best parameters, we need to mathematically define what “best” means.
- We use an **error function** (aka **loss function**) to evaluate model predictions on certain data and parameters.
  - ➡ Sum squared error: $\sum_i \left(f(\boldsymbol{x_i}, \boldsymbol{\theta}) - y_i\right)^2$
  - ➡ Log loss: $-\sum_i \log p(y_i|\boldsymbol{x_i}, \boldsymbol{\theta})$

Parameter Estimation

- Once you know your model (function) and have selected an error function, you're ready to choose the best parameters.
- There are many methods to choose from that vary in applicability, difficulty of implementation, and speed of convergence.
  - ➡ Analytic solutions: Lagrange multipliers
  - ➡ Matrix-based optimization: least squares, singular value decomposition
  - ➡ Sampling-based methods: Monte Carlo Markov Chains, Gibbs sampling
  - ➡ Probabilistic inference: Expectation Maximization (EM), variational inference

What is Machine Learning?

"**Machine Learning** is a collection of methods for using *data* to select a *model* which can make *decisions* in the face of uncertainty."

Now we can say what it means to select a model. First, the analyst chooses:

- A set of features to represent the data
- A model (function) which maps a feature vector to a prediction value, given some parameters
- An error/loss function to tell you the quality of some particular parameters
- A parameter estimation method to select the ideal parameters

The estimation method then finds parameters that minimize the error function, given some training data.

Is Machine Learning Hard?

- It depends on what you want to do. The easy parts:
  - There are many ML libraries you can download and run without needing to know their details.
  - Many ML algorithms are simple to implement.
- The hard parts:
  - Developing a broad knowledge of ML methods, and when to use which.
  - Devising a custom model and parameter estimation method that outperforms out-of-the-box methods for your data or task.
  - Improving the state of the art in accuracy and speed of parameter estimation methods.
  - (Sometimes) Reading the under-explained math notation in ML papers :)

Let’s look at the main ML task used for IR.

Classification

Classification means choosing the correct label for a document.

- **Binary Classification** uses just two labels. You often choose based on whether the model output exceeds some threshold. (e.g. relevant or not)
- **Multi-class Classification** uses more than two labels. One way is to train a binary classifier for each label and pick the one with the highest predicted value. (e.g. relevance grades)
- **Multi-label Classification** can apply multiple labels to the same document. (Less useful for our task.)

Data

docid	tf(tropical)	tf(fish)	tf(lincoln)	Rel?
d1	5	10	0	**Yes**
d2	0	3	15	**No**
d3	7	0	0	**Yes**
d4	0	2	3	**No**

$$\boldsymbol{X} = \begin{bmatrix} 5 & 10 & 0 \\ 0 & 3 & 15 \\ 7 & 0 & 0 \\ 0 & 2 & 3 \end{bmatrix} \quad \boldsymbol{Y} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Classification

$$\text{Is } \begin{bmatrix} 3 & 2 & 7 \end{bmatrix} \text{ relevant?}$$

Binary Classification

- Let's see an example of binary classification in IR.
- We will train a classifier to predict whether a new document is relevant to a particular query.
- Our features are term frequency counts for our three-word vocabulary.
- We have four training examples.

Data

docid	tf(tropical)	tf(fish)	tf(lincoln)	Rel?
d1	5	10	0	**Yes**
d2	0	3	15	**No**
d3	7	0	0	**Yes**
d4	0	2	3	**No**

$$\boldsymbol{X} = \begin{bmatrix} 5 & 10 & 0 \\ 0 & 3 & 15 \\ 7 & 0 & 0 \\ 0 & 2 & 3 \end{bmatrix} \quad \boldsymbol{Y} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Decision

$$\text{Is } \begin{bmatrix} 3 & 2 & 7 \end{bmatrix} \text{ relevant?}$$

Binary Classification

- We will use a method called *logistic regression*.
- Our model, shown on the right, is an exponential function of a linear combination of features.
- When the sum is a lot less than zero, the prediction is nearly zero. When it's a lot more, the prediction is nearly one.
- The parameters choose how quickly the transition from zero to one occurs, and which features matter more.
- We will use log loss for our error function.

Model

$$p(y|\boldsymbol{x}, \boldsymbol{\theta}) = \frac{1}{1 + e^{\sum_i \theta_i \cdot x_i}}$$

Error Function

$$-\sum_i \log p(y_i|\boldsymbol{x_i}, \boldsymbol{\theta})$$

Binary Classification

- After we fit the data, we get the parameters:

$$\theta \approx \begin{bmatrix} 0.478 & 0.079 & -0.534 \end{bmatrix}$$

- Feature one is evidence for relevance, three evidence against, and two more or less neutral.
- You can see the predicted value for each training record, and the total log loss on the right.
- Why isn't this perfect?

Predictions for Data

$$\boldsymbol{X} = \begin{bmatrix} 5 & 10 & 0 \\ 0 & 3 & 15 \\ 7 & 0 & 0 \\ 0 & 2 & 3 \end{bmatrix} \hat{\boldsymbol{Y}} \approx \begin{bmatrix} 0.490 \\ 0.269 \\ 0.491 \\ 0.308 \end{bmatrix}$$

Log Loss

$$-\sum_i \lg \hat{Y}_i$$

$$\approx 1.028 + 1.894 + 1.024 + 1.698$$

$$= 5.644$$

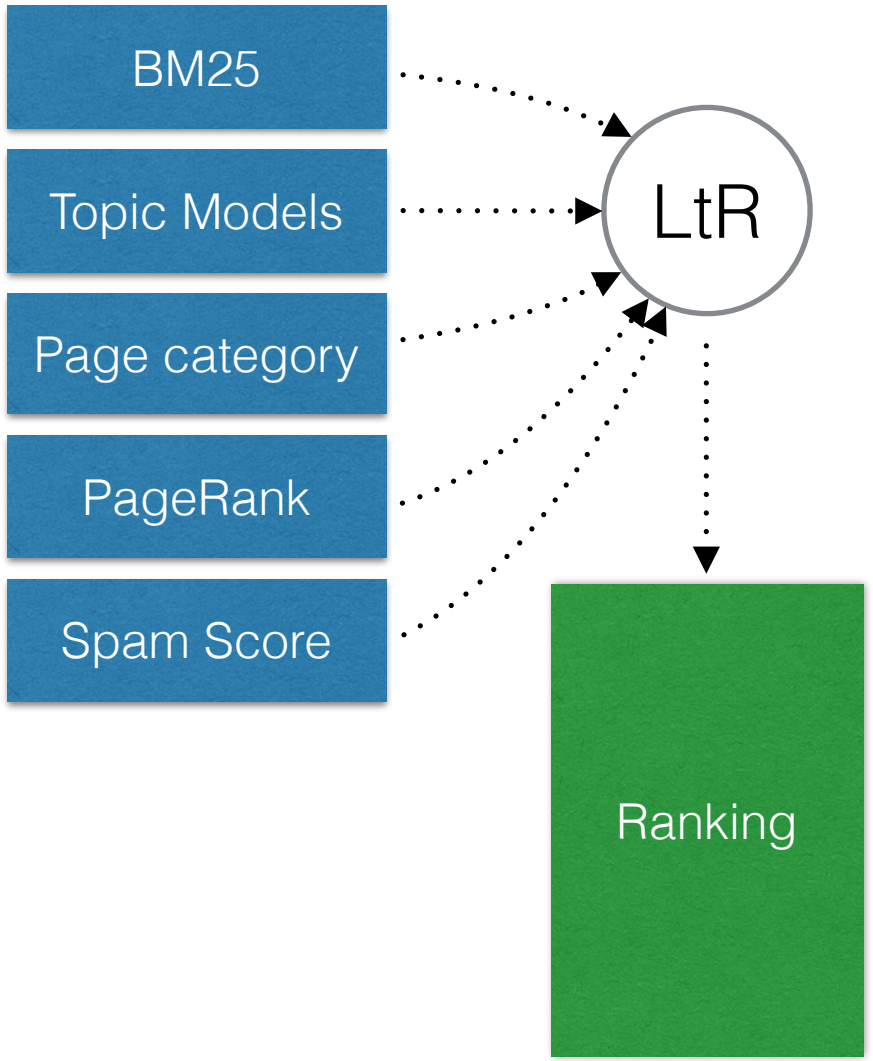
and now, IR at last

Learning to Rank

Probability | Machine Learning
Learning to Rank | Features for Ranking

Learning to Rank

- A key challenge in IR is combining evidence from multiple sources to produce a quality document ranking.
- Making predictions from different features is a natural setting for ML research, and dozens of methods have been developed.
- These methods are collectively known as **Learning to Rank** methods.



Ranking as Classification

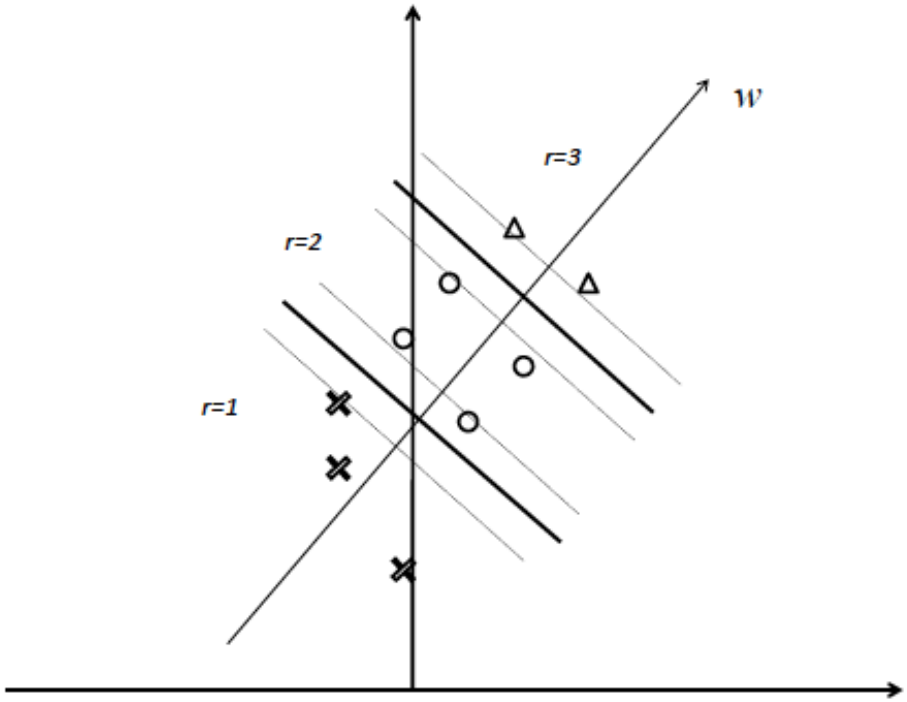
- There are a lot of ways to cast ranking as a classification problem. The three general approaches are:
  - **Pointwise**: predict the relevance grade for each document
  - **Pairwise**: given two documents, predict which is more relevant
  - **Listwise**: predict a relevance grade for each document, then optimize for some metric of the implied ranking

Pointwise LtR

- Document features are combined into a feature vector, and standard ML techniques are used to predict a relevance grade.
- Advantages:
  - Easy to implement: can use out-of-the-box methods (though tailored methods also exist)
  - Easy to interpret: you can look at mistakes on a document-by-document basis
- Disadvantages:
  - Ignores the list structure of ranking (which is the final output)
  - Hard to map goal onto evaluation measures (MAP, NDCG, etc.)

Example: OC SVM

- Support Vector Machines are a standard ML technique which find *support vectors* to separate your data.
- Support vectors are tangent to hyperplanes that are between data points of different classes, and as far as possible from those points.
- In SVM for Ordinal Classification, we search for *parallel* hyperplanes to establish the relevance grades.



Shapes represent documents
w is the support vector
The bold lines are the hyperplanes

Pointwise Methods

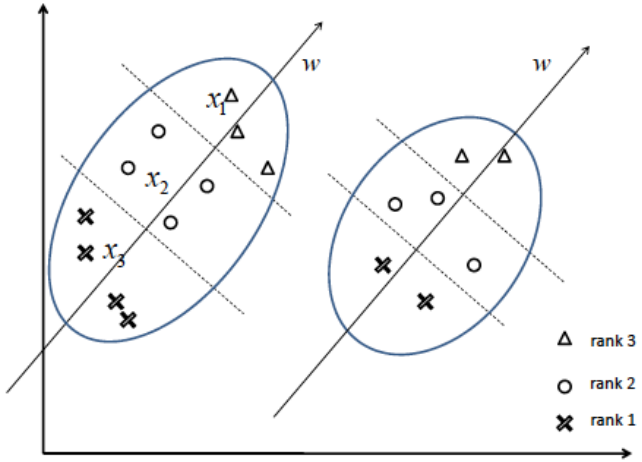
- Example methods for further reading:
  - Prank – Pranking with Ranking, NIPS 2001
  - OC SVM – Ranking with large margin principle: Two approaches, NIPS 2002
  - Subset Ranking – Subset Ranking using Regression, COLT 2006
  - McRank – McRank: Learning to rank using multiple classification and gradient boosting, NIPS 2007

Pairwise LtR

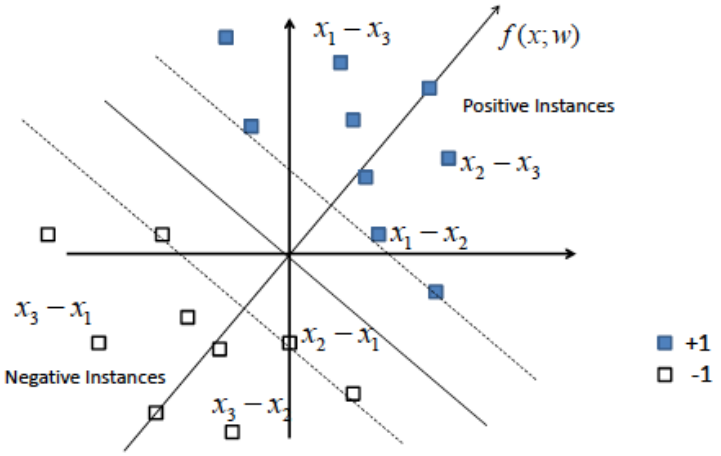
- Features represent a pair of documents, rather than a single document. (For instance, use difference between individual document features.)
- Ranking is done using a sorting algorithm, with pairwise classification used to compare docs.
- Advantages:
  - Works better than Pointwise in many cases
  - Can distinguish between documents which Pointwise would assign to the same class
- Disadvantages:
  - Often no direct relationship to IR evaluation measures

Example: Ranking SVM

- Given a document collection with relevance grades, you form pairwise data by choosing pairs of documents with different relevance grades and subtracting the features of one from the other.
- If the subtracted document has a smaller grade, the label is +; otherwise, it is -.



Ranking for two queries



Transformed to pairwise classification

Pairwise Methods

- Example methods for further reading:
  - Ranking SVM – Large Margin Rank Boundaries for Ordinal Regression, MIT Press 2000
  - RankBoost – An efficient boosting algorithm for combining preferences, JMLR 2003
  - RankNet – Learning to rank using gradient descent, ICML 2005
  - LambdaRank – Learning to rank with nonsmooth cost functions, NIPS 2006
  - GBRank – A general boosting method and its application to learning ranking functions for web search, NIPS 2008
  - LambdaMART – Adapting boosting for information retrieval measures, IR 2010

Listwise LtR

- This approach trains using the ranking directly, instead of training on documents or pairs and inferring a ranking.
- Advantages:
  - You are learning with the object you ultimately want
  - You can optimize directly for evaluation measures (MAP, NDCG, etc.)
  - Often outperforms Pointwise and Pairwise (at least, for the target measure)
- Disadvantages:
  - Evaluation measures are typically non-smooth and non-differentiable, so most parameter estimation techniques don't work.
  - Optimizing for one measure doesn't mean you do well on others. Which do you pick?

Example: SVM MAP

- SVM MAP directly optimizes for mean average precision
- It does this using a error function tailored for MAP
  - A ranking is a permutation π of the documents
  - For two permutations π_i and π_j,

$$map(\pi_i) > map(\pi_j) \implies err(\pi_i) < err(\pi_j)$$

- The problem: there is an exponential number of permutations; this creates an exponential number of optimization constraints.

Example: SVM MAP

- To solve this, SVM MAP uses a *working set* of constraints:

    1. Train a model on current working set of constraints.

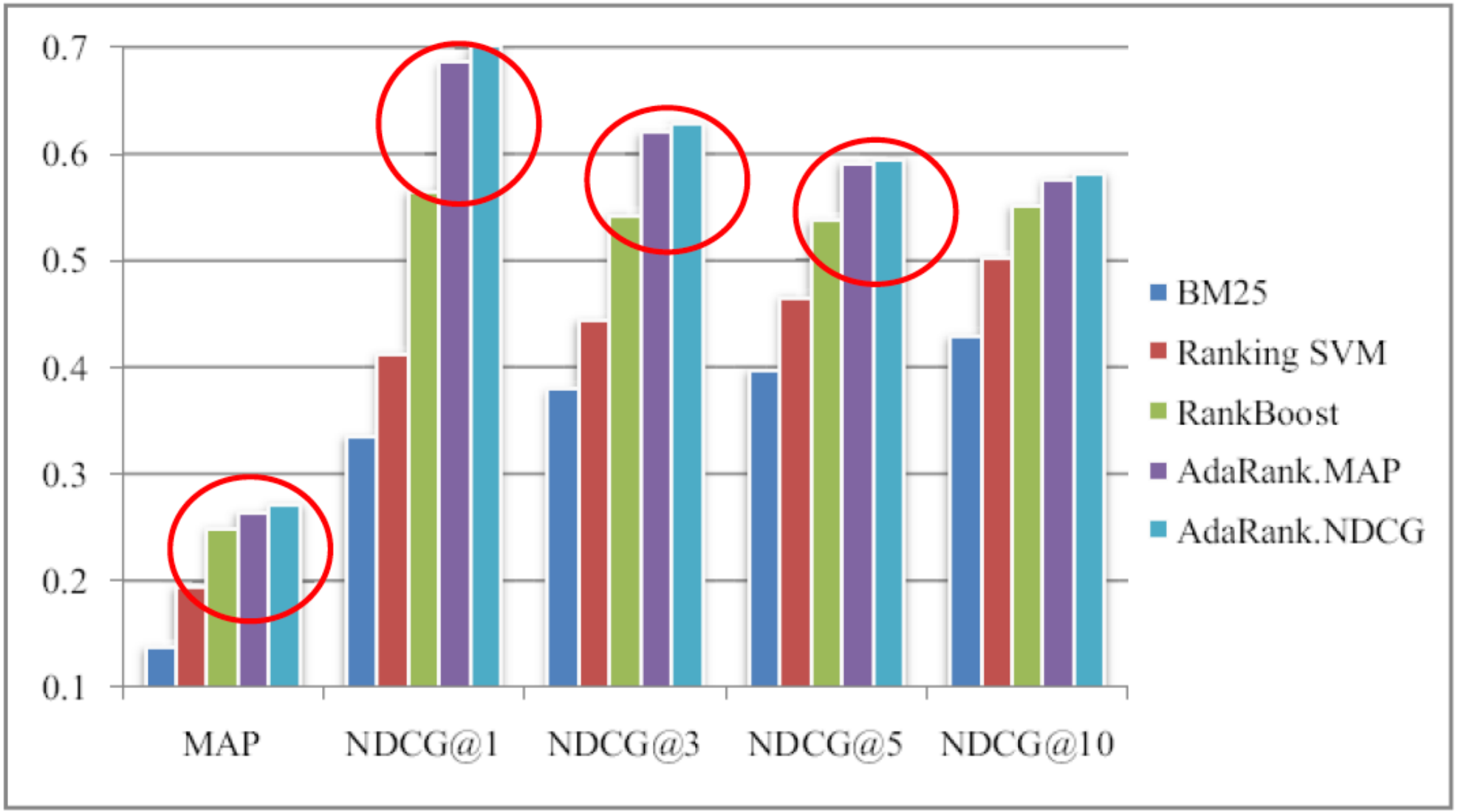
    2. Use this model to find the *most violated constraint* not currently in the working set.

    3. If this constraint is more violated than the most violated constraint in the working set, add it to the working set and start over.

- This is proven to loop for at most a polynomial number of iterations.

Listwise Methods

- Example methods for further reading:
  - ListNet – Learning to rank: from pairwise approach to listwise approach, ICML 2007
  - AdaRank – AdaRank: a boosting algorithm for information retrieval, SIGIR 2007
  - SVM MAP – A support vector method for optimizing average precision, SIGIR 2007
  - ListMLE – Listwise approach to learning to rank: theory and algorithm, ICML 2008
  - Soft-Rank – Soft-Rank: optimizing non-smooth rank metrics, WSDM 2008



Comparing Approaches

Listwise > Pairwise > Pointwise > BM25… this time.
Data from 2003.

Features for Ranking

Probability | Machine Learning
Learning to Rank | **Features for Ranking**

The Importance of Features

- The primary motivation behind developing learning to rank algorithms is to combine the wealth of features available for ranking.
- Many of these algorithms are very good at selecting the more useful features and ignoring less useful ones.
- In Machine Learning, choosing good features is critical to achieving good learning performance. Your features comprise all the information your model has from which to infer document relevance.

Feature Types

- Commercial search engines use hundreds of features for search. These are generated for a (document, query) pair and include:
  - **Text Match** – how well the text of the document and query match each other, e.g. BM25 score.
  - **Topical Matching** – how well the *topic* of the document and query match each other.
  - **Web Graph features** – graph-theoretic properties of a page, such as its PageRank, number of in-links, etc.
  - **Document Statistics** – number of words in different types of tags, number of slashes in URL, etc.

More Feature Types

- **Document Classifier** – document categories: spam, language, news, adult content, page quality, etc.
- **Click Data** – probability of click, probability of skipping it, dwell time, click count, etc.
- **External References** – supporting evidence from other sites, such as Delicious tags, Facebook likes, etc. (Is this page trending right now?)
- **Time** – page freshness, date of first appearance on web, update frequency, etc.

- Aside: BM25 on its own doesn't do that well. All these other features are one huge difference between your project two and Bing.

Summary

- Here are some good summary papers and tutorials on LtR.
  - A Short Introduction to Learning to Rank. Hang Li, 2011.
  - Learning to Rank for Information Retrieval. Tie-Yan Liu, 2008.
  - From RankNet to LambdaRank to LambdaMART: An Overview. Christopher J.C. Burges, 2010.
  - The whens and hows of learning to rank for web search. Craig Macdonald, 2012.
  - Yahoo! Learning to Rank Challenge Overview. Olivier Chapelle, Yi Chang, 2011.