CS 572: Information Retrieval

Classification → Learning to Rank

Acknowledgements
Some slides in this lecture are adapted from Chris Manning (Stanford), Jan Pedersen (Yahoo→MSFT), Ysong Yue (Cornell) and Filip Radlinski (MSR)
“Real World” Ranking

• Many different possible sources of evidence:
  – **Relevance**: Is the page relevant to the query?
  – **Page quality**: Is this a reliable source/site?
  – **Freshness**: How old is the index record for result?
  – **Spam**: Is this page likely to be optimized or spammed?
  – **Clickthrough**: how often do people click on this result? Why?
  – **Context**: Is this a reformulation of previous query?
Today's Plan

• Classification Overview

• Learning to Rank (LTR)
  – Regression/Trees
  – RankSVM
  – Neural Nets
  – Research Issues
Standing queries

• From IR → classification:
  – You have an information need to monitor, say:
    • Unrest in the Niger delta region
  – You want to rerun an appropriate query periodically to find new news items on this topic
  – You will be sent new documents that are found
    • I.e., it’s text classification not ranking

• Such queries are called **standing queries**
  – Long used by “information professionals”
  – A modern mass instantiation is **Google Alerts**

• Standing queries are (hand-written) text classifiers
Relevance feedback revisited

• In relevance feedback, the user marks a few documents as relevant/nonrelevant
• The choices can be viewed as classes or categories
• For several documents, the user decides which of these two classes is correct
• The IR system then uses these judgments to build a better model of the information need
• So, relevance feedback can be viewed as a form of text classification (deciding between several classes)
• The notion of classification is very general and has many applications within and beyond IR
Categorization/Classification

• Given:
  – A description of an instance, \( d \in X \)
    • \( X \) is the *instance language* or *instance space*.
      – Issue: how to represent text documents.
      – Usually some type of high-dimensional space
  – A fixed set of classes:
    \[ C = \{ c_1, c_2, \ldots, c_J \} \]

• Determine:
  – The category of \( d \): \( \gamma(d) \in C \), where \( \gamma(d) \) is a *classification function* whose domain is \( X \) and whose range is \( C \).
    • We want to know how to build classification functions ("classifiers").
Text Categorization Applications (2)

• Assigning documents to a fixed set of categories.

• Applications:
  – Web pages
    • Recommending
    • Yahoo-like classification
  – Newsgroup Messages
    • Recommending
    • spam filtering
  – News articles
    • Personalized newspaper
  – Email messages
    • Routing
    • Prioritizing
    • Folderizing
    • spam filtering
Supervised Classification

• Given:
  – A description of an instance, \( d \in X \)
    • \( X \) is the *instance language* or *instance space*.
  – A fixed set of classes:
    \( C = \{ c_1, c_2, \ldots, c_J \} \)
  – A training set \( D \) of labeled documents with each labeled document \( \langle d, c \rangle \in X \times C \)

• Determine:
  – A learning method or algorithm which will enable us to learn a classifier \( \gamma : X \rightarrow C \)
  – For a test document \( d \), we assign it the class \( \gamma(d) \in C \)
General Learning Issues

- Many hypotheses are usually consistent with the training data.
- Bias
  - Any criteria other than consistency with the training data that is used to select a hypothesis.
- Classification accuracy (% of instances classified correctly).
  - Measured on independent test data.
- Training time (efficiency of training algorithm).
- Testing time (efficiency of subsequent classification).
Generalization

• Hypotheses must generalize to correctly classify instances not in the training data.
• Simply memorizing training examples is a consistent hypothesis that does not generalize.
• Occam’s razor:
  – Finding a *simple* hypothesis helps ensure generalization.
Bayes’ Rule for text classification

- For a document $d$ and a class $c$

\[ P(c,d) = P(c \mid d)P(d) = P(d \mid c)P(c) \]

\[ P(c \mid d) = \frac{P(d \mid c)P(c)}{P(d)} \]
Naive Bayes Classifiers

Task: Classify a new instance $d$ based on a tuple of attribute values into one of the classes $c_j \in C$

$$d = \langle x_1, x_2, \ldots, x_n \rangle$$

$$c_{MAP} = \arg \max_{c_j \in C} P(c_j \mid x_1, x_2, \ldots, x_n)$$

$$= \arg \max_{c_j \in C} \frac{P(x_1, x_2, \ldots, x_n \mid c_j) P(c_j)}{P(x_1, x_2, \ldots, x_n)}$$

$$= \arg \max_{c_j \in C} P(x_1, x_2, \ldots, x_n \mid c_j) P(c_j)$$

MAP is “maximum a posteriori” = most likely class
Naïve Bayes Classifier: Naïve Bayes Assumption

- \( P(c_j) \)
  - Can be estimated from the frequency of classes in the training examples.

- \( P(x_1, x_2, \ldots, x_n | c_j) \)
  - \( O(|X|^n \cdot |C|) \) parameters
  - Could only be estimated if a very, very large number of training examples was available.

Naïve Bayes Conditional Independence Assumption:

- Assume that the probability of observing the conjunction of attributes is equal to the product of the individual probabilities \( P(x_i | c_j) \).
Using Multinomial Naive Bayes Classifiers to Classify Text: Basic method

- Attributes are text positions, values are words.

\[ c_{NB} = \arg \max_{c_j \in C} P(c_j) \prod_i P(x_i \mid c_j) \]

\[ = \arg \max_{c_j \in C} P(c_j) P(x_1 = "our" \mid c_j) \cdots P(x_n = "text" \mid c_j) \]

- Still too many possibilities
- Assume that classification is independent of the positions of the words
  - Use same parameters for each position
  - Result is bag of words model (over tokens not types)
Naive Bayes: Learning

- From training corpus, extract *Vocabulary*
- Calculate required $P(c_j)$ and $P(x_k \mid c_j)$ terms
  - For each $c_j$ in $C$ do
    - $docs_j \leftarrow$ subset of documents for which the target class is $c_j$
    - $P(c_j) \leftarrow \frac{|docs_j|}{|\text{total # documents}|}$

- $Text_j \leftarrow$ single document containing all $docs_j$
- for each word $x_k$ in *Vocabulary*
  - $n_k \leftarrow$ number of occurrences of $x_k$ in $Text_j$
  - $P(x_k \mid c_j) \leftarrow \frac{n_k + \alpha}{n + \alpha |\text{Vocabulary}|}$
Naive Bayes: Classifying

- positions ← all word positions in current document which contain tokens found in Vocabulary
- Return $c_{NB}$, where

$$c_{NB} = \arg\max_{c_j \in C} P(c_j) \prod_{i \in \text{positions}} P(x_i \mid c_j)$$
Underflow Prevention: using logs

- Multiplying lots of probabilities, which are between 0 and 1 by definition, can result in floating-point underflow.
- Since \( \log(xy) = \log(x) + \log(y) \), it is better to perform all computations by summing logs of probabilities rather than multiplying probabilities.
- Class with highest final un-normalized log probability score is still the most probable.

\[
c_{NB} = \arg\max_{c_j \in C} \left[ \log P(c_j) + \sum_{i \in \text{positions}} \log P(x_i | c_j) \right]
\]

- Note that model is now just max of sum of weights...
Naive Bayes Classifier

$$c_{NB} = \arg\max_{c_j \in C} [\log P(c_j) + \sum_{i \in \text{positions}} \log P(x_i | c_j)]$$

- interpretation: Each conditional parameter $\log P(x_i | c_j)$ is a weight that indicates how good an indicator $x_i$ is for $c_j$.
- The prior $\log P(c_j)$ is a weight that indicates the relative frequency of $c_j$.
- The sum is then a measure of how much evidence there is for the document being in the class.
- We select the class with the most evidence for it.
Evaluating Categorization

- Evaluation must be done on test data that are independent of the training data (usually a disjoint set of instances).
  - Sometimes use cross-validation (averaging results over multiple training and test splits of the overall data)
- It’s easy to get good performance on a test set that was available to the learner during training (e.g., just memorize the test set).
- Measures: precision, recall, F1, classification accuracy
- **Classification accuracy**: \( \frac{c}{n} \) where \( n \) is the total number of test instances and \( c \) is the number of test instances correctly classified by the system.
  - Adequate if one class per document
  - Otherwise F measure for each class
Recall: Vector Space Representation

• Each document is a vector, one component for each term (= word).

• Normally normalize vectors to unit length.

• High-dimensional vector space:
  – Terms are axes
  – 10,000+ dimensions, or even 100,000+
  – Docs are vectors in this space

• How can we do classification in this space?
Classification Using Vector Spaces

- As before, the training set is a set of documents, each labeled with its class (e.g., topic)
- In vector space classification, this set corresponds to a labeled set of points (or, equivalently, vectors) in the vector space
- **Premise 1:** Documents in the same class form a contiguous region of space
- **Premise 2:** Documents from different classes don’t overlap (much)
- We define surfaces to delineate classes in the space
Documents in a Vector Space

- Government
- Science
- Arts
Test Document of what class?

- Government
- Science
- Arts
Test Document = Government

Is this similarity hypothesis true in general?
Using Rocchio for text classification

• Relevance feedback methods can be adapted for text categorization
  – As noted before, relevance feedback can be viewed as 2-class classification
    • Relevant vs. nonrelevant documents

• Use standard tf-idf weighted vectors to represent text documents

• For training documents in each category, compute a prototype vector by summing the vectors of the training documents in the category.
  – Prototype = centroid of members of class

• Assign test documents to the category with the closest prototype vector based on cosine similarity.

\[
\bar{\mu}(c) = \frac{1}{|D_c|} \sum_{d \in D_c} \tilde{v}(d)
\]
Illustration of Rocchio Text Categorization
Rocchio Properties

- Forms a simple generalization of the examples in each class (a prototype).
- Prototype vector does not need to be averaged or otherwise normalized for length since cosine similarity is insensitive to vector length.
- Classification is based on similarity to class prototypes.
- Does not guarantee classifications are consistent with the given training data.

Why not?
Linear classifiers and binary and multiclass classification

• Consider 2 class problems
  – Deciding between two classes, perhaps, government and non-government
    • One-versus-rest classification

• How do we define (and find) the separating surface?

• How do we decide which region a test doc is in?
Linear classifier: Example

- Class: “interest” (as in interest rate)
- Example features of a linear classifier
  
  $w_i \cdot t_i$
  
- 0.70 prime
- 0.67 rate
- 0.63 interest
- 0.60 rates
- 0.46 discount
- 0.43 bundesbank
- –0.71 dlr
- –0.35 world
- –0.33 sees
- –0.25 year
- –0.24 group
- –0.24 dlr

- To classify, find dot product of feature vector and weights
Linear Classifiers

• Many common text classifiers are linear classifiers
  – Naïve Bayes
  – Perceptron
  – Rocchio
  – Logistic regression
  – Support vector machines (with linear kernel)
  – Linear regression with threshold

• Despite this similarity, noticeable performance differences
  – For separable problems, there is an infinite number of separating hyperplanes. Which one do you choose?
  – What to do for non-separable problems?
  – Different training methods pick different hyperplanes

• Classifiers more powerful than linear often don’t perform better on text problems. Why?
Separation by Hyperplanes

• A strong high-bias assumption is linear separability:
  – in 2 dimensions, can separate classes by a line
  – in higher dimensions, need hyperplanes

• Can find separating hyperplane by linear programming
  (or can iteratively fit solution via perceptron):
  – separator can be expressed as $ax + by = c$
Find $a, b, c$, such that

- $ax + by > c$ for red points
- $ax + by < c$ for blue points.
Which Hyperplane?

In general, lots of possible solutions for $a, b, c$. 
Which Hyperplane?

- Lots of possible solutions for $a, b, c$.
- Some methods find a separating hyperplane, but not the optimal one [according to some criterion of expected goodness]
  - E.g., perceptron
- Most methods find an optimal separating hyperplane
- Which points should influence optimality?
  - All points
    - Linear/logistic regression
    - Naïve Bayes
  - Only “difficult points” close to decision boundary
    - Support vector machines
Linear classifiers: Which Hyperplane?

• Lots of possible solutions for $a, b, c$.
• Some methods find a separating hyperplane, but not the optimal one [according to some criterion of expected goodness]
  – E.g., perceptron
• Support Vector Machine (SVM) finds an optimal solution.
  – Maximizes the distance between the hyperplane and the “difficult points” close to decision boundary
  – One intuition: if there are no points near the decision surface, then there are no very uncertain classification decisions

This line represents the decision boundary: $ax + by - c = 0$
Another family of linear algorithms

**Intuition** (Vapnik, 1965)

If the classes are linearly separable:

- Separate the data
- Place hyper-plane “far” from the data: **large margin**
- Statistical results guarantee **good generalization**
Large margin classifier

• **Intuition** (Vapnik, 1965) if linearly separable:
  - Separate the data
  - Place hyperplane “far” from the data: **large margin**
  - Statistical results guarantee **good generalization**

\[ \rightarrow \text{Maximal Margin Classifier} \]
If not linearly separable

- **Allow** some **errors**
- Still, try to place hyperplane “far” from each class
Another intuition

- If you have to place a fat separator between classes, you have less choices, and so the capacity of the model has been decreased
Support Vector Machine (SVM)

- Large Margin Classifier
- Linearly separable case
- Goal: find the hyperplane that maximizes the margin

\[ w^T x + b = 0 \]
\[ w^T x_a + b = 1 \]
\[ w^T x_b + b = -1 \]

Support vectors
Maximum Margin: Formalization

• \( \mathbf{w} \): decision hyperplane normal vector
• \( \mathbf{x}_i \): data point \( i \)
• \( y_i \): class of data point \( i \) (+1 or -1)  \( \text{NB: Not 1/0} \)
• Classifier is: \( f(\mathbf{x}_i) = \text{sign}(\mathbf{w}^T\mathbf{x}_i + b) \)
• Functional margin of \( \mathbf{x}_i \) is: \( y_i (\mathbf{w}^T\mathbf{x}_i + b) \)
  – But note that we can increase this margin simply by scaling \( \mathbf{w}, b \)....
• Functional margin of dataset is twice the minimum functional margin for any point
  – The factor of 2 comes from measuring the whole width of the margin
Geometric Margin

- Distance from example to the separator is $r = y \frac{w^T x + b}{||w||}$
- Examples closest to the hyperplane are support vectors.
- Margin $\rho$ of the separator is the width of separation between support vectors of classes.

Derivation of finding $r$:
Dotted line $x' - x$ is perpendicular to decision boundary so parallel to $w$.
Unit vector is $w/||w||$, so line is $rw/||w||$.
$x' = x - yrw/||w||$.
$x'$ satisfies $w^T x' + b = 0$.
So $w^T(x - yrw/||w||) + b = 0$.
Recall that $||w|| = \sqrt{w^T w}$.
So, solving for $r$ gives:
$r = y(w^T x + b)/||w||$
Linear SVM Mathematically

The linearly separable case

• Assume that all data is at least distance 1 from the hyperplane, then the following two constraints follow for a training set \( \{(x_i, y_i)\} \)

\[
\begin{align*}
w^T x_i + b &\geq 1 \quad \text{if } y_i = 1 \\
w^T x_i + b &\leq -1 \quad \text{if } y_i = -1
\end{align*}
\]

• For support vectors, the inequality becomes an equality

• Then, since each example’s distance from the hyperplane is

\[
r = y \frac{w^T x + b}{\|w\|}
\]

• The margin is:

\[
\rho = \frac{2}{\|w\|}
\]
Linear Support Vector Machine (SVM)

• **Hyperplane**
  \[ w^T x + b = 0 \]

• **Extra scale constraint:**
  \[ \min_{i=1,\ldots,n} |w^T x_i + b| = 1 \]

• **This implies:**
  \[ w^T(x_a - x_b) = 2 \]
  \[ \rho = \frac{1}{2} |x_a - x_b|_2 = \frac{2}{||w||_2} \]
Soft Margin Classification

- If the training data is not linearly separable, *slack variables* $\xi_i$ can be added to allow misclassification of difficult or noisy examples.
- Allow some errors
  - Let some points be moved to where they belong, at a cost
- Still, try to minimize training set errors, and to place hyperplane “far” from each class (large margin)
Classification with SVMs

- Given a new point $\mathbf{x}$, we can score its projection onto the hyperplane normal:
  - I.e., compute score: $\mathbf{w}^T \mathbf{x} + b = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$
  - Can set confidence threshold $t$.

Score $> t$: yes
Score $<-t$: no
Else: don’t know
Linear SVMs: Summary

- The classifier is a *separating hyperplane*.
- The most “important” training points are the support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points $x_i$ are support vectors with non-zero Lagrangian multipliers $\alpha_i$.
- Both in the dual formulation of the problem and in the solution, training points appear only inside inner products:

\[
\begin{align*}
\text{Find } \alpha_1 \ldots \alpha_N \text{ such that } \\
\mathcal{Q}(\mathbf{\alpha}) &= \sum \alpha_i - \frac{1}{2} \sum \alpha_i \alpha_j y_i y_j x_i^T x_j \text{ is maximized and } \\
(1) \quad &\sum \alpha_i y_i = 0 \\
(2) \quad &0 \leq \alpha_i \leq C \text{ for all } \alpha_i
\end{align*}
\]

\[
f(\mathbf{x}) = \sum \alpha_i y_i x_i^T \mathbf{x} + b
\]
“Real World” Ranking

- **Many** different possible sources of evidence:
  - **Relevance**: Is the page relevant to the query?
  - **Page quality**: Is this a reliable source/site?
  - **Freshness**: How old is the index record for result?
  - **Spam**: Is this page likely to be optimized or spammed?
  - **Clickthrough**: How often do people click on this result? Why?
  - **Context**: Is this a reformulation of previous query?
Solution: Machine Learning for Ranking

• We’ve looked at methods for ranking documents in IR
  – Cosine similarity, inverse document frequency, pivoted document length normalization, Pagerank, ...

• We’ve looked at methods for classifying documents using supervised machine learning classifiers
  – Naïve Bayes, Rocchio, kNN, SVMs

• 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”
Goal: Construct a ranking function

- **Input:**
  - Large number training examples (labels from editors, clicks, ...)
  - MANY Features that predict relevance
  - Relevance metrics (labels from editors, clicks, ...)

- **Output:**
  - Ranking function

Enables rapid experimental cycle

- **Scientific investigation of**
  - Modifications to existing features
  - New feature
Machine learning for IR ranking

• This “good idea” has been actively researched – and actively deployed by the major web search engines – in the last 5 years

• Why didn’t it happen earlier?
  – Modern supervised ML has been around for about 15 years...
  – Naïve Bayes has been around for about 45 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.
• Resistance to “black box” approaches (c.f. Fuhr SIGIR 2012 keynote).

Value of Theoretic Models

• Deeper insight (scientific interest)
• General validity as basis for broad range of applications
• Make better predictions (engineer's view)
• Within well-defined application range

But what is the application range?

• Defined by underlying assumptions
• Which can be verified
Why weren’t early attempts successful?

• 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 (BM25!!!)  ➔  Project 1 😊
Why is ML needed now

• Modern systems use many 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?
    – 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
    • \(x = (\alpha, \omega)\)  \(\alpha\) is cosine similarity, \(\omega\) is minimum query window size
      – \(\omega\) is the shortest text span that includes all query words

<table>
<thead>
<tr>
<th>example</th>
<th>docID</th>
<th>query</th>
<th>cosine score</th>
<th>(\omega)</th>
<th>judgment</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\Phi_1)</td>
<td>37</td>
<td>linux operating system</td>
<td>0.032</td>
<td>3</td>
<td>relevant</td>
</tr>
<tr>
<td>(\Phi_2)</td>
<td>37</td>
<td>penguin logo</td>
<td>0.02</td>
<td>4</td>
<td>nonrelevant</td>
</tr>
<tr>
<td>(\Phi_3)</td>
<td>238</td>
<td>operating system</td>
<td>0.043</td>
<td>2</td>
<td>relevant</td>
</tr>
<tr>
<td>(\Phi_4)</td>
<td>238</td>
<td>runtime environment</td>
<td>0.004</td>
<td>2</td>
<td>nonrelevant</td>
</tr>
<tr>
<td>(\Phi_5)</td>
<td>1741</td>
<td>kernel layer</td>
<td>0.022</td>
<td>3</td>
<td>relevant</td>
</tr>
<tr>
<td>(\Phi_6)</td>
<td>2094</td>
<td>device driver</td>
<td>0.03</td>
<td>2</td>
<td>relevant</td>
</tr>
<tr>
<td>(\Phi_7)</td>
<td>3191</td>
<td>device driver</td>
<td>0.027</td>
<td>5</td>
<td>nonrelevant</td>
</tr>
</tbody>
</table>
Simple example:
Using classification for ad hoc IR

• A linear score function is then

\[ \text{Score}(d, q) = \text{Score}(\alpha, \omega) = a\alpha + b\omega + c \]

• And the linear classifier is

Decide relevant if \( \text{Score}(d, q) > \theta \)

• ... just like when we were doing text classification
Simple example: Using classification for ad hoc IR

Decision surface

Term proximity $\omega$

Cosine score $\alpha$

R R R R N N N N N

Sec. 15.4.1
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
How to Train?

- **Classification & Regression**
  - Learn $f(x) \rightarrow R$ in conventional ways
  - Sort by $f(x)$ for all docs for a query
  - Typically does not work well

- **2 Major Problems**
  - Labels have ordering
    - Additional structure compared to multiclass problems
  - Severe class imbalance
    - Most documents are not relevant
“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 $\psi_i$ 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)
**Machine Learning for Ranking: Regression**

- **Approach: Regression**
  - Estimate editorial relevance given ranking features

- **Query Dependent features**
  - Term overlap between query and
    - Meta-data
    - Content

- **Query Independent Features**
  - Quality (e.g., text/word features)
  - Spamminess
Ranking Features

• A0 - A4 anchor text score per term
• W0 - W4 term weights
• L0 - L4 first occurrence location
  (encodes hostname and title match)
• SP spam index: logistic regression of 85 spam filter variables
  (against relevance scores)
• F0 - F4 term occurrence frequency within document
• DCLN document length (tokens)
• ER Eigenrank
• HB Extra-host unique inlink count
• ERHB ER*HB
• A0W0 etc. A0*W0
• QA Site factor –
  logistic regression of 5 site link and url count ratios
• SPN Proximity
• FF family friendly rating
• UD url depth
Implementation: (Tree 0)

- \( A_0w_0 < 22.3 \)
  - Y
    - \( L_0 < 18+1 \)
      - \( L_1 < 18+1 \)
        - R = -0.0545
      - W_0 < 856
        - R = -0.0039
        - \( F_2 < 1 + 1 \)
          - R = -0.1604
        - R = -0.0790
    - \( R = 0.0015 \)
  - \( N \)
    - \( L_1 < 509+1 \)
      - F_0 < 2 + 1
        - R = -0.0199
        - R = -0.1185
      - R = -0.2368
Conventional multiclass learning does not incorporate ordinal structure of class labels.
Conventional multiclass learning does not incorporate ordinal structure of class labels.
Ordinal Regression

• Assume class labels are ordered
  – True since class labels indicate level of relevance

• Learn hypothesis function $f(x) \rightarrow \mathbb{R}$
  – Such that the ordering of $f(x)$ agrees with label ordering
  – Ex: given instances $(x, 1), (y, 1), (z, 2)$
    • $f(x) < f(z)$
    • $f(y) < f(z)$
    • Don’t care about $f(x)$ vs $f(y)$
Ordinal Regression

• Compare with classification
  – Similar to multiclass prediction
  – But classes have ordinal structure

• Compare with regression
  – Doesn’t necessarily care about value of \( f(x) \)
  – Only care that ordering is preserved
Ordinal Regression Approaches

• Option 1: Learn multiple thresholds

• Option 2: Learn multiple classifiers

• Option 3: Optimize pairwise preferences
Option 1: Multiple Thresholds

• Maintain T thresholds \((b_1, \ldots, b_T)\)
• \(b_1 < b_2 < \ldots < b_T\)
• Learn model parameters \(+ (b_1, \ldots, b_T)\)

• Goal
  – Model predicts a score on input example
  – Minimize threshold violation of predictions
Ordinal SVM Example

[Chu & Keerthi, 2005]
Ordinal SVM Formulation

\[
\arg \min_{w,b,\xi^+,\xi^-} \frac{1}{2} w^2 + \frac{C}{N} \sum_{j=1}^{T}\left( \sum_i \xi_{i,j}^+ + \sum_i \xi_{i,j}^- \right)
\]

Such that for j = 0..T:

\[
\begin{align*}
    w^T x_i - b_j & \geq 1 - \xi_{i,j}^- , \quad \forall i : y_i = j \\
    w^T x_i - b_j & \leq -1 + \xi_{i,j-1}^+ , \quad \forall i : y_i = j - 1 \\
    \xi_{i,j}^- , \xi_{i,j-1}^+ & \geq 0, \quad \forall i
\end{align*}
\]

And also: \( b_1 < b_2 < \ldots < b_T \)

[Chu & Keerthi, 2005]
Learning Multiple Thresholds

• Gaussian Processes
  – [Chu & Ghahramani, 2005]

• Decision Trees
  – [Kramer et al., 2001]

• Neural Nets
  – RankProp [Caruana et al., 1996]

• SVMs & Perceptrons
  – PRank [Crammer & Singer, 2001]
  – [Chu & Keerthi, 2005]
Option 2: Voting Classifiers

- Use T different training sets
- Classifier 1 predicts 0 vs 1,2,...T
- Classifier 2 predicts 0,1 vs 2,3,...T
- ... Classifier T predicts 0,1,...,T-1 vs T
- Final prediction is combination
  - E.g., sum of predictions
- Recent work
  - McRank [Li et al., 2007]
  - [Qin et al., 2007]
Option 2: Classification: Problem

- Severe class imbalance
- Near perfect performance by always predicting 0
Option 3: Pairwise Preferences

- Most popular approach for IR applications
- Learn model to minimize pairwise disagreements
- $(\% \text{(Pairwise Agreements)}) = \text{ROC-Area}$
Utility Function: Pairwise Disagreement

2 pairwise disagreements
Optimizing Pairwise Preferences

- Consider instances \((x_1, y_1)\) and \((x_2, y_2)\)
- Label order has \(y_1 > y_2\)
Optimizing Pairwise Preferences

• Consider instances \((x_1, y_1)\) and \((x_2, y_2)\)
• Label order has \(y_1 > y_2\)

• Create new training instance
  \(- (x', +1) \text{ where } x' = (x_1 - x_2)\)

• Repeat for all instance pairs with label order preference
Optimizing Pairwise Preferences

• Result: new training set!
  – Often represented implicitly

• Has only positive examples

• Mispredicting means that a lower ordered instance received higher score than higher order instance.
Pairwise SVM Formulation

\[
\arg\min_{w, \xi} \frac{1}{2} w^2 + \frac{C}{N} \sum_{i,j} \xi_{i,j}
\]

Such that:

\[
w^T x_i - w^T x_j \geq 1 - \xi_{i,j}, \quad \forall i, j : y_i > y_j
\]

\[
\xi_{i,j} \geq 0, \quad \forall i, j
\]

[Herbrich et al., 1999]

Can be reduced to \(O(n \log(n))\) time [Joachims, 2005].
Optimizing Pairwise Preferences

• Neural Nets
  – RankNet [Burges et al., 2005]

• Boosting & Hedge-Style Methods
  – [Cohen et al., 1998]
  – RankBoost [Freund et al., 2003]
  – [Long & Servidio, 2007]

• SVMs
  – [Herbrich et al., 1999]
  – SVM-perf [Joachims, 2005]
  – [Cao et al., 2006]
Rank-Based Measures

- Pairwise Preferences not quite right
  - Assigns equal penalty for errors no matter where in the ranking

- People (mostly) care about top of ranking
  - IR community use rank-based measures which capture this property.
Point-wise learning

- Goal is to learn a threshold to separate each rank
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
  $$f(\psi_i) = w \cdot \psi_i$$
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):

\[ c_i > c_k \text{ iff } w \cdot (\psi_i - \psi_k) > 0 \]

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

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

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

• We can build model over just cases for which \( z_u = -1 \)

• From training data \( S = \{\Phi_u\} \), we train an SVM
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 $w$ and $\xi_u \geq 0$ such that
  – $\frac{1}{2}w^Tw + C \sum \xi_u$ is minimized, and
  – for all $\Phi_u$ such that $z_u < 0$, $w \cdot \Phi_u \geq 1 - \xi_u$

• 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
The SVM loss function

• The minimization
  \[ \min_w \frac{1}{2} w^T w + C \sum \xi_u \]
  and for all \( \Phi_u \) such that \( z_u < 0, w \cdot \Phi_u \geq 1 - \xi_u \)

• can be rewritten as
  \[ \min_w (1/2C)w^T w + \sum \xi_u \]
  and for all \( \Phi_u \) such that \( z_u < 0, \xi_u \geq 1 - (w \cdot \Phi_u) \)

• Now, taking \( \lambda = 1/2C \), we can reformulate this as
  \[ \min_w \sum [1 - (w \cdot \Phi_u)]_+ + \lambda w^T w \]

• Where \([]_+\) is the positive part (0 if a term is negative)
The SVM loss function

- The reformulation

\[
\min_w \sum [1 - (w \cdot \Phi_u)]_+ + \lambda w^T w
\]

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

• Training data (labeled)
• Features
• Algorithm
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
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 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
Based on the LETOR test collection

- From Microsoft Research Asia
- An openly available standard test collection with pregenerated features, baselines, and research results for learning to rank
- Its 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]
IR Evaluation Measures

• Some evaluation measures strongly weight doing well in highest ranked results:
  – MAP (Mean Average Precision)
  – NDCG (Normalized Discounted Cumulative Gain)

• NDCG has been especially popular in machine learned relevance research
  – It handles multiple levels of relevance (MAP doesn’t)
  – It seems to have the right kinds of properties in how it scores system rankings
Normalized Discounted Cumulative Gain (NDCG) evaluation measure

- Query: $q_i$
- DCG at position $m$: $N_i = Z_i \sum_{j=1}^{m}(2^{r(j)} - 1)/\log(1 + j)$
- NDCG at position $m$: average over queries

Example
- $(3, 3, 2, 2, 1, 1, 1)$ rank $r$
- $(7, 7, 3, 3, 1, 1, 1)$ gain $2^{r(j)} - 1$
- $(1, 0.63, 0.5, 0.43, 0.39, 0.36, 0.33)$ discount $1/\log(1 + j)$
- $(7, 11.41, 12.91, 14.2, 14.59, 14.95, 15.28)$

$Z_i$ normalizes against best possible result for query, the above, versus lower scores for other rankings

- Necessarily: High ranking number is good (more relevant)
Recap: Two Problems with Direct Application of the Ranking SVM

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

  \[ \text{d: definitely relevant, p: partially relevant, n: not relevant} \]

  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: \(2 \times (d, p) + 4 \times (d, n) + 8 \times (p, n) = 14\)
  q2 pairs: \(6 \times (d, p) + 10 \times (d, n) + 15 \times (p, n) = 31\)
These problems are solved with a new Loss function

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

• \( \tau \) weights for type of rank difference
  – Estimated empirically from effect on NDCG

• \( \mu \) weights for size of ranked result set
  – Linearly scaled versus biggest result set

![Graph showing loss vs. \( z \cdot f(\tilde{x}^{(1)} - \tilde{x}^{(2)}) \) for different values of \( \tau \cdot \mu \).]
Neural Nets

• **RankNet**: Burges et al., [ICML 2005]
  – Scalable Neural Net implementation
  – Input: feature vectors and relevance labels

• **LambdaRank**: extension to RankNet that directly optimizes IR measures (MRR, MAP, nDCG).

• **LambdaMART**: adds boosting w/ weighted trees (inspired by GBDT success).
Training RankNet

• For query results 1 and 2, present pair of vectors and labels, $\text{label}(1) > \text{label}(2)$
RankNet [Burges et al. 2005]

- For query results 1 and 2, present pair of vectors and labels, \( \text{label}(1) > \text{label}(2) \)
RankNet [Burges et al. 2005]

• For query results 1 and 2, present pair of vectors and labels, label(1) > label(2)
RankNet [Burges et al. 2005]

• For query results 1 and 2, present pair of vectors and labels, label(1) > label(2)

Error is function of both outputs (Desire output1 > output2)
Predicting with RankNet

• Present individual vector and get score
The Limitation 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.
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!]

And there is every reason to think that the importance of machine learning in IR will only increase in the future.
Resources

• Many Learning to Rank Tutorials:

• LETOR benchmark datasets (Microsoft)
  – Website with data, links to papers, benchmarks, etc.
  – http://research.microsoft.com/users/LETOR/
  – Everything you need to start research in this area!

• Yahoo Learning to Rank challenge:
Multi-class classification

- **Given:** some data items that belong to one of $M$ possible classes
- **Task:** Train the classifier and predict the class for a new data item
- **Geometrically:** harder problem, no more simple geometry
Multi-class classification
Multi-class classification: Examples

- Author identification
- Language identification
- Text categorization (topics)
More Than Two Classes

- **Any-of or multivalue** classification
  - Classes are independent of each other.
  - A document can belong to 0, 1, or >1 classes.
  - Decompose into $n$ binary problems
  - Quite common for documents

- **One-of or multinomial or polytomous** classification
  - Classes are mutually exclusive.
  - Each document belongs to exactly one class
  - E.g., digit recognition is polytomous classification
    - Digits are mutually exclusive
Set of Binary Classifiers: Any of

- Build a separator between each class and its complementary set (docs from all other classes).
- Given test doc, evaluate it for membership in each class.
- Apply decision criterion of classifiers independently
- Done

— Though maybe you could do better by considering dependencies between categories
Set of Binary Classifiers: One of

- Build a separator between each class and its complementary set (docs from all other classes).
- Given test doc, evaluate it for membership in each class.
- Assign document to class with:
  - maximum score
  - maximum confidence
  - maximum probability
- Why different from multiclass/classification?
(Some) Algorithms for Multi-class classification

• Linear
  – Parallel class separators: Decision Trees
  – Non parallel class separators: Naïve Bayes

• Non Linear
  – K-nearest neighbors
  – Decision Trees
  – Neural Networks
Linear, parallel class separators (ex: Decision Trees)
Linear, NON parallel class separators
(ex: Naïve Bayes)