CS 730: Text Mining for Social Media & Collaboratively Generated Content

Mondays 10am-12:30pm
Emerson E510 (Conference room)
Instructor: Eugene Agichtein (http://www.mathcs.emory.edu/~eugene/ )
Lecture Plan

• Language Models (conclusion, applications)
• Smoothing
• Text classification review
• Word Sense Disambiguation
• Part of Speech Tagging
Logistics/Plans

- Language models (word, n-gram, ...)
- Statistical NLP:
  - Language models
  - Classification and sequence models
    - Part-of-speech tagging, entity tagging, information extraction
- Next week: parsing
- Next->Next week: semantics
- Part II: Social media (research papers start)
Recap: N-order Markov Models

- First order Markov assumption = bigram
  \[ P(w_k|w_1 \ldots w_{k-1}) \approx P(w_k|w_{k-1}) = \frac{P(w_{k-1}w_k)}{P(w_{k-1})} \]
- Similarly, \( n \)-th order Markov assumption
- Most commonly, trigram (2nd order):
  \[ P(w_k|w_1 \ldots w_{k-1}) \approx P(w_k|w_{k-2}, w_{k-1}) = \frac{P(w_{k-2}w_{k-1}w_k)}{P(w_{k-2}, w_{k-1})} \]
Sparsity

- Problems with n-gram models:
  - New words appear all the time:
    - Synaptitude
    - 132,701.03
    - fuzzificational
  - New bigrams: even more often
  - Trigrams or more – still worse!

- Zipf’s Law
  - Types (words) vs. tokens (word occurrences)
  - Broadly: most word types are rare
  - Specifically:
    - Rank word types by token frequency
    - Frequency inversely proportional to rank
  - Not special to language: randomly generated character strings have this property
Smoothing Overview

- We often want to make estimates from sparse statistics:
  
  P(w | denied the)
  3 allegations
  2 reports
  1 claims
  1 request
  7 total

- Smoothing flattens spiky distributions so they generalize better

  P(w | denied the)
  2.5 allegations
  1.5 reports
  0.5 claims
  0.5 request
  2 other
  7 total

- Very important all over NLP, but easy to do badly!
- We'll illustrate with bigrams today (h – previous word, could be anything).
Smoothing

• Estimating multinomials
• We want to know what words follow some history h
  – There’s some true distribution \( P(w \mid h) \)
  – We saw some small sample of N words from \( P(w \mid h) \)
  – We want to reconstruct a useful approximation of \( P(w \mid h) \)
  – Counts of events we didn’t see are always too low (\( 0 < N \cdot P(w \mid h) \))
  – Counts of events we did see are in aggregate too high

• Two issues:
  – Discounting: how to reserve mass what we haven’t seen
  – Interpolation: how to allocate that mass amongst unseen events
Laplace smoothing

• Idea: pretend we saw every word once more than we actually did [Laplace]
  – Corresponds to a uniform prior over vocabulary
  – Think of it as taking items with observed count $r > 1$ and treating them as having count $r^* < r$
  – Holds out $V/(N+V)$ for “fake” events
  – $N1+/N$ of which is distributed back to seen words
  – $N0/(N+V)$ actually passed on to unseen words (nearly all!)
  – Actually tells us not only how much to hold out, but where to put it

• Works poorly in practice

• Quick fix: add a small $\delta$ instead of 1 [Lidstone, Jefferys]

• Slightly better, holds out less mass, still a bad idea
Linear Interpolation

- One way to ease the sparsity problem for n-grams is to use less-sparse n-1-gram estimates.
- General linear interpolation:

\[ P(w | w_1) = \left[ 1 - \lambda(w, w_1) \right] \hat{P}(w | w_1) + \left[ \lambda(w, w_1) \right] P(w) \]

- Having a single global mixing constant is generally not ideal:

\[ P(w | w_{-1}) = [1 - \lambda] \hat{P}(w | w_{-1}) + [\lambda] P(w) \]

- Solution: have different constant buckets
  - Buckets by count
  - Buckets by average count (better)
Good-Turing Smoothing

- Intuition: Can judge rate of novel events by rate of singletons.

- Let $N_r = \#\ of\ word\ types\ with\ r\ training\ tokens$
  - e.g., $N_0 = \#\ of\ unobserved\ words$
  - e.g., $N_1 = \#\ of\ singletons$

- Let $N = \sum r \ N_r = total\ \#\ of\ training\ tokens$
Good-Turing Smoothing (2)

- Let $N_r$ = # of word types with $r$ training tokens
- Let $N = \sum r N_r$ = total # of training tokens
- Naïve estimate: if $x$ has $r$ tokens, $p(x) =$ ?
  - Answer: $r/N$
- Total naïve probability of all words with $r$ tokens?
  - Answer: $N_r r / N$.

- Good-Turing estimate of this total probability:
  - Defined as: $N_{r+1} (r+1) / N$
  - So proportion of novel words in test data is estimated by proportion of singletons in training data.
  - Proportion in test data of the $N_1$ singletons is estimated by proportion of the $N_2$ doubletons in training data. Etc.
Backoff Smoothing

• Why are we treating all novel events as the same?

• \( p(zygote \mid \text{see the}) \) vs. \( p(baby \mid \text{see the}) \)
  – Suppose both trigrams have zero count

• baby beats zygote as a unigram
• the baby beats the zygote as a bigram
• see the baby beats see the zygote?

• As always for backoff:
  – Lower-order probabilities (unigram, bigram) aren’t quite what we want
  – But we do have enuf data to estimate them & they’re better than nothing.
• Backoff smoothing
  – Holds out same amount of probability mass for novel events
  – But divide up *unevenly* in proportion to backoff prob.
  – When defining $p(z \mid xy)$, the backoff prob for novel $z$ is $p(z \mid y)$
    • Novel events are types $z$ that were never observed after $xy$.
  – When defining $p(z \mid y)$, the backoff prob for novel $z$ is $p(z)$
    • Here novel events are types $z$ that were never observed after $y$.
    • Even if $z$ was never observed after $xy$, it may have been observed after the shorter, more frequent context $y$. Then $p(z \mid y)$ can be estimated without further backoff. If not, we back off further to $p(z)$.
  – When defining $p(z)$, do we need a backoff prob for novel $z$?
    • What are novel $z$ in this case? What could the backoff prob be? What if the vocabulary is known and finite? What if it’s potentially infinite?
How much to subtract?

- Remember the key discounting problem:
  - What count should \(r^*\) should we use for an event that occurred \(r\) times in \(N\) samples?
  - \(r\) is too big

- Idea: held-out data [Jelinek and Mercer]
  - Get another \(N\) samples
  - See what the average count of items occurring \(r\) times is (e.g. doubletons on average might occur 1.78 times)
  - Use those averages as \(r^*\)
  - Much better than add-one, etc.
Importance of Held-Out Data

- Important tool for getting models to generalize:

  \[ LL(w_1...w_n \mid M(\lambda_1...\lambda_k)) = \sum_i \log P_{M(\lambda_1...\lambda_k)}(w_i \mid w_{i-1}) \]

- When we have a small number of parameters that control the degree of smoothing, we set them to maximize the (log-)likelihood of held-out data.

- Can use any optimization technique (line search or EM usually easiest)

- Examples:

  \[ P_{\text{LIN}(\lambda_1,\lambda_2)}(w \mid w_{-1}) = \lambda_1 \hat{P}(w \mid w_{-1}) + \lambda_2 \hat{P}(w) \]

  \[ P_{\text{UNI-PRIOR}(\delta)}(w \mid w_{-1}) = \frac{c(w, w_{-1}) + \delta \hat{P}(w)}{c(w_{-1}) + \delta} \]
Held-Out Reweighting

- What’s wrong with unigram-prior smoothing?
- Let’s look at some real bigram counts [Church and Gale 91]:

<table>
<thead>
<tr>
<th>Count in 22M Words</th>
<th>Actual $c^*$ (Next 22M)</th>
<th>Add-one’s $c^*$</th>
<th>Add-0.00000027’s $c^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.448</td>
<td>2/7e-10</td>
<td>~1</td>
</tr>
<tr>
<td>2</td>
<td>1.25</td>
<td>3/7e-10</td>
<td>~2</td>
</tr>
<tr>
<td>3</td>
<td>2.24</td>
<td>4/7e-10</td>
<td>~3</td>
</tr>
<tr>
<td>4</td>
<td>3.23</td>
<td>5/7e-10</td>
<td>~4</td>
</tr>
<tr>
<td>5</td>
<td>4.21</td>
<td>6/7e-10</td>
<td>~5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mass on New</th>
<th>9.2%</th>
<th>~100%</th>
<th>9.2%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratio of 2/1</td>
<td>2.8</td>
<td>1.5</td>
<td>~2</td>
</tr>
</tbody>
</table>

- Big things to notice:
  - Add-one vastly overestimates the fraction of new bigrams
  - Add-0.00000027 still underestimates the ratio $2^*/1^*$
- One solution: use held-out data to predict the map of $c$ to $c^*$
What actually works?

- **Trigrams:**
  - Unigrams, bigrams too little context
  - Trigrams much better (when there's enough data)
  - 4-, 5-grams usually not worth the cost (which is more than it seems, due to how speech recognizers are constructed)
- **Good-Turing-like methods for count adjustment**
  - Absolute discounting, Good-Turing, held-out estimation, Witten-Bell
- **Kneser-Ney equalization for lower-order models**
- See [Chen+Goodman] reading for tons of graphs!

[Graphs from Joshua Goodman]
Review of Text Classification
LMs: Language Identification

How can we tell what language a document is in?

The 38th Parliament will meet on Monday, October 4, 2004, at 11:00 a.m. The first item of business will be the election of the Speaker of the House of Commons. Her Excellency the Governor General will open the First Session of the 38th Parliament on October 5, 2004, with a Speech from the Throne.


How to tell the French from the English?

- Treat it as word-level textcat?
  - Overkill, and requires a lot of training data
  - You don’t actually need to know about words!

Σύμφωνο σταθερότητας και ανάπτυξης
Patto di stabilità e di crescita

- Option: build a character-level language model
Text Classification

- Want to classify documents into broad semantic topics (e.g. politics, sports, etc.)

Democratic vice presidential candidate John Edwards on Sunday accused President Bush and Vice President Dick Cheney of misleading Americans by implying a link between deposed Iraqi President Saddam Hussein and the Sept. 11, 2001 terrorist attacks.

While No. 1 Southern California and No. 2 Oklahoma had no problems holding on to the top two spots with lopsided wins, four teams fell out of the rankings — Kansas State and Missouri from the Big 12 and Clemson from the Atlantic Coast Conference and Oregon from the Pac-10.

- Which one is the politics document? (And how much deep processing did that decision take?)
- One approach: bag-of-words and Naïve-Bayes models
- Another approach later…
- Usually begin with a labeled corpus containing examples of each class
Naïve Bayes Model

- Idea: pick a topic, then generate a document using a language model for that topic.
- Naïve-Bayes assumption: all words are independent given the topic.

\[ P(c, w_1, w_2, \ldots w_n) = P(c) \prod_i P(w_i | c) \]

We have to smooth these!

- Compare to a unigram language model:

\[ P(w_1, w_2, \ldots w_n) = \prod_i P(w_i) \]
Maximum a posteriori Hypothesis

\[ h_{\text{MAP}} \equiv \arg\max_{h \in H} P(h \mid D) \]

\[ h_{\text{MAP}} = \arg\max_{h \in H} \frac{P(D \mid h)P(h)}{P(D)} \]

\[ h_{\text{MAP}} = \arg\max_{h \in H} P(D \mid h)P(h) \]
Maximum likelihood Hypothesis

If all hypotheses are a priori equally likely, we only need to consider the $P(D|h)$ term:

$$h_{ML} \equiv \arg\max_{h \in H} P(D \mid h)$$
Naive Bayes Classifiers

Task: Classify a new instance based on a tuple of attribute values

\[ \langle x_1, x_2, \ldots, x_n \rangle \]

\[
c_{MAP} = \arg\max_{c_j \in C} P(c_j | x_1, x_2, \ldots, x_n)
\]

\[
c_{MAP} = \arg\max_{c_j \in C} \frac{P(x_1, x_2, \ldots, x_n | c_j)P(c_j)}{P(c_1, c_2, \ldots, c_n)}
\]

\[
c_{MAP} = \arg\max_{c_j \in C} P(x_1, x_2, \ldots, x_n | c_j)P(c_j)
\]
Naïve Bayes Classifier: Assumptions

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

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

**Conditional Independence Assumption:**

⇒ Assume that the probability of observing the conjunction of attributes is equal to the product of the individual probabilities.
The Naïve Bayes Classifier

- **Conditional Independence Assumption:**
  features (words) are independent of each other given the class:

\[
P(X_1, \ldots, X_5 \mid C) = P(X_1 \mid C) \cdot P(X_2 \mid C) \cdot \cdots \cdot P(X_5 \mid C)
\]
Learning the Model

- Common practice: maximum likelihood
  - simply use the frequencies in the data

\[
\hat{P}(c_j) = \frac{N(C = c_j)}{N}
\]

\[
\hat{P}(x_i \mid c_j) = \frac{N(X_i = x_i, C = c_j)}{N(C = c_j)}
\]
Problem with Max Likelihood

- What if we have seen no training cases where patient had no flu and muscle aches?

\[ P(X_1, \ldots, X_5 \mid C) = P(X_1 \mid C) \cdot P(X_2 \mid C) \cdot \cdots \cdot P(X_5 \mid C) \]

- Zero probabilities cannot be conditioned away, no matter the other evidence!

\[ \ell = \arg \max_c \hat{P}(c) \prod_i \hat{P}(x_i \mid c) \]
Smoothing to Avoid Overfitting

\[
\hat{P}(x_i \mid c_j) = \frac{N(X_i = x_i, C = c_j) + 1}{N(C = c_j) + k}
\]

- Somewhat more subtle version

\[
\hat{P}(x_{i,k} \mid c_j) = \frac{N(X_i = x_{i,k}, C = c_j) + mp_{i,k}}{N(C = c_j) + m}
\]

# of values of \(X_i\)

overall fraction in data where \(X_i=x_{i,k}\)

extent of “smoothing”
Underflow Prevention

• 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.
Two NB Formulations

- Two NB models for text categorization
  - The class-conditional unigram model, a.k.a. multinomial model
    - One node per word in the document
    - Driven by words which are present
    - Multiple occurrences, multiple evidence
    - Better overall – plus, know how to smooth
  - The binominal (binary) model
    - One node for each word in the vocabulary
    - Incorporates explicit negative correlations
    - Know how to do feature selection (e.g. keep words with high mutual information with the class variable)
Beyond Naïve Bayes

- So far we’ve looked at “generative models”
  - Language models, Naive Bayes, IBM MT
- In recent years there has been extensive use of conditional or discriminative probabilistic models in NLP, IR, and Speech
- Because:
  - They give high accuracy performance
  - They make it easy to incorporate lots of linguistically important features
  - They allow automatic building of language independent, retargetable NLP modules
Categorization

• Given:
  – A description of an instance, \( x \in X \), where \( X \) is the instance language or instance space.
  – A fixed set of categories: \( C = \{c_1, c_2, \ldots, c_n\} \)

• Determine:
  – The category of \( x \): \( c(x) \in C \), where \( c(x) \) is a categorization function whose domain is \( X \) and whose range is \( C \).
Some Terminology: Features

- In these slides and most maxent work: *features* are elementary pieces of evidence that link aspects of what we observe \( d \) with a category \( c \) that we want to predict.

- A feature has a (bounded) real value: \( f: C \times D \rightarrow \mathbb{R} \)

- Usually features specify an indicator function of properties of the input and a particular class (*every one we present is*). They pick out a subset.

  - \( f_i(c, d) = [\Phi(d) \land c = c_i] \quad \text{[Value is 0 or 1]} \)

- We will freely say that \( \Phi(d) \) is a feature of the data \( d \), when, for each \( c_i \), the conjunction \( \Phi(d) \land c = c_i \) is a feature of the data-class pair \((c, \ d)\).
Features

For example:

- $f_1(c, d) = [c^- \text{“NN”} \land \text{islower}(w_0) \land \text{ends}(w_0, \text{“d”})]$
- $f_2(c, d) = [c^- \text{“NN”} \land w_{-1}^- \text{“to”} \land t_{-1}^- \text{“TO”}]$
- $f_3(c, d) = [c^- \text{“VB”} \land \text{islower}(w_0)]$

![Circle diagram showing parts of speech: IN NN for in bed, TO NN for to aid, TO VB for to aid, IN JJ for in blue.]

- Models will assign each feature a weight.
- Empirical count (expectation) of a feature:
  \[
  \text{empirical } E(f_i) = \sum_{(c,d) \in \text{observed}(C,D)} f_i(c,d)
  \]
- Model expectation of a feature:
  \[
  E(f_i) = \sum_{(c,d) \in (C,D)} P(c,d) f_i(c,d)
  \]
Sample Category Learning Problem

- **Instance language:** <size, color, shape>
  - size ∈ {small, medium, large}
  - color ∈ {red, blue, green}
  - shape ∈ {square, circle, triangle}

- **C =** {positive, negative}

- **D:**

<table>
<thead>
<tr>
<th>Example</th>
<th>Size</th>
<th>Color</th>
<th>Shape</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>small</td>
<td>red</td>
<td>circle</td>
<td>positive</td>
</tr>
<tr>
<td>2</td>
<td>large</td>
<td>red</td>
<td>circle</td>
<td>positive</td>
</tr>
<tr>
<td>3</td>
<td>small</td>
<td>red</td>
<td>triangle</td>
<td>negative</td>
</tr>
<tr>
<td>4</td>
<td>large</td>
<td>blue</td>
<td>circle</td>
<td>negative</td>
</tr>
</tbody>
</table>
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).
Binary 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 test which region a test doc is in?
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 $ax + by = c$
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 regression
    • Naïve Bayes
  – Only “difficult points” close to decision boundary
    • Support vector machines
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
  • $w_i \cdot t_i$
    - -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
  - (Simple) perceptron neural networks

- 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. Why?
Naive Bayes is a linear classifier

- Two-class Naive Bayes. We compute:

\[
\log \frac{P(C \mid d)}{P(C \mid d)} = \log \frac{P(C)}{P(C)} + \sum_{w \in d} \log \frac{P(w \mid C)}{P(w \mid C)}
\]

- Decide class \( C \) if the odds is greater than 1, i.e., if the log odds is greater than 0.
- So decision boundary is hyperplane:

\[
\alpha + \sum_{w \in V} \beta_w \times n_w = 0 \quad \text{where} \quad \alpha = \log \frac{P(C)}{P(C)};
\]

\[
\beta_w = \log \frac{P(w \mid C)}{P(w \mid C)}; \quad n_w = \# \text{ of occurrence } s \text{ of } w \text{ in } d
\]
High Dimensional Data

- Pictures like the one at right are absolutely misleading!
- Documents are zero along almost all axes
- Most document pairs are very far apart (i.e., not strictly orthogonal, but only share very common words and a few scattered others)
- In classification terms: virtually all document sets are separable, for most any classification
- This is part of why linear classifiers are quite successful in this domain
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 string, 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/any of classification?
Evaluating Categorization

- Evaluation must be done on test data that are independent of the training data (usually a disjoint set of instances).

- **Classification accuracy**: $c/n$ where $n$ is the total number of test instances and $c$ is the number of test instances correctly classified by the system.

- Results can vary based on sampling error due to different training and test sets.

- Average results over multiple training and test sets (splits of the overall data) for the best results.
Other Evaluation Metrics

- **Precision**: fraction of identified entities that are correct = $P(\text{correct} \mid \text{selected})$
- **Recall**: fraction of correct entities that are identified = $P(\text{selected} \mid \text{correct})$

<table>
<thead>
<tr>
<th></th>
<th>Correct</th>
<th>Not Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selected</td>
<td>TP</td>
<td>FP</td>
</tr>
<tr>
<td>Not Selected</td>
<td>FN</td>
<td>TN</td>
</tr>
</tbody>
</table>

- Precision $P = \frac{tp}{tp + fp}$
- Recall $R = \frac{tp}{tp + fn}$
A combined measure: $F$

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

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

- Usually use balanced $F_1$ measure
  - i.e., with $\beta = 1$ or $\alpha = 1/2$: \( F = 2PR/(P+R) \)
Naïve Bayes has found a home in spam filtering
  – Graham’s *A Plan for Spam*
    • And its mutant offspring...
  – Naive Bayes-like classifier with weird parameter estimation
  – Widely used in spam filters
    • Classic Naive Bayes superior when appropriately used
    • According to David D. Lewis
  • Many email filters use NB classifiers
    – But also many other things: black hole lists, etc.
NB on Spam

% Correct

Training Examples

RIPPER
BAYES-INDP-BVF
MOST-COMMON-CATEGORY
Naive Bayes is Not So Naive

- Naive Bayes: First and Second place in KDD-CUP 97 competition, among 16 (then) state of the art algorithms
  - Goal: Financial services industry direct mail response prediction model: Predict if the recipient of mail will actually respond to the advertisement – 750,000 records.

- Benefits:
  - Robust to Irrelevant Features
    - Irrelevant Features cancel each other without affecting results
  - Very good in domains with many equally important features
    - Decision Trees suffer from fragmentation in such cases – especially if little data
  - A good dependable baseline for text classification (but not the best)!
  - Very Fast: Learning with one pass over the data; testing linear in the number of attributes, and document collection size
  - Low Storage requirements

- Optimal if the Independence Assumptions hold: If assumed independence is correct, then it is the Bayes Optimal Classifier for problem
Beyond Naïve Bayes

- So far we’ve looked at “generative models”
  - Language models, Naive Bayes, IBM MT
- In recent years there has been extensive use of conditional or discriminative probabilistic models in NLP, IR, and Speech
- Because:
  - They give high accuracy performance
  - They make it easy to incorporate lots of linguistically important features
  - They allow automatic building of language independent, retargetable NLP modules
Joint vs Conditional Models

- **Joint (generative) models** place probabilities over both observed data and the hidden stuff (generate the observed data from hidden stuff):
  - All the best known StatNLP models:
    - $n$-gram models, Naive Bayes classifiers, hidden Markov models, probabilistic context-free grammars
  - Discriminative (conditional) models take the data as given, and put a probability over hidden structure given the data:
    - Logistic regression, conditional loglinear models, maximum entropy markov models, (SVMs, perceptrons)
Lecture Plan

• Language Models (conclusion, applications)

• Smoothing

• Text classification review

➢ Word Sense Disambiguation (WSD)

• Part of Speech Tagging

• If time: chunking
Motivating Example: WSD

- Words have multiple distinct meanings, or senses:
  - Plant: living plant, manufacturing plant, ...
  - Title: name of a work, ownership document, form of address, material at the start of a film, ...

- Many levels of sense distinctions
  - Homonymy: totally unrelated meanings (river bank, money bank)
  - Polysemy: related meanings (star in sky, star on tv)
  - Systematic polysemy: productive meaning extensions (organizations to their buildings) or metaphor
  - Sense distinctions can be extremely subtle (or not)

- Granularity of senses needed depends a lot on the task

- Why is it important to model word senses?
  - Translation, parsing, information retrieval?
Word Sense Disambiguation

- Example: living plant vs. manufacturing plant

- How do we tell these senses apart?
  - "context"

  The manufacturing plant which had previously sustained the town’s economy shut down after an extended labor strike.

  - Maybe it’s just text categorization
  - Each word sense represents a topic
  - Run the naive-bayes classifier from last class?

- Bag-of-words classification works ok for noun senses
  - 90% on classic, shockingly easy examples (line, interest, star)
  - 80% on senseval-1 nouns
  - 70% on senseval-1 verbs
Verb WSD

- Why are verbs harder?
  - Verbal senses less topical
  - More sensitive to structure, argument choice

- Verb Example: “Serve”
  - [function] The tree stump serves as a table
  - [enable] The scandal served to increase his popularity
  - [dish] We serve meals for the homeless
  - [enlist] He served his country
  - [jail] He served six years for embezzlement
  - [tennis] It was Agassi’s turn to serve
  - [legal] He was served by the sheriff
Modeling the word “serve”

- So what do we need to model to handle “serve”?
  - There are distant topical cues
    - ... point ... court ..................... serve .......... game ...

\[
P(c, w_1, w_2, \ldots, w_n) = P(c) \prod_{i} P(w_i | c)
\]
Weighted Windows with NB

**Distance conditioning**
- Some words are important only when they are nearby
  - .... as .... point ... court ........... serve ........ game .......
  - ........................................ serve as ........

\[
P(c, w_{-k}, \ldots, w_{-1}, w_0, w_{1}, \ldots w_{+k'}) = P(c) \prod_{i=-k}^{k'} P(w_i \mid c, \text{bin}(i))
\]

**Distance weighting**
- Nearby words should get a larger vote
  - ... court ...... serve as........... game ...... point

\[
P(c, w_{-k}, \ldots, w_{-1}, w_0, w_{1}, \ldots w_{+k'}) = P(c) \prod_{i=-k}^{k'} P(w_i \mid c)^{\text{boost}(i)}
\]
Complex Features with NB

- **Example:** Washington County jail served 11,166 meals last month - a figure that translates to feeding some 120 people three times daily for 31 days.

- So we have a decision to make based on a set of cues:
  - `context:jail, context:county, context:feeding, ...`
  - `local-context:jail, local-context:meals`
  - `subcat:NP, direct-object-head:meals`

- Not clear how build a generative derivation for these:
  - Choose topic, then decide on having a transitive usage, then pick “meals” to be the object’s head, then generate other words?
  - How about the words that appear in multiple features?
  - Hard to make this work (though maybe possible)
  - No real reason to try
Discriminative Approach

- View WSD as a discrimination task (regression, really)

  \[ P(\text{sense} \mid \text{context: jail}, \text{context: county}, \]
  \[ \text{context: feeding}, \ldots \]
  \[ \text{local-context: jail, local-context: meals} \]
  \[ \text{subcat: NP, direct-object-head: meals, \ldots} \]

- Have to estimate multinomial (over senses) where there are a huge number of things to condition on
  - History is too complex to think about this as a smoothing / back-off problem

- Many feature-based classification techniques out there
- We tend to need ones that output distributions over classes (why?)
Discriminative Models for WSD

Example: Washington County jail served 11,166 meals last month - a figure that translates to feeding some 120 people three times daily for 31 days.

So we have a decision to make based on a set of cues:
- context:jail, context:county, context:feeding, ...
- local-context:jail, local-context:meals
- subcat:NP, direct-object-head:meals

Not clear how build a generative derivation for these:
- Choose topic, then decide on having a transitive usage, then pick “meals” to be the object’s head, then generate other words?
- How about the words that appear in multiple features?
- Hard to make this work (though maybe possible)
- No real reason to try
Feature Representation

\[ d \]

Washington County jail served 11,166 meals last month - a figure that translates to feeding some 120 people three times daily for 31 days.

- Features are indicator functions \( f_i \) which count the occurrences of certain patterns in the input

- We map each input to a vector of feature predicate counts

\( \{f_i(d)\} \)

context:jail = 1
context:county = 1
context:feeding = 1
context:game = 0
...
local-context:jail = 1
local-context:meals = 1
...
subcat:NP = 1
subcat:PP = 0
...
object-head:meals = 1
object-head:ball = 0
Linear Classification

- For a pair \((c,d)\), we take a weighted vote for each class:

\[
\text{vote}(c \mid d) = \exp \sum_i \lambda_i(c) f_i(d)
\]

<table>
<thead>
<tr>
<th>Feature</th>
<th>Food</th>
<th>Jail</th>
<th>Tennis</th>
</tr>
</thead>
<tbody>
<tr>
<td>context:jail</td>
<td>-0.5 * 1</td>
<td>+1.2 * 1</td>
<td>-0.8 * 1</td>
</tr>
<tr>
<td>subcat:NP</td>
<td>+1.0 * 1</td>
<td>+1.0 * 1</td>
<td>-0.3 * 1</td>
</tr>
<tr>
<td>object-head:meals</td>
<td>+2.0 * 1</td>
<td>-1.5 * 1</td>
<td>-1.5 * 1</td>
</tr>
<tr>
<td>object-head:years = 0</td>
<td>-1.8 * 0</td>
<td>+2.1 * 0</td>
<td>-1.1 * 0</td>
</tr>
<tr>
<td>TOTAL</td>
<td>+3.5</td>
<td>+0.7</td>
<td>-2.6</td>
</tr>
</tbody>
</table>

- There are many ways to set these weights
  - Perceptron: find a currently misclassified example, and nudge weights in the direction of a correct classification
  - Other discriminative methods usually work in the same way: try out various weights until you maximize some objective
Maximum Entropy Classifiers

- **Exponential (log-linear, maxent, logistic, Gibbs) models:**
  - Turn the votes into a probability distribution:

    \[
    P(c \mid d, \lambda) = \frac{\exp \sum_i \lambda_i(c) f_i(d)}{\sum_{c'} \exp \sum_i \lambda_i(c') f_i(d)}
    \]

    - Makes votes positive.
    - Normalizes votes.

  - For any weight vector \( \{\lambda_i\} \), we get a conditional probability model \( P(c \mid d, \lambda) \).
  - We want to choose parameters that **maximize the conditional (log) likelihood** of the data:

    \[
    \log P(C \mid D, \lambda) = \sum_{(c,d) \in (C,D)} \log P(c \mid d, \lambda) = \sum_{(c,d) \in (C,D)} \log \frac{\exp \sum_i \lambda_i(c) f_i(d)}{\sum_{c'} \exp \sum_i \lambda_i(c') f_i(d)}
    \]
Building MaxEnt models

- **How to define features:**
  - Features are patterns in the input which we think the weighted vote should depend on
  - Usually features added incrementally to target errors
  - If we’re careful, adding some mediocre features into the mix won’t hurt (but won’t help either)

- **How to learn model weights?**
  - Maxent just one method
  - Use a numerical optimization package
  - Given a current weight vector, need to calculate (repeatedly):
    - Conditional likelihood of the data
    - Derivative of that likelihood wrt each feature weight
Fitting Model Parameters

- To find the parameters \( \lambda_1, \lambda_2, \lambda_3 \)
  write out the conditional log-likelihood of the training data and maximize it

\[
C\text{LogLik}(D) = \sum_{i=1}^{n} \log P(c_i | d_i)
\]

- The log-likelihood is concave and has a single maximum; use your favorite numerical optimization package

- Good large scale techniques: conjugate gradient or limited memory quasi-Newton
Parameter Estimation: Brute force vs Hill climbing

• Last lecture: brute force/exhaustive search

• Better (in this case): hill climbing / greedy search to maximize data log likelihood ($LL(D)$)
  1. Start with some values for $\{\lambda_1, \lambda_2, \lambda_3, \ldots\}$
  2. While $LL(D)$ increases (or max iterations)
     1. Foreach $x$ in $\{\lambda_1, \lambda_2, \lambda_3, \ldots\}$
        1. Increase or decrease $x$ by small amount
        2. $LL' = LL(D, x, \lambda_1, \lambda_2, \lambda_3, \ldots)$
        3. Delta = $LL'(D) - LL(D)$
        4. Remember parameters with highest delta
     2. Update parameters, $LL(D)$

• Guaranteed to find local maximum
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.
Generalization (continued)

- Design general features
  - Design (or infer!) feature classes:
    - [0-9.\-]+ $\rightarrow$ NUM
    - [A-Z] $\rightarrow$ CAP
  - Feature selection
    - Keep only salient and frequent features
    - Helps avoid overfitting

- Design algorithms to generalize
  - Penalize model complexity
  - Smoothing
Feature Generalization [Borkar et al, SIGMOD 2001]

Use cross-validation to prune hierarchy
Feature Selection: Why?

- Text collections have a large number of features
  - 10,000 – 1,000,000 unique words ... and more
- May make using a particular classifier feasible
  - Some classifiers can’t deal with 100,000 of features
- Reduces training time
  - Training time for some methods is quadratic or worse in the number of features
- Can improve generalization (performance)
  - Eliminates noise features
  - Avoids overfitting
Feature selection: how?

• Two ideas:
  – Hypothesis testing statistics:
    • Are we confident that the value of one categorical variable is associated with the value of another
    • Chi-square test
  – Information theory:
    • How much information does the value of one categorical variable give you about the value of another
    • Mutual information

• They’re similar, but $\chi^2$ measures confidence in association, (based on available statistics), while MI measures extent of association (assuming perfect knowledge of probabilities)
\( \chi^2 \) statistic (CHI)

- \( \chi^2 \) computes \((f_o - f_e)^2 / f_e\) summed over all table entries: is the observed number what you’d expect?

\[
\chi^2 (j, a) = \sum (O - E)^2 / E = (2 - 0.25)^2 / 0.25 + (3 - 4.75)^2 / 4.75
+ (500 - 502)^2 / 502 + (9500 - 9498)^2 / 9498 = 12.9 (p < .001)
\]

- The null hypothesis is rejected with confidence .999,
- since 12.9 > 10.83 (the value for .999 confidence).

<table>
<thead>
<tr>
<th></th>
<th>Term = jaguar</th>
<th>Term ≠ jaguar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class = auto</td>
<td>2 (0.25)</td>
<td>500 (502)</td>
</tr>
<tr>
<td>Class ≠ auto</td>
<td>3 (4.75)</td>
<td>9500 (9498)</td>
</tr>
</tbody>
</table>
Mutual Information

- Define *Mutual Information* as
  \[
  I(x,y) = \sum_{i,j} p(x_i,y_j) \log \left[ \frac{p(x_i,y_j)}{p(x_i)p(y_j)} \right]
  \]
- When \( x \) and \( y \) are independent \( p(x_i,y_j) = p(x_i)p(y_j) \), so \( I(x,y) \) is zero.
- When \( x \) and \( y \) are the same, the mutual information of \( x,y \) is the same as the information conveyed by \( x \) (or \( y \)) alone, which is just \( H(x) \).
- Mutual information can also be expressed as
  \[
  I(x,y) = H(x) - H(x|y) = H(y) - H(y|x)
  \]
- Mutual information is nonnegative.
- Mutual information is symmetric; i.e., \( I(x,y) = I(y,x) \).
Feature selection via Mutual Information

• In training set, choose \( k \) words which best discriminate (give most info on) the categories.

• The Mutual Information between a word, class is:

\[
I(w, c) = \sum_{e_w \in \{0,1\}} \sum_{e_c \in \{0,1\}} p(e_w, e_c) \log \frac{p(e_w, e_c)}{p(e_w)p(e_c)}
\]

— For each word \( w \) and each category \( c \)
Feature selection via MI (continued)

- For each category we build a list of $k$ most discriminating terms.
- For example (on 20 Newsgroups):
  - `sci.electronics`: circuit, voltage, amp, ground, copy, battery, electronics, cooling, ...
  - `rec.autos`: car, cars, engine, ford, dealer, mustang, oil, collision, autos, tires, toyota, ...
- Greedy: does not account for correlations between terms
- Why?
Feature Selection: MI vs. Chi^2

• Mutual Information
  – Clear information-theoretic interpretation
  – May select rare uninformative terms

• Chi-square
  – Statistical foundation
  – May select very slightly informative frequent terms that are not very useful for classification

• Just use $K$ most frequent terms?
  – No particular foundation
  – In practice, this is often 90% as good
Feature selection for NB

- In general feature selection is *necessary* for binomial NB.

- Otherwise you suffer from noise, multi-counting

- “Feature selection” really means something different for multinomial NB: **dictionary truncation**

- This “feature selection” normally isn’t needed for multinomial NB, but may help a little with quantities that are badly estimated (e.g., trigrams!)
Hidden Markov Model (HMM)

- HMMs allow you to estimate probabilities of unobserved events
- Given plain text, which underlying parameters generated the surface
- E.g., in speech recognition, the observed data is the acoustic signal and the words are the hidden parameters
HMMs and their Usage

• HMMs are very common in NLP:
  – Speech recognition (observed: acoustic signal, hidden: words)
  – Handwriting recognition (observed: image, hidden: words)
  – Part-of-speech tagging (observed: words, hidden: part-of-speech tags)
  – Machine translation (observed: foreign words, hidden: words in target language)
HMM Example (FSNLP, 9.2)

- Crazy soda machine: is turned on, runs for a while... is in unknown state at time when you put in the money

![HMM diagram showing states and transitions]

- Coke Pepsi Sprite
  CP: 0.6 0.1 0.3
  PP: 0.1 0.7 0.2

**Problem:** What is the prob. of seeing (pepsi, sprite)?

**Solution:** Sum all paths through HMM that result in this observed sequence
Part-of-Speech (POS) tagging

Input: the lead paint is unsafe
Output: the/Det lead/N paint/N is/V unsafe/Adj

Uses:
- text-to-speech (how do we pronounce “lead”?)
- can write regexps like (Det) Adj* N+ over the output
- preprocessing to speed up parser (but a little dangerous)
- if you know the tag, you can back off to it in other tasks
Part of Speech (POS) Tagging

- One basic kind of linguistic structure: syntactic word classes

Open class (lexical) words
- Nouns
  - Proper: IBM, Italy
  - Common: cat, cats, snow
- Verbs
  - Main: see, registered
- Adjectives: yellow
- Adverbs: slowly
- Numbers: 122,312, one
- Modals: can, had

Closed class (functional)
- Determiners: the, some
- Conjunctions: and, or
- Pronouns: he, its
- Prepositions: to, with
- Particles: off, up

…more
Parts of Speech: English

- DT: determiner
- EX: existential there
- FW: foreign word
- IN: preposition or conjunction, subordinating
- JJ: adjective or numeral, ordinal
- JJR: adjective, comparative
- JJJS: adjective, superlative
- MD: modal auxiliary
- NN: noun, common, singular or mass
- NNP: noun, proper, singular
- NNP’s: noun, proper, plural
- NNS: noun, common, plural
- POS: genitive marker
- PRP: pronoun, personal
- PRP$: pronoun, possessive
- RB: adverb
- RBR: acverb. comparative
- RBS: adverb, superlative
- RP: particle
- TO: "to" as preposition or infinitive marker
- UH: interjection
- VD: verb, base form
- VBD: verb, past tense
- VBG: verb, present participle or gerund
- VBN: verb, past participle
- VBP: verb, present tense, not 3rd person singular
- VBZ: verb, present tense, 3rd person singular
- WDT: WH-determiner
- WP: WH-pronoun
- WP$: WH-pronoun, possessive
- WRB: WH-adverb

and both but either or
mid-1090 nine-thirty 0.5 one
all an every no that the
there
gemeinschaft hund ich jeux
among whether out on by if
third ill-mannered regrettable
braving cheaper taller
bravest cheapest tallest
can may might will would
cabbage thermostat investment subhumanity
Motown Cougar Yvette Liverpool
Americans Materials States
undergraduates brio-a-brac averages
"s
here himself it we them
her his mine my our ours ther thy your
occasionally maddeningly adventurously
further gloomier heavier less-perfectly
best biggest nearest worst
aboard away back by on open through
to
huh howdy uh whammo shucks hok
ask bring fire see take
pleaded swiped registered saw
stirring focusing approaching erasing
diabolical imitated reunified unsettled
twist appear comprise mold postpone
bases reconstructs marks uses
that what whatever which Whinever
that what whatever which who whom
whose
however wherever where why
Degree of Supervision

- **Supervised**: Training corpus is tagged by humans
- **Unsupervised**: Training corpus isn’t tagged
- **Partly supervised**: Training corpus isn’t tagged, but you have a dictionary giving possible tags for each word

- We’ll start with the supervised case and move to decreasing levels of supervision.
Current Performance

Input: the lead paint is unsafe
Output: the/Det lead/N paint/N is/V unsafe/Adj

• How many tags are correct?
  – About 97% currently
  – But baseline is already 90%
    • Baseline is performance of stupidest possible method
    • Tag every word with its most frequent tag
    • Tag unknown words as nouns
What Should We Look At?

**correct tags**

<table>
<thead>
<tr>
<th>PN</th>
<th>Verb</th>
<th>Det</th>
<th>Noun</th>
<th>Prep</th>
<th>Noun</th>
<th>Prep</th>
<th>Det</th>
<th>Noun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill</td>
<td>directed</td>
<td>a</td>
<td>cortege of autos</td>
<td>through</td>
<td>the</td>
<td>dunes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each unknown tag is **constrained** by its word and by the tags to its immediate left and right. But those tags are unknown too ...
HMMs (for POS tagging)

- We want a model of sequences $s$ and observations $w$

\[
P(s, w) = \prod_{i} P(s_i | s_{i-1}) P(w_i | s_i)
\]

- **Assumptions:**
  - States are tag n-grams
  - Usually a dedicated start and end state / word
  - Tag/state sequence is generated by a Markov model
  - Words are chosen independently, conditioned only on the tag/state
  - These are totally broken assumptions: why?
Example (from Grishman, Fall 2004)

• Consider a random pig:
  – Observations:
    • squeeze it 10 times, and 8 times it goes "oink" and 2 times it goes "wee"
  – Model topology:
    • assume it is a stateless pig (no hidden states)
  – Parameter estimation:
    • Maximum likelihood estimate (MLE) to model the pig as a random emitter with $P("oink") = 0.8$ and $P("wee") = 0.2$. Simply, $P(x) = \text{count}(X) / \text{total number of trials}$
    • What you are doing for Project 1.1
Training an HMM (Supervised)

1. Define model topology (states, possible arcs)
2. Obtain labeled/tagged data
3. Estimate HMM parameters:
   1. Transition probabilities
   2. Emission probabilities
4. Validate on hold-out data
5. Done.
Transitions and Emissions
Transitions

- Transitions $P(s|s')$ encode well-formed tag sequences
  - In a bigram tagger, states = tags
    \[
    s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_n
    \]
    \[
    w_1 \rightarrow w_2 \rightarrow \ldots \rightarrow w_n
    \]
  - In a trigram tagger, states = tag pairs
    \[
    s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_n
    \]
    \[
    w_1 \rightarrow w_2 \rightarrow \ldots \rightarrow w_n
    \]
Estimating Transition Probabilities

- Use standard smoothing methods to estimate transitions:

\[ P(t_i | t_{i-1}, t_{i-2}) = \lambda_2 \hat{P}(t_i | t_{i-1}, t_{i-2}) + \lambda_1 \hat{P}(t_i | t_{i-1}) + (1 - \lambda_1 - \lambda_2) \hat{P}(t_i) \]

- Can get a lot fancier (e.g. KN smoothing), but in this case it doesn’t buy much

- One option: encode more into the state, e.g. whether the previous word was capitalized (Brants 00)
Estimating Emission Probabilities

\[ P(s, w) = \prod_{i} P(s_i | s_{i-1}) P(w_i | s_i) \]

- Emissions are trickier:
  - Words we’ve never seen before
  - Words which occur with tags we’ve never seen
  - One option: break out the Good-Turning smoothing
  - Issue: words aren’t black boxes:
    - 343,127.23 11-year Minteria reintroducibly
  - Unknown words usually broken into word classes
    - D^+, D^+, D^+  D^+-x^+  Xx^+  x^+“ly”
  - Another option: decompose words into features and use a maxent model along with Bayes’ rule
    \[ P(w \mid t) = P_{MAXENT}(t \mid w)P(w) / P(t) \]
Decoding

- Given these two multinomials, we can score any word / tag sequence pair

\[
\begin{align*}
<&\star,\star> &<&\star,\text{NNP}> &<&\text{NNP},\text{VBZ}> &<&\text{VBZ},\text{NN}> &<&\text{NN},\text{NNS}> &<&\text{NNS},\text{CD}> &<&\text{CD},\text{NN}> &<&\text{STOP}> \\
\text{NNP} & & & & \text{VBZ} & & \text{NN} & & \text{NNS} & & \text{CD} & & \text{NN} & . \\
\end{align*}
\]

Fed raises interest rates 0.5 percent.

\[
P(\text{NNP} | <\star,\star>) P(\text{Fed} | \text{NNP}) P(\text{VBZ} | \text{NNP}, <\star>) P(\text{raises} | \text{VBZ}) P(\text{NN} | \text{VBZ}, \text{NNP}).
\]

- In principle, we’re done – list all possible tag sequences, score each one, pick the best one (the Viterbi state sequence)

\[
\begin{align*}
\text{NNP} & \text{VBZ} & \text{NN} & \text{NNS} & \text{CD} & \text{NN} & \rightarrow & \log P = -23 \\
\text{NNP} & \text{NNS} & \text{NN} & \text{NNS} & \text{CD} & \text{NN} & \rightarrow & \log P = -29 \\
\text{NNP} & \text{VBZ} & \text{VB} & \text{NNS} & \text{CD} & \text{NN} & \rightarrow & \log P = -27
\end{align*}
\]
Finding the Best Trajectory

- Too many trajectories (state sequences) to list
- Option 1: Beam Search
  - Fed:NNP
  - Fed:VBN
  - Fed:VBD
  - Fed:NNP raises:NNS
  - Fed:NNP raises:VBZ
  - Fed:VBN raises:NNS
  - Fed:VBN raises:VBZ

- A beam is a set of partial hypotheses
- Start with just the single empty trajectory
- At each derivation step:
  - Consider all continuations of previous hypotheses
  - Discard most, keep top k, or those within a factor of the best, (or some combination)

- Beam search works relatively well in practice
  - … but sometimes you want the optimal answer
  - … and you need optimal answers to validate your beam search
Best Trajectory
Example

\[ p(\text{word seq, tag seq}) \]

The best path:

\textbf{Start} Det Adj Adj Noun \textbf{Stop} = 0.32 \times 0.0009 \ldots

det the cool directed autos
The Trellis Shape $\leftrightarrow$ Cross-Product Construction

All paths here are 5 words

So all paths here must have 5 words on output side
Actually, Trellis Isn’t Complete

$p(\text{word seq}, \text{tag seq})$

Trellis has no Det \(\rightarrow\) Det or Det \(\rightarrow\) Stop arcs; why?

The best path:

\textbf{Start} Det Adj Adj Noun \textbf{Stop} = 0.32 \times 0.0009 \ldots

the cool directed autos
Actually, Trellis Isn’t Complete

\[ p(\text{word seq}, \text{tag seq}) \]

Lattice is missing some other arcs; why?

The best path:

**Start** Det Adj Adj Noun **Stop** = 0.32 * 0.0009 ...

the cool directed autos
Actually, Trellis Isn’t Complete

\[
p(\text{word seq, tag seq})
\]

Lattice is missing some states; why?

The best path:

```
Start Det Adj Adj Noun Stop = 0.32 * 0.0009 ...
the cool directed autos
```

9/13/2010 CS730: Text Mining for Social Media, F2010
Find best path from Start to Stop

- Use dynamic programming (Viterbi alg)
  - What is best path from Start to each node?
  - Work from left to right
  - Each node stores its best path from Start (as probability plus one backpointer)

- Special acyclic case of Dijkstra’s shortest-path alg.
- Faster if some arcs/states are absent
Viterbi Algorithm

- Dynamic program for computing
  \[ \delta_i(s) = \max_{s_0 \ldots s_{i-1}} P(s_0 \ldots s_{i-1}, s, w_1 \ldots w_{i-1}) \]
  - The score of a best path up to position i ending in state s
  \[ \delta_0(s) = \begin{cases} 1 & \text{if } s = \langle \bullet, \bullet \rangle \\ 0 & \text{otherwise} \end{cases} \]
  \[ \delta_i(s) = \max_{s'} P(s | s') P(w | s') \delta_{i-1}(s') \]
  - Also store a backtrace
  \[ \psi_i(s) = \arg \max_{s'} P(s | s') P(w | s') \delta_{i-1}(s') \]
- Memoized solution
- Iterative solution
In Summary

• We are modeling $p(\text{word seq}, \text{tag seq})$
• The tags are hidden, but we see the words
• Is tag sequence $X$ likely with these words?
• Noisy channel model is a “Hidden Markov Model”:

- **Find $X$ that maximizes probability** product
Partially Supervised Tagging

- Accuracy of tagging degrades outside of domain; Often make errors on important words (e.g., protein names)
  \[ \lambda = (A, B, \pi) \]

- We assumed that we know the underlying model

- Often these parameters are estimated on annotated training data, which has two drawbacks:
  - Annotation is difficult and/or expensive
  - Training data is different from the current data

- We want to maximize the parameters with respect to the current data, i.e., we’re looking for a model \( \lambda' \), such that
  \[ \lambda' = \arg \max_{\lambda} P(O \mid \lambda) \]
Can we train HMM on un-labeled corpus?

- Goal: select parameters maximizing likelihood of training corpus (observations)
  - $a_{ij}$ (transition probability from state i to state j)
  - $b_j(w)$ (probability of emitting word w from state i)

- Procedure more complicated:
  - Estimates for emission probs depend on hidden states (not explicitly given)
  - No closed-form solution to estimate parameters known
  - Use particular form of hill climbing:
    - Expectation Maximization (EM)
      - Baum-Welch of Forward-Backward algorithm (M&S p. 333)
Baum-Welch (1)

- Want to estimate:
  \[ a_{ij} \] (transition probability from state i to state j)
- If we had tagged corpus:
  \[ a_{ij} = \frac{\text{count(transition from state i to state j)}}{\text{count(transition from state i to any state)}} \]
- Don’t have one. So:
  - **Assume some initial values** for \( a_{ij} \) and \( b_j(w) \)
  - Use \( a_{ij} \) and \( b_j(w) \) to compute expected counts \( (E) \)
  - Then use \( E \) to *re-estimate* \( a_{ij} \rightarrow a'_{ij} \)
  - \( a'_{ij} = \frac{E(\text{transition from state i to state j})}{E(\text{transition from state i to any state})} \)
Forward Recurrence

- What’s the probability of being in state $s_i$ at time $t$ and going to state $s_j$, given the current model and parameters?
Backward Recurrence

\[ \beta_t(i) = \sum_j \beta_{t+1}(j) a_{ij} b_j(o_{t+1}) \]

Diagram showing the backward recurrence process with states \( q_1, q_2, q_3, \ldots, q_N \) and emissions \( o_t, o_{t-1}, o_{t+1} \). The recurrence relation is illustrated through the graph, showing transitions and emissions at each state.
Notation List

• **Forward probability:** $\alpha_t(i)$
  The probability of being in state $s_i$, given the partial observation $o_1, \ldots, o_t$

• **Backward probability:** $\beta_t(i)$
  The probability of being in state $s_i$, given the partial observation $o_{t+1}, \ldots, o_T$

• **Transition probability:** $a_{ij}$
  The probability of going from state $s_i$, to state $s_j$, given the complete observation $o_1, \ldots, o_T$

• **Emission probability:** $b_j(w)$

• **State probability:** $\gamma_t(i)$
  The probability of being in state $s_i$, given the complete observation $o_1, \ldots, o_T$
Baum-Welch (2)  
\[ a_{ij} = \frac{\text{count}(\text{transition from state } i \text{ to state } j)}{\text{count}(\text{transition from state } i \text{ to any state})} \]

- To compute the expected values, we use the values computed by the Viterbi algorithm.
  - Assume the input is \( w_1, \ldots, w_T \), and the states are numbered 1 to \( N \).
  - The forward probability \( \alpha_j(t) \) is the probability of being in state \( j \) and generating the first \( t \) words of the input.
  - The backward probability \( \beta_i(t) \) is the mirror image: the probability, starting in state \( i \), of generating the words from word \( t \) through the end of the input.
  - Let \( \tau_t(i,j) \) be the probability of being in state \( i \) for word \( t \) and state \( j \) for word \( t+1 \):
    \[ \tau_t(i,j) = \alpha_j(t) a_{ij} b_j(w_{t+1}) \beta_j(t+1) / \alpha_N(T) \]
    then:
    \[ a'_{ij} = \frac{\sum_{t=1 \text{ to } T-1} \tau_t(i,j)}{\sum_{t=1 \text{ to } T-1} \sum_{j=1 \text{ to } N} \tau_t(i,j)} \]
Baum-Welch (4): Emission Probabilities

- Emission probabilities:
  - \( b_j(w) = \frac{\text{count}(\text{emissions } w \text{ from state } j)}{\text{count}(\text{all words emitted from state } j)} \)

- Emission probabilities are re-estimated as
  \[ \hat{b}_i(k) = \frac{\text{expected number of times in state } s_i \text{ and observe symbol } v_k}{\text{expected number of times in state } s_i} \]

- Formally:
  \[ \hat{b}_i(k) = \frac{\sum_{t=1}^{T} \delta(o_t, v_k) \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} \]

  Where \( \delta(o_t, v_k) = 1, \text{ if } o_t = v_k, \text{ and } 0 \text{ otherwise} \)

  * Note that \( \delta \) here is the Kronecker delta function
Re-estimating Initial State Probabilities

- Initial state distribution: $\pi_i$ is the probability that $s_i$ is a start state.
- Re-estimation is easy:
  \[ \hat{\pi}_i = \text{expected number of times in state } s_i \text{ at time } 1 \]
- Formally:
  \[ \hat{\pi}_i = \gamma_1(i) \]
The Updated Model (Summary)

- Starting from \( \lambda = (A, B, \pi) \) we get to \( \lambda' = (\hat{A}, \hat{B}, \hat{\pi}) \) by the following update rules:

\[
\hat{a}_{i,j} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \\
\hat{b}_{i}(k) = \frac{\sum_{t=1}^{T} \delta(o_t,v_k) \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} \\
\hat{\pi}_{i} = \gamma_1(i)
\]
Summary – Baum-Welch Algorithm

• Start with initial HMM
• Calculate, using F-B, the likelihood to get our observations given that a certain hidden state was used at time i.
• Re-estimate the HMM parameters
• Continue until convergence
• Baum showed this to monotonically improve data likelihood (not necessarily accuracy!)
Does It Work for POS tagging?

- Using forward-backward for training a model without a tagged corpus: the Xerox tagger
  - Doug Cutting; Julian Kupiec; Jan Pedersen; Penelope Sibun. A Practical Part-of-Speech Tagger. ANLP 1992.
  - 96% accuracy on Brown corpus
Does it work (2)?

- Some (discouraging) experiments [Merialdo 94]

- Setup:
  - You know the set of allowable tags for each word
  - Fix k training examples to their true labels
    - Learn $P(w|t)$ on these examples
    - Learn $P(t|t_1, t_2)$ on these examples
  - On n examples, re-estimate with EM

- Note: we know allowed tags but not frequencies
### Meriado 94: Results

<table>
<thead>
<tr>
<th>Iter</th>
<th>Correct tags (% words) after ML on 1M words</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>77.0</td>
</tr>
<tr>
<td>1</td>
<td>80.5</td>
</tr>
<tr>
<td>2</td>
<td>81.8</td>
</tr>
<tr>
<td>3</td>
<td>83.0</td>
</tr>
<tr>
<td>4</td>
<td>84.0</td>
</tr>
<tr>
<td>5</td>
<td>84.8</td>
</tr>
<tr>
<td>6</td>
<td>85.3</td>
</tr>
<tr>
<td>7</td>
<td>85.8</td>
</tr>
<tr>
<td>8</td>
<td>86.1</td>
</tr>
<tr>
<td>9</td>
<td>86.3</td>
</tr>
<tr>
<td>10</td>
<td>86.6</td>
</tr>
</tbody>
</table>
Expectation Maximization

- The forward-backward algorithm is an instance of the more general EM algorithm
  - The E Step: Compute the forward and backward probabilities for a given model
  - The M Step: Re-estimate the model parameters
  - To be continued next lecture...
POS-tagging Summary

- The task

- Supervised tagging
  - Church 1989: HMM
  - MaxEnt:
    - Ratnaparki 1999
  - ENCG (rule-based)

- Unsupervised (HMMs with Baum Welch)

- Partially supervised (TBL)
  - Brill 1994
POS Tagging final thoughts (1)

• There is no unique set of part-of-speech tags.
  – Words can be grouped in different ways to capture different generalizations
  – Coarser or finer categories
  – The first large (1 M words) tagged corpus was the Brown Corpus (in the 1960’s); used a set of 87 tags.
  – Currently the most widely used tag sets are those for the Penn Tree Bank (45 tags)
  – Having a relatively small tag set makes it somewhat easier for people (and programs) to tag the text, but it loses distinctions which may be grammatically important.
    • E.g., Penn Tree Bank does not distinguish between prepositions and subordinating conjunctions, or between auxiliary and main verbs.
POS tagging final thoughts (2)

- Naïve memorization baseline:
  - Memorize (and predict) most common tag for each word
    - **Accuracy**: 90%
    - With good unknown word model: 93.7%

- Rule-based approach (English-specific)
  - Constraint grammar by Fred Karlsson et al. at University of Helsinki.
  - Used tag set avoided some of the ambiguities of other tag sets
  - Tagging procedure:
    1. Dictionary look-up to assign each word all possible parts of speech
    2. Apply set of 'constraint rules' to eliminate a particular part of speech.
  - These rules were all written by hand, using the annotated corpus to check the correctness of the rules. **The final ENGCG (English constraint grammar) system had 3600 rules!**
  - **Accuracy**: 98.5% **on their tag set**
POS Tagging Final Thoughts (3)

- Supervised (train on Brown or Penn Tree Bank):
  - TNT: HMM trigram (2-tags back) tagger
    - Careful smoothing
    - Suffix trees for emissions
    - 96.7% accurate on WSJ
    - Note: at least 2% error in annotated data.
  
    - For a sentence with words $w_1, ..., w_n$ and tags $t_1, ..., t_n$, the features combined with $t_i$ were $w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}, t_{i-1}$, and $t_{i-2}t_{i-1}$.
    - Feature selection: frequency $\geq 10$
    - Used beam search (not full Viterbi – why?), keeping the N best tag sequences at each word.
    - Accuracy: 96.5%.
    - Ratnaparkhi points out that MaxEnt has the advantage of allowing specialized features (like TBL) and providing probabilities (like an HMM).
      - As an example of specialized features, he tried using conjunctions of features for difficult words, but found very little gain.
      - He noted that he could improve performance to 97% by using training and test data from only one of the Treebank annotators.

Baseline: 90% naïve or 93.7% better
POS Tagging Final Thoughts (4)

• TBL (Brill 95): Transformation-Based tagger
  – Label training set with most frequent tags
    DT MD VBD VBD.
    The can was rusted.
  – Add transformation rules which reduce training mistakes
    MD → NN : DT __
    VBD → VBN : VBD __.
  – Stop when no transformations do sufficient good
  – Probably the most widely used tagger
  – Accuracy: 96.6% / 82.0 %

Baseline: 90% naïve or 93.7% better
POS Tagging Final Thoughts (6)

Baseline: 90% naïve or 93.7% better

- Unsupervised POS taggers
  - Xerox tagger:
    - Start with initial HMM
    - Use Baum-Welch form of Expectation-Maximization to iteratively re-estimate parameter values (maximize likelihood of data).
    - Continue until convergence
    - Accuracy: 96%
  
  - Meriado:
    - Showed that unsupervised training (Baum-Welch) only helpful for small amount of training data. Beyond that, hurts

  - Brill: Unsupervised TBL
    - Words are initially assigned all their possible parts of speech, based on a dictionary.
    - Transformations are scored by the formula:
      \[
      \text{incontext}(Y, C) - \left( \frac{\text{freq}(Y)}{\text{freq}(X)} \right) \times \text{incontext}(X, C)
      \]
    - where \( \text{freq}(A) \) is the number of words in the corpus unambiguously tagged with part-of-speech \( A \), and \( \text{incontext}(A, C) \) is the number of words unambiguously tagged with part-of-speech \( A \) in context \( C \).
    - When some tagged data is available, Brill first applies his unsupervised tagger, and then his supervised tagger.
• Penn tag set now the standard for assessing English tagging, but forces annotators to make hard decisions \(\rightarrow\) inter-annotator error of 3%

• There are many supervised learning methods which can get 96-97% accuracy on held-out Wall Street Journal data;
  – HMMs (variants), TBL, and MaxEnt. The error in the Penn Treebank probably masks differences between the methods

• Unsupervised methods can do quite well using a dictionary, bootstrapping from those (many) words which have only a single part-of-speech;
  – unsupervised TBL can do almost as well as supervised methods
  – adding a bit of supervised training gets near the performance of supervised training on the entire PTB;
  – Baum-Welch (Xerox tagger) can do almost as well, but needs clever grouping of ambiguity classes

• ENGCG tags can be more consistently assigned, and hand-written ENGCG rules can get 98%+ accuracy
Big Picture (1): view 1

• Words: “atomic” units of meaning
  – Language modeling: predicting next character, phoneme, word
  – Parts of speech, word classes, collocations
  – Applications: text classification, IR, ...

• Sentence structure
  – Context free grammars (CFGs)
  – Parsing sentences
  – Partial parsing: chunking
  – Full parsing methods
  – Applications: IE, Machine Translation, grammar checking

• Putting it all together: Semantics / “meaning”
  – Named Entities
  – Semantic role labeling
  – Co-reference resolution
  – Entity disambiguation
  – Relation|Fact|Event Extraction
  – Applications: IR, question answering, text/data mining, intelligence
Readings for Next Week (9/20)

• FSNLP: Chapters 6, 7, 9, 10.