Reminder: HMMs

- 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?

Finding the Best Trajectory

- Too many trajectories (state sequences) to list
- Option 1: Beam Search
  - 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
  - Beam search works ok in practice
    - … but sometimes you want the optimal answer
    - … and you need optimal answers to validate your beam search
    - … and there’s usually a better option than naïve beams

The State Lattice / Trellis

- Too many trajectories (state sequences) to list
- Option 1: Beam Search
  - 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

- Beam search works ok in practice
  - … but sometimes you want the optimal answer
  - … and you need optimal answers to validate your beam search
  - … and there’s usually a better option than naïve beams
The Viterbi Algorithm

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

Unsupervised Tagging?

- AKA part-of-speech induction
- Task:
  - Raw sentences in
  - Tagged sentences out
- Obvious thing to do:
  - Start with a (mostly) uniform HMM
  - Run Expectation Maximization
  - Inspect results

Clustering / Pattern Detection

- Soccer team wins match
- Stocks close up 3%
- Investing in the stock market has …
- The first game of the world series …

- Problem: There are many patterns in the data, most of which you don’t care about.

Clustering vs. Classification

- Classification: we specify which pattern we want, features uncorrelated with that pattern are idle

|           | P(w|sports) | P(w|politics) | P(w|headline) | P(w|story) |
|-----------|-------------|---------------|---------------|------------|
| the       | 0.1         | 0.1           | 0.05          | 0.1        |
| game      | 0.02        | 0.005         | 0.01          | 0.01       |
| win       | 0.02        | 0.01          | 0.01          | 0.01       |

- Clustering: the clustering procedure locks on to whichever pattern is most salient, statistically
  - \( P(\text{content words} \mid \text{class}) \) will learn topics
  - \( P(\text{length, function words} \mid \text{class}) \) will learn style
  - \( P(\text{characters} \mid \text{class}) \) will learn “language”
Model-Based Clustering

- Clustering with probabilistic models:
  - Unobserved (Y) Observed (X)
  - ? LONDON -- Soccer team wins match
  - ? NEW YORK -- Stocks close up 3%
  - ? Investing in the stock market has …
  - ? The first game of the world series …

  Build a model of the domain: \( P(x, y, \theta) \)
  Often: find \( \theta \) to maximize: \( P(x|\theta) = \sum_y P(x, y|\theta) \)

- Problem 2: The relationship between the structure of your model and the kinds of patterns it will detect is complex.

Learning Models with EM

- Hard EM:
  E-step: Find best “completions” \( Y \) for fixed \( \theta \)
  M-step: Find best parameters \( \theta \) for fixed \( Y \)

- Example: K-Means

- Problem 3: Data likelihood (usually) isn’t the objective you really care about
- Problem 4: You can’t find global maxima anyway

Hard EM for Naïve-Bayes

- Procedure: (1) we calculate best completions:
  \[ y^* = \arg\max_y P(y) \prod_i P(x_i|y) \]

  (2) compute relevant counts from the completed data:
  \[ c(w, y) = \sum_{x \in D} \sum_i [1(x_i = w, y^* = y)] \]

  (3) compute new parameters from these counts (divide)
  (4) repeat until convergence
  Can also do this when some docs are labeled

EM: More Formally

- Hard EM: \( \arg\max_{\theta, y} P(y, \theta|x) \)

- Improve completions
  \[ y^* = \arg\max_y P(y, \theta^*|x) = \arg\max_y P(y|x, \theta^*) \]

- Improve parameters
  \[ \theta^* = \arg\max_{\theta} P(y^*, \theta|x) = \arg\max_{\theta} P(\theta|x, y^*) \]

- Each step either does nothing or increases the objective
Soft EM for Naïve-Bayes

- Procedure: (1) calculate posteriors (soft completions):
\[ P(y|x) = \frac{P(y) \prod_i P(x_i|y)}{\sum_{y'} P(y') \prod_i P(x_i|y')} \]

- (2) compute expected counts under those posteriors:
\[ c(w, y) = \sum_{x \in D} P(y|x) \sum_i [1(x_i = w, y)] \]

- (3) compute new parameters from these counts (divide)
- (4) repeat until convergence

EM in General

- We’ll use EM over and over again to fill in missing data
- Convenience Scenario: we want P(x), including y just makes the model simpler (e.g. mixing weights for language models)
- Induction Scenario: we actually want to know y (e.g. clustering)
- NLP differs from much of statistics / machine learning in that we often want to interpret or use the induced variables (which is tricky at best)

- General approach: alternately update y and \( \theta \)
  - E-step: compute posteriors \( P(y|x, \theta) \)
    - This means scoring all completions with the current parameters
    - Usually, we do this implicitly with dynamic programming
  - M-step: fit \( \theta \) to these completions
    - This is usually the easy part – treat the completions as (fractional) complete data
  - Initialization: start with some noisy labelings and the noise adjusts into patterns based on the data and the model
  - We’ll see lots of examples in this course

- EM is only locally optimal (why?)

EM for HMMs: Process

- Alternate between recomputing distributions over hidden variables (the tags) and reestimating parameters
- Crucial step: we want to tally up how many (fractional) counts of each kind of transition and emission we have under current params:

\[ \text{count}(s \rightarrow s') = \sum_i P(t_{i-1} = s, t_i = s'|w) \]
\[ \text{count}(w, s) = \sum_{i:w_i = w} P(t_i = s|w) \]

- But we need a dynamic program to help, because there are too many sequences to sum over to compute these marginals

EM for HMMs: Quantities

- Cache total path values:

\[ \alpha_i(s) = P(w_0 \ldots w_i, s_i) = \sum_{s_{i-1}} P(s_i | s_{i-1}) P(w_i | s_i) \alpha_{i-1}(s_{i-1}) \]
\[ \beta_i(s) = P(w_i + 1 \ldots w_n | s_i) = \sum_{s_{i+1}} P(s_{i+1} | s_i) P(w_{i+1} | s_{i+1}) \beta_{i+1}(s_{i+1}) \]

- Can calculate in \( O(s^2n) \) time (why?)
EM for HMMs: Process

- From these quantities, can compute expected transitions:

\[
\text{count}(s \rightarrow s') = \frac{\sum_i \alpha_i(s)P(s'|s)P(w_i|s)\beta_{i+1}(s')}{P(w)}
\]

- And emissions:

\[
\text{count}(w, s) = \frac{\sum_i:w_i=w \alpha_i(s)\beta_{i+1}(s)}{P(w)}
\]
Merialdo: Setup

- 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

Merialdo: Results

<table>
<thead>
<tr>
<th>Iter</th>
<th>Correct tags (% words) after ML on 1M words</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>90.0 95.4 96.2 96.6 96.9 97.0</td>
</tr>
<tr>
<td>1</td>
<td>82.5 92.6 95.8 96.3 96.6 96.7 96.8</td>
</tr>
<tr>
<td>2</td>
<td>84.8 93.0 95.7 96.1 96.3 96.4 96.4</td>
</tr>
<tr>
<td>3</td>
<td>83.0 93.5 95.4 95.8 96.1 96.2 96.2</td>
</tr>
<tr>
<td>4</td>
<td>84.0 93.0 95.2 95.5 95.8 96.0 96.0</td>
</tr>
<tr>
<td>5</td>
<td>84.8 92.9 95.1 95.4 95.6 95.8 95.8</td>
</tr>
<tr>
<td>6</td>
<td>85.3 92.8 94.9 95.2 95.5 95.6 95.7</td>
</tr>
<tr>
<td>7</td>
<td>85.8 92.8 94.7 95.1 95.3 95.5 95.5</td>
</tr>
<tr>
<td>8</td>
<td>86.1 92.7 94.6 95.0 95.2 95.4 95.4</td>
</tr>
<tr>
<td>9</td>
<td>86.3 92.6 94.5 94.9 95.1 95.3 95.3</td>
</tr>
<tr>
<td>10</td>
<td>86.6 92.6 94.4 94.8 95.0 95.2 95.2</td>
</tr>
</tbody>
</table>

Domain Effects

- Accuracies degrade outside of domain
  - Up to triple error rate
  - Usually make the most errors on the things you care about in the domain (e.g. protein names)

- Open questions
  - How to effectively exploit unlabeled data from a new domain (what could we gain?)
  - How to best incorporate domain lexica in a principled way (e.g. UMLS specialist lexicon, ontologies)

Tagging Search Queries [Ganchev et al. '12]

ADJ or NOUN?
Global Discriminative Taggers

- Newer, higher-powered discriminative sequence models
  - CRFs (also perceptrons, M3Ns)
  - Do not decompose training into independent local regions
  - Can be slow to train – require repeated inference on training set

- Differences can be substantial for some sequence tasks

- However: one issue worth knowing about in local models
  - “Label bias” and other explaining away effects
  - MEMM taggers’ local scores can be near one without having both good “transitions” and “emissions”
  - This means that often evidence doesn’t flow properly
  - Why isn’t this a big deal for POS tagging?
  - Also: in decoding, condition on predicted, not gold, histories

Perceptron Taggers [Collins ’01]

- Linear models:
  \[ \text{score}(t|w) = \lambda^T f(t, w) \]
  \[ = \lambda^T \sum_i f(t_i, t_{i-1}, w, i) \]
  \[ \text{... allow us to predict with the Viterbi algorithm} \]
  \[ t^* = \arg \max_t \text{score}(t|w) \]
  \[ \text{... which means we can train with the perceptron algorithm (or related updates, like MIRA)} \]

Label Bias

- The mass that arrives at the state must be distributed among the possible successor states

- States with one outgoing transition ignore observation

\[ \begin{array}{c}
0 \quad r_c- \\
1 \quad e_- \\
2 \quad \text{b:rib} \\
3 \\
4 \quad \text{b:rob} \\
5 \quad o_- \\
\end{array} \]

CRFs

- Make a maxent model over entire taggings
  - MEMM
    \[ P(t|w) = \prod_i \frac{1}{Z(i)} \exp(\lambda^T f(t_i, t_{i-1}, w, i)) \]
  - CRF
    \[ P(t|w) = \frac{1}{Z(w)} \exp(\lambda^T f(t, w)) \]
    \[ = \frac{1}{Z(w)} \exp(\lambda^T \sum_i f(t_i, t_{i-1}, w, i)) = \frac{1}{Z(w)} \prod_i \phi_i(t_i, t_{i-1}) \]
CRFs

- Like any maxent model, derivative is:
  \[
  \frac{\partial L(\lambda)}{\partial \lambda} = \sum_k \left( f_k(t^k) - \sum_t P(t|w_k) f_k(t) \right)
  \]

- So all we need is to be able to compute the expectation each feature, for example the number of times the label pair DT-NN occurs, or the number of times NN-interest occurs in a sentence.

- How many times does, say, DT-NN occur at position 10? The ratio of the scores of trajectories with that configuration to the score of all trajectories.

- This requires exactly the same forward-backward score ratios as for EM, but using the local potentials \(\phi\) instead of the local probabilities.

Advantages of CRFs

- Allow easy integration of millions of features:
  - At each state, can condition on entire input
  - Observation dependent transitions

- Training is easily parallelizable

- Verify versatile use-cases

- Give state-of-the-art performance on a variety of tasks, in various fields (NLP, Vision, Bio)

NP Chunking

- “Shallow Parsing,” identify all noun phrases
- Good for Information Extraction tasks
- Typically use second order sequence models

Model | Accuracy
--- | ---
MEMM | 93.70
Voted Perceptron | 94.09
CRF | 94.38

Named Entity Recognition

- Find “Names”
- Organizations, Locations, People, etc.
- Capitalization strong cue in English, but what about other languages?

<table>
<thead>
<tr>
<th>Language</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>75%-90%</td>
</tr>
<tr>
<td>German</td>
<td>60%-80%</td>
</tr>
</tbody>
</table>
Generative vs. Discriminative

- Generative models tend to have higher asymptotic error, but they approach their asymptotic error faster than discriminative ones with number of training examples logarithmic in the number of parameters rather than linear.
- Discriminative models make no independence assumptions for observations, and are therefore more flexible for incorporating overlapping features.
- Training time for generative models usually much lower. Testing time comparable.

Overview