Word Senses

- 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
    - [tennis] It was Agassi’s turn to serve
    - [legal] He was served by the sheriff

Various Approaches to WSD

- Unsupervised learning
  - Bootstrapping (Yarowsky 95)
  - Clustering

- Indirect supervision
  - From thesauri
  - From WordNet
  - From parallel corpora

- Supervised learning
  - Most systems do some kind of supervised learning
  - Many competing classification technologies perform about the same (it’s all about the knowledge sources you tap)
  - Problem: training data available for only a few words

Resources

- WordNet
  - Hand-build (but large) hierarchy of word senses
  - Basically a hierarchical thesaurus

- SensEval
  - A WSD competition, of which there have been 3 iterations
  - Training/test sets for a wide range of words, difficulties, and parts-of-speech
  - Bake-off where lots of labs tried lots of competing approaches

- SemCor
  - A big chunk of the Brown corpus annotated with WordNet senses

- Other Resources
  - The Open Mind Word Expert
  - Parallel texts
  - Flat thesauri
**Knowledge Sources**

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

\[ P(c, w_1, w_2, \ldots, w_n) = P(c) \prod_{i=1}^{k} P(w_i | c) \]

**Better Features**

- There are smarter features:
  - Argument selectional preference:
    - serve NP(meals) vs. serve NP(paper) vs. serve NP(country)
  - Subcategorization:
    - [function] serve PP(as)
    - [enable] serve VP(to)
    - [tennis] serve <transitive>
  - [food] serve NP(NP[food])
  - Can capture poorly (but robustly) with local windows
  - ... but we can also use a parser and get these features explicitly
  - Other constraints (Yarowsky 95)
    - One-sense-per-discourse (only true for broad topical distinctions)
    - One-sense-per-collocation (pretty reliable when it kicks in: manufacturing plant, flowering plant)

**Weighted Windows with NB**

- Distance conditioning
  - Some words are important only when they are nearby

\[ P(c, w_1, \ldots, w_n) = P(c) \prod_{i=1}^{k} P(w_i | c, bin(i)) \]

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

**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

**A Discriminative Approach**

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

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

- 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?)

**Feature Representations**

\[ f(d) \]

- 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 \( f \) which count the occurrences of certain patterns in the input

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

- For a pair \((c, d)\), we take a weighted vote for each class:
\[
vote(c | d) = \exp \sum_i \lambda_i f_i(d)
\]

<table>
<thead>
<tr>
<th>Feature</th>
<th>Food</th>
<th>Jail</th>
<th>Tennis</th>
</tr>
</thead>
<tbody>
<tr>
<td>object-head:meals</td>
<td>2.0*1</td>
<td>1.5*1</td>
<td>1.5*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: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>0.3</td>
<td>-2.6</td>
</tr>
</tbody>
</table>

- There are many ways to set these weights:
  - Perception: 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.

Building a Maxent Model

- 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):
    - Derivative of that likelihood wrt each feature weight.

The Likelihood Value

- The (log) conditional likelihood is a function of the iid data \((C, D)\) and the parameters \(\lambda\):
\[
\log P(C | D, \lambda) = \log \prod_{i \in D} P(c | d, \lambda) = \sum_{i \in D} \log P(c | d, \lambda)
\]
- If there aren’t many values of \(c\), it’s easy to calculate:
\[
\log P(C | D, \lambda) = \sum_{i \in \text{instances}} \log \exp \sum_c \exp \lambda_i f_i(d) = \sum_{i \in \text{instances}} \log \sum_c \exp \lambda_i f_i(d)
\]
- We can separate this into two components:
\[
\log P(C | D, \lambda) = \sum_{i \in \text{instances}} \log \exp \sum_c \lambda_i f_i(d) - \sum_{i \in \text{instances}} \log \sum_c \exp \lambda_i f_i(d)
\]

Maximum-Entropy Classifiers

- Exponential (log-linear, maxent, logistic, Gibbs) models:
  - Turn the votes into a probability distribution:
\[
P(c | d, \lambda) = \frac{\exp \sum_i \lambda_i f_i(d)}{\sum_c \exp \sum_i \lambda_i f_i(d)}
\]
  - Makes votes positive.
  - Normalizes votes.

- For any weight vector \(\{\lambda\}\), we get a conditional probability model \(P(c | d, \lambda)\).
- We want to choose parameters that maximize the conditional (log) likelihood of the data:
\[
\log P(C | D, \lambda) = \sum_{i \in \text{instances}} \log P(c | d, \lambda) = \sum_{i \in \text{instances}} \log \sum_c \exp \lambda_i f_i(d)
\]

The Derivative I: Numerator

\[
\frac{\partial N(\lambda)}{\partial \lambda_i} = \sum_{d \in D} \frac{\partial \sum \exp \lambda_i f_i(d)}{\partial \lambda_i} = \sum_{d \in D} \frac{f_i(d)}{\sum_{c \in C} \exp \lambda_i f_i(d)}
\]

The Derivative II: Denominator

\[
\frac{\partial M(\lambda)}{\partial \lambda_i} = \sum_{d \in D} \frac{\partial \sum \exp \lambda_i f_i(d)}{\partial \lambda_i} = \sum_{d \in D} \frac{\sum \exp \lambda_i f_i(d)}{\prod_{c \in C} \exp \lambda_i f_i(d)}
\]

Derivative of the numerator is the empirical count \(f_i \propto c \wedge d\)

E.g.: we actually saw the word “dish” with the “food” sense 3 times (maybe twice in one example and once in another).
The Derivative III

\[ \frac{\partial \log P(c \mid D, \lambda)}{\partial \lambda(c)} = \text{actual count}(f, c) - \text{predicted count}(f, c) \]

The optimum parameters are the ones for which each feature’s predicted expectation equals its empirical expectation. The optimum distribution is:
- Always unique (but parameters may not be unique)
- Always exists (if features counts are from actual data).

Smoothing: Issues of Scale

- Lots of features:
  - NLP maxent models can have over 1M features.
  - Even storing a single array of parameter values can have a substantial memory cost.
- Lots of sparsity:
  - Overfitting very easy – need smoothing!
  - Many features seen in training will never occur again at test time.
- Optimization problems:
  - Feature weights can be infinite, and iterative solvers can take a long time to get to those infinities.

Smoothing: Issues

- We have a function to optimize:
  \[ \log P(c \mid D, \lambda) = \sum_{(c,d) \in D} \exp \left( \lambda(c) f(d) \right) \]
- We know the function’s derivatives:
  \[ \frac{\partial \log P(c \mid D, \lambda)}{\partial \lambda(c)} = \text{actual count}(f, c) - \text{predicted count}(f, \lambda) \]
- Ready to feed it into a numerical optimization package...
- What did any of that have to do with entropy?

Smoothing: Issues

- Assume the following empirical distribution:

<table>
<thead>
<tr>
<th>Heads</th>
<th>Tails</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0.8</td>
</tr>
</tbody>
</table>

- Features: \{Heads\}, \{Tails\}
- We’ll have the following model distribution:
  \[
P_{\text{HEADS}} = \frac{e^{\lambda h}}{e^{\lambda h} + e^{\lambda t}} \quad P_{\text{TAILS}} = \frac{e^{\lambda t}}{e^{\lambda h} + e^{\lambda t}}
\]
- Really, only one degree of freedom (\(\lambda = \lambda_H - \lambda_T\))

\[
P_{\text{HEADS}} = \frac{e^\lambda}{e^\lambda + e^{-\lambda}} \quad P_{\text{TAILS}} = \frac{e^{-\lambda}}{e^\lambda + e^{-\lambda}}
\]

Smoothing: Early Stopping

- In the 4/0 case, there were two problems:
  - The optimal value of \(\lambda\) was \(\infty\), which is a long trip for an optimization procedure.
  - The learned distribution is just as spiked as the empirical one – no smoothing.
- One way to solve both issues is to just stop the optimization early, after a few iterations.
  - The value of \(\lambda\) will be finite (but presumably big).
  - The optimization won’t take forever (clearly).
  - Commonly used in early maxent work.
Smoothing: Priors (MAP)

- What if we had a prior expectation that parameter values wouldn’t be very large?
- We could then balance evidence suggesting large parameters (or infinite) against our prior.
- The evidence would never totally defeat the prior, and parameters would be smoothed (and kept finite!).
- We can do this explicitly by changing the optimization objective to maximum posterior likelihood:

\[
\log P(C, \lambda | D) = \log P(\lambda) + \log P(C | D, \lambda)
\]

Posterior Prior Evidence

If we use gaussian priors:

- Trade off some expectation-matching for smaller parameters.
- When multiple features can be recruited to explain a data point, the more common ones generally receive more weight.
- Accuracy generally goes up!

Change the objective:

\[
\log P(C, \lambda | D) = \log P(C | D, \lambda) - \log P(\lambda)
\]

\[
\log P(C|D,\lambda) = \sum_{i \in \{D\}} \log \frac{1}{\sqrt{2\pi} \sigma} \exp \left( \frac{(\lambda_i - \mu)^2}{2\sigma^2} \right)
\]

Change the derivative:

\[
\frac{\partial \log P(C, \lambda | D)}{\partial \lambda_i} = \text{actual}(f_i, C) - \text{predicted}(f_i, \lambda) - (\lambda_i - \mu) / \sigma^2
\]

Example: NER Smoothing

Because of smoothing, the more common prefixes have larger weights even though entire-word features are more specific.

Local Context

<table>
<thead>
<tr>
<th>Feature</th>
<th>Feature Type</th>
<th>PERS</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous word</td>
<td>-0.75</td>
<td>0.94</td>
<td></td>
</tr>
<tr>
<td>Current word</td>
<td>0.05</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>Beginning tag</td>
<td>0.46</td>
<td>0.80</td>
<td></td>
</tr>
<tr>
<td>Current POS tag</td>
<td>-0.42</td>
<td>-0.15</td>
<td></td>
</tr>
<tr>
<td>Prev and cur tags</td>
<td>0.13</td>
<td>0.55</td>
<td></td>
</tr>
<tr>
<td>Previous state</td>
<td>-0.79</td>
<td>-0.91</td>
<td></td>
</tr>
<tr>
<td>Current signature</td>
<td>0.80</td>
<td>0.44</td>
<td></td>
</tr>
<tr>
<td>Prev state, cur sig</td>
<td>0.68</td>
<td>0.37</td>
<td></td>
</tr>
<tr>
<td>Prev-curr sig</td>
<td>-0.30</td>
<td>0.37</td>
<td></td>
</tr>
<tr>
<td>Prev-curr sig</td>
<td>0.30</td>
<td>0.82</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>-0.58</td>
<td>2.48</td>
<td></td>
</tr>
</tbody>
</table>