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
    - 80% 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

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

- OtherResources
  - 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 ...

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

Weighted Windows with NB

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

\[
P(c, w_1, \ldots, w_{i-1}, w_i, w_{i+1}, \ldots, w_n) = P(c) \prod_{j=i}^{k} P(w_j \mid c, \text{bin}(j))
\]

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

\[
P(c, w_1, \ldots, w_{i-1}, w_i, w_{i+1}, \ldots, w_n) = P(c) \prod_{j=i}^{k} P(w_j \mid c)^{\text{boost}(i) + \text{relative position}(i)}
\]

Better Features

- There are smarter features:
  - Argument selectional preference:
    - serve NP[meals] vs. serve NP[papers] vs. serve NP[country]
  - Subcategorization:
    - [function] serve PP[as]
    - [enable] serve VP[to]
    - [tennis] serve <intransitive>
  - [food] serve NP [PP[to]]
  - 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)

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: 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)

\[
P(\text{sense} \mid \text{context:jail}, \text{context:county}, \text{context:feeding}, \ldots, \text{local-context:jail}, \text{local-context:meals}, \ldots)
\]

- 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?)

Feature Representations

- 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

\[
\{f_i(d)\}
\]

\[
\text{context:jail} = 1
\]
\[
\text{context:county} = 1
\]
\[
\text{context:feeding} = 1
\]
\[
\text{context:game} = 0
\]
\[
\text{local-context:jail} = 1
\]
\[
\text{local-context:meals} = 1
\]
\[
\text{subcat:NP} = 1
\]
\[
\text{subcat:PP} = 0
\]
\[
\text{object-head:meals} = 1
\]
\[
\text{object-head:ball} = 0
\]
Linear Classifiers

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

<table>
<thead>
<tr>
<th>Feature</th>
<th>Food</th>
</tr>
</thead>
<tbody>
<tr>
<td>context: jail</td>
<td>+0.1</td>
</tr>
<tr>
<td>context: jail</td>
<td>+0.2</td>
</tr>
<tr>
<td>context: jail</td>
<td>+0.8</td>
</tr>
<tr>
<td>object: head: meals</td>
<td>+2.0</td>
</tr>
<tr>
<td>object: head: meals</td>
<td>+1.5</td>
</tr>
<tr>
<td>object: head: meals</td>
<td>+1.6</td>
</tr>
<tr>
<td>object: head: years</td>
<td>+3.8</td>
</tr>
<tr>
<td>object: head: years</td>
<td>+2.1</td>
</tr>
<tr>
<td>object: head: years</td>
<td>+1.1</td>
</tr>
<tr>
<td>TOTAL</td>
<td>+3.5</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:
    \[
    P(c \mid d, \lambda) = \sum_c \exp \sum_i \lambda_i f_i(d) \quad \text{normalizes votes.}
    \]
  - For any weight vector \(\lambda\), 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:
    \[
    \log P(C \mid D, \lambda) = \log \prod_{i \in C, d} P(c \mid d, \lambda) = \sum_{i \in C, d} \log \sum_c \exp \sum_i \lambda_i f_i(d)
    \]

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):
      - Conditional likelihood of the data
      - 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 \(\lambda\):
  \[
  \log P(C \mid D, \lambda) = \log \prod_{i \in C, d} P(c \mid d, \lambda) = \sum_{i \in C, d} \log \sum_c \exp \sum_i \lambda_i f_i(d)
  \]
  - If there aren't many values of \(c\), it's easy to calculate:
    \[
    \log P(C \mid D, \lambda) = \sum_{i \in C, d} \log \exp \sum_c \lambda_i f_i(d) - \sum_{i \in C, d} \log \sum_c \exp \sum_i \lambda_i f_i(d)
    \]
  - We can separate this into two components:
    \[
    \log P(C \mid D, \lambda) = \sum_{i \in C, d} \log \exp \sum_c \lambda_i f_i(d) - \sum_{i \in C, d} \log \sum_c \exp \sum_i \lambda_i f_i(d)
    \]

The Derivative I: Numerator

- Derivative of the numerator is the empirical count\(f_i(d)\)

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 II: Denominator

- Derivative of the denominator is the conditional (log) likelihood of the data:

\[
\frac{\partial M(\lambda)}{\partial \lambda_i(c)} = \sum_{i \in C, d} \sum_c \lambda_i f_i(d) \frac{\partial \sum_i \lambda_i f_i(d)}{\partial \lambda_i(c)} = \sum_{i \in C, d} f_i(d)
\]

\[
\text{predicted count}(f_i, \lambda) = \sum_c P(c \mid d_i, \lambda) f_i(d_i)
\]
The Derivative III

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

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)

The context-word:jail) feature: actual = 1 empirical = 1.2

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

Assume the following empirical distribution:

<table>
<thead>
<tr>
<th>Heads</th>
<th>Tails</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Features: (Heads), (Tails)

The data likelihood in this model is:

\[ \log P(h, t \mid \lambda) = h \log P_{\text{HEADS}} + t \log P_{\text{TAILS}} \]

\[ \log P(h, t \mid \lambda) = h \lambda - (t + h) \log(1 + e^\lambda) \]

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:

\[
\log P(C, \lambda | D) = \log P(\lambda) + \log P(C | D, \lambda) \\
\text{Posterior} \quad \quad \text{Prior} \quad \quad \text{Evidence}
\]

Smoothing: Priors

- 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!

\[
\log P(C, \lambda | D) = \log P(\lambda) - \frac{1}{\sigma^2} \sum_i (\lambda_i - \mu)^2 + k
\]

- Change the derivative:

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

Example: NER Smoothing

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

<table>
<thead>
<tr>
<th>Feature Type</th>
<th>Feature</th>
<th>PERs</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous word</td>
<td>at</td>
<td>-0.73</td>
<td>0.94</td>
</tr>
<tr>
<td>Current word</td>
<td>Grace</td>
<td>0.03</td>
<td>0.01</td>
</tr>
<tr>
<td>Beginning bigram</td>
<td>&lt;G</td>
<td>0.45</td>
<td>-0.04</td>
</tr>
<tr>
<td>Current POS tag</td>
<td>NNP</td>
<td>0.41</td>
<td>0.48</td>
</tr>
<tr>
<td>Prev and cur tags</td>
<td>IN</td>
<td>-0.10</td>
<td>0.15</td>
</tr>
<tr>
<td>Previous state</td>
<td>Other</td>
<td>-0.76</td>
<td>-0.92</td>
</tr>
<tr>
<td>Current signature</td>
<td>Xx</td>
<td>0.80</td>
<td>0.48</td>
</tr>
<tr>
<td>Prev state, cur sig</td>
<td>O-x</td>
<td>0.68</td>
<td>0.37</td>
</tr>
<tr>
<td>Previous next sig</td>
<td>x-O-x</td>
<td>-0.69</td>
<td>0.27</td>
</tr>
<tr>
<td>P. state - p-cur sig</td>
<td>O-x-x</td>
<td>-0.20</td>
<td>0.95</td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td>-0.58</td>
<td>2.68</td>
</tr>
</tbody>
</table>