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

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

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 (organisations 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?

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

![Diagram showing weighted windows with NB](image)

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

**Weighted Windows with NB**

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

![Distance conditioning diagram](image)

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

- Distance weighting
  - Nearby words should get a larger vote

![Distance weighting diagram](image)

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

**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 [PPP[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-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

**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?

- Example: living plant vs. manufacturing plant

**Word Sense Disambiguation**

- 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
  - [legal] He was served by the sheriff
- Rest of today: a maximum entropy approach

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 ... 

Better Features

- There are smarter features:
  - Argument selectional preference:
    - serve NP[meals] vs. serve NP[papers] vs. serve NP[country]
  - Subcategorization:
    - [function] serve VP[as]
    - [enable] serve VP[to]
    - [legal] serve <intransitive>
    - [food] serve NP [PP[ing]]
  - 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-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)
  \[ P(\text{sense} | \text{context:jail, context:county, context:feeding, ...}) \]
- 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

\[ f_i(d) \]

- 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

Linear Classifiers

- For a pair \((c,d)\), we take a weighted vote for each class:
  \[ \text{vote}(c | d) = \exp \sum_j \lambda_j(c) f_j(d) \]

Maximum-Entropy Classifiers

- Exponential (log-linear, maxent, logistic, Gibbs) models:
  - Turn the votes into a probability distribution:
    \[ P(c | d, \lambda) = \frac{\exp \sum c \lambda_j(c) f_j(d)}{\sum_j \exp \sum_c \lambda_j(c) f_j(d)} \]
  - For any weight vector \( \lambda_j \), we get a conditional probability model \( P(c | d, \lambda) \).
  - We want to choose parameters that maximize the conditional (log) likelihood of the data:
    \[ \log P(C | D, \lambda) = \sum_{c,d} \log P(c | d, \lambda) = \sum_{c,d} \log \frac{\exp \sum c \lambda_j(c) f_j(d)}{\sum_j \exp \sum_c \lambda_j(c) f_j(d)} \]

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 \(\lambda\):
  \[
  \log P(C \mid D, \lambda) = \log \prod_{(c,d,i,j) \in C,D} P(c \mid d, \lambda) = \sum_{(c,d,i,j) \in C,D} \log P(c \mid d, \lambda)
  \]
- If there aren’t many values of \(c\), it’s easy to calculate:
  \[
  \log P(C \mid D, \lambda) = \sum_{c \in \text{values}} \log \exp \left( \sum_{d} \lambda(c,f(d)) f(d) \right)
  \]
- We can separate this into two components:
  \[
  \log P(C \mid D, \lambda) = N(\lambda) - M(\lambda)
  \]

The Derivative I: Numerator

\[
\frac{\partial N(\lambda)}{\partial \lambda(c)} = \frac{\partial}{\partial \lambda(c)} \sum_{d} \log \exp \left( \sum_{f(d)} \lambda(c,f(d)) f(d) \right) = \sum_{d} \lambda(c,f(d)) f(d)
\]

Derivative of the numerator is the empirical count \(f_i, c\)

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

\[
\frac{\partial M(\lambda)}{\partial \lambda(c)} = \frac{\partial}{\partial \lambda(c)} \sum_{i} \log \exp \left( \sum_{f(d)} \lambda(c,f(d)) f(d) \right) = \sum_{f(d)} \lambda(c,f(d)) f(d)
\]

\[
= \sum_{k} P(c \mid d_k, \lambda) f_i(d_k) = \text{predicted count}(f_i, c)
\]

The Derivative III

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

Summary

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

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>h</td>
<td>t</td>
</tr>
</tbody>
</table>
  
• Features: {Heads}, {Tails}
• We’ll have the following model distribution:

\[ P_{\text{Heads}} = \frac{e^\lambda}{e^{2\lambda} + e^\lambda}, \quad P_{\text{Tails}} = \frac{e^{-\lambda}}{e^{2\lambda} + e^{-\lambda}} \]

Really, only one degree of freedom (\( \lambda = \lambda_t - \lambda_h \))

\[ P_{\text{Heads}} = \frac{e^\lambda}{e^{2\lambda} + e^\lambda}, \quad P_{\text{Tails}} = \frac{e^{-\lambda}}{e^{2\lambda} + e^{-\lambda}} = e^\lambda \]

Smoothing: Issues

• The data likelihood in this model is:

\[ \log P(h, t | \lambda) = h \log P_{\text{Heads}} + t \log P_{\text{Tails}} \]

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

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:

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

Posterior Prior 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!

• Change the objective:

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

\[ \log P(C, \lambda | D) = \sum_{d \in D} \log P(d | \lambda) - \frac{(\lambda - \mu)^2}{2\sigma^2} + k \]

• Change the derivative:

\[ \frac{\partial \log P(C, \lambda | D)}{\partial \lambda} = \text{actual}(f_C, C) - \text{predicted}(f_C, \lambda) - (\lambda - \mu) / \sigma^2 \]
Because of smoothing, the more common prefixes have larger weights even though entire-word features are more specific.

<table>
<thead>
<tr>
<th>State</th>
<th>Other</th>
<th>Prev</th>
<th>Cur</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word</td>
<td>at</td>
<td>Grace</td>
<td>Road</td>
<td></td>
</tr>
<tr>
<td>Tag</td>
<td>IN</td>
<td>NNP</td>
<td>NNP</td>
<td></td>
</tr>
<tr>
<td>Sig</td>
<td>x</td>
<td>Xx</td>
<td>Xx</td>
<td></td>
</tr>
</tbody>
</table>

Feature Weights

<table>
<thead>
<tr>
<th>Feature Type</th>
<th>Feature</th>
<th>PERS</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix 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>0.00</td>
</tr>
<tr>
<td>Beginning tag</td>
<td>&lt;G</td>
<td>0.45</td>
<td>-0.04</td>
</tr>
<tr>
<td>Current POS tag</td>
<td>NNP</td>
<td>0.47</td>
<td>0.46</td>
</tr>
<tr>
<td>Prev and cur tags</td>
<td>NNP</td>
<td>-0.10</td>
<td>0.54</td>
</tr>
<tr>
<td>Previous state</td>
<td>Other</td>
<td>-0.70</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-Xx</td>
<td>0.68</td>
<td>0.37</td>
</tr>
<tr>
<td>Prev-cur state sig</td>
<td>a-Xx-Xx</td>
<td>-0.89</td>
<td>0.37</td>
</tr>
<tr>
<td>Prev state - p-cur sig</td>
<td>O-a-Xx</td>
<td>-0.20</td>
<td>0.82</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>-0.58</td>
<td>2.68</td>
</tr>
</tbody>
</table>