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
  - [endeavor] 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
- 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
- SenseEval
  - 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 …

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

Weighted Windows with NB

- Distance conditioning
  - Some words are important only when they are nearby
    - … as … point … court …serve … game …
  - \[
P(c, w_2, \ldots, w_k | c) = P(c) \prod_{i=2}^{k} P(w_i | c, \text{bin}(i))
\]
- Distance weighting
  - Nearby words should get a larger vote
    - … point … serve as … game …
  - \[
P(c, w_2, \ldots, w_k | c) = P(c) \prod_{i=2}^{k} P(w_i | c)^{\text{rel}(i)}
\]

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

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

- 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.

Example:
- Local-context: jail, local-context: meals.
- Subcategory: NP, PP.
- Object-head: meals, ball.

Linear Classifiers

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

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

<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>local-context:jail</td>
<td>+10 * 1</td>
<td>+10 * 1</td>
<td>+10 * 1</td>
</tr>
<tr>
<td>object-head:meals</td>
<td>+20 * 0</td>
<td>+10 * 0</td>
<td>+15 * 0</td>
</tr>
<tr>
<td>object-head:years</td>
<td>-1.5 * 0</td>
<td>+2.1 * 0</td>
<td>-1.1 * 0</td>
</tr>
<tr>
<td>TOTAL</td>
<td>+3.5</td>
<td>+8.7</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:
    \[ \frac{\exp \left( \sum \lambda(c) f_i(d) \right)}{\sum \exp \left( \sum \lambda(c) f_i(d) \right)} \]

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 w.r.t each feature weight.

The Likelihood Value

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

The Derivative I: Numerator

- Derivative of the numerator is the empirical count \( f_i(c) \):
  \[ \frac{\partial N(\lambda)}{\partial \lambda(c)} = \sum_{k \in c} \frac{\partial \sum \lambda(c) f_i(d_k)}{\partial \lambda(c)} = \sum_{k \in c} \sum \lambda(c) f_i(d_k) \]
  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

\[
\frac{\partial M(\lambda)}{\partial \lambda(c)} = \frac{\sum \log \exp \left( \sum \lambda(c') f_i(d_i) \right)}{\sum \exp \left( \sum \lambda(c') f_i(d_i) \right)} - \frac{\sum \exp \left( \sum \lambda(c') f_i(d_i) \right)}{\sum \exp \left( \sum \lambda(c') f_i(d_i) \right)} \frac{\delta \sum \lambda(c') f_i(d_i)}{\delta \lambda(c)}
\]

\[
= \sum_i P(c \mid d_i, \lambda) f_i(d_i) = \text{predicted count}(f_i, \lambda)
\]

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:
  \[
  \log P(C \mid D, \lambda) = \sum_{c,d,i} \log \exp \left( \sum \lambda(c') f_i(d_i) \right)
  \]

- 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>i</td>
</tr>
</tbody>
</table>

- Features: {Heads}, {Tails}

- We’ll have the following model distribution:

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

- Really, only one degree of freedom (\( \lambda = \lambda_1 - \lambda_2 \))

  \[
P_{\text{Heads}} = \frac{e^{\lambda_1}}{e^{\lambda_1} + e^{\lambda_2}}, \quad P_{\text{Tails}} = \frac{e^{\lambda_2}}{e^{\lambda_1} + e^{\lambda_2}}
  \]

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:

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

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:

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

- Change the derivative:

$$\frac{\partial \log P(C, \lambda | D)}{\partial \lambda} = \text{predicted}(f, \lambda) - \text{actual}(f, \lambda) / \sigma^2$$

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 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>0.93</td>
<td>0.06</td>
</tr>
<tr>
<td>Beginning state</td>
<td>&lt;0</td>
<td>0.45</td>
<td>-0.54</td>
</tr>
<tr>
<td>Current POS tag</td>
<td>NNP</td>
<td>0.47</td>
<td>0.48</td>
</tr>
<tr>
<td>Prev and cur tags</td>
<td>In help</td>
<td>-0.10</td>
<td>0.18</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 tags, sig</td>
<td>x-Xx+xX</td>
<td>0.89</td>
<td>0.37</td>
</tr>
<tr>
<td>P state, p-out sig</td>
<td>O-x-Xx</td>
<td>-0.20</td>
<td>0.82</td>
</tr>
</tbody>
</table>

Total: -0.58 2.68

Example: Sensors

**NB FACTORS:**
- $P(s) = 1/2$
- $P(+|s) = 1/4$
- $P(+) = 1/4$
- $P(+|r) = 3/4$
- $P(s|+) = 9/10$
- $P(s|+) = 1/10$

**PREDICTIONS:**
- $P(r,+,+) = (1/2)(1/4)(3/4)$
- $P(s,+,+) = (1/2)(1/4)(1/4)$
- $P(+,+,r) = 3/8$
- $P(+,+,s) = 1/8$
Example: Sensors

- Problem: NB multi - c ounts the evidence.
  \[
  P(r \mid +, \ldots) = P(r) \cdot P(+ \mid r) / P(+) = P(s) \cdot P(+ \mid s) / P(+) = P(s) \cdot P(+ \mid s) / P(+) = P(s) \cdot P(+ \mid s) / P(+)
  \]

- Maxent behavior:
  - Take a model over \((M_1, \ldots, M_n, R)\) with features:
    \(f_r: M_i=+, R=r\) weight: \(\lambda_i\)
    \(f_s: M_i=+, R=s\) weight: \(\lambda_i\)
  \(\exp(\lambda_i(r) - \lambda_i(s))\) is the factor analogous to \(P(+\mid r)/P(+\mid s)\)
  - But instead of being 3, it will be \(3^{1/n}\)
  - ... because if it were 3, \(E[f_{r,r}]\) would be far higher than the target of 3/8!

---

Example: Stoplights

- Reality
  - Lights Working
  - Lights Broken

- NB Model
  - NB FACTORS:
    - \(P(w) = 6/7\)
    - \(P(r \mid w) = 1/2\)
    - \(P(g \mid w) = 1/2\)
    - \(P(b) = 1/7\)
    - \(P(r \mid b) = 1\)
    - \(P(g \mid b) = 0\)
  - \(P(b \mid r, r) = 4/10\) (what happened?)

---

Example: Stoplights

- What does the model say when both lights are red?

  - P(b, r, r) = (1/7)(1/1) = 1/7 = 4/28
  - P(w, r, r) = (6/7)(1/2)(1/2) = 6/28 = 6/28
  - P(w|f, r) = 6/10!

  We'll guess that \((r, r)\) indicates lights are working!
  - Imagine if \(P(b)\) were boosted higher, to 1/2:
    - P(b, r, r) = (1/2)(1/1) = 1/2 = 4/8
    - P(w, r, r) = (1/2)(1/2)(1/2) = 1/8 = 1/8
    - Changing the parameters, bought accuracy at the expense of data likelihood

---

Causes and Effects

- Effects
  - Children (the \(d_i\) here) are effects in the model.
  - When two arrows exit a node, the children are (independent) effects.

- Causes
  - Parents (the \(d_i\) here) are causes in the model.
  - When two arrows enter a node (a v-structure), the parents are in causal competition.

---

Explaining-Away

- When nodes are in causal competition, a common interaction is explaining-away.
- In explaining-away, discovering one cause leads to a lowered belief in other causes.

Example: I buy lottery tickets A and B. You assume neither is a winner. I then do a crazy jig. You then believe one of my two lottery tickets must be a winner, 50\% 50\%. If you then find that ticket A did indeed win, you go back to believing that B is probably not a winner.

---

Data and Causal Competition

- Problem in NLP in general:
  - Some singleton words are noise.
  - Others are your only only glimpse of a good feature.

- Maxent models have an interesting, potentially NLP-friendly behavior:
  - Optimization goal: assign the correct class.
  - Process: assigns more weight ("blame") to features which are needed to get classifications right.
  - Maxent models effectively have the structure shown, putting features into causal competition.
**Example WSD Behavior I**

- line₂ (a phone line)
  - A) "thanks anyway, the transatlantic line₂ died."
  - B) "... phones with more than one line₂, plush robes, exotic flowers, and complimentary wine."

- In A, "died" occurs with line₂ 2/3 times.
- In B, "phone(s)" occurs with line₂ 191/193 times.
- "transatlantic" and "flowers" are both singletons in data.

- We’d like "transatlantic" to indicate line₂ more than "flowers" does...

**Example WSD Behavior II**

- Both models use “add one” pseudocount smoothing
- With Naïve-Bayes:
  \[
  \frac{P_{NB}(\text{flowers} \mid 2)}{P_{NB}(\text{flowers} \mid 1)} = 2 \quad \frac{P_{NB}(\text{transatlantic} \mid 2)}{P_{NB}(\text{transatlantic} \mid 1)} = 2
  \]
- With a word-featured maxent model:
  \[
  \frac{P_{ME}(\text{flowers} \mid 2)}{P_{ME}(\text{flowers} \mid 1)} = 2.05 \quad \frac{P_{ME}(\text{transatlantic} \mid 2)}{P_{ME}(\text{transatlantic} \mid 1)} = 3.74
  \]
- Of course, “thanks” is just like “transatlantic”!

**What’s Next**

- Next class:
  - Intro to Sequence Models
  - Part-of-Speech Tagging
  - HMMs
- Reading: M+S 9-10 (over the next week)