Word Sense Disambiguation

Linguistics 420
Statistical NLP
Spring 2006

Word Sense Disambiguation (WSD)

Word Sense Disambiguation (WSD) is the task of determining the proper sense of an ambiguous word in a given context

e.g., Given the word *bank*, is it:

- the rising ground bordering a body of water
- an establishment for exchanging funds

Not always clear what the distinct senses of a word are, but still we will move on, focusing mostly on supervised WSD

Bayesian WSD

- Look at a context of surrounding words, call it c, within a window of a particular size
- Select the best sense s from among the different senses

$$\begin{aligned} s &= \arg\max_{s_k} P(s_k|c) \\ &= \arg\max_{s_k} \frac{P(c|s_k)P(s_k)}{P(c)} \\ &= \arg\max_{s_k} P(c|s_k)P(s_k) \end{aligned} \quad (1)$$

It is actually computationally simpler to calculate logarithms, giving us:

$$s = \arg\max_{s_k}[\log P(c|s_k) + \log P(s_k)] \quad (2)$$

Naive Bayes assumption

How do we handle those features v_j that make up our context c?

Well, we make the **Naive Bayes assumption** that every surrounding word v_j is independent of the other ones:

$$P(c|s_k) = \prod_{v_j \in c} P(v_j|s_k) \quad (3)$$

This means that we have:

$$s = \arg\max_{s_k}[\sum_{v_j \in c} \log P(v_j|s_k) + \log P(s_k)] \quad (4)$$

We get maximum likelihood estimates from the corpus for $P(s_k)$ and $P(v_j|s_k)$

(High-level) Bayes example

Try to disambiguate *drug*: 'medication' vs. 'illegal substance'

- See *prices* in the context
- Compare $P(prices|$'medication'$)$ and $P(prices|$'illegal substance'$)$

  $\rightarrow P(prices|$'medication'$)$ wins out, so if *prices* is in the context, we are more likely to select 'medication'

Information-theoretic approach

Instead of using all contextual features—which we assume are independent—an information-theoretic approach tries to find one disambiguating feature

Take a set of possible indicators and determine which is the best, i.e., which gives the highest mutual information in the training data

Possible indicators:

- object of the verb
- the verb tense
- word to the left
- word to the right
- etc.

And then when sense tagging, find the value of that indicator and tag appropriately

Partitioning

More specifically, you have to determine what the values (x_i) of the indicator indicate, i.e. what sense (s_i) they point to.

- Assume two senses (P_1 and P_2), which can be captured in subsets $Q_1 = \{x_i|x_i$ indicates sense 1$\}$ and $Q_2 = \{x_i|x_i$ indicates sense 2$\}$
- We will have a set of indicator values Q; our goal is to partition Q into these two sets

The partition we choose is the one which maximizes the mutual information scores $I(P_1, Q_1)$ and $I(P_2, Q_2)$

- The Flip-Flop algorithm is used when you have to automatically determine your senses (e.g., if using parallel text)

Disambiguation

After determining the best indicator and partitioning the values, disambiguating is easy:

1. Determine the value x_i of the indicator for the ambiguous word.
2. If x_i is in Q_1, assign it sense 1; otherwise, sense 2.

This method is also applicable for determining which indicators are best for a set of translation words

Dictionary-based WSD

Use general characterizations of the senses to aid in disambiguation

Intuition: words found in a particular sense definition can provide contextual cues, e.g., for *ash*:

Sense	Definition
s_1: tree	a tree of the olive family
s_2: burned stuff	the solid residue left when combustible material is burned

If we find *tree* in the context of *ash*, the sense is more likely s_1

The algorithm

Actually, we look at words within the sense definition and the words within the definitions of context words, too (unioning over different senses)

1. Take all senses s_k of a word w and gather the set of words in the definition → treat it as a **bag of words**
2. Gather all the words in the definitions of the surrounding words, within some context window
3. Calculate the overlap
4. Choose the sense with the higher overlap

Example

(5) This cigar **burns** slowly and creates a stiff *ash*.
(6) The *ash* is one of the last **trees** to come into leaf.

- So, sense s_2 goes with the first sentence and s_1 with the second
- Note that for this to work, we need to look at lemmata
- Note also that, depending on the dictionary, *leaf* might also be a contextual cue for sense s_1 of *ash*

Problems with dictionary-based WSD

- Not very accurate: 50%-70%
- Highly dependent upon the choice of dictionary
- Not always clear whether the dictionary definitions align with what we think of as different senses

Thesaurus-based WSD

Use essentially the same set-up as dictionary-based WSD, but now:

- instead of requiring context words to have overlapping dictionary definitions
- we require surrounding context words to list the focus word w (or the subject code of w) as one of their topics

e.g., If an ANIMAL or INSECT appears in the context of *bass*, we choose the *fish* sense instead of the musical one

NB: Something like WordNet could be used for this kind of WSD, to pick out the appropriate topics

Extension to thesaurus-based WSD

One nice extension is to automatically figure out new words which fit into a particular topic

- The problem is that context words may indicate some topic, but will not in the thesaurus specify that topic
- e.g., *Navratilova* indicates a SPORTS topics

Using a Naive Bayes classifier to find such new topics for words, i.e., what topics appear more than we would expect for a given word.

Translation-based WSD

Idea: when disambiguating a word w, look for a combination of w and some contextual word which translates to a particular pair, indicating a particular sense

- *interest* can be 'legal share' (*Beteiligung* in German) or 'concern' (*Interesse*)
- In the phrase *show concern*, we are more likely to translate to *Interesse zeigen* than *Beteiligung zeigen*
- So, in this English context, the German context tells us to go with the sense that corresponds to *Interesse*

Some constraints

Examining your data can tell you that certain things are true about word senses

- **One sense per discourse:** the sense of a word is highly consistent within a given document
- **One sense per collocation:** collocations rarely have multiple senses associated with them

Using collocational knowledge

Rank senses based on what collocations the word appears in, e.g., *show interest* might be strongly correlated with the 'attention, concern' usage of *interest*

- The collocational feature could actually be a surrounding POS tag, or a word in the object position, or whatever
- For a given context, select which collocational feature will be used to disambiguate, based on which feature is strongest indicator
  - Avoid having to combine different pieces of information this way

Rankings are based on the following, where f is a collocational feature:

(7) $\frac{P(s_{k_1}|f)}{P(s_{k_2}|f)}$

Calculating collocations

1. Initially, calculate the collocations for s_k
2. Calculate the contexts in which an ambiguous word is assigned to s_k, based on those collocations
3. Calculate the set of collocations that are most characteristic of the contexts for s_k, using the formula:

   (8) $\frac{P(s_{k_1}|f)}{P(s_{k_2}|f)}$

4. Repeat steps 2 & 3 until a threshold is reached.

Using discourse knowledge

We also want to integrate discourse knowledge—in a given discourse, we are unlikely to see senses flip around

... the existence of *plant* and animal life ... classified as either *plant* or animal
... Although bacterial and *plant* cells are enclosed ...

So, we can take the output of a WSD algorithm (such as "one sense per collocation") and change it so that every ambiguous word in a particular discourse is given the majority sense

- Or every word beneath some threshold of likelihood ...
- Or just surrounding words ...

Unsupervised WSD

Without any information, you really cannot sense tag something ... but you can perform sense *discrimination*, or *clustering*

- In other words, you can group comparable senses together (even if you cannot give it the correct label)

We will look briefly at the EM (Expectation-Maximization) algorithm for this task, based on a Bayesian model

EM algorithm: Bayesian review

Remember how we did Bayesian WSD for supervised learning:

- Look at a context of surrounding words, call it c (v_j = word in context), within a window of a particular size
- Select the best sense s from among the different senses

$$
\begin{aligned}
s &= \arg\max_{s_k} P(s_k|c) \\
&= \arg\max_{s_k} \frac{P(c|s_k)P(s_k)}{P(c)} \\
&= \arg\max_{s_k} P(c|s_k)P(s_k) \\
&= \arg\max_{s_k} [\log P(c|s_k) + \log P(s_k)] \\
&= \arg\max_{s_k} [\sum_{v_j \in c} \log P(v_j|s_k) + \log P(s_k)]
\end{aligned} \tag{9}
$$

We need some other way to get estimates of $P(s_k)$ and $P(c|s_k)$

EM algorithm

1. **Intialize** the parameters randomly, i.e., the probabilities for all senses and contexts

   And decide K, the number of senses you want → determines how fine-grained your distinctions are

2. While still improving:

(a) **Expectation:** re-estimate the probability of s_k generating the context c

$$\hat{P}(c_i|s_k) = \frac{P(c_i|s_k)}{\sum_{k=1}^{K} P(c_i|s_k)} \tag{10}$$

And remember that all contextual words v_j (i.e., $P(v_j|s_k)$) will be used to calculate the context

EM algorithm (cont.)

(b) **Maximization:** Use the expected probabilities to re-estimate the parameters:

$$P(v_j|s_k) = \frac{\sum_{\{c_i : v_j \in c_i\}} \hat{P}(c_i|s_k)}{\sum_k \sum_{\{c_i : v_j \in c_i\}} \hat{P}(c_i|s_k)} \tag{11}$$

→ Of all the times that v_j occurs in a context of any of this word's senses, how often does v_j indicate s_k?

$$P(s_k) = \frac{\sum_i \hat{P}(c_i|s_k)}{\sum_k \sum_i \hat{P}(c_i|s_k)} \tag{12}$$

→ Of all the times that any sense generates c_i, how often does s_k generate it?

Unsupervised learning

Pros:

- Can choose your own granularity of senses, which may or may not match up with dictionaries
- Performance can be quite decent, depending on how much data is used—and does not, of course, require any labeled data

Cons:

- Performance can vary quite a bit based on your initial parameters

Clustering more generally

Lots of different tasks require us to put groups into clusters, as we saw, e.g., with text categorization (in an unsupervised fashion)

- Represent data abstractly, e.g., by features
- Define similarity, such that similar items get put into the same group

We'll look at two main kinds of clustering:

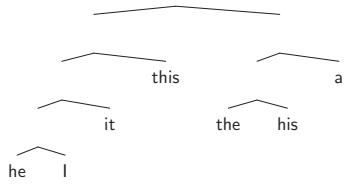
- Hierarchical clustering: make clusters of clusters
- K-means clustering, a form of non-hierarchical (flat) clustering

See Table 14.1 (p. 500) for a summary of differences

Hierarchical clustering

The goal of **hierarchical clustering** is to build a tree that captures relationships between your objects.

Here is a tree that crudely captures the idea of a part of speech:



Agglomerative clustering

We can build a tree in bottom-up fashion; this is called **agglomerative clustering**

1. Assign each item to its own cluster
2. Find the clusters which have the maximum amount of *similarity* between them
3. Form a new cluster by taking the union of those two clusters
4. Repeat steps 2 and 3 until every cluster is grouped into the same cluster

Divisive clustering

Likewise, we can build a tree in top-down fashion; this is called **divisive clustering**

1. Assign every item into the same cluster
2. Determine which cluster has the least amount of *coherence* (defined in terms of similarity between items)
3. Split the least coherent cluster, based on the item which is the most dissimilar to the others
4. Repeat steps 2 and 3 until every item is in its own cluster

Similarity functions

Hierarchical clustering depends upon how you calculate similarity, and there are a few main techniques:

- **Single link** clustering = cluster similarity defined by similarity of two most similar members
- **Complete link** clustering = cluster similarity defined by similarity of two least similar members
- **Group average** clustering = cluster similarity defined by average similarity between members

Coherence is based upon which similarity measure is being used

K-means clustering

Non-hierarchical clustering can be good because it might allow you to do **soft clustering** = assign an item to more than one cluster

However, we'll look at K-means clustering, which is also a **hard clustering** algorithm = every item is assigned to only one cluster

K-means clustering allows for k clusters, and uses the mean, or the center, of each cluster to define what makes up the cluster

K-means clustering algorithm

Once you know what value of k you want, i.e., the number of clusters you want,

1. Select k different means $(f_1, ..., f_k)$
2. Assign items to the cluster whose mean is closest to the item's value
3. Re-calculate the means $(f_1, ..., f_k)$, based on the values of the items which are now in the cluster
4. Repeat steps 2 and 3 until a stopping criterion is reached