NLP project 2: Word sense disambiguation

Fred Callaway Luke Goodman

We plan to use a supervised Naive Bayes approach for word sense disambiguation. The basic strategy is to first create a function that extracts features from a context, and then use Naive Bayes to discover which features are predictive of the sense of a word in context. We hope to achieve strong performance by using linguistically informed features. In particular, we plan to use the Spacy library to extract word lemmas, brown clusters, parts of speech, and syntactic dependencies. We will consider some or all of these features for every word in the context. Additionally, we will consider the following word positions as especially relevant:

- the target word w_i, as well as w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2}
- any words that are either a predicate or argument of the target word

Features assigned to these words will receive their own place in the large feature vector. Thus, some possible features include:

- The target word is tagged as `NOUN`. This could distinguish between the two senses of “bank”.
- The lemma of the subject of the target word is “stick”. This could distinguish between two senses of the word “snap”: the hand gesture vs. breaking.
- A word with brown cluster 1061 occurs in the context. For example, the model may have learned to associate “friend” with one sense of the word “bar”. It can then use this information when it sees the word “buddy” in the context of a test case.

For each position, there could be up to N features, where N is the number of words. However, most words will not occur in most of the specified slots, so the resulting vector will be considerably smaller than this. Furthermore, each vector will be very sparse, a trait we can capitalize on when designing our algorithms.

The next step is perform smoothing to account for data sparsity: The fact that a feature occurred only once for some sense should not make that feature a trump-all predictor of that sense. Finally we will calculate $p(f|s)$ for each feature, from which we can find $p(s|\vec{f})$ using Bayes’ theorem. Pseudopython for the entire process is on the following page.

Pseudocode

```
# get sense and sense-feature counts
sense_features = {}  # senses -> feature_names -> counts
sense_counts = {}
for context, sense in training_data:
    features = extract_features(context)  # a list of feature IDs
    for feature in features:
        sense_features[sense][feature] += 1
        sense_counts[sense] += 1

# get sense-feature probabilities
sense_features = smooth(sense_features)  # account for zero-probability features
probabilities = {}
for sense, count in sense_counts.items():
    for feature in sense_features[sense]:
        probabilities[sense][features] = sense_features[sense][feature] / count

def classify(context):
    features = extract_features(context)  # a list of feature IDs
    for sense in senses:
        for feature in features:
            # the implementation will be more complex for efficiency reasons
            p_features_given[sense] *= probabilities[sense][feature]
    return argmax_sense(p_features_given)
```