CS 4740 Introduction to NLP
Spring 2013
Word Sense Disambiguation

Proposal: Due via CMS by Thu, March 7th, 1:25pm
Results and Report: Due via CMS by Fri, March 15th 1:25pm

1 Introduction

Word Sense Disambiguation (WSD) is a task to find the correct meaning of a word given context. As many words in natural language are polysemous, humans perform WSD based on various cues from the context including both verbal and non-verbal. In this assignment, you are going to implement a WSD system by using two different methods: one *supervised* method and one *dictionary-based* method. Since many problems in natural language understanding are related to knowing the sense of each word in a document, WSD can be utilized as a building block for a variety of high-level tasks in the future.

Through this assignment, you will use the English Lexical Sample task from Senseval-3 for training and testing your system. Training and evaluating on these data, you will analyze the pros and cons of two methods. We will also setup a Kaggle competition so that you can submit the prediction results of your WSD system.

2 Dataset

To get an initial sense, visit the Senseval-3 web site[1] and read the overview of the English Lexical Sample task on pages 25-28. This is a paper written by Mihalcea, Chklovski and Kilgarriff. You can find it in the following link.[2]

The data files for this project are lightly preprocessed and shuffled from the original English Lexical Sample task in Senseval-3. They are available via CMS and consist of training data, test data, and an XML formatted dictionary that describes commonly used senses for each word. Every lexical item in the dictionary contains multiple sense items, and the training data tags the correct sense of the target word in terms of a binary encoding where each bit corresponds to

[1] `http://www.senseval.org/senseval3`
[2] `http://www.cse.unt.edu/~rada/senseval/senseval3/proceedings/pdf/mihalcea2.pdf`

one sense in order of appearance in the dictionary. Concretely, each example of training and test data has the following format:

```
word.pos t0 t1 ... tk @ prev-context @ head @ next-context
```

- `word` is the target word for which we are to predict the sense.
- `pos` is the part-of-speech of the target word where 'n', 'v', and 'a' stand for noun, verb, and adjective respectively.
- `t0 t1 ...tk` is the binary encoding of tagged senses indicating whether or not each sense given in the dictionary applies in this example. The first sense $t0$ is always reserved for indicating 'unclear' or 'none of the above' when it is activated to 1. The rest $t1$ through tk correspond to k possible senses defined in the dictionary. Having $ti = 1$ means that the target word is used according to the i-th sense given in the dictionary. Note that some examples may contain multiple 1's in their encodings if the target words express multiple meanings at the same time. However, at least one of senses $t1$ ... tk will be activated to 1 in all examples.
- `@` is a delimiter.
- `prev-context` is the actual text given before the target word for which we are predicting the sense.
- `head` is the actual appearance of the target word. Note that it may be morphologically varied into different forms with respect to the semantic and syntactic context. For example, the `word` "begin.v" could show up as "beginning" instead of "begin" to denote a participle in the `head` position.
- `next-context` is the actual text given after the target word for which we are predicting the sense.

Note that binary encodings of test examples are all erased to 0 in order to measure the prediction accuracy of your own system. We will explain the scoring schemes in the later section.

In the dictionary file, every sense item contains a gloss field to indicate the corresponding definition. Each gloss consists of commonly used definitions[3] delimited by a semicolon, and may have multiple real examples wrapped by quotation marks being also delimited by a semicolon. Because the format of the gloss field is not highly uniform, however, you may find some glosses that do not have real examples.

[3] It is possible to explain a certain sense in multiple different ways inside a single sense item entry

3 Supervised WSD

This section will show a simple probabilistic approach called the Naive-Bayes model[4] to perform supervised WSD. This model takes a word in context as an input and outputs a probability distribution over predefined senses, indicating how likely each sense corresponds to the correct meaning of the target word within the given context. Specifically, it picks the best sense by the following equation:

$$\hat{s} = argmax_{s \in S(w)} P(s|\vec{f})$$

In the above equation, $S(w)$ is the predefined set of senses for the target word w and $\vec{f}$ is a feature vector extracted from the context surrounding w. Thus the equation says we are going to choose the most probable sense as the correct meaning of the target word w. By applying Bayes rule,

$$P(s|\vec{f}) = \frac{P(\vec{f}|s)P(s)}{P(\vec{f})}$$

As the denominator does not change with respect to $s \in S(w)$, the best sense $\hat{s}$ is determined by

$$\hat{s} = argmax_{s \in S(w)} P(s|\vec{f}) = argmax_{s \in S(w)} P(\vec{f}|s)P(s)$$

Here the model *naively* assumes[5] that features in the feature vector $\vec{f}$ are conditionally independent given the sense of the word s. This assumption yields the following decomposition:

$$P(\vec{f}|s) = \prod_{j=1}^{n} P(f_j|s) \quad where \;\; f = (f_1, f_2, ..., f_n)$$

In other words, the probability of a feature vector given sense can be estimated by the product of the probability of its individual features given that sense under our assumption. Hence the best sense is determined by

$$\hat{s} = argmax_{s \in S(w)} P(\vec{f}|s)P(s) = argmax_{s \in S(w)} P(s) \prod_{j=1}^{n} P(f_j|s)$$

What you have to implement for this model is given below. **Read the following instructions carefully**.

1. In order to train the above model, you should learn the model parameters: 1) the prior probability of each sense $P(s)$ and 2) the individual feature probabilities $P(f_j|s)$. Those are computed by the Maximum Likelihood

[4] See section 20.2.2 in the textbook.
[5] This is why the model is called Naive-Bayes.

Estimation (MLE) which simply counts the number of occurrences in the training set. In particular for the i-th sense s_i of a word w,

$$P(s_i) = \frac{count(s_i, w)}{count(w)} \qquad P(f_j|s_i) = \frac{count(f_j, s_i)}{count(s_i)}$$

For instance, let's assume there are 1,000 training examples corresponding to the word "bank". Among them, 750 occurrences stand for $bank_1$ which covers the financial sense, and 250 occurrences for $bank_2$ which covers the river sense. Then the prior probabilities are

$$P(s_1) = \frac{750}{1000} = 0.75 \qquad P(s_2) = \frac{250}{1000} = 0.25$$

If the first feature "credit" occurs 195 times within the context of $bank_1$, but only 5 times within the context of $bank_2$,

$$P(f_1 = "credit"|s_1) = \frac{195}{750} = 0.26 \qquad P(f_1 = "credit"|s_2) = \frac{5}{250} = 0.02$$

2. Though the basic setup is given above, the performance of your WSD system will mainly rely on how well you generate feature vectors from the contex. Note that target words to be disambiguated are always provided within a sufficiently long sentence. As you have seen in the above toy example, extracting informative features from surrounding context will have the model parameters discriminate the unlikely senses from the correct sense. In our model, this process of deciding model parameters corresponds to the *training*.[6] Of course, **you have to train a separate model per each target word in the training data**.

3. When learning your model, be sure that you never use the test data for training. Note that the binary labels given in the test data are all erased to 0, which means those are no longer true labels. Instead of marking the predicted senses directly into the testing file, you are going to generate a separate output file consisting of the predicted senses for test data. If you upload it to Kaggle, you will get back your score until the submission deadline. You can find the output specification in Section 5.

4. If you want to evaluate the performance of your system before submitting to Kaggle, or you designd multiple different models based on the Naive-Bayes model, you may reserve a certain portion of training data as a validation set in order to measure the performance of your system (or models). Since you know the true senses for the training data, testing on the validation set will let you guess how well your system (or models) works for the test data. This process is called *validation*. Note that you must not train on the validation set if you use a validation step.

[6] Indeed the performance depends largely on the feature engineering in the discriminative learning setting as SVMs.

4 Dictionary-based WSD

This section will show a dictionary-based WSD approach which is different from the previous supervised setting. Rather than training on the human-tagged dataset of true senses, dictionary-based approaches utilize definitions given in the dictionary. See the following example of disambiguating *"pine cone"*.

- pine (the context)
    1. a kind of **evergreen tree** with needle-shaped leaves
    2. to waste away through sorrow or illness
- cone (the target word)
    1. A solid body which narrows to a point
    2. Something of this shape, whether solid or hollow
    3. Fruit of certain **evergreen trees**

As bold faced in the above, 3rd sense of the target word matches the most with the 1st sense of the context word among all possible combinations. This process shows the original Lesk algorithm to disambiguate senses based only on the cross-comparing the definitions. However, it is able to be extended to utilize examples in the dictionary to get wider matching. **Read the following instructions carefully**.

1. Design a metric that rewarding more to consecutive overlaps rather than treating one overlap of two consecutive words equivalent to two distant overlaps of a single word. Note that there would be morphological variations in the definitions and examples. In order to increase the matching, stemming or lemmatizing could be useful.[7]
2. Implement a dictionary-based WSD system that disambiguates the sense by comparing the definitions of the target word to the definitions of relevant words in the context. As the performance of the supervised system depends largely on the feature generation, here your design decision of finding relevant words will determine the performance of the dictionary-based system in combining with the metric you designed above.
3. In contrast to the supervised settings, there is no training process involved here because we mainly use definitions and examples in the dictionary to figure out the correct sense. As explained in the Section 2, the dictionary file contains a certain amount of glosses and examples. If you conclude those are not enough to perform the dictionary-based approach, using the WordNet dictionary could be an alternative.[8]

[7] You can find such tools in the WordNet.

[8] If it is hard to find the mapping between predefined senses of our dictionary and WordNet, you may first find the correct sense in WordNet, and then may choose the best sense in our dictionary based on the WordNet sense with its examples and definitions. Though you may have to perform matching algorithm twice, incorporating WordNet may be helpful due to its richness and APIs.

4. Since there is no training process, you could verify the performance of your dictionary-based system on the entire training set. You could also compare the performance of two WSD systems via testing on the same validation set you reserved from the training set. Similar to supervised WSD, you have to submit prediction results for the test data to Kaggle.

5 Scoring and Extensions

We use *accuracy*[9] as a score. Since a target word may contain multiple meanings at the same time, a single prediction could be counted as incorrect one unless your system predicts exactly same senses with the ground-truth labels. Suppose that the target word in a test example has k predefined senses. Instead of treating it as a single sequence prediction task, we are going to consider it as $(k + 1)$ simple prediction tasks in which your system have to guess whether each of k-senses is used in the target word. Note that first prediction is always reserved for indicating 'unclear' or 'none of the above'.

1. The prediction file you will have to submit to Kaggle consists of a single 0-1 number per line. For each test example, the prediction file should contain multiple lines with respect to the predefined number of senses for the target word. Concretely speaking, if an example refers to a word with k *senses*, your system has to output $k+1$ lines for that example. The first output is always reserved to indicate the 'unclear' or 'none of the above' as we saw in the training set; having 0 or 1 in the remaining$k$ will be decided by our system. For example, if the first word in the test data has 4 senses and your system predicts senses 1 & 4 are appropriate, then the first 5 lines of your submission file should be the following:

   ```
   0
   1
   0
   0
   1
   ```

2. Since you have to submit hard scores of 0 or 1, it would be worth thinking about how to allow multiple 1's rather than choosing just one best sense. Typically this is done by thresholding, but largely depends upon your design decision.

3. *(Optional)* Instead of voting only 0 or 1 to each sense, you could design another scoring scheme that partially votes to each sense with respect to its appropriateness.[10] For the supervised WSD using the Naive-Bayes model, it is easy to vote partially because the system guesses how likely each sense

[9] Accuracy = # of correct predictions / # of total predictions
[10] This is called soft score.

is correct, whereas you may have to do some normalization for soft-scoring in the dictionary-based method. Evaluate the prediction result **on the validation set** by the same format with using the hard-score: performing multiple simple predictions rather than doing a single sequence prediction. Analyze the difference between the two scoring schemes.

4. *(Optional)* In the supervised setting, it is highly interesting to see the effect of the training set size. Drawing the learning curve against the different number of training samples, and analyze the result. You could do this by testing on the validation set or by testing on Kaggle. In case that you did the previous extension, report two learning curves by using both hard-scoring and soft-scoring via testing on the validation set.

6 Proposal

Describe your WSD system and implementation plan in 1 page: what kinds of features are you planning to extract from the context for supervised WSD? What are you going to do for dictionary-based WSD? Provide a **clear and brief** explanation of your system and algorithm. (You don't have to repeat the basic model that described in this document) Rather than writing up every single detail, try to explain the motivation of your design decisions by providing the intuition via examples. Note that the more examples you brainstorm, the higher accuracies you will get through this assignment.

7 Report

You should submit a short document (5-6 pages will suffice) that consists of the following sections. (Of course, you can add more sections if you wish!)

1. Approach: Explain your approaches for two different WSD systems. Try to justify your design decisions by providing intuitive examples.

2. Software: List any software that your system uses, but you did not write by yourself. Make clear which parts of system rely on the software if it is not obvious.

3. Results: Explain what kinds of experiments you perform for this assignment. Summarize the performance of your system on the test data. Minimally, you could compare your results to the baseline system that always predicts to the most frequent sense. Note that having no comparisons will not justify the experimental performance of your system. Please put the results into clearly labeled tables and diagrams including a written summary of the results.

4. Discussion: You should include observations that you achieve during the experiment. One essential discussion is to analyze why and which features

are informative based on the real examples. Please report the top three features with the selected real examples. [11] Discuss the difference between the supervised and the dictionary-based WSD systems. Which system is more appropriate for which cases?

8 Guideline

1. This is a relatively big project. We suggest that you work in groups of at least four, ideally five. We may allow you to form a group of six students in case that your proposal shows reasonable amount of implementation plan. Recruiting people from various backgrounds will highly benefit both implementation and analysis for this assignment.

2. It is not allowed to use other prebuilt WSD systems to implement or fortify your own system. In contrast, using toolkits such as NLTK or OpenNLP is encouraged. Note that you should not use machine learning algorithm such as SVMs for this assignment.

3. Start early and try to discuss the training and dictionary data together before writing a proposal. Leave enough time to provide an analysis of your results.

4. **Submitting to Kaggle is not optional, but mandatory for every team. Detailed information about Kaggle will be updated via Piazza**.

5. Grading
   - Proposal: [10 pts]
   - Implementation: supervised [25 pts] / dictionary-based [25 pts]
   - Report: [50 pts]
   - Optional Extensions: [10 pts]

6. What to submit
   - Proposal (pdf into CMS, hardcopy at class)
   - Source code (only the code that YOU wrote, not for any of the packages that you used).
   - Prediction output (the files you submit to Kaggle)
   - The report (pdf into CMS, hardcopy submission will be notified later)
   - Archive all of these except the proposal, and upload it to CMS.

[11] You don't need to implement anything complicated such as measuring information gain. Measure how much accuracy drops as you remove a certain feature each time.