CS 4740/5740 Introduction to NLP
Spring 2014
Word Sense Disambiguation

Proposal: Due via CMS and bf hardcopy by Thu, March 6th, 1:25pm.
Results and Report: Due via CMS by Weds, March 19th 11:59pm.
Hardcopy of report due in class on Thurs, March 20th.

1 Introduction

The goal of the Word Sense Disambiguation (WSD) task is to find the correct meaning of a word in context. Because many words in natural language are polysemous, humans perform WSD based on various cues from the context including verbal (and presumably non-verbal) signals. In this assignment, you are to implement a WSD system 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 language-processing tasks.

In this assignment, you will build two WSD classifiers (from the training data in the case of supervised methods) and analyze their pros and cons by evaluating them on the validation data. We will also launch a Kaggle competition using the test data so that you can determine the accuracy of the predictions of your WSD system. **Note that WSD is known as one of the most challenging tasks in the world of NLP!**

2 Dataset

To get an initial picture, 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 at the following link.[2]

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

The data files are lightly preprocessed for the class project. They are available via CMS and consist of training, validation, and test data with an XML-formatted dictionary that describes the commonly used senses for each word. For each lexical element (i.e. word), the dictionary lists its senses, assigning one integer id per sense. Consider, for example, the following entry from the dictionary:

```
<lexelt item="future.n" num="4">
<sense id="1" wordnet="1" gloss="time to come" examples="In the future we will drive flying cars.
| What will you do in the future." />
<sense id="2" wordnet="2" gloss="verb tense" examples="The paper was written in future tense." />
<sense id="3" wordnet="3" gloss="commodities" examples="He made his living trading in futures." />
<sense id="4" wordnet="" gloss="personal time to come" examples="My future is bright. | What will
you do with your future?" />
</lexelt>
```

It describes the noun *future*. Four different senses exist for this word. Each sense has its own *gloss(definition)* and several *examples* that are separated by the | symbol. The corresponding WordNet 2.1 senses are also indicated. Because our words senses are different from WordNet's, senses might be mapped to multiple WordNet senses or possibly no WordNet sense (e.g., see the fourth sense in the above example).

Each example in the training data specifies the correct sense of the target word along with the context in which it appeared. Concretely, each line of training data has the the following format:

```
word.pos | sense-id | prev-context %% target %% next-context
```

- `word` is the original form of the target word for which we are to predict the sense. You will use it find the associated entry in the dictionary.
- `pos` is the part-of-speech of the word where 'n', 'v', and 'a' stand for noun, verb, and adjective, respectively.
- `sense-id` is an integer that denotes the correct sense from our dictionary. Note that each word has a different number of senses.
- | and `%%` are delimiters.
- `prev-context` is the text that precedes the target word in the example.
- `target` is the actual appearance of the word. Note that it may be a morphological variant of the target word. For example, the `word` "begin.v" could show up as "beginning" instead of "begin" to denote a participle at the given position.

- `next-context` is the text that follows the target word.

Note that the gold-standard sense-ids in the test set are all set to 0. Also, you will need to do additional processing of each example to extract the features (i.e., attribute-value pairs) you decide to include in your system.

3 Supervised WSD

This section will show a simple probabilistic approach called the Naive-Bayes model[3] 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 that we are going to choose the most probable sense as the correct meaning of the target word w. By applying Bayes rule,

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

Because 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[4] that each feature in the feature vector $\vec{f}$ is 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 a particular sense can be estimated by the product of the probabilities of its individual features given that sense under our assumption. Hence the best sense is

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

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

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

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 Estimation (MLE) which simply counts the number of actual occurrences in the training set. Particularly 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-related 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 $bank_2$,

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

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

3. When learning (i.e., training) your model, be sure that you never use the test data for training. Note that the correct sense ids are deliberately "erased" (i.e., changed to 0) in the test data. Instead of marking the predicted senses directly into the test file, you are going to generate a separate output file consisting only of the predicted sense ids 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 design multiple different variations based on the Naive-Bayes model and want to compare their performance on non-training data), we provide a validation set, which contains examples randomly reserved from the original test data. Since the true senses are given for

the validation data, testing on the validation set will let you make an educated guess as to how well your system (or models) will work for the test data. This process is called *validation*. Note that you must not train on the validation set if you are currently in a validation step. **However, you can add the validation set to the training data for the Kaggle competition later.**

4 Dictionary-based WSD

This section will show a dictionary-based WSD approach that is different from the previous supervised setting. Rather than training a system using the human-annotated 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**

Shown in **boldface** above, the 3rd sense of the target word matches more terms across the sense definitions of the context word and so should be selected as the correct sense for "cone". This process is the original Lesk algorithm described in class and in the readings: it disambiguates senses based only on the cross-comparison of the sense definitions. However, the rich examples provided in the dictionary can also be utilized to extend the Lesk model toward a more flexible matching scheme. **Read the following instructions carefully**.

1. Design a metric that rewards consecutive overlaps more than two distant overlaps of a single word. Note that there can be morphological variations in the definitions and examples. In order to increase chances of matching, stemming or lemmatizing could be useful.[5]
2. Implement a dictionary-based WSD system that disambiguates the target word by comparing its definitions to the definitions of relevant words in the context. Just as the performance of the supervised WSD system depends largely on the features employed, your decision for how to find relevant words will determine the performance of the dictionary-based system in combination with the metric you designed above.

[5] You can find such tools in some libraries such as NLTK and WordNet.

3. In contrast to the supervised settings, there is no real "training" process involved here because we mainly use definitions and examples in the dictionary to figure out the correct sense. Although the dictionary file contains glosses and examples (see Section 2), you might conclude that these are not enough to perform the dictionary-based approach. In this case, using WordNet could be an alternative.[6]

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

5 Scoring and Extensions

We use *accuracy*[7] as a score. Since no examples are mapped to multiple senses, a prediction will be counted as correct only if it exactly matches the sense given in the ground-truth labels.

1. Assuming the target word in a test example has k different senses based on the dictionary, the prediction file for Kaggle must consist of one $1 \sim k$ integer per line. Concretely speaking, if the test set consists of three examples with target words that have 7, 3, and 5 different senses, respectively, your system should output one line for each of three test examples like the following:

   7
   1
   4

2. *(Optional)* You are predicting only the single most-likely sense, but it would be interesting to think about how (and when you might want) to keep track of multiple candidate senses, and pick the second or third best senses (instead of the top-ranked sense) when some conditions are satisfied. There is no one right answer or approach here, so be creative in designing and testing multiple different approaches. Explain your choice of approach based on real examples (i.e., examples from our dataset), and analyze whether or not it improves the performance.

3. *(Optional)* Instead of a scoring approach that assumes the prediction of a single sense, you could design another scoring scheme that partially

[6] The WordNet senses that we associated here in the dictionary file are based only on the version 2.1 rather than the most recent distribution. You can download WordNet 2.1 from the web, including raw data files and its own API. Because using WordNet is not mandatory, we will not be able to answer questions about how to install it or how to use the API.

[7] Accuracy = # of correct predictions / # of test examlples

votes for each sense with respect to its confidence based on your model.[8] For the supervised WSD using the Naive-Bayes model, it is easy to vote partially because the system guesses how likely each sense is correct as a probability distribution, whereas you may have to do some normalization for soft-scoring in the dictionary-based method.

Note that this scoring can be explained as an expected score: for example, if your best answers are *sense-1* with 70% confidence and *sense-2* with 30% confidence, you will gain only 0.7 (rather than 1.0) if *sense-1* is a right answer, whereas you will lose only 0.7 (rather than 1.0) if sense-2 is a right answer. Evaluate the new soft-scoring approach **on the validation set** and compute the average accuracy. Analyze the difference between the two scoring schemes and discuss which one seems more beneficial with supporting reasons.

4. *(Optional)* In the supervised setting, it is highly interesting to see the effect of the training set size on performance. To do this, show the "learning curve" for the Naive Bayes approach: plot the accuracy of a model (y-axis) vs. the number of examples used to train it (x-axis), varying the training set size from a small number to larger and larger numbers. Analyze the results. The learning curve could be plotted by testing on the validation set or by testing directly on Kaggle. If you completed the previous extension, report two learning curves by using both hard-scoring and soft-scoring.

6 Proposal

Describe your implementation plans for each of the two WSD systems:

- What kinds of features are you planning to extract from the surrounding context for supervised WSD?
- What are you going to do for finding relevant words in the context for dictionary-based WSD?
- Provide a brief implementation schedule.

Provide a **clear and brief** explanation of your planned systems and algorithms. (Do not repeat the basic models that are already described in this document) Rather than writing up every single detail, **try to explain the motivation of your design decisions by investigating the provided dataset and illustrating the intuition that you discovered from the real examples**. Note that better features (typically identified by looking at the training data), the higher accuracies you will likely achieve.

[8] This is called a soft score.

7 Report

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

1. Approach: Explain your approaches for each of the two WSD systems. (Your final designs might, and probably will, change from the ones given in the proposal.) Try to justify your design decisions by providing various intuitive real examples.

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

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

4. Discussion: For the supervised system, you should include observations or findings that result from the experiment. One essential discussion is to analyze why and which features are informative based on the real examples. Please report the top three most informative features and provide examples of their usefulness via selected real examples.[9] Discuss the difference between the supervised and the dictionary-based WSD systems. Which system is more appropriate for which cases?

Above all, we highly encourage you to read this problem statement carefully, and find the relevant questions to answer by yourself!

8 Guidelines

1. This is a relatively big project. We suggest that you work at least in groups of three, ideally four. Recruiting people from various backgrounds will be beneficial in both implementation and analysis particularly 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[10] or OpenNLP is encouraged. Note that you should not use machine learning algorithms such as SVMs for this assignment unless they are used in addition to Naive Bayes.

[9] 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, instead.

[10] WordNet corpus given inside NLTK is likely to be more recent version than 2.1.

3. Start early and try to discuss the training and dictionary data together before writing a proposal. Reserve 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 competition will be updated via Piazza**.

5. Grading

   - Proposal: [10 pts]
   - Implementation: supervised [20 pts] / dictionary-based [20 pts]
   - Report: [45 pts]
   - Optional Extensions: [15 pts]

6. What to submit

   - Proposal (pdf into CMS, hardcopy at class)
   - Source code (only the code that YOUR TEAM wrote, not for any of the common packages that your team used).
   - Prediction output (the csv files you submit to Kaggle)
   - The report (pdf into CMS, hardcopy at class)
   - Archive all of these except the proposal, and upload it to CMS.