Introduction

In this lab you will investigate the Lexical Sample Task data taken from Senseval 3 and you will build a supervised decision list classifier, trained on the provided training data and tested on the provided test data.

Examining the data

The data provided by the Senseval 3 competition is in /data/cs65/senseval3/. Training and testing data are in separate subdirectories. Before you begin work on the lab, you need to spend a few minutes looking at the source data. In particular, examine the files train/EnglishLS.train and test/EnglishLS.test. If you look through the training data file closely, you’ll find that some words are tagged with two senses, indicating that annotators believed that both senses were represented, and you’ll find that some words that are tagged as “U”, or “Unassignable”. This means that the word was left unlabeled because annotators did not think the sense inventory contained an entry for the correct sense. There are even some cases where the word was given two sense labels: one an actual sense label for the word, the other “U”, meaning that some of the annotators agreed on one sense but other annotators left it unassigned.

Parsing the data

The source data for the Senseval task is tricky to parse correctly. I have provided you with a file, parse.py, that will do this parsing for you. You can use this file directly as such:

```python
from parse import getData
if __name__=='__main__':
    trainingFile = '/data/cs65/senseval3/train/EnglishLS.train'
    trainData = getData(trainingFile)
```

You’re also welcome to modify the parse.py file, though I’m not sure it’s necessary. Note that parsing the XML file is not particularly fast (it takes about 8 seconds each time), so you may want to run tests of your code in the Python interpreter, e.g.

```python
python -i warmup.py
>>> trainData.keys()
[u'begin.v', u'rule.v', u'play.v', u'argument.n', u'bank.n', ...]
>>> trainData['play.v'].keys()
[u'play.v.bnc.00018254', u'play.v.bnc.00026527', ...]
```

See the comments at the top of the parse.py file for further examples of how to use the data returned by the getData function.

Each word is labeled with a key that does not map directly to the version of WordNet installed on the lab machines. (There may be a way to map this to the current WordNet version, version 3.0, but you
do not need to do that for this lab. If you are really interested in doing this, you are welcome to talk to me for pointers, but I don’t have an implemented solution you can use directly: you’d have to do the hard work yourself.)

Exploring the training data

You should store the solutions to questions 1-11 of the lab in warmup.py.

1. How many different “lexelts” (lexical elements) are there? The lexelts represent the polysemous word types that you are being asked to disambiguate.

2. What is the breakdown of part of speech of these lexelts? (How many verbs, nouns, and adjectives?)

3. How many training examples are there for the noun “organization” (organization.n)?

4. How many senses of organization.n are represented in the training data? (Do not count “U” as a sense for now.)

5. One common baseline for this task is to assume that we guess randomly. However, rather than actually making random guesses, you just assume that you will get $\frac{1}{n}$ of the instances for a particular lexelt correct, where $n$ is the number of senses for that lexelt. For example, if there 5 senses for a word $w$, the random baseline will get 20% of the instances of the word $w$ correct. (It doesn’t matter if the number of instances for $w$ is not a multiple of $n$. If there are 37 instances for a word with 5 senses, you’ll get 7.4 of them correct.) The question for you is as follows: what percentage of the words in the training data would you be able to label correctly if you just guessed randomly? (Note that this makes some assumptions about how you’d score instances with multiple answers and how you’d score instances with “U”, but just ignore that for now.)

6. What is the most frequent sense for the organization.n? (Specify this in the form of how the answer is written in the training data, e.g. organization%1:04:02::)

7. Another common baseline is the “most frequent sense” baseline. In this baseline, we find the most frequent sense for each word and the label every instance with that most frequent sense. What percentage of the instances in the training data would you label correctly using the most frequent sense baseline? (You may need to think for a bit about how to actually implement this.) Is this higher or lower than you expected? Discuss.

Exploring the test data

8. How many examples of organization.n are there in the test data?

9. There are two similar questions that are not repeats of one another:

   (a) Are there any senses of organization.n that are represented in the training data but not in the test data? Assuming the answer is yes, would this be a problem? Discuss briefly.

   (b) Are there any senses of organization.n that are represented in the test data but not in the training data? Assuming the answer is yes, would this be a problem? Discuss briefly.

10. For each word in the training data, find the most frequent sense. Repeat this for the test data. How many words have the same most frequent sense in the training data and the test data? Also report this as a percentage of the words (types). You can choose how to handle ties. Discuss the implications of the disparities you found between the MFS in the training and the MFS in the test.
11. Use the MFS sense from the training data to label the test data. How well does this do? Compare to the random baseline (Question 5) and the MFS baseline (Question 7) performed when applied to the training data. How does this match your intuition? Discuss.

Important scoring notes:

- For words in the test data with multiple sense labels, count it as a match if you match at one of the labels.
- “U” is treated as the correct labeling for the sense, and your system is supposed to be able to predict when “U” is appropriate for the instance, so if you predict an actual sense label (e.g. \texttt{organization?1:04:02::}) and the true label is “U”, you are incorrect.

Writing a supervised decision list classifier

Put the code to this question in \texttt{declist.py}. Put your answers to this question in \texttt{declist.txt} (or \texttt{declist.tex} if you want to keep practicing \LaTeX).

Here we will develop a supervised decision list classifier. (See the Jurafsky and Martin textbook, page 643 for more details.) We want to identify which words in a bag of words (+k from the head word) and which collocations (here, two word phrases beginning or ending with the head word) are high-quality indicators of one sense over another.

First, let’s see how this works in the case where there are only two senses you are trying to decide between (a binary classifier). For each feature $f$, which could be a word in the bag of words or a collocation, you first compute $P(\text{sense}_1|f)$ and $P(\text{sense}_2|f)$, and then compute a score of how good this feature is in distinguishing the two senses. This score is calculated by taking the absolute value of the log of their ratios:

$$\text{score} = |\log \frac{P(\text{sense}_1|f)}{P(\text{sense}_2|f)}|$$

High scores indicate that the feature is good at distinguishing the senses. Low scores indicate that the feature is not good.

$P(\text{sense}_1|f)$ can be estimated using maximum likelihood estimation as:

$$\frac{\text{count}(f \text{ is present in sense}_1)}{\text{count}(f \text{ is present in either sense})}$$

Since we’re going to divide $P(\text{sense}_1|f)$ by $P(\text{sense}_2|f)$, and since their denominators will be the same, they cancel each other out, giving us this:

$$\frac{P(\text{sense}_1|f)}{P(\text{sense}_2|f)} = \frac{\text{count}(f \text{ is present in sense}_1)}{\text{count}(f \text{ is present in sense}_2)}$$

SMOOTHING: In some cases, the numerator or denominator might be zero, so we will use Laplace smoothing (it’s easy!) but instead of +1 we will use $+\alpha$ smoothing, adding $\alpha = 0.1$ to both the numerator and denominator in order to avoid division by zero errors, and to avoid taking the log of 0.

In the lexical sample data we have, we can’t directly use the formulation above because we don’t have a binary classification: we have to decide which sense label to assign when there might be many more than 2 choices. Therefore, we don’t use the formulation shown above; rather, we use the following modification for each sense $i$:

$$\text{score} = \log \frac{P(\text{sense}_i|f)}{P(\text{NOT sense}_i|f)}$$

Notice that the absolute value is gone: we only care about cases where the log likelihood ratio is positive since we can’t classify words as “not sense;”. 

Similar to what we showed above, we can rewrite the ratio of probabilities as follows:

\[
P(\text{sense}_i|f) = \frac{\text{count}(f \text{ is present in sense}_i)}{\text{count}(f \text{ is present all other senses except sense}_i)}
\]

To create a decision list from the features, you will first calculate a score for each (sense, feature) pair. For some senses, a feature will be a good indicator; for other senses it will not. Sort these (sense, feature) pairs so that the (sense, feature) pair with the highest score comes first. That’s the decision list.

To use the decision list on the training data, you will begin with an instance of a word. (Here, “instance” refers to the item in the test data that includes the text and the identified head word.) Now, walk down the decision list, which is a sorted list of (sense, feature) pairs. If the instance contains the first feature, label it as that sense. If not, proceed to the next feature. Continue this process until you have exhausted all features in your decision list. If you get to the end of your decision list, just label the instance with the MFS from the training data.

**Important:** In case it was not obvious from the description above, you need to create a separate decision list for each word (e.g. one decision list for bank.n, another for organization.n, etc.)

12. Implement the decision list described above, apply it to the test data, and report your accuracy. Does your decision list outperform the MFS baseline?

13. Examine the rules in the decision tree for organization.n.

   - What is problematic about some of the rules you have generated? Give some examples.
   - Add code to your decision list generator so that very common words (also known as “stopwords”) are excluded from being used as part of features. How you decide what “very common” means is up to you, but it’s probably best if its automatically generated rather than hand-entered. Does this help or hurt your performance? (Give me some numbers!)
   - Thinking about this task, can you think of any reason why removing some frequent words might be a bad idea? Provide an example if you can!
   - Implement case-folding (making everything lower-case) for the bag-of-words and collocations. Does this always make sense to do? Does this help or hurt performance? (Give me some numbers!)
   - What is your new accuracy now that you have added support for stopwords? Does your decision list now/still outperform the MFS baseline?

14. Is it possible that stopping before you get to the end of the decision list would improve results? In other words, perhaps instead of making it all the way to the end of your decision list, you stop whenever the score falls below 1.0 – or perhaps you stop after seeing 100 rules. Once you reach the end of the rules you’re willing to consider, fall back on the MFS baseline. Try a couple of scenarios, see what happens, and discuss what you see.