Natural Language Processing

Lab 11: Word Sense Disambiguation

---

1 Introduction

In this lab, we are going to explore the use of machine learning for word sense disambiguation. More precisely, we will use NLTK to build Naive Bayes classifiers to disambiguate selected words using data from Senseval-2. All files are available in `/local/kurs/nlp/semantics/`.

Acknowledgment: Thanks to Alex Lascarides for letting us reuse and modify an older lab, including the code in `wsd_code.py`.

2 Data

The Senseval-2 corpus consists of text from a mixture of places, including the British National Corpus and the Wall Street Journal section of the Penn Treebank. Each word in the corpus is tagged with its part of speech, and the senses of the following target words are also annotated: the noun *interest*; the verb *serve*; and the adjective *hard*. The senses used to annotate each target word come from the Longman Dictionary of Contemporary English and WordNet and are shown in the table below.[1]

Word	Sense	Definition	Example
interest	interest_1	readiness to give attention	"an interest in music"
	interest_2	quality of causing attention to be given to	"they said nothing of great interest"
	interest_3	activity, etc. that one gives attention to	"she has many interests"
	interest_4	advantage, advancement or favor	"in the interest of safety"
	interest_5	a share in a company or business	"they have interests all over the world"
	interest_6	money paid for the use of money	"how much interest do you pay on your mortgage?"
hard	hard_1	not easy; requiring great physical or mental effort to accomplish or comprehend or endure	"a hard task"
	hard_2	dispassionate	"a hard bargainer"
	hard_3	resisting weight or pressure	"a hard rock"
serve	serve_2	do duty or hold offices; serve in a specific function	"she served in Congress for two terms"
	serve_6	provide (usually but not necessarily) food	"we serve meals for the homeless"
	serve_{10}	work for or be a servant to	"may I serve you?"
	serve_{12}	be sufficient; be adequate, either in quality or quantity	"nothing else will serve"

Let us now start exploring the sense-tagged data using the Python interpreter. Copy the file `wsd_code.py` from `/local/kurs/nlp/semantics/` to your working directory and start the Python interpreter with the command `python`.

[1] These are subsets of the full set of senses defined in WordNet, which is why the numbering of senses is not (always) contiguous.

```
>>> import wsd_code as wc
>>> wc.senses('hard.pos')
['HARD3', 'HARD2', 'HARD1']
>>> hard = wc.senseval.instances('hard.pos')
>>> hard[0]
SensevalInstance(word='hard-a', position=20, context=[('``', '``'), ('he', 'PRP'),
('may', 'MD'), ('lose', 'VB'), ('all', 'DT'), ('popular', 'JJ'), ('support', 'NN'),
(',', ','), ('but', 'CC'), ('someone', 'NN'), ('has', 'VBZ'), ('to', 'TO'),
('kill', 'VB'), ('him', 'PRP'), ('to', 'TO'), ('defeat', 'VB'), ('him', 'PRP'),
('and', 'CC'), ('that', 'DT'), ("'s", 'VBZ'), ('hard', 'JJ'), ('to', 'TO'),
('do', 'VB'), ('.', '.'), ("''", "''")], senses=('HARD1',))
```

The session above shows how to use the method `senses()` to list the possible senses of the word *hard* and how to retrieve the set of all sense-tagged instances of this word. After using the method `instances()`, the value of `hard` is a list containing all the instances of *hard* in the corpus, and we can retrieve the first instance as `hard[0]`. An instance has four different attributes:

1. `word` specifies the target word together with its syntactic category; for example, `hard-a` means that the word is *hard* and that its category is *adjective*;
2. `position` gives its position within the sentence (starting with position 0);
3. `context` represents the sentence as a list of pairs, each consisting of a token and its tag;
4. `senses` is a tuple, each item in the tuple being a sense for the target word; typically, this tuple consists of only one argument, but there are a few examples in the corpus where there is more than one, representing the fact that the annotator couldn't decide which sense to assign to the word; for simplicity, we are going to ignore any non-first arguments to the attribute `senses`

Make sure you understand the representation of instances. Look at a few more instances of *hard* and then do the same for *interest* and *serve* (by replacing `'hard.pos'` with `'interest.pos'` and `'serve.pos'`, respectively).

3 Baseline classifiers

Before we build sense classifiers for *interest*, *hard* and *serve*, we need to establish baselines for comparison. Two commonly used baselines are the *random* baseline and the *majority* baseline. The random baseline is the accuracy that we can expect from guessing senses randomly.

1. What is the random baseline for *hard* given our current sense inventory?
2. Is the random baseline the same for *interest* and *serve*? Why (not)?

The majority baseline is the accuracy we obtain if we always guess the most frequent sense. We can derive the majority baseline for *hard* using NLTK methods as follows:

```
>>> import wsd_code as wc
>>> import nltk
>>> hard_dist = nltk.FreqDist([i.senses[0] for i in wc.senseval.instances('hard.pos')])
>>> hard_dist
FreqDist({'HARD1': 3455, 'HARD2': 502, 'HARD3': 376})
>>> hard_baseline = hard_dist.freq('HARD1')
>>> hard_baseline
0.797369028386799
```

We use the method `nltk.FreqDist()` to compute the distribution of different senses of *hard* in the Senseval corpus. The output is stored in the variable `hard_dist` and shows that `'HARD1'` has 3455 occurrences, while `'HARD2'` and `'HARD3'` occur only 502 and 376 times, respectively. Hence, by always guessing that *hard* has the first sense, we reach an accuracy (on this sample) equal to 3455 / (3455 + 502 + 376) = 0.797, or 79.7%.

3. Compute the majority baselines for *interest* and *serve*, respectively.

4 Naive Bayes classifiers

We are now going to compare the performance of different classifiers that perform word sense disambiguation, using the method `wsd_classifier()`, which must have at least the following arguments specified by you:

1. a trainer, in this case `NaiveBayesClassifier.train`;
2. a target word: `'hard.pos'`, `'interest.pos'` or `'serve.pos'`;
3. one of two feature representations:
   - `wsd_word_features`: a bag-of-words representation defined over the 300 most frequent words that co-occur with the target word in the training corpus (after filtering out stop words);
   - `wsd_context_features`: a collocational feature representation consisting of 3 preceding and 3 succeeding word-tag pairs;

The method splits the data available for a target word into a *training set* and a *test set*, trains a Naive Bayes classifier on the training set, and evaluates its accuracy on the test set. Here is how you use the method to evaluate the bag-of-words model for the target word *hard*:

```
>>> import wsd_code as wc
>>> import nltk
>>> nb_hard = wc.wsd_classifier(nltk.NaiveBayesClassifier.train, 'hard.pos', wc.wsd_word_features)
Reading data...
 Senses: HARD1 HARD2 HARD3
Training classifier...
Testing classifier...
Accuracy: 0.8108
```

You should do the following:

1. Evaluate both feature representations for all three target words.
2. Compare the results for different classifiers, including your baselines.