Homework 4

Statistical NLP

Due: Tuesday, March 13

1 HMM Tagging

In this task you will go through the steps required to build a HMM-based statistical POS-tagger for a language much like English (but with fewer parts of speech!). Accompanying this homework is a file `tagseq.txt` which has a sequence of tagged English-like sentences. You task is to "build" a tagger for this language based on this sample. This involves three steps.

- Collect the POS transition probabilities—P(V | N), P(V | DET), etc.—from the file
- Collect the word-emission probabilities—P(the | DET), P(a | DET), P(man | N), etc.
- Put these each in a table (a transition probability table and an emission probability table)
- For the word sequence: *buffalo buffalo buffalo buffalo buffalo* determine what the most likely POS sequence is using the Viterbi algorithm (show the steps you go through - that is, build the delta table and demonstrate how you back-track through the table to get the tag).

2 Statistical Parser

In this problem you will go through the steps required to build a statistical parser. The first part involves using a tree-bank to extract statistics for a simple probabilistic phrase-structure grammar. The second part involves evaluating quantitatively the parses generated by the grammar.

Induce a grammar from the Penn Treebank. There is an easy way to do this, based on the part of the Penn Treebank distributed with the NLTK. The obvious steps are:

```
>>> productions=[]
>>> for tree in nltk.corpus.treebank.parsed_sents():
            productions+=tree.productions()
>>> parser=nltk.parse.viterbi.ViterbiParser(nltk.grammar.induce_pcfg(nltk.Nonterminal('S'),productions)
>>> parser.parse(str.split("the man left ."))
```

You should, however, make a couple modifications:

- First: before you induce the grammar, reserve some of the sentences (10 % of them) for evaluation.
- Many of the rules contain numerical indices which are part of the way the Penn Treebank treats "filler-gap" dependencies.
  (eg. `S -> NP-SBJ-1 VP, NP-SBJ-1 -> DT CD JJ NNS`)
  You should eliminate these indices (e.g. simply eliminate the `-1` in the rules above).
- Finally: You should expand the lexicon. Do this by using the whole tagged Penn Treebank (/home/Corpora/PTB/), extracting the open class elements (verbs, nouns, adjectives and adverbs), and generating lexical productions (`NNP -> 'Sony'`) that you can add to the productions you extracted from the nltk.corpus.treebank productions to build a grammar with more lexical coverage

Now evaluate your parser by having it parse the sentences that you have reserved. (If you did the lexical extension part right, your grammar should cover the words). You should do this by using the PARSEVAL measure (you will have to write evaluation code - or find some [hint: there is at least one piece of code available to do this task in python which I am aware of])

3 Baum-Welsh (forward-backward) Algorithm

Using the file `eisenerHMM.xls` which illustrates the workings of the Expectation Maximization strategy to learning the parameters of a Hidden Markov Model you should: Inspect the model and explain what the sums represented in bold under the first iteration represent. (**14.679 18.321 9.931 3.212 1.537 1.069 7.788 9.463 12.855 1.695 1.599 15.85**) and how these are used to reestimate the model's parameters (which are listed below) after a single iteration?

	p(...|C)	p(...|H)	p(...|START)
p(1|...)	0.6765	0.0584	
p(2|...)	0.2188	0.4251	
p(3|...)	0.1047	0.5165	
p(C|...)	0.8757	0.0925	0.1291
p(H|...)	0.1090	0.8652	0.8709
p(STOP|...)	0.0153	0.0423	0