Computational Linguistics Practicals, Hilary Term 2016

PRACTICALS INFO: `http://www.cs.ox.ac.uk/teaching/timetables/`

The aim of the practical is to design and implement a bigram Hidden Markov Model tagger of the type described in lecture 2 or the references cited there.

Your starting point is a training data file `WSJ-2-12.tgz` consisting of sections 2-12 of the tagged Wall Street Journal texts from LDC Treebank 3. PLEASE DO NOT REDISTRIBUTE THIS DATA: it is under licence. Note that this version also contains NP chunk boundaries. For tagging, you will have to write a program that reads in this data and removes irrelevant information. Note that there are several other subtleties regarding breaks between sentences and other phenomena that will need care.

Once you've got the data in a suitable form you will have to write a program which estimates the quantities $P(Cat_i|Cat_{i-1})$ and P(Word|Cat) by counting instances in the training data, and working out relative frequencies as described in the lectures. You may at this stage want to think what to do about unobserved bigrams: there is an extensive literature on 'smoothing' for HMMs, but a simple method is to add 1 to each bigram count and then treat unobserved bigrams as if they had actually been seen once (Google for "add 1 smoothing").

Next you will have to implement the Viterbi algorithm for tagging new sentences. You may want to extend it a little to cope with the case where you encounter an unknown word.

When you have built the training routines and the bigram tagger, test its accuracy by a technique called 10-fold 'cross-validation'. To do this you split the corpus 90%/10% into training and test sections. Train your tagger on the 90% and test on the 10%. Now take a different 90%/10% split and do the same, and so on, rotating the 90%/10% split so that eventually each bit of the corpus has been used as testing data. You can then average the results for each 10% test set.

Carry out the test under TWO conditions: one where the P(Word|Cat) quantity (but not the bigrams) is derived from the whole 100%, so that there are NO UNKNOWN WORDS; and one test where just the 90% training data is used, so that there are SOME UNKNOWN WORDS.

You can use whatever programming language you like, but make sure that your code is **fully and clearly documented**, so that someone who may not be completely fluent with that language can understand what you have done.

If you finish the exercise early, you may want to do one or more of the following, although these are ENTIRELY OPTIONAL AND NOT NECESSARY.

- extend the tagger to use trigrams, using the representation method mentioned in the lectures.

- implement a Naive Bayes or other classifier to guess the POS of unknown words. You can use features of the word itself, and/or features from the context. Train and test it using cross-validation as before.
- using the data provided, implement an HMM-based NP chunker along the lines described in the lectures.
- experiment with different smoothing methods, in particular the Kneser-Ney method as described in

  `http://www.cs.berkeley.edu/~klein/cs294-5/chen_goodman.pdf`.

You need to get one of the demonstrators to sign off your working system by the end of week 8.

Stephen.Pulman@cs.ox.ac.uk