1 Introduction

This assignment will make use of the Natural Language Tool Kit (NLTK) for Python. NLTK is a platform for writing programs to process human language data, that provides both corpora and modules. For more information on NLTK, please visit: http://www.nltk.org/.

NB: This assignment will be marked, you will get feedback, but the mark will not count towards your overall mark for the course. Material introduced in lectures and covered in this assignment may well appear on the final exam.

1.1 Getting Started

Before continuing with this assignment, download a copy of the assignment template assignment2.py from the FNLP course website. This template contains code that you must use as a starting point when attempting the questions for this assignment.

1.2 Submitting Your Assignment

Submit your assignment code using the submit system:

   $ submit fnlp 2 assignment2.py

The notional deadline for this assignment is 16 March. Work submitted after this date may not be marked or given feedback.

Before submitting your assignment:

- Ensure that your code works on DICE. Your assignment must work when the following command is run:

  $ python2.7 assignment2.py

- Test your code thoroughly. If your code crashes when run, up to 25 marks may be deducted for code that fails to run (on DICE), depending on the severity of the coding error(s).

- Please use the tests provided in the template to do a sanity check for your code. A version of the interim results checker may become available to give more detailed feedback.

- Ensure that you include comments in your code where appropriate.

- Important: Whenever you use corpus data in this assignment, you must lowercase the data, so that e.g. the original tokens “Freedom” and “freedom” are made equal. Do this throughout the assignment.

1.3 Good Scholarly Practice

Please remember the University requirement as regards all assessed work. Details about this can be found at:
Section A: Training a Hidden Markov Model (15 marks)

In this part of the assignment you have to train a Hidden Markov Model (HMM) for part-of-speech (POS) tagging. Review Lab 3, particularly Exercises 3 and 4, if you need a reminder of what you have to compute.

Access to the training and test data you must use are provided in the template. An object of the class HMM will be created in the main function and the training and test data will be passed as arguments. The labeled sentences are annotated with the Universal POS tagset. Having a small number of labels (states) will make Viterbi decoding faster.

Question 1 (5 marks)

Estimate the emission model. Use ConditionalProbDist with the estimator: LidstoneProbDist with +0.01 added to the sample count for each bin, adding one to the number of bins already present. Lowercase all the observations (words).

Fill in the function emission_model(self, train_data) from the HMM class. Store the emission model in the variable self.emission_PD. Save the states (types) that were seen in training in the variable self.states. Both these variables will be used by the Viterbi algorithm in Section B.

Question 2 (10 marks)

Estimate the transition model. Use ConditionalProbDist with the estimator: LidstoneProbDist with +0.01 added to the sample count for each bin, and one to the number of bins already present. Add a start state <s> and an end state </s>

Fill in the function transition_model(self, train_data) from the HMM class. Store the transition model in the variable self.transition_PD. This variable will be used by the Viterbi algorithm in Section B.

Testing (No marks)

The template provides two functions to test this part of the assignment: test_emission(self) and test_transition(self). These tests are called from the main function. The expected output for these functions is written as a comment in the template. Please check your solutions with the test functions. Consider this a sanity check for your code.

Section B: Implementing the Viterbi Algorithm (55 marks)

In this part of the assignment you have to implement the Viterbi algorithm. The pseudo code of the algorithm can be found in figure 10.8 of chapter 10 of the online 3'd edition draft of Speech and Language Processing or in the 2nd edition book in chapter 5, Figure 5.17. Follow the pseudo-code to guide your implementation.
In the pseudo-code the $b$ probabilities correspond to the **emission model** implemented in part A, question 1 and the $a$ probabilities correspond to the **transition model** implemented in part A, question 2. You should use **costs** (negative $\log_2$ probabilities) throughout. Therefore instead of multiplication of probabilities (as in the pseudo-code) you will do addition of **costs**, and where the pseudo-code uses $\max$, you should use $\min$.

**Question 1 (15 marks)**

Implement the **initialization step** of the algorithm. The algorithm uses two data structures that have to be initialized for each sentence that is being tagged: the **viterbi** data structure (10 marks) and the **backpointer** data structure (5 marks). Use **costs** when initializing the viterbi data structure.

Fill in the function `initialize(self, observation)`. The argument `observation` is the first word of the sentence to be tagged ($o_1$ in the pseudo-code). Describe the data structures with comments.

**Question 2 (40 marks)**

Implement the **recursion step** (20 marks) and the **termination step** (10 marks) of the algorithm. Reconstruct the tag sequence corresponding to the best path using the **backpointer** structure (10 marks).

Fill in the function `tag(self, observations)`. The argument `observations` is a list of words representing the sentence to be tagged. Remember to use **costs**. The two loops of the recursion step have been provided in the template. Fill in the code inside the loops for the recursion step. Fill in the code for the termination step outside the loops. Describe your implementation with comments.

**Testing (No marks)**

The template provides a test for this part of the assignment: accuracy over the test set is computed using your implementation of the `tag` function. The test data is tagged with the Universal tagset. These test is called from the `main` function. The expected output for the functions is written as a comment in the template. *Please use this test as a sanity check for your code.*

**Section C: Comparing Tagging Accuracy (30 marks)**

**Question 1 (30 marks)**

Compare the tagging accuracy when using different label sets. Use `HiddenMarkovModelTagger` to train an HMM tagger on the training data annotated with the Brown tagset and another HMM tagger on the training data annotated with the Universal tagset (10 marks). Evaluate the taggers on the test set (5 marks). Observe the difference in accuracy and answer the following questions:

- Why do you think the accuracy for one of the taggers is lower than for the other, considering the training data has the same size? (10 marks)
- How does the training size affect each of the taggers and why? Change the size of the training data in the code to test your hypothesis. (5 marks)
Testing (No marks)

The template provides a test for this part of the assignment: accuracy is compared to expected value. These test is called from the main function. The expected output for the accuracy for each system is written as a comment in the template. Please use this test as a sanity check for your code.