FNLP: Assignment 2

Hidden Markov Models: Part-of-speech Tagging

Deadline: 4pm, 15th March 2018

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: `www.nltk.org`.

1.1 Getting Started

Before continuing with this assignment, please **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

Your submission will be a python file. Please name it with your matriculation number: e.g., s0123456.py. Then submit your assignment code using the submit system:

$ submit fnlp cw2 s0123456.py

Before submitting your assignment:

- Ensure that your code works on DICE. Your assignment must work when the following command is run: $ python s0123456.py
- Test your code thoroughly. If your code crashes when run you may be awarded a mark of 0. If your code does not run to completion then it will make it very difficult for us to test your answers, we cannot be expected to fix your code while marking.

- Ensure that you include comments in your code where appropriate. This makes it easier for the markers to understand what you have done and makes it more likely that partial marks can be awarded.
- Any character limits will be strictly enforced. Answers will be passed through an automatic filter that only keeps the first x characters, where x is the character limit. Therefore, anything after the character cut-off will not be read.

1.3 Good Scholarly Practice

Please remember the University requirement as regards all assessed work. Details about this can be found at:

`http://www.ed.ac.uk/schools-departments/academic-services/students/undergraduate/discipline/academic-misconduct`

and at:

`http://www.inf.ed.ac.uk/admin/ITO/DivisionalGuidelinesPlagiarism.html`

1.4 Late Coursework Policy

You can submit more than once up until the submission deadline. All submissions are timestamped automatically. Identically named files will overwrite earlier submitted versions, so we will mark the latest submission that comes in before the deadline.

If you submit anything before the deadline, you may not resubmit afterward. (This policy allows us to begin marking submissions immediately after the deadline, without having to worry that some may need to be re-marked).

If you do not submit anything before the deadline, you may submit *exactly once* after the deadline, and a late penalty will be applied to this submission unless you have received an approved extension. Please be aware that late submissions may receive lower priority for marking, and marks may not be returned within the same timeframe as for on-time submissions.

Warning: Unfortunately the `submit` command will technically allow you to submit late even if you submitted before the deadline (i.e. it does not enforce the above policy). Don't do this! We will mark the version that we retrieve just after the deadline, and (even worse) you may still be penalized for submitting late because the timestamp will update.

For additional information about late penalties and extension requests, see the School web page below. Do **not** email any course staff directly about extension requests; you must follow the instructions on the web page.

http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests

Section A: TRAINING A HIDDEN MARKOV MODEL (25 Marks)

In this part of the assignment you have to train a Hidden Markov Model (HMM) for part-of-speech (POS) tagging. Look at the solutions from Lab 3, Exercise 3 and Exercise 4 as a reminder for what you have to compute.

The training and test data 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 fewer number of labels (states) will make Viterbi decoding faster.

Question 1 (10 Marks)

Estimate the *Emission model*. Use ConditionalProbDist with the estimator: Lidstone probability distribution with +0.01 added to the sample count for each bin. 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 (15 Marks)

Estimate the *Transition model*. Use ConditionalProbDist with the estimator: Lidstone probability distribution with +0.01 added to the sample count for each bin. Add a start state $< s >$ and an end state $< /s >$ to each sentence in the corpus.

Fill in the function $transition\_model(self, train\_data)$ from the HMM class. Store the Sensor 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: $test\_sensor(self)$ and $test\_transition(self)$. These test 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 (75 Marks)

In this part of the assignment you have to implement the Viterbi algorithm. The pseudo code of the algorithm can be found in the Speech and Language Processing book in the chapter 5, Figure 5.17. Follow the pseudo-code to solve the questions.

In the pseudo-code the b probabilities correspond to the *sensor model* implemented in part A, question 1 and the a probabilities correspond to the *transition model* implemented in part A, question 2. **You will have to use *log-probabilities***. Therefore instead of multiplication of probabilities (as in the pseudo-code) you will do addition of log-probabilities.

Question 3 (20 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 (*10 Marks*). Use *log-probabilities* when initializing the viterbi data structure.

Fill in the function $initiatlize(self, observation)$. 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 4 (50 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 *log-probabilities*. 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.**

Question 4b (5 Marks)

Modify the test code to inspect some of the incorrect tagged sequences. Why do you think these might have been tagged incorrectly? Pick an incorrectly tagged sentence and give a short answer describing why you think it was tagged incorrectly. Write down your answer in the *answer_question4b* function.

There is a 280 character limit on this question.