FNLP: Assignment 2

Hidden Markov Models: Part-of-speech Tagging

Deadline: Thursday 14th March 2019 1600

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/.

Getting Started

Before continuing with this assignment, please download a copy of the assignment template[1]. **This template contains code that you must use as a starting point when attempting the questions for this assignment.**

Submitting Your Assignment

Submit your assignment code using the submit system:

```
$ submit fnlp cw2 assignment2.py
```

Before submitting your assignment:

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

  ```
  (fnlp)... $ python assignment2.py
  ```

  See the lab introduction for instructions on how to enable the **fnlp** environment.

- 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, under the **fnlp** environment, using python3), depending on the severity of the coding error(s).

- 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, whether it's explicitly stated or not.

[1] http://www.inf.ed.ac.uk/teaching/courses/fnlp/coursework/assignment2.py

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-projec
late-coursework-extension-requests

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/academic-services/students/conduct/academic-misconduct

and at:

http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct

Section A: TRAINING A HIDDEN MARKOV MODEL (20 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.

You will need to create and train two models - an **Emission Model** and a **Transition Model** as described in lectures.

Use labelled sentences from the 'news' part of the Brown corpus. These are annotated with parts of speech, which you will convert into the Universal POS tagset. Having a smaller number of labels (states) will make Viterbi decoding faster.

Question 1 (10 Marks)

Estimate the **Emission model**: Fill in the `emission_model` method of the `HMM` class. Use a `ConditionalProbDist` with a `LidstoneProbDist` estimator with `+0.01` added to the sample count for each bin. Lowercase all the observations (words). Store the result in the variable `self.emission_PD`. Save the **states** (POSes) 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**. Fill in the `transition_model` method of the `HMM` class. Use a `ConditionalProbDist` with a `LidstoneProbDist` estimator 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 training data. Store the result in the variable `self.transition_PD`. This variable will be used by the Viterbi algorithm in Section B.

Section B: IMPLEMENTING THE VITERBI ALGORITHM (80 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 Jurafsky & Martin 3rd edition book in Appendix A Figure A.4: use it as a guide for 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 probabilities)*. Therefore instead of multiplication of probabilities (as in the pseudo-code) you will do addition of costs, and instead of *max* and *argmax* you will use *min* and *argmin*.

Question 3 (25 Marks)

Implement the **initialization step** of the algorithm by filling in the `initialise` method. 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.

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 **costs** when initializing the viterbi data structure.

Fill in the model construction and training parts of the test code in the `answers` function and check the results for plausibility (*5 Marks*).

Question 4a (40 Marks)

Implement the **recursion step** (*20 Marks*) and the **termination step** (*10 Marks*) of the algorithm by filling in the `tag` method. Reconstruct the tag sequence corresponding to the best path using the `backpointer` structure (*8 Marks*).

The argument `observations` is a list of words representing the sentence to be tagged. Remember to use **costs**. The nested 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 after the loops. Describe your implementation with comments.

Fill in the sample tag example and accuracy computation parts of the test code in the `answers` function and check the results for plausibility (*2 Marks*).

Question 4b (5 Marks)

Modify the test code so it saves the first 10 incorrectly tagged test sentences along with their correct versions. Inspect these. Why do you think these might have been tagged incorrectly? Pick one 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 the answer to this question.

Question 5 (10 Marks)

Suppose you have a hand-crafted grammar and lexicon that has 100% coverage on constructions but less than 100% lexical coverage. How could you use a pre-trained POS tagger to ensure that the grammar produces a parse for any well-formed sentence, even when it doesn't recognise the words within that sentence?

Will your approach always do as well or better than the original parser on its own? Why or why not?

Write down your answer in the `answer_question5` function.

There is a 500 character limit on the answer to this question.