FNLP: Assignment 2

Hidden Markov Models: Part-of-speech Tagging

DEADLINE: 4 PM, 12th March 2020

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/.

1.1 Getting Started

Before starting this assignment, please download a copy of the assignment materials[1] This contains additional Python modules used in this coursework, together with a file named `template.py`, **which you must use as a starting point when attempting the questions for this assignment**.

During development, we have provided an interim progress checker[2] where you can check your results and get some feedback as you work on your answer.

In a change from the Assignment 1 version, when you use the `--answers` command-line switch to produce an `answers.py` file for checking, the answer processing will try very hard to run to completion whether all the necessary functions have been completed or not. When run in this mode, error notices are logged to a file `userErrs.txt`, so be sure to check this if any of the lines in `answers.py` have no value.

1.2 Submitting Your Assignment

When ready to submit, create a directory that should be called `fnlp-ass2-<UUN>`, where `<UUN>` is your matriculation number e.g.: s1234567. In this directory put your modified **template.py** file, renamed with your matriculation number, e.g. **s1234567.py**.

Submit your assignment by creating a gzipped tar file from your `fnlp-ass2-<UUN>` directory. You do this using the following command in a DICE machine:

```
tar -cvzf fnlp-ass2-<UUN>.tar.gz fnlp-ass2-<UUN>
```

You can check that this file stores the intended data with the following command, which lists all the files one would get when extracting the original directory (and its files) from this file:

```
tar -tv fnlp-ass2-<UUN>.tar.gz
```

Submit this file via LEARN by uploading the file using the interface on the LEARN website for the course FNLP. If you have trouble please refer to this blogpost:

https://blogs.ed.ac.uk/ilts/2019/09/27/assignment-hand-ins-for-learn-guidance-for-students/

Before submitting your assignment:

- Ensure that your code works on DICE. Your modified **template.py** should fully execute using `python3`.

[1] http://www.inf.ed.ac.uk/teaching/courses/fnlp/coursework/assignment2.tar.gz
[2] https://homepages.inf.ed.ac.uk/cgi/ht/fnlp/interim2.py

- Test your code thoroughly. If your code crashes when run on DICE you may be awarded a mark of **0** and you'll get no qualitative feedback other than "the code didn't run".
- 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 to open questions will be strictly enforced. Answers will be passed through an automatic filter that only keeps the first N characters, where N is the character limit given in a question.
- **Important:** Whenever you use corpus data in this assignment, you *must* convert the data to lowercase, 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.

1.3 Late Coursework Policy

You can submit more than once up until the submission deadline. All submissions are timestamped automatically. We will mark the latest submission that comes in before the deadline.

If you do not submit anything before the deadline, then you won't get the formative feedback from this assignment unless you have received an approved extension.

For additional information about late penalties and extension requests, see the School web page[3]. Do **not** email any course staff directly about extension requests; you must follow the instructions on the web page.

1.4 Good Scholarly Practice

Please remember the University requirements as regards all assessed work. Detail about this can be found on guides for academic misconduct[4] and plagiarism[5].

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 (NLTK uses the smaller version of this set defined by Petrov et al.[6]). Having a smaller number of labels (states) will make Viterbi decoding faster.

Use the last 500 sentences from the corpus as the test set and the rest for training. This split corresponds roughly to a 90/10% division. Do not shuffle the data before splitting.

Failure to follow these instructions exactly will render most of your answers incorrect.

[3] http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests
[4] https://www.ed.ac.uk/academic-services/students/conduct/academic-misconduct
[5] https://www.ed.ac.uk/academic-services/students/conduct/academic-misconduct/plagiarism
[6] https://github.com/slavpetrov/universal-pos-tags

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.

Review the help text for the `ConditionalProbDist` class. Note that the `probdist_factory` argument to its `__init__` method can be a function that takes a frequency distribution and returns a smoothed probability distribution based on it, a.k.a. an estimator.

Review the help text for the `LidstoneProbDist` class and look particularly at the arguments to the `__init__` method. You should implement a function to pass to `ConditionalProbDist` which creates and returns a `LidstoneProbDist` based on the input frequency distribution, using `+0.01` for smoothing and adding an extra bin.

Convert all the observations (words) to lowercase.

Store the result in the variable `self.emission_PD`. Save the **states** (POS tags) that were seen in training in the variable `self.states`. Both these variables will be used by the Viterbi algorithm in Section B.

Define an access function `elprob` for the emission model by filling in the function template provided.

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 using the same approach as in Question 1.

When using the training data for this step, add a start token `(<s>,<s>)` and an end token `(</s>,</s>)` to each sentence, so that the resulting matrix has useful transition probabilities for transitions from `<s>` to the *real* POS tags and from the real POS tags to `</s>`. For example, for this sentence from the training data:

```
[('Ask', 'VERB'), ('jail', 'NOUN'), ('deputies', 'NOUN')]
```

you would use this as part of creating the transition model:

```
[(<s>,<s>),('Ask', 'VERB'), ('jail', 'NOUN'), ('deputies', 'NOUN'),
 (</s>,</s>)]
```

Store the model in the variable `self.transition_PD`. This variable will be used by the Viterbi algorithm in Section B.

Define an access function `tlprob` for the transition model by filling in the function template provided.

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 the Jurafsky & Martin 3rd edition book in Appendix A[7] Figure A.9 in section A.4: use it as a guide for your implementation.

However you *will* need to *add* to J&M's algorithm code to make use of the transition probabilities to `</s>` which are also now in the *a* matrix.

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 (20 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. *5 Marks*

Note that per the instructions for Question 2 above, you will *not* need a separate π tabulation for the initialisation step, because we've included start state probabilities in the *a* matrix as transitions from `<s>` and you can use that.

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 access functions `get_viterbi_value` and `get_backpointer_value` so that your Viterbi implementation can be queried without the caller needing to know how you implemented your data structures.

Then, fill in the model construction and training parts of the test code in the `answers` function and check the results for plausibility.

Question 4a (35 Marks)

Implement the **recursion step** (*20 Marks*) 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. Describe your implementation with comments.

Note that as J&M don't include end-of-sentence handling, you will need separate code for the **termination step** after the loops (*5 Marks*), but as for the beginning-of-sentence case, you have the numbers you need in the transition model.

Finally, fill in the accuracy computation part of the test code in the `answers` function and check the results for plausibility (*2 Marks*).

[7] `http://web.stanford.edu/~jurafsky/slp3/A.pdf`

Section C: Short answer questions (25 Marks)

Question 4b (5 Marks)

Modify the accuracy code so it prints the first 10 tagged test sentences which did not match up with their ‘correct’ versions (taken from the tagged test data as supplied). Inspect these. Why do you think these have been tagged differently by the model? Pick *one* sentence and give a short answer discussing this. Be sure to consider the possibility that in some cases it’s the *corpus* tag which is ‘incorrect’.

Write down your answer in the `answer_question4b` function, returning the sentence as tagged by your model, the official ’correct’ version and your commentary.

You do *not* need to include code to choose the sentence pair, just copy-paste them from your output lists of 10 into `answer_question4b`.

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.

Question 6 (10 Marks)

Why else, besides the speedup already mentioned above, do you think we converted the original Brown Corpus tagset to the Universal tagset? What do you predict would happen if we hadn’t done that? Why?

Write down your answer in the `answer_question6` function.

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