Coursework 2 – Foundations of Natural Language Processing

Deadline: 12 noon, Monday 18th March 2024

Introduction

This coursework consists of four parts in which you demonstrate your understanding of fundamental concepts and methods of Natural Language Processing. The coursework contains open questions and programming tasks with Python using NLTK. This coursework is worth 12.5% of your final mark for the course. It is marked out of 100.

The files with the template code you need are on the LEARN page for the assignment (click on "Assessment" on the LHS menu, then "Coursework 2 - Hidden Markov Models"). You will download a file called `assignment2.tar.gz` which can be unpacked using the following command to a shell prompt:

```
tar -xf assignment2.tar.gz
```

This will create a directory `assignment2` which 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. If running `template.py` was throwing an error because it doesn't find some datasets, we recommend running `python download.py` to fix it. The code should run without errors on DICE (`nltk==3.4.1`)

There is an interim checker that runs some automatic tests that you can use to get partial feedback on your solution: https://sweb.inf.ed.ac.uk/ititov/fnlp_interim_2.cgi. Please note that, for the text questions (Question 2.3, Question 3.2, Question 4.1, Question 4.2, Question 4.3), the interim checker does not give any feedback as they are not submitted via the code file `template.py`.

Submission

Before submitting your assignment:

- Ensure that your code works on DICE. Your modified `template.py` should fully execute using python3 with or without the answer flag.
- Test your code thoroughly. If your code crashes when run, you may be awarded a mark of 0 for the code-based questions and you'll get no feedback other than "the code did not 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 word limits to open questions will be strictly enforced. We use the command `wc -w`, which is equivalent to `len(text.split())` in Python, to calculate the word count. If your answer exceeds the word limit, you may be awarded a mark of 0 and you will get no feedback other than "the answer exceeds the word limit".
- Note that we may use automated tools to check your code and/or your text answers for evidence of academic misconduct.

When you are ready to submit, rename your modified `template.py` with your matriculation number: e.g., `s1234567.py`. Then do the following:

1. Click on the "Gradescope CW2 Submission". This opens a page with two links: "Coursework 2 - Hidden Markov Models (programming)"; and "Coursework 2 - Hidden Markov Models (essay)".
2. Click "Coursework 2 - Hidden Markov Models (programming)" and upload your .py file.
3. Click "Coursework 2 - Hidden Markov Models (essay)" and write your answers to the text questions in the boxes provided (Question 2.3, Question 3.2, Question 4.1, Question 4.2, Question 4.3). When answering the questions on Gradescope, please refer to the question numbers in parentheses, which reflect the question numbers in this document. Notice that there is a "Total word count" question at the end of each question. Please provide the combined word count across all subquestions in the form of an integer.

If you have trouble submitting your .py file, then please refer to this blogpost: https://blogs.ed.ac.uk/ilts/2019/09/27/assignment-hand-ins-for-learn-guidance-for-students/

The deadline for submission is **12 noon, Monday 18th March 2023**.

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 afterwards. (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 time frame as for on-time submissions.

For information about late penalties and extension requests, see the School web page: http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests. Do not email any course staff directly about extension requests; you must follow the instructions on the web page.

Good Scholarly Practice: Please remember the School's requirements regarding all assessed work for credit. Details about this can be found at: http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct

Furthermore, you are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work on a public repository (e.g., GitHub), then you must set access permissions appropriately (for this coursework, that means only you should be able to access it).

1 Training a Hidden Markov Model (10 Marks)

In this part of the assignment you have to implement a Hidden Markov Model (HMM) class and train it for part-of-speech (POS) tagging. Look at the solutions from Lab 3, Exercise 3 and Exercise 4 for examples of the output of a POS-tagger.

You will need to create and train two models—an **Emission Model** and a **Transition Model** as described in lectures. Use tagged sentences from the 'news' part of the Brown corpus, with all the words converted to lowercase. 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.[1]). Having a smaller number of labels (states) will make Viterbi decoding faster.

We will 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.

Question 1.1 (5 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 (i.e. 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 +0.001 for smoothing and adding an extra bin (which will be used for all "unknown" tokens, i.e. tokens not seen in the training set).

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 2.

Define an access function elprob for the emission model by filling in the function template provided. If you want to compute the logarithm base 2 of a number p, use math.log(p,2).

Question 1.2 (5 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 but without the extra 'bin' (because we assume all POS tags are seen in training).

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 2.

Define an access function tlprob for the transition model by filling in the function template provided. If you want to compute the logarithm base 2 of a number p, use math.log(p,2).

[1] https://github.com/slavpetrov/universal-pos-tags

2 Implementing The Viterbi Algorithm (50 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[2] 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 Question 1.1 and the a probabilities correspond to the **transition model** implemented in Question 1.2. *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 2.1 (15 Marks)

Implement the **initialisation 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. (*4 Marks*)

Note that per the instructions for Question 1.2 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 (*7 Marks*) and the `backpointer` data structure (*4 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.

Note: in order to use the interim checker, you first need to complete the question below and fill in `tag_sentence`.

Question 2.2 (25 Marks)

Implement the **recursion step** (*15 Marks*) in the `tag` method. Reconstruct the tag sequence corresponding to the best path using the `backpointer` structure (*5 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 `tag_sentence`, which takes a sentence as input, initialises the HMM, runs the Viterbi algorithm and returns the tags. Don't forget to lower case the tokens and **don't** clear the memory of the viterbi and backpointer data structures because the automarking procedure will call `get_viterbi_value` and `get_backpointer_value`.

Performance (5 Marks)

We will test your implementation of the Viterbi algorithm to determine if it shows the expected runtime complexity. It should have a runtime complexity of $O(n \cdot T^2)$, where n is the length of the

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

sentence to be tagged and T is the size of the tagset. That is, we will not *not* evaluate based on the absolute runtime, as this also depends on the machine, but how it varies as a function of the input length and the size of the tagset.

Question 2.3 (5 Marks)

Modify `compute_acc` 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': such examples, with incorrect corpus tags, are *not* interesting for the purpose of this question.

Include in your answer the sentence as tagged by your model, the official 'correct' version and your commentary.

There is a 75 word limit on the answer to this question, *not* including the tagged sentences.

3 Expectation Maximisation (10 Marks)

Question 3.1 (5 Marks)

Sometimes we don't have a lot of labeled training data but we do have access to unlabeled data. In such a scenario, we can use semi-supervised methods. Here we simulate such a scenario with only 20 sentences labeled with POS tags and the remaining sentences from the training set as unlabeled data. You are going to apply the "hard EM" algorithm (see Lecture 8). In contrast to the real EM algorithm you will not use soft counts but hard counts; this makes it much easier to implement. Use the following pseudocode as guidance to implement `hard_em`:

Require: Labeled data L, unlabeled data U, number of iterations k.

```
1: procedure SEMI-SUPERVISED HARD EM(L, U, k)
2:     T_0 ← train HMM on labeled data L
3:     for i ∈ ⟨0, ..., k − 1⟩ do
4:         Let P_{i+1} be U labelled according to T_i
5:         T_{i+1} ← HMM trained on L and P_{i+1}
```

Question 3.2 (5 Marks)

Below is a sentence from the corpus with the gold POS tags it was annotated with, the most likely tags according to the model which was trained on the 20 sentences (T_0) and the most likely tags according to a model trained with hard EM for 3 iterations (T_3):

Sentence	In	fact	he	seemed	delighted	to	get	rid	of	them	.
Gold POS	ADP	NOUN	PRON	VERB	VERB	PRT	VERB	ADJ	ADP	PRON	.
T_0	PRON	VERB	NUM	ADP	ADJ	PRT	VERB	NUM	ADP	PRON	.
T_3	PRON	VERB	PRON	VERB	ADJ	PRT	VERB	NUM	ADP	NOUN	.

1. T_0 erroneously tagged "he" as "NUM" and T_3 correctly identifies it as "PRON". Speculate why additional unlabeled data might have helped in that case. Refer to the training data (inspect the 20 sentences!).
2. Where does T_3 mislabel a word but T_0 is correct? Why do you think hard EM went wrong in that case?

There is a 150 word limit for this question.

4 Text questions (30 Marks)

Question 4.1 (5 Marks)

You are tasked with overseeing a Machine Translation model between English and a low-resource language. Explain and justify what translation you expect to be better: going from the low-resource language to English or the other way around.

There is a 100 word limit for this question.

Question 4.2 (10 Marks)

Explain the challenges associated with using synonym words in a Naive Bayes classifier. Given a set of synonym words S, suggest a modification to enhance the Naive Bayes model's handling of synonyms. Evaluate the potential drawbacks of these adjustments. Conclude by explaining how a logistic regression model addresses the issue of synonyms.

There is a 200 word limit on the answer to this question.

Question 4.3 (15 Marks)

In his 1950 paper *Computing Machinery and Intelligence*, Alan Turing addressed the question of whether a machine would be able to think. However, he deemed this question as absurd, and proposed to instead consider a framework in which a machine would be challenged to pretend to be a human through a text-only channel. While he mentioned the difficulty of this Imitation Game

> The game may perhaps be criticised on the ground that the odds are weighted too heavily against the machine. If the man were to try and pretend to be the machine he would clearly make a very poor showing.

he also predicted that future machines would be able to perform well in his test

> I believe that in about fifty years' time it will be possible, to programme computers, [...] to make them play the imitation game so well that an average interrogator will not have more than 70 per cent chance of making the right identification after five minutes of questioning.

Imagine that you own a Language Model which has the potential to generate human-like text. Discuss in detail different methods or strategies that could be implemented to detect text as generated by your Language Model.

There is a 400 word limit on the answer to this question.