Programming Assignment 4: Tagging with HMMs

Due Apr 12 by 10pm **Points** 100 **Available** Mar 22 at 12pm - Apr 19 at 10pm 28 days

This assignment was locked Apr 19 at 10pm.

Released: March 22, 2021

Table of Contents.

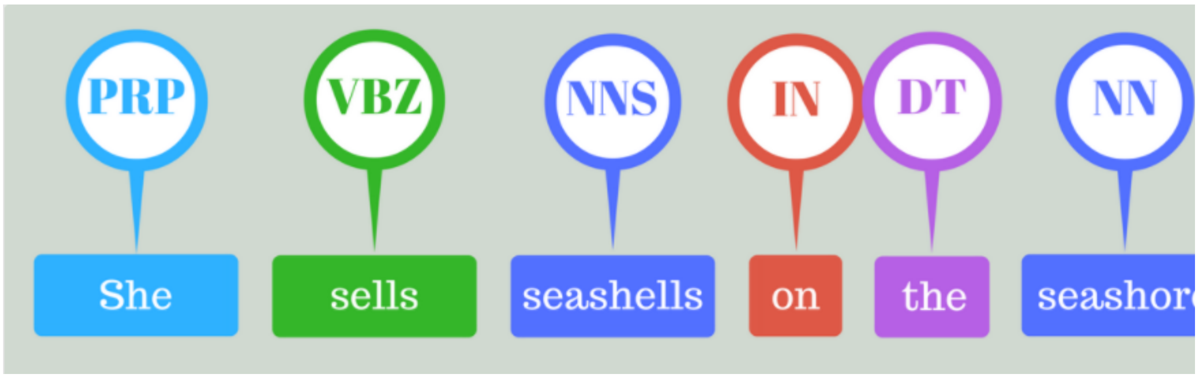

Fig 1. One Example of Part-of-Speech Tags for the tongue twister "She sells seashells on t seashore"

Warning (Please read this)

As ever, we are aware that solutions or nearly-solutions to this problem may exist online. **Do not use outside solutions as this would be plagiarism.** To earn marks on this assignment you must develop your own solutions, and to ensure that everybody is meeting the University's academic integrity standards, we will be running software similarity checks on all submissions. So please, do your own work.

Also please consider the following points, as you did when implementing prior assignments:

- **Do not add any non-standard imports in the python files you submit** (all imports already in the starter code must remain). All imports that are available on teach.cs are considered to be standard.

- **Make certain that your code runs on teach.cs using python3**. You should all have an account on teach.cs and you can log in, download all of your code (including all of the supplied code) to a subdirectory of your home directory, and use the command `python3 autograder.py` and test it there before you submit. Your code will be graded by running it on teach.cs, so the fact that it runs on your own system but not on teach is not a legitimate reason for a regrade.
- The test cases used in the autograder are a simple example of the test cases that we will use during marking. These test cases are meant as a sanity check to ensure that your HMM is working, but the autograder we use will be run on

much larger training and test files.

- We will also look for certain things in the assignments (e.g., running them through software plagiarism checkers, looking at assignments that fail all tests, etc.). If we have **good reasons** we will change your grade from that given by the autograder either up or down.

Introduction

Natural Language Processing (NLP) is a subset of AI that focuses on the understanding and generation of written and spoken language. This involves a series of tasks from low-level speech recognition on audio signals up to high-level semantic understanding and inferencing on the parsed sentences.

One task within this spectrum is Part-Of-Speech (POS) tagging. Every word and punctuation symbol is understood to have a syntactic role in its sentence, such as nouns (denoting people, places or things), verbs (denoting actions), adjectives (which describe nouns) and adverbs (which describe verbs), just to name a few. Each word in a piece of text is therefore associated with a part-of-speech tag (usually assigned by hand), where the total number of tags can depend on the organization tagging the text.

For this assignment, we will be using a subset of the British National Corpus (the word "corpus" is Latin for "body", in this case a body of literary works). This corpus contains a variety of texts across science, art and literature, with every word tagged (usually by hand) with the associated part-of-speech. More information about the texts in the British National Corpus (including word frequencies, domains and authour info) can be found **here (http://www.natcorp.ox.ac.uk/docs/URG.xml)** .

What You Need To Do:

Your task for this assignment is to create an Hidden Markov Model whose input is a file containing untagged text and whose output is a file containing each word from the input file, along with the appropriate POS tag. More information about the expected format of these tagged and untagged files is described in more detail in the section **Training File Representation**.

To do this task, your code will need to train the probability tables (initial, transition and emission) for your HMM before it can do the tagging. To train your HMM, it will take in a series of tagged *training files* as the input for the training stage. Once the probability tables have been built, your final grade for this assignment will depend on the accuracy of the tags that your HMM assigns to the untagged file (also known as the *test file*).

The Challenge Of This Assignment:

While the task is simple to describe, the challenge increases as your model's accuracy increases. Achieving over 90% accuracy is very good; getting 100% accuracy is practically impossible. Therein lies the challenge. You can create a completely naïve tagger using just the emission probability table and still pass the assignment. Or you can implement the HMM and improve your accuracy significantly. There are no restrictions to where you go from there though. While we teach you the basics about HMMs, Bayes Nets and the associated algorithms, the only restriction on your submission is that your POS tagger needs to a) take in a untagged file and b) return a tagged one, given a set of training files. What you do and how you do it is entirely up to you.

Starter Code

Update (April 2, 2021): Responding to issues that were raised with the training and test files, these files have been updated in the starter code to a) fix non-standard characters and b) remove stray XML code that remained from the file creation process. Your HMMs should work even better now :)

The starter code contains one Python starter file and a number of training and test files. You can download all the code and supporting files as a **zipped file archive** **(https://q.utoronto.ca/courses/197877/files/13700731/download?download_frd=1)** . In that archive you will find the following files:

Project File (the file you will edit and submit on Markus):

`tagger.py`	The file where you will implement your POS tagger; this is the only file to be submitted and graded.

Training Files (look, but don't modify):

`training1.txt - training10.txt`	Training files, containing large texts with part-of-speech tags on each word.
`test1.txt - test10.txt`	Test files, identical to the training files but without the part-of-speech tags.

Autograder Files (the files used to sanity test your HMM):

`autograder.py`	The autograding script that will tell you how accurate your POS tagger is.
`autotraining.txt`	The training file containing the word and POS tags for a sample training text.
`autotest.txt`	The test file with untagged words.
`autosolution.txt`	The test file with tagged words.

Note that the autograder expects that it is located in the same directory as the files `autotraining.txt`, `autotest.txt` and `autosolution.txt`. Also note the earlier caveat about the autograder, that it is meant as a sanity check to see how your HMM performs on a simple sample text. Performing 100% on the autograder means that your HMM is not broken, but only the larger training and test files will give you an idea of your overall performance.

With this in mind, feel free to modify the autograder to use other files for your own training and testing.

Running the Code

You can run the POS tagger by typing the following at a command line:

```
$ python3 tagger.py -d <training file names> -t <test file name> -o <output file name>
```

where the parameters consist of:

- a series of training files (pairings between words and POS tags),
- a single input test file, and
- a single output file (containing the words from the test file, paired with each word's POS tag).

Here is an example of what this might look like to train on three files (`trainingA.txt`, `trainingB.txt`, `trainingC.txt`) and create an output file called `output.txt` based on the words in the input file `test.txt`:

```
$ python3 tagger.py -d trainingA.txt trainingB.txt trainingC.txt -t test.txt -o output.txt
```

Training File Representation

Each line in a training file consists of a single word and POS tag, separated by a single space, colon character and a second space. A sample training file would contain something like the following:

```
Detective : NP0
Chief : NP0
Inspector : NP0
John : NP0
McLeish : NP0
gazed : VVD
doubtfully : AV0
at : PRP
the : AT0
plate : NN1
before : PRP
him : PNP
. : PUN
Having : VHG
thought : VVN
he : PNP
was : VBD
hungry : AJ0
, : PUN
he : PNP
now : AV0
realized : VVD
that : CJT
actually : AV0
```

In addition to being the format for the training file, this is also the expected format for the output of your POS tagger. So if you run your tagger on the file `test1.txt` that we provide, the optimal output should be a file identical to `training1.txt`.

The Part-Of-Speech Tags

While this task falls under the domain of Natural Language Processing, having prior language experience doesn't offer any particular advantage. In the end, the main task is to create a model that can figure out a sequence of underlying states, given a sequence of observations.

With that being said, it's useful to know that there are 57 basic tags and 4 punctuation tags in the tag set, along with 30 "ambiguity tags" (an ambiguity tag is a combination of two tags like AJ0-NN1 where the people conducting the tagging couldn't decide between the two tags). To find out more about the names of these tags and what they mean, consult the **Guidelines to Wordclass Tagging** (http://www.natcorp.ox.ac.uk/docs/URG.xml?ID=posGuide#guidelines) provided by the British National Corpus.

Making the POS Tagger

Creating a HMM for POS tagging involves the creation of the conditional probability tables that define it, and then using these tables to calculate the most likely tags for a given sequence of words. While it's up to you how to implement this, here are a few suggestions to inform your implementation.

Naïve Tagger

If you're running out of time and have other assignments to get through, a naïve tagger only uses emission probabilities and doesn't use transition probabilities at all. The way it decides on the tag for a word is to choose the most frequent POS tag seen for this word during training.

Hard to say what the accuracy for this will be. 50%...maybe higher? This approach will have difficulty making the right call when a word has two possible interpretations or when it encounters a word that has never been seen before. All you can do is choose a random tag in that case, or improve your performance by considering the context of the word, namely....

HMM Tagger

The Hidden Markov Model tagger also considers the initial probability table as well as the transition probability table when deciding on a part of speech tag, in addition to the emission probability table. The probability values for these tables are deciding on a part-of-speech tag, in addition to the emission probability table. The probability values for these tables are calculated during training. The Viterbi algorithm (or your own variation on it) is used to calculate the most likely sequence of states, given the sequence of words.

This approach will take more time to run but will increase the accuracy of the tagging.

Enhancements

What will you do when the basic HMM implementation reaches its limits of performance? What can you do to speed up the tagging process? This is where you can investigate extensions to the basic HMM, such as:

- Segmenting the training and test text into sentences,
- Implementing trigram models,
- Preloading your CPT with default starting values,
- investigating new ideas like Maximum Entropy Markov Models (MEMMs) that train on the features of a word,
- Whatever crazy idea you have that will fix those mislabeled words.

This is the last assignment of the course, so all we're doing here is providing you with the task to perform. You have free rein to design or hack whatever solution you'd like.

Mark Breakdown

Your performance on the tagging task will determine your mark for this assignment:

- If you get 90% accuracy on the test file, your mark for this assignment will be 100%.
- If your accuracy is less than 90%, we will divide your accuracy by 90% to determine the mark you receive.
  - If you achieve 72% accuracy, your mark will be (72/90) = 80%

What training and test sets will be used in the grading?

We provide you with 10 training files and 10 corresponding test files. Our grading script will provide your tagger with 3-4 of these training files and one test file from this set. We want you to have some idea of what to expect, which is why we will limit our training and testing to the data files provided.

What to Submit

You will be using MarkUs to submit your assignment. You will submit one file:

- Your modified `tagger.py`

Make sure to document the code to help our TAs understand the approach you used. In particular, if you decided to implement enhancements beyond the basic HMM/Viterbi algorithm, make sure to document that in the comments at the top of your solution.

The assignment is due on **April 12 at 10pm!**

GOOD LUCK!