Programming Assignment 4: Tagging with HMMs

Due Monday by 10pm **Points** 100 **Available** Jul 26 at 12pm - Aug 16 at 10pm 21 days

Released: July 26, 2021

Table of Contents.

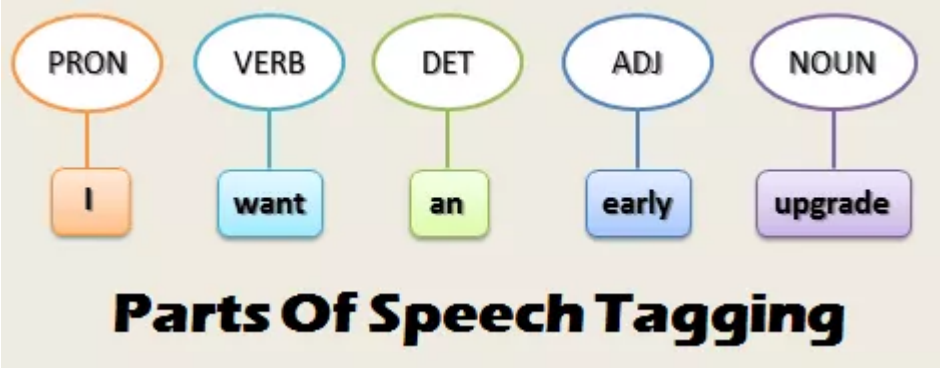

Fig 1. One Example of Part-of-Speech Tags for the sentence "I want an early upgrade"

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 grader.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 grader.py are a simple example of the test cases which is meant as a sanity check to ensure that your HMM is working, but we will use a different grader and

training/testing files during our grading.
- 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 graderT 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.

While this task falls under the domain of NLP, having prior language experience doesn't offer any particular advantage. To minimize the requirement of the prior knowledge for NLP, we use a very simple set of POS tags: {"VERB" , "NOUN" , "PRON" , "ADJ" , "ADV" , "ADP" , "CONJ" , "DET" , "NUM" , "PRT" , "X" , "."}. In the end, the main task is to create a HMM model that can figure out a sequence of underlying states, given a sequence of observations.

What You Need To Do:

Your task for this assignment is to create a Hidden Markov Model (HMM) for POS tagging including：

1. Training probability tables (i.e., initial, transition and emission) for HMM from training files containing text-tag pairs
2. Doing inference with your trained HMM to predict appropriate POS tags for untagged text.

Your solution will be graded based on the learned probability tables and the accuracy on our (private) test files, as well as the computational efficiency of your algorithm. See **Mark Breakdown** for more details.

Starter Code

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 zip file **starter-code.zip**. 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.

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

`data/train-public.txt`	Containing large texts with POS tags on each word.
`data/train-public.ind`	Containing starting indices for each sentence of the training text.

Public Testing Files (look, but don't modify):

`data/test-public-{small/large}.txt`	Test files, identical to the training files but without the POS tags. The version with a large text is used for **testing the efficiency** of your solution.
`data/test-public-{small/large}.ind`	Identical to those for training.
`data/test-public-{small/large}.soln`	The ground-truth POS labels for the testing files.

Grader Files (the files used to sanity test your solution):

`grader.py`	The public autograding script for testing your solution with provided training/testing files.

Running the Code

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

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

where the parameters consist of:

- a training file name **without suffix**
- a test file name **without suffix**

The script will read `<training file name>.txt` and `<training file name>.ind` for training, and similarly for testing. The test output (POS predictions) will be written to `<test file name>.pred` automatically. Here is an example:

```
$ python3 tagger.py -d data/train-public -t data/test-public-small
```

Making HMM Tagger

Creating a HMM for POS tagging involves the creation of probability tables (i.e., initial, transition, emission) that define it which are calculated during training. Then these tables are used to calculate the most likely tags for a given sequence of words with inference algorithms like the Viterbi algorithm (or your own variation). You are required to split the training and inference procedure in `tagger.py`. In particular:

- In `train_HMM`, the input is the training file name and it is expected to return 3 objects, the **prior and transition distributions which will both be tables** and an **emission distribution which will be a dictionary**. Detailed specifications can be found in its docstring.
- In the function `tag`, you are expected to call `train_HMM` and then use the trained model to do POS prediction on test data.

It is recommended that during training/testing, the whole text file be split into sentences using the provided sentence indices (in the `*.ind` files) for training and prediction.

Mark Breakdown

Your mark for this assignment will be split as:

- Correctness of the training procedure (42%): this is based on the learned HMM probability tables (14% for each).
- POS prediction accuracy on a **small** test file (35%)
- POS prediction accuracy on a **large** test file (23%): this will test the computational efficiency of your solution. You may lose these marks if your code runs out of time during our grading.

To get full marks on any single accuracy test we want to see your code predict correct tags at least 85% of the time. If your accuracy is less than 85%, the mark you receive will reflect your accuracy. For example, if you achieve 72% accuracy on a test worth 10 points you will get (72/10) = 7.2 points on that test.

Sanity Check Your Solution

We provide a grader file for sanity checking your solution which you can run with:

```
$ python3 grader.py
```

This code is only meant as a sanity check to see how your HMM performs; getting 100% on the grader means that your HMM is not broken. But it does not mean you will get 100% as your final mark! You will want to create tests of your own that vary training/testing files to see how your code generalizes.

Different training and testing files than those used in the grader.py file will be used to grade your solution.

What to Submit

You will be using MarkUs to submit your assignment. You will submit only one file: your modified `tagger.py`.

Make sure to document the code to help our TAs understand the approach you used.

The assignment is due on **August 16 (last day of class) at 10pm!**

GOOD LUCK!