Assignment 4: HMM Part-of-Speech Tagging

- **Due** Thursday by 11:59p.m.
- **Points** 100
- **Available** after Nov 14 at 9a.m.

starter-code-1.zipDownload starter-code-1.zip

Document History

- Released Nov 14 2022
- Updated starter-code.zip: removed <w c5="AV0" hw="rather" pos="ADV"> and <w c5="AT0" hw="a" pos="ART"> from training1.txt in the validation folder. Also fixed tagger_validate.py. Switched <output[index]> and <solution[index]> in the print statements.

Table of Contents.

Fig 1. One Example of Part-of-Speech Tags for

Warning (Please read this)

Making assignments requires us to strike a balance between being novel and being practical. With that in mind, we are aware that solutions or near-solutions to this particular task may exist online. **Do not use outside solutions as this would be plagiarism, which is an academic offense.** To earn marks on this assignment, you must develop your own solutions. To ensure that everybody meets the University's academic integrity standards, we will run 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 non-standard imports in the python files you submit** (all imports already in the starter code must remain). If you have a question about an import, ask on Piazza.
- **Make certain that your code runs using python3**. As with past assignments, we have included a validation program for testing your code before submitting it. Your code will be run and tested on the teach.cs machines, so the fact that it runs on your own system but not others is not a legitimate reason for a remark request.
- The test cases provided with the validation file are meant to ensure that your HMM is compiling and producing the expected output. As always, we will be using a more sophisticated autotest program and additional training/testing files during our grading.

Overview

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), 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.

A list of all the part-of-speech tags can be found here.

While this task falls under the domain of NLP, having prior language experience doesn't offer any particular advantage. In the end, the main task is to create an 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. Performing 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 test files, as well as the efficiency of your algorithm. See Mark Breakdown for more details.

Starter Code & Validation Program

The starter code contains one Python starter file, a validation program and several training and test files. You can download 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 t and graded.

Training Files (don't modify):

`data/training1.txt - training5.txt`	Training files (in text format) containing large texts with POS tags on eac

Testing Files (don't modify):

`data/test1.txt - test5.txt`	Test files (in text format), identical to the training files but without the PO

Validation Files:

`validation/tagger-validate.py`	The public validation script for testing your solution with a set of provide files.

Running the Code

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

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

where the parameters consist of:

- one or more training file names, separated by a spaces

- a single test file name
- a single output file name

The script will read in `<training files>` for training, and `<test file>` for testing. The test output (POS predictions) will be written to `<output file>`. Here is an example:

```
$ python3 tagger.py -d data/training1.txt data/training2.txt -t data/test1.txt -o dat
a/output1.txt
```

Making the HMM Tagger

In the hidden Markov model, the hidden variables represent the POS tags, and the evidence variables represent the words in the sentences.

During the **training** phase, you will need to learn three probability tables described below:

- The initial probabilities over the POS tags (how likely each POS tag appears at the beginning of a sentence)
- The transition probabilities from one POS tag to another.
- The emission probabilities from each POS tag to each observed word.

Learning these probabilities will require you to perform counting. For instance, to determine the initial probability for the POS tag `NP0`, you need to divide the number of times `NP0` begins a sentence by the total number of sentences. How you store these transitions and their associated probabilities is up to you. Be aware of the storage and performance tradeoffs associated with each design choice.

During the **test** phase, you will use these probability tables to calculate the most likely tag for each word in a sentence using inference algorithms like the Viterbi algorithm (or your own variation).

You are required to split the training and test procedures in `tagger.py`.

You can assume independence between sentences for the purpose of this assignment. As a result, you can split the text files into individual sentences during training and testing without losing information in the training and prediction process.

Mark Breakdown

The majority of your mark for this assignment will be calculated based on the accuracy of your Part-of-Speech tagger:

Marking Scheme

Component	Weight
Correctness (% words tagged correctly)	90%
Heuristic file	10%

As shown in this table, 10% of your mark for this assignment is earned by sharing the interesting heuristics or techniques that you used to enhance the performance of your HMM. These are meant to be part of your solution and described in a file called `heuristics.pdf`, which you will submit to Markus along with your `tagger.py` file. This mark is assigned separately from your correctness mark (i.e. you can get full marks for your heuristics even if you don't get full correctness marks).

To get full marks on our correctness tests, we want to see your code predict correct tags at least 90% of the time. If your accuracy is less than 90%, your correctness mark will be calculated as (your accuracy / 90). For example, if you achieve 72% accuracy on a test worth 10 points, you will get (72/90)*10 = 8 points on that test.

Our runtime expectations will be ~5 minutes for a single training & test file but will be relaxed for cases involving multiple training files. That being said, be mindful of the space and runtime performance of your data structures.

Validating Your Solution

As always, we have provided a validation script that you can use to check your solution before submitting it. Remember that this code is only meant to confirm that it runs as specified and gives you a basic idea of how your HMM performs. Getting 100% on the validation test only means that your HMM is not broken. It does not mean you are guaranteed 100% as your final mark. You will want to create tests of your own that vary the combination of training and testing files to see how your code generalizes. While we will use some of the same training and test files as provided to you here, there will also be unseen training and test files used to grade your solution.

Help! My HMM Isn't Working!

If you've been reviewing the slides on HMMs and the Viterbi algorithm and can't seem to get things to work, don't panic. In the end, your mark for this assignment is based on the accuracy of your part-of-speech tagger, not whether you complete the Viterbi algorithm exactly right. As with past assignments, some of the intermediate stages to Viterbi can also perform the tagging task in a less accurate but more straightforward way.

For instance, you could do the following:

- Use the emission probability table to label each word with the most likely POS tag, ignoring the context of the words around it.

- Use the transition probability table to calculate the most likely POS tag for the current word, given the POS tag for the previous word.

Viterbi is just the more advanced form of this approach, calculating the likelihood of each POS label for the current word, given the previous sequence of words in the sentence. But if you're getting stuck and running out of time, the alternatives above will serve in a pinch.

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!