Project Option 2: Hidden Markov Models and POS Tagging

CSE 473: Introduction to Artificial Intelligence
The University of Washington, Seattle, Autumn 2020

Due Friday, December 4, via Gradescope by 11:59 PM
You can choose to do this assignment up to group of 2 students. If you do this assignment in a group of 2, make sure when one of you submits to Gradescope, you tag your partner's name as a group member so both of you receive due credit.
Choice of project option and partner due by midnight, November 27 (via Canvas "quiz" -- counts for up to 3 points of participation credit to submit this on time).

Overview

Part of understanding natural language often involves "part-of-speech" (POS) tagging: classifying words into groups according to their syntactic role. In this project option we'll take advantage of the POS tagging problem to gain a detailed look at the Hidden Markov Model structure and a couple of algorithms that work with it: the Forward algorithm and the Viterbi algorithm.

A graphical visualizer for HMMs is provided with the starter code. We'll also be using a corpus of Twitter data that has already been tagged.

This project may be done individually or in partnerships of 2.

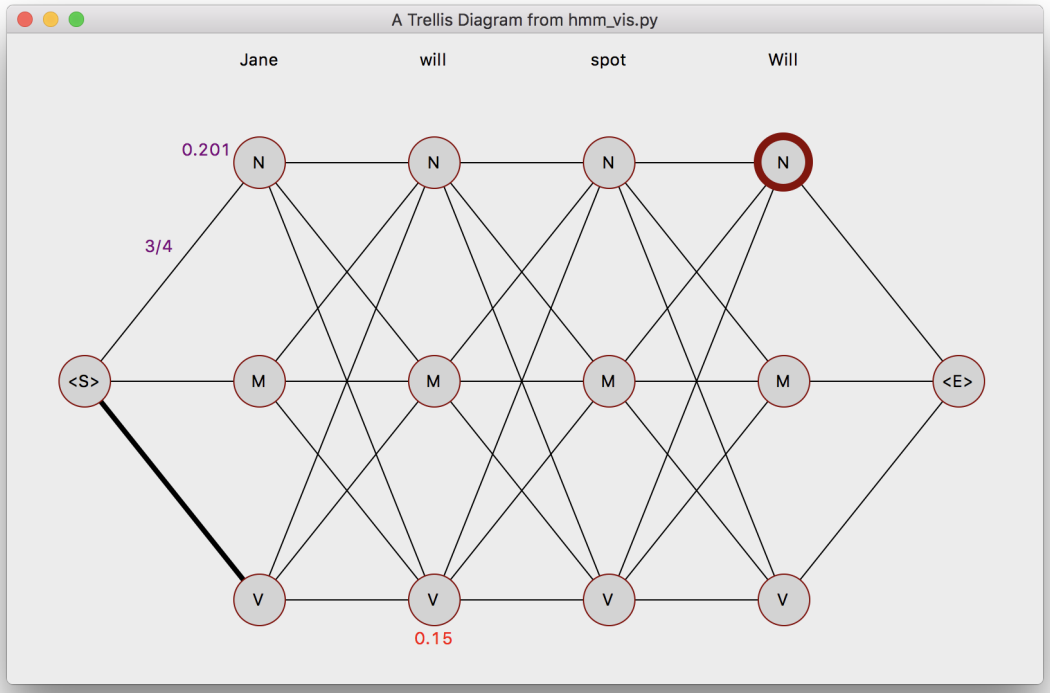
Note: This project option requires the use of the Python graphics module known as tkinter, but the version compatible with Python 3.x.

How to Get Started

Download the starter code.

If you are using the command line, all files can be run as Python scripts (e.g., **`python hmm_vis.py`**).

If you are using IDLE, start up the IDLE integrated development environment that ships with the standard distribution of Python from Python.org. In IDLE, open the file **`hmm_vis.py`**. Then, using the Run menu on this file's window, try running the starter code by selecting "Run module F5". This will import a file **`hmm_vis.py`** and start up a graphical display for a simple HMM, assuming that you are running the standard distribution of Python 3.6, 3.7, or 3.8 which includes the Tkinter user-interface technology. You should see a window that looks like this:



PART A: Implement the Forward algorithm and the Viterbi algorithm (50 points).

For Part A, you will implement two HMM algorithms by filling out the template code provided for you in **`hmm.py`**. Your functions will need access to the HMM parameters. These are loaded into the HMM class from a saved JSON file by either specifying the JSON file in the call to the HMM constructor or by calling the **`load_parameters`** function. See the bottom of **`hmm.py`** in the starter code for an example of how this is done.

Your functions will also need to visualize the results of your algorithm using a trellis diagram by calling functions in **`hmm_vis.py`**. Take a look at the documentation at the top of **`hmm_vis.py`** to see which functions are available for you to use, and look at the bottom of the files **`hmm_vis.py`** and **`hmm.py`** for examples of how to visualize the trellis diagram and highlight certain nodes and edges. You can run each of these files as a Python script to generate example trellis diagrams.

1. **`forward_algorithm(obs_sequence, show=False)`**

   Implement the Forward algorithm. Here **`obs_sequence`** is a list of elements from the set of possible observations. Using the current HMM parameters (set of states, P_trans, P_emission), run the Forward algorithm for n steps where n is the length of **`obs_sequence`**. In this assignment, this function should return a list of lists of probability values, having the same length as **`obs_sequence`**. The i'th list should give the belief vector for the state of the system at time i, taking into account all observations up to and including the one at step i. If **`show`** is True, then the graphical display should be updated as the algorithm progresses, and the resulting belief values should be shown on the display next to their associated nodes. Note that there should not be probability values associated with any special start and end states (like '⟨S⟩' and '⟨E⟩').

2. **`viterbi_algorithm(obs_sequence, show=False)`**

Implement the Viterbi algorithm. This function is like the Forward algorithm one, except that it should return only a list of states representing the most likely sequence of states given the observations made. If **`show`** is True, the sequence of states and the corresponding edges should be highlighted in the display, and the relevant scores should be shown next to the nodes.

When implementing the Viterbi algorithm, it is common to use logarithms of probabilities and add them to correspond to multiplying probability values. Since the main result of the Viterbi algorithm is a most probable state sequence, it can be obtained as the argmax either of the computed probabilities of the sequences (jointly with the observation sequence) or as the argmax of the corresponding scores. Logarithms tend to be more numerically stable for computing the kinds of values needed than the probability values themselves, which can quickly become so small in magnitude that accuracy suffers - sometimes catastrophically.

However, the use of logarithms is NOT required. Due to the short item sequences in Tweets, we do not expect you to experience serious numerical underflow, but be aware that this might be an issue if you experience problems applying your implemented code to longer sequences.

PART B: Develop and Test an HMM for Twitter POS Tagging (50 points)

For Part B, you will generate an HMM transition model and emission model from Twitter data in a training set, and then use the resulting HMM to find most-likely POS sequences of tweets in a test set.

There are three data files for your use in Part B. Your HMM transition and emission models should be generated using the data in the file **`twt.train.json`**. The file **`twt.dev.json`** is intended for your use in debugging as you develop program for building the HMM model. It is only 10 percent as large as **`twt.train.json`**, the file used to actually create your HMM transition model and emission model. Finally, the file **`twt.test.json`** contains the examples you should use to evaluate the accuracy of your resulting HMM.

For this part of the assignment, you will need to read data from JSON files as well as write out HMM parameters to a JSON file. You can learn more about formatting Python objects in JSON format here. To learn more about reading from and writing to files in Python, you can look at the documentation here or refer to the provided reference for Assignment 0 called *Python as a Second Language*.

- Develop the model by writing code in a file named **`build_twitter_hmm.py`** that reads in the training data and processes it, writing out a file called **`twitter_pos_hmm.json`**. After running **`build_twitter_hmm.py`**, the **`twitter_pos_hmm.json`** file should contain the HMM states, possible emissions, transition model, and emission model similar to the example file **`toy_pos_tagger.json`** that we have provided for you. The following YouTube video may be helpful in understanding how to build your transition model from data: POS Tagging (Luis Serrano).

- Create the transition model.

  The state space is a 25-element set of tags in the training data. Base your probabilities on counts of "bigrams". For example, if your model-builder sees a ["fun", "N"] item immediately followed by a ["with", "P"] item in the Twitter training data, then this counts 1

towards the number of occurrences of the POS bigram ("N", "P") (which means noun followed by preposition). Express the transition model as a **`dict`** object, as demonstrated in **`toy_pos_tagger.json`** in the starter code.

- Create the emission model.

  First set up your code to count the number of occurrences of each word in the training data, and sort your observation set into a sequence in order of decreasing frequency of occurrence. You can print out the first 50 elements of this sequence in order as a sanity check, to see if they correspond to words that might be most common.

  For each (word | state) pair, compute P(word | state). If P(word, state) = 0, you can ignore the (word, state) entry and simply treat the probability P(word | state) as zero by default, when looking up probabilities. Put the results into a dictionary. Again, see the starter code file **`toy_pos_tagger.json`** for an example of this.

  The transition model is a dictionary whose keys are states, and each value is a dictionary with states as keys and probabilities as values. The emission model is also a dictionary whose keys are states, with each value being a dictionary with emissions as keys and probabilities as values. Both the transition model and emission model should be written out to the file **`twitter_pos_hmm.json`** as the values of P_trans and P_emission.

  The set of states for the POS tagging of Twitter data is a list S where the elements of the list are the POS tags, in the order given here:

  ```
  N  O  S  ^  Z  L  M  V  A  R  !  D  P  &  T  X  Y  #  @  ~  U  E  $  ,  G
  ```

  In Python, this list is as follows:

  ```
  S = ['N', 'O', 'S', '^', 'Z', 'L', 'M', 'V', 'A', 'R', '!', 'D',\
       'P', '&', 'T', 'X', 'Y', '#', '@', '~', 'U', 'E', '$', ',', 'G']
  ```

- Create a Python script **`run_pos_test.py`** that prints the test set accuracy of POS tagging using the HMM you generated from the training data. First, compute the total count of correctly-tagged words and the total count of all words over all test set tweets. Then, use these values to compute the percentage of correctly-tagged words across the entire test set. Your script should output these values in the following format:

  ```
  Total count of correctly-tagged words in the test set: <print value here>
  Total count of words in the test set: <print value here>
  Test set accuracy: <print value here, as a percentage out of 100>
  ```

  To write this script, you will need to load the HMM parameters you saved in the **`twitter_pos_hmm.json`** file into an HMM model. See the bottom of **`hmm.py`** in the starter code for how to do this.

- Use the GUI to demonstrate the POS tagging of one of the test data tweets by your HMM model. Turn in a PNG or JPG image of a trellis diagram showing the results of POS tagging via the Viterbi algorithm of an example tweet having at least 4 items (e.g., words) but not more than 7 items. You could use the first tweet in the test set (having 4 items) or some other, more interesting tweet from the test set. Your trellis diagram should highlight the most likely sequence of nodes (corresponding to the POS tags) as well as the corresponding

edges between nodes.

- Create a plain text document **`Part_B_Report.txt`** with the following elements (you can copy/paste the text below and fill in the specified parts). Note that numbers 2, 3, and 4 will be the values printed by the **`run_pos_test.py`** script.

```
1. <your name here>
2. Total count of correctly tagged words in the test data: <your answer here>
3. Total count of words in the test data: <your answer here>
4. Percentage correct (calculated from the previous two values): <your answer here>
5. Something you learned doing this assignment:
   <your answer here>
6. Biggest challenge you faced doing this assignment:
   <your answer here>
```

Note: There is no autograder at this time. However, the visualization facility in **`hmm_vis.py`** and the demo HMM parameters in **`toy_pos_tagger.json`** are provided to scaffold your development of code for this assignment. Using the visualization techniques, you should be able to figure out quickly not only whether your code is doing the right thing, but if not, where it is going wrong.

The tagged Twitter data used in this assignment comes from research done at Carnegie-Mellon University described in the paper *Part-of-Speech Tagging for Twitter: Annotation, Features, and Experiments* (Gimpel et al.). This was published in the Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL) in 2011 (the last author on the paper, Noah Smith, is now a professor in the Allen School).

What to Turn In

For Part A, turn in your file **`hmm.py`**. Do not turn in any of the other starter code files.

For Part B, turn in your code files **`build_twitter_hmm.py`** and **`run_pos_test.py`**, and your saved HMM parameters file **`twitter_pos_hmm.json`**. In addition, turn in your **`Part_B_Report.txt`** report file. Finally, turn in a screen shot image, in PNG or JPG format, of a trellis diagram that shows the results of the Viterbi algorithm run on one of the tweets from the test data set. Remember, this sample tweet should have between 4 and 7 words in it -- not trivially short, but not so long as to require a very wide image. The most probable state sequence and the edges on that path should be highlighted. Name your image file **`Sample_Tweet_POS_Trellis.png`** or **`Sample_Tweet_POS_Trellis.jpg`**.

Updates and Corrections

If necessary, updates and corrections will be posted here and/or mentioned in class or on Ed.