CMPSC 442: Homework 6 [100 points]

Release Date	Tuesday, April 6, 2021, 12:00 am
Due Date	Tuesday, April 25, 2021, 11:59 pm

TO SUBMIT HOMEWORK

To submit homework for a given homework assignment:

1. You *must* download the homework template file from Canvas, located in Files/Homework Templates and Pdfs, and modify this file to complete your homework. Each template file is a python file that will give you a headstart in creating your homework python script. For a given homework number N, the template file name is homeworkN_cmpsc442.py. The template for homework #6 is homework6_cmpsc442.py. IF YOU DO NOT USE THE CORRECT TEMPLATE FILE, YOUR HOMEWORK CANNOT BE GRADED AND YOU WILL RECEIVE A ZERO. There is also a text file in the Canvas assignment page that you will need, called:
   - brown-corpus.txt
2. You *must* rename the file by replacing the file root using your PSU id that consists of your initials followed by digits. This is the same as the part of your PSU email that precedes the "@" sign. For example, your instructor's email is rjp49@cse.psu.edu, and her PSU id is rjp49. Your homework files for every assignment will have the same name, e.g., rjp49.py. IF YOU DO NOT RENAME YOUR HOMEWORK FILE CORRECTLY, IT WILL NOT BE GRADED AND YOU WILL RECEIVE A ZERO. Do not be alarmed if you upload a revision, and it is renamed to include a numeric index, e.g., rjp49-1.py or rjp49-2.py. We can handle this automatic renaming.
3. You *must* upload your homework to the assignments area in Canvas by 11:59 pm on the due date. You will have two opportunities (NO MORE) to submit up to two days late. IF YOU DO NOT UPLOAD YOUR HOMEWORK TO THE ASSIGNMENT FOLDER BY THE DUE DATE (OR THE TWO-DAY GRACE PERIOD IN SOME CASES), IT CANNOT BE GRADED AND YOU WILL RECEIVE A ZERO.

Instructions

In this assignment, you will gain experience working with hidden Markov models for part-of-speech tagging.

A skeleton file `homework6_cmpsc442.py` containing empty definitions for each question has been provided. Since portions of this assignment will be graded automatically, none of the names or function signatures in this file should be modified. However, you are free to introduce additional variables or functions if needed.

You may import definitions from any standard Python library, and are encouraged to do so in case you find yourself reinventing the wheel.

You will find that in addition to a problem specification, most programming questions also include a pair of examples from the Python interpreter. These are meant to illustrate typical use cases to clarify the assignment, and are not comprehensive test suites. In addition to performing your own testing, you are strongly encouraged to verify that your code gives the expected output for these examples before submitting.

You are strongly encouraged to follow the Python style guidelines set forth in PEP 8, which was written in part by the creator of Python. However, your code will not be graded for style.

1. Hidden Markov Models [100 points]

In this section, you will develop a hidden Markov model for part-of-speech (POS) tagging, using the Brown corpus as training data. The tag set used in this assignment will be the universal POS tag set, which is composed of the twelve POS tags Noun (noun), Verb (verb), Adj (adjective), Adv (adverb), Pron (pronoun), Det (determiner or article), Adp (preposition or postposition), Num (numeral), Conj (conjunction), Prt (particle), '.' (punctuation mark), and X (other).

As in previous assignments, your use of external code should be limited to built-in Python modules, which **excludes packages such as NumPy and NLTK**.

1. **[10 points]** Write a function `load_corpus(path)` that loads the corpus at the given path and returns it as a list of POS-tagged sentences. Each line in the file should be treated as a separate sentence, where sentences consist of sequences of whitespace-separated strings of the form `"token=POS"`. Your function should return a list of lists, with individual entries being 2-tuples of the form (token, POS).

```
>>> c = load_corpus("brown_corpus.txt")
>>> c[1402]
[('It', 'PRON'), ('made', 'VERB'),
 ('him', 'PRON'), ('human', 'NOUN'),
 ('.', '.')]
```

```
>>> c = load_corpus("brown_corpus.txt")
>>> c[1799]
[('The', 'DET'), ('prospects', 'NOUN'),
 ('look', 'VERB'), ('great', 'ADJ'),
 ('.', '.')]
```

2. **[20 points]** In the `Tagger` class, write an initialization method `__init__(self, sentences)` which takes a list of sentences in the form produced by `load_corpus(path)` as input and initializes the internal variables needed for the POS tagger. In particular, if $\{ t_1, t_2, \ldots, t_n \}$ denotes the set of tags and $\{ w_1, w_2, \ldots, w_m \}$ denotes the set of tokens found in the input sentences, you should at minimum compute:

   - The initial tag probabilities $\pi(t_i)$ for $1 \leq i \leq n$, where $\pi(t_i)$ is the probability that a sentence begins with tag t_i.

   - The transition probabilities $a(t_i \rightarrow t_j)$ for $1 \leq i, j \leq n$, where $a(t_i \rightarrow t_j)$ is the probability that tag t_j occurs after tag t_i.

- The emission probabilities $b(t_i \rightarrow w_j)$ for $1 \leq i \leq n$ and $1 \leq j \leq m$, where $b(t_i \rightarrow w_j)$ is the probability that token w_j is generated given tag t_i.

It is imperative that you use Laplace smoothing where appropriate to ensure that your system can handle novel inputs, but the exact manner in which this is done is left up to you as a design decision. Your initialization method should take no more than a few seconds to complete when given the full Brown corpus as input.

3. **[30 points]** In the `Tagger` class, write a method `most_probable_tags(self, tokens)` which returns the list of the most probable tags corresponding to each input token. In particular, the most probable tag for a token w_j is defined to be the tag with index $i^* = argmax_i\, b(t_i \rightarrow w_j)$.

```
>>> c = load_corpus("brown_corpus.txt")
>>> t = Tagger(c)
>>> t.most_probable_tags(
...     ["The", "man", "walks", "."])
['DET', 'NOUN', 'VERB', '.']
```

```
>>> c = load_corpus("brown_corpus.txt")
>>> t = Tagger(c)
>>> t.most_probable_tags(
...     ["The", "blue", "bird", "sings"])
['DET', 'ADJ', 'NOUN', 'VERB']
```

4. **[40 points]** In the `Tagger` class, write a method `viterbi_tags(self, tokens)` which returns the most probable tag sequence as found by Viterbi decoding. Recall from lecture that Viterbi decoding is a modification of the Forward algorithm, adapted to find the path of highest probability through the trellis graph containing all possible tag sequences. Computation will likely proceed in two stages: you will first compute the probability of the most likely tag sequence, and will then reconstruct the sequence which achieves that probability from end to beginning by tracing backpointers.

```
>>> c = load_corpus("brown_corpus.txt")
>>> t = Tagger(c)
>>> s = "I am waiting to reply".split()
>>> t.most_probable_tags(s)
['PRON', 'VERB', 'VERB', 'PRT', 'NOUN']
>>> t.viterbi_tags(s)
['PRON', 'VERB', 'VERB', 'PRT', 'VERB']
```

```
>>> c = load_corpus("brown_corpus.txt")
>>> t = Tagger(c)
>>> s = "I saw the play".split()
>>> t.most_probable_tags(s)
['PRON', 'VERB', 'DET', 'VERB']
>>> t.viterbi_tags(s)
['PRON', 'VERB', 'DET', 'NOUN']
```