CIS 521: Homework 8 [50 points]

Release Date	Tuesday, April 5, 2016
Due Date	11:59 pm on Tuesday, April 12, 2016

Instructions

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

A skeleton file `homework8.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, and should not be taken as comprehensive test suites.

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.

Once you have completed the assignment, you should submit your file on Eniac using the following `turnin` command, where the flags `-c` and `-p` stand for "course" and "project", respectively.

```
turnin -c cis521 -p hw8 homework8.py
```

You may submit as many times as you would like before the deadline, but only the last submission will be saved. To view a detailed listing of the contents of your most recent submission, you can use the following command, where the flag `-v` stands for "verbose".

```
turnin -c cis521 -p hw8 -v
```

1. Hidden Markov Models [45 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. **[5 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. **[10 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 $\{t_1, t_2, \cdots, t_n\}$ denotes the set of tags and $\{w_1, w_2, \cdots, w_m\}$ denotes the set of tokens found in the input sentences, you should at minimum compute:

   - The initial tag probabilities $\pi(t_i)$ for $1 \le i \le n$, where $\pi(t_i)$ is the probability that a sentence begins with tag t_i.
   - The transition probabilities $a(t_i \to t_j)$ for $1 \le i, j \le n$, where $a(t_i \to t_j)$ is the probability that tag t_j occurs after tag t_i.
   - The emission probabilities $b(t_i \to w_j)$ for $1 \le i \le n$ and $1 \le j \le m$, where $b(t_i \to w_j)$ 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. **[10 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 $i^* = \text{argmax}_i\, b(t_i \to w_j)$.

```
>>> 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. **[20 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']
```

2. Feedback [5 points]

1. **[1 point]** Approximately how long did you spend on this assignment?
2. **[2 points]** Which aspects of this assignment did you find most challenging? Were there any significant stumbling blocks?
3. **[2 points]** Which aspects of this assignment did you like? Is there anything you would have changed?