Project 4: Part of Speech Tagging

Due: Wednesday Oct. 21st, 11:59am (Eastern Time)

Deliverables

- `writeup.tex` and the corresponding compiled `writeup.pdf` with answers to all questions.
- A completed `hmm.py` with all required functions filled in.

LaTeX

For this (and likely future labs) you will submit your project writeups using LaTeX. You should have some experience with LaTeX from CS35, but part of my intention here is to help get you comfortable working with LaTeX before you'll need to use it more substantially with the final project/paper.

If you haven't worked with LaTeX very much before you may find these guides (**here** and **here**) put together by Prof. Newhall and Prof. Brody respectively, to be helpful. If you are having trouble figuring out how to get the typesetting for something specific to work, please try searching online first (you're very likely to find someone on StackExchange with **similar questions**) but you are also welcome to ask on **Piazza**.

As with previous labs, include your names at the top of your `writeup` file. There are no strict requirements for the formatting, and you won't be graded on it[1], but do your best to present something readable. If you want a starting point here's a sample **simple template** for problem sets and here is a **slightly fancier one** with more examples.

1 HMM Exercises

The following tagging model has the two words *'boxes'* and *'books'*, and the two POS tags *'noun'* and *'verb'* plus the end-of-sentence delimiters $\triangleright$ and $\triangleleft$.

$$p(\textsc{noun}|\triangleright) = 1/2 \quad p(\textsc{verb}|\triangleright) = 1/2 \quad p(boxes|\textsc{noun}) = 1/2$$
$$p(\textsc{noun}|\textsc{noun}) = 1/2 \quad p(\textsc{verb}|\textsc{noun}) = 1/6 \quad p(boxes|\textsc{verb}) = 3/4$$
$$p(\textsc{noun}|\textsc{verb}) = 1/2 \quad p(\textsc{verb}|\textsc{verb}) = 1/6 \quad p(books|\textsc{noun}) = 1/2$$
$$p(\triangleleft|\textsc{noun}) = 1/3 \quad p(\triangleleft|\textsc{verb}) = 1/3 \quad p(books|\textsc{verb}) = 1/4$$

[1] Although I can provide some minor extra credit for particularly nice submission writeups!

Questions

1. What is the *total probability* of the output sequence *'boxes books'*? i.e. what probability does our HMM assign to the string given any possible tag sequence. Show your work.
2. What is the probability of the *most likely tag sequence* for *'boxes books'*? Show your work.

If you're unsure how to approach the *total probability* take a look at **section A.3** in the textbook.

2 HMM Programming

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

I have provided a skeleton file (`hmm.py`) containing empty definitions for each programming question. Do not modify the names or signatures of those existing functions, but feel free to introduce additional variables and/or functions as needed.

You may import definitions from any standard Python library, and are encouraged to do so in case you find yourself reinventing the wheel. However for this assignment, your use of external code should be limited to built-in Python modules, which excludes packages such as NumPy and NLTK.

All of the data you'll need for this project is outlined below and already included in the associated GitHub repo:

Data

- Starter code is available in the `hmm.py` Python file of the Lab4 **GitHub repo**.
- `brown_corpus.txt` is a txt file with a POS-tagged version of the Brown corpus.

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 we will use is 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).

I encourage you to develop some unit tests to verify that your implementations of the required methods are functioning correctly[2], but you don't need to directly report the output of any particular test.

2.1 load_corpus()

Questions

3. Write a function `load_corpus(path)` that loads the corpus at the given path and returns it as a list of POS-tagged sentences. Describe the logic of your implementation (rough pseudo-code or prose) in the writeup.

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] Looking ahead of the final project, being creative about reasonable ways to evaluate NLP models is a useful skill

2.2 Tagger initialization

Questions

4. 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. Describe the logic of your approach in the writeup. In particular, if $\{s_1, s_2, ..., s_n\}$ denotes the set of states (tags) and $\{w_1, w_2, ..., w_m\}$ denotes the set of words found in the input sentences, you should at minimum compute:
   - The initial tag probabilities $\pi(s_i)$ for $1 \leq i \leq n$, where $\pi(s_1)$ is the probability that a sentence begins with state s_1. (On the lecture slides I had assumed every sentence starts with $\triangleright$.)
   - The transition probabilities $a(s_i \rightarrow s_j)$ for $1 \leq i \leq n$, where $a(s_i \rightarrow s_j)$ is the probability that state s_j occurs after state s_i. (This is the same as the state-to-state $\sigma_{i,j}$ estimates on the lecture slides.)
   - The emission probabilities $b(s_i \rightarrow t_j)$ for $1 \leq i \leq n$ and $1 \leq j \leq m$, where $b(s_i \rightarrow t_j)$ is the probability that token t_j is generated given state s_i. (This is the same as the state-to-tag τ estimates on the lecture slides.)

It is imperative that you use Laplace (add-α) 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.

2.3 most_probable_tags()

Questions

5. In the `Tagger` class, write a method `most_probable_tags(self, tokens)` which returns the list of the most probable tag states corresponding to each input token.

Note that this is the *"stupid baseline"* that I noted in lecture. In particular, the most probable tag y for a token t_j is defined to be the tag with largest value τ_{y,t_j} (using the formalism from the slides). This could also be written as $\arg\max_y b(y \rightarrow t_j)$ following the Jurafsky & Martin formalism.

```
>>> 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']
```

2.4 Viterbi

Questions

6. In the `Tagger` class, write a method `viterbi_tags(self, tokens)` which returns the most probable tag sequence as found by Viterbi decoding. Describe your implementation in the writeup.

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. You can also find a description of the Viterbi algorithm in the **textbook section 8.4.5**. 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']
```

3 Feedback

Please provide answers to these in `Writeup.tex` just like the rest of the questions. There are no right or wrong answers here of course :)

Questions

7. Approximately how many hours did you spend on this assignment?

8. Which aspects of this assignment did you find most challenging? Were there any significant stumbling blocks?

9. Which aspects of this assignment did you like? Is there anything you would have changed?

4 Optional Extra credit

Describe any extra credit questions you chose to implement / answer in an additional section at the end of the `writeup` file.

4.1 A bonus pen and paper question

Suppose for some HMM when applied to the sequence $\mathbf{x}$, $P(\mathbf{x}|\mathbf{y}) = 0$ for all $\mathbf{y} =< ..., y_i = a, ... >$. That is, any sequence of states that goes through state a at position i assigns zero probability to the string $\mathbf{x}$. Does it follow that $\tau_{a,x_i} = 0$? Why or why not?

4.2 A bonus programming task

Write an evaluation script that executes your completed `hmm.py` and reports the accuracy for the following:

- How well does the model perform if you 'cheat' by training and then testing on the whole `brown_corpus.txt` file?
- Now implement a **k-fold cross validation scheme** ($k = 10$) and report the mean accuracy?
- Finally explore the effect of limiting the training set size. Pick a random 10% of the total `brown_corpus` to serve as the test data. Then generate a plot showing the outcome accuracy that results from training are various proportions of the remaining training data (the maximum possible training data is the set theoretic difference between `brown_corpus` and your designated test subset.