Part-of-speech tagging

A basic problems in Natural Language Processing is *part-of-speech tagging*, in which the goal is to mark every word in a sentence with its part of speech (noun, verb, adjective, etc.). Sometimes this is easy: a sentence like "Blueberries are blue" clearly consists of a noun, verb, and adjective, since each of these words has only one possible part of speech (e.g., "blueberries" is a noun but can't be a verb).

But in general, one has to look at all the words in a sentence to figure out the part of speech of any individual word. For example, consider the — grammatically correct! — sentence: "Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo." To figure out what it means, we can parse its parts of speech:

Buffalo	buffalo	Buffalo	buffalo	buffalo	buffalo	Buffalo	buffalo.
Adjective	Noun	Adjective	Noun	Verb	Verb	Adjective	Noun

(In other words: the buffalo living in Buffalo, NY that are buffaloed (intimidated) by buffalo living in Buffalo, NY buffalo (intimidate) buffalo living in Buffalo, NY.)

That's an extreme example, obviously. Here's a more mundane sentence:

Her	position	covers	a	number	of	daily	tasks	common	to	any	social	director.
DET	NOUN	VERB	DET	NOUN	ADP	ADJ	NOUN	ADJ	ADP	DET	ADJ	NOUN

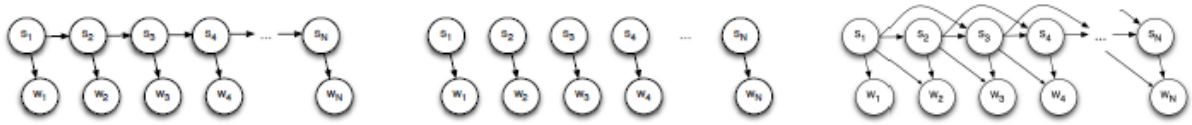
where DET stands for a determiner, ADP is an adposition, ADJ is an adjective, and ADV is an adverb.[1] Many of these words can be different parts of speech: "position" and "covers" can both be nouns or verbs, for example, and the only way to resolve the ambiguity is to look at the surrounding words. Labeling parts of speech thus involves an understanding of the intended meaning of the words in the sentence, as well as the relationships between the words.

Fortunately, statistical models work amazingly well for NLP problems. Consider the Bayes net shown in Figure 1(a). This Bayes net has random variables $S = \{S_1, \ldots, S_N\}$ and $W = \{W_1, \ldots, W_N\}$. The W's represent observed words in a sentence. The S's represent part of speech tags, so $S_i \in \{\text{VERB}, \text{NOUN}, \ldots\}$. The arrows between W and S nodes model the relationship between a given observed word and the possible parts of speech it can take on, $P(W_i|S_i)$. (For example, these distributions can model the fact that the word "dog" is a fairly common noun but a very rare verb.) The arrows between S nodes model the probability that a word of one part of speech follows a word of another part of speech, $P(S_{i+1}|S_i)$. (For example, these arrows can model the fact that verbs are very likely to follow nouns, but are unlikely to follow adjectives.)

Data. To help you with this assignment, we've prepared a large corpus of labeled training and testing data. Each line consists of a sentence, and each word is followed by one of 12 part-of-speech tags: ADJ (adjective), ADV (adverb), ADP (adposition), CONJ (conjunction), DET (determiner), NOUN, NUM (number), PRON (pronoun), PRT (particle), VERB, X (foreign word), and . (punctuation mark).[2]

[1] If you didn't know the term "adposition", neither did I. The adpositions in English are prepositions; in many languages, there are postpositions too. But you won't need to understand the linguistic theory between these parts of speech to complete the assignment; if you're curious, check out the "Part of Speech" Wikipedia article for some background.

[2] This dataset is based on the Brown corpus. Modern part-of-speech taggers often use a much larger set of tags – often over



(a) (b) (c)

Figure 1: Bayes Nets for part of speech tagging: (a) HMM, and (b) simplified model, and (c) complicated model.

What to do. Your goal in this part is to implement part-of-speech tagging in Python, using Bayes networks.

1. To get started, consider the simplified Bayes net in Figure 1(b). To perform part-of-speech tagging, we'll want to estimate the most-probable tag s_i^* for each word W_i,

$$s_i^* = \arg\max_{s_i} P(S_i = s_i|W).$$

   Implement part-of-speech tagging using this simple model.

2. Now consider Figure 1(a), a richer Bayes net that incorporates dependencies between words. Implement Viterbi to find the maximum a posteriori (MAP) labeling for the sentence,

$$(s_1^*, \ldots, s_N^*) = \arg\max_{s_1,\ldots,s_N} P(S_i = s_i|W).$$

3. Consider the Bayes Net of Figure 1c, which could be a better model because it incorporates richer dependencies between words. But it's not an HMM, so we can't use Viterbi. Implement Gibbs Sampling to sample from the posterior distribution of Fig 1c, $P(S|W)$. Then estimate the best labeling for each word (by picking the maximum marginal for each word, $s_i^* = \arg\max_{s_i} P(S_i = s_i|W)$. (To do this, just generate many (thousands?) of samples and, for each individual word, check which part of speech occurred most often.)

Your program should take as input a training filename and a testing filename. The program should use the training corpus to estimate parameters, and then display the output of Steps 1-3 on each sentence in the testing file. For the result generated by each of the three approaches (Simple, HMM, Complex), as well as for the ground truth result, your program should output the logarithm of the joint probability $P(S, W)$ for each solution it finds under each of the three models in Figure 1. It should also display a running evaluation showing the percentage of words and whole sentences that have been labeled correctly so far. For example:

```
[djcran@raichu djc-sol]$ python3 ./label.py training_file testing_file
Learning model...
Loading test data...
Testing classifiers...
                  Simple     HMM  Complex  Magnus     ab integro seclorum nascitur  ordo .
  0. Ground truth -48.52  -64.33   -73.43    noun   verb     adv     conj     noun  noun .
    1. Simplified -47.29  -66.74   -75.29    noun   noun    noun      adv     verb  noun .
          2. HMM  -47.48  -63.83   -74.12    noun   verb     adj     conj     noun  verb .
    3.    Complex -47.50  -64.21   -72.02    noun   verb     adv     conj     noun  noun .

==> So far scored 1 sentences with 17 words.
```

100 tags, depending on the language of interest – that carry finer-grained information like the tense and mood of verbs, whether nouns are singular or plural, etc. In this assignment we've simplified the set of tags to the 12 described here; the simple tag set is due to Petrov, Das and McDonald, and is discussed in detail in their 2012 LREC paper if you're interested.

```
                Words correct:     Sentences correct:
  0. Ground truth       100.00%             100.00%
    1. Simplified        42.85%               0.00%
          2. HMM         71.43%               0.00%
       3. Complex       100.00%             100.00%
```

We've already implemented some skeleton code to get you started, in three files: label.py, which is the main program, pos_scorer.py, which has the scoring code, and pos_solver.py, which will contain the actual part-of-speech estimation code. You should only modify the latter of these files; the current version of pos_solver.py we've supplied is very simple, as you'll see. In your report, please make sure to include your results (accuracies) for each technique on the test file we've supplied, `bc.test`. Your code should finish within about 10 minutes.