B551 Assignment 3: Probability

Fall 2016
Due: Sunday November 6, 11:59PM
(You may submit up to 48 hours late for a 10% penalty.)

This assignment will give you a chance to practice probabilistic inference for some real-world problems.

You'll work in a group of 1-4 people for this assignment; we've already assigned you to a group (see details below) based on the input you provided. We tried to accomodate as many of your requests as possible, but we could not satisfy all of them. You should only submit **one** copy of the assignment for your team, through GitHub, as described below. All people on the team will receive the same grade on the assignment, except in unusual circumstances; we will collect feedback about how well your team functioned in order to detect these circumstances. Please read the instructions below carefully; we cannot accept any submissions that do not follow the instructions given here. Most importantly: please **start early,** and ask questions on Piazza or in office hours.

The following problems require you to write programs in Python. You may import standard Python modules for routines not related to AI, such as basic sorting algorithms and basic data structures like queues. You must write all of the rest of the code yourself. If you have any questions about this policy, please ask us. We recommend using the CS Linux machines (e.g. `burrow.soic.indiana.edu`). You may use another development platform (e.g. Windows), but make sure that your code works on the CS Linux machines before submission.

For each programming problem, please include a detailed comments section at the top of your code that describes: (1) a description of how you formulated the problem, including precisely defining the abstractions (e.g. HMM formulation); (2) a brief description of how your program works; (3) a discussion of any problems, assumptions, simplifications, and/or design decisions you made; and (4) answers to any questions asked below in the assignment.

Academic integrity. You and your teammates may discuss the assignment with other people at a high level, e.g. discussing general strategies to solve the problem, talking about Python syntax and features, etc. You may also consult printed and/or online references, including books, tutorials, etc., but you must cite these materials (e.g. in source code comments). However, the work and code that you and your partners submit must be your group's own work, which your group personally designed and wrote. You may not share written answers or code with any other students except your own partner, nor may you possess code written by another student who is not your partner, either in whole or in part, regardless of format.

Part 0: Getting started

You can find your assigned teammate by logging into IU Github, at `http://github.iu.edu/`. In the upper left hand corner of the screen, you should see a pull-down menu. Select `cs-b551-fa2016`. Then in the yellow box to the right, you should see a repository called *userid1-userid2-userid3*-a3, where the other user IDs correspond to your teammate(s).

To get started, clone the github repository:

`git clone https://github.iu.edu/cs-b551-fa2016/`*your-repo-name*`-a3`

where *your-repo-name* is the one you found on the GitHub website above.

Part 1: Natural Language Processing

Natural language processing (NLP) is an important research area in artificial intelligence, with a history dating back to at least the 1950's. One of the most basic problems in NLP 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.). This is

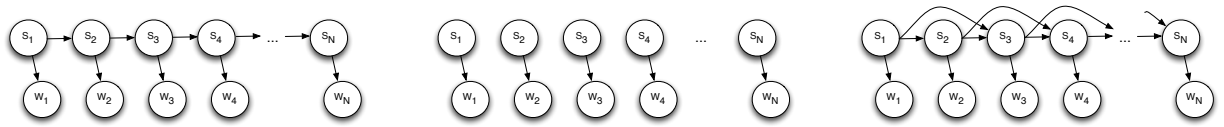


Figure 1: Three Bayes Nets for part of speech tagging: (a) an HMM, (b) a simplified model, and (c) a more complicated model.

a first step towards extracting semantics from natural language text. For example, consider the following sentence:

Her position covers a number of daily tasks common to any social director.

Part-of-speech tagging here is not easy because many of these words can take on different parts of speech depending on context. For example, *position* can be a noun (as in the above sentence) or a verb (as in "They position themselves near the exit"). In fact, *covers*, *number*, and *tasks* can all be used as either nouns or verbs, while *social* and *common* can be nouns or adjectives, and *daily* can be an adjective, noun, or adverb. The correct labeling for the above sentence is:

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] 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, it turns out that a relatively simple Bayesian network can model the part-of-speech problem surprisingly well. In particular, consider the Bayesian network shown in Figure 1(a). This Bayes net has a set of N random variables $S = \{S_1, \ldots, S_N\}$ and N random variables $W = \{W_1, \ldots, W_N\}$. The W variables represent observed words in a sentence, so $W_i \in \{w | w \text{ is a word in the English language}\}$. The variables in S represent part of speech tags, so $S_i \in \{\text{VERB}, \text{NOUN}, \ldots\}$. The arrows between W and S nodes model the probabilistic 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.)

We can use this model to perform part-of-speech tagging as follows. Given a sentence with N words, we construct a Bayes Net with $2N$ nodes as above. The values of the variables W are just the words in the sentence, and then we can perform Bayesian inference to estimate the variables in S.

Your goal in this part is to implement a part-of-speech tagger in Python, using Bayesian networks.

Data. To help you get started, we've prepared a large corpus of labeled training and testing data, consisting of nearly 1 million words and 50,000 sentences. The file format of the datasets is quite simple: each line consists of a word, followed by a space, followed by one of 12 part-of-speech tags: ADJ (adjective), ADV (adverb), ADP (adposition), CONJ (conjunction), DET (determiner), NOUN, NUM (number), PRON

[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 lingustic theory between these parts of speech to complete the assignment; if you're curious, you might check out the "Part of Speech" Wikipedia article for some background.

(pronoun), PRT (particle), VERB, X (foreign word), and . (punctuation mark). Sentence boundaries are indicated by blank lines.[2]

1. First you'll need to estimate the conditional probability tables encoded in the Bayesian network above, namely $P(S_1)$, $P(S_{i+1}|S_i)$, and $P(W_i|S_i)$. To do this, use the labeled *training* corpus file we've provided.

2. Your goal now is to label new sentences with parts of speech, using the probability distributions learned in step 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, and display the result as well as the actual marginal probability $P(S_i = s_i^*|W)$ for each chosen state. (Note that this probability is like a "confidence" score, indicating how certain the algorithm feels about each word's part of speech.)

   Hint: This is easy; if you don't see why, try running Variable Elimination by hand on the Bayes Net in Figure 1(b).

3. Now consider the Bayes net of Figure 1a, which is a better model that incorporates dependencies between words. Implement the Viterbi algorithm to find the maximum a posteriori (MAP) labeling for the sentence – i.e. the most likely state sequence:

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

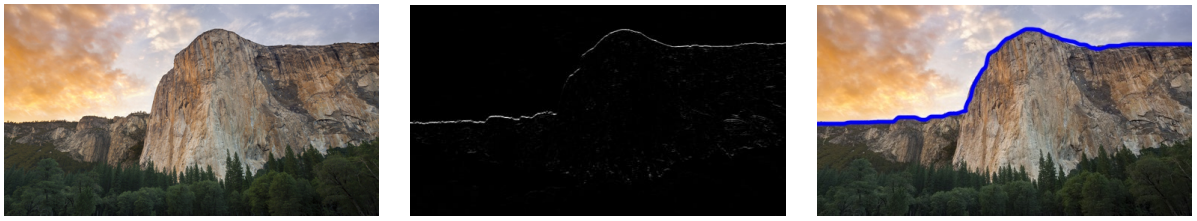
4. Consider the Bayes Net of Figure 1c, which is a better model of language because it incorporates some longer-term dependencies between words. However, it's no longer an HMM, so one can't use Viterbi. Use Variable Elimination to estimate the best labeling for each word (by picking the maximum marginal for each word, $s_i^* = \arg\max_{s_i} P(S_i = s_i|W)$, as in step 2), as well as the "confidence score" for each word, $P(S_i = s_i|W)$.

Your program should be called label.py and should take as input two filenames: a training file and a testing file. The program should use the training corpus for Step 1, and then display the output of Steps 2 through 4 on each sentence in the testing file. It should also display the logarithm of the posterior probability for each solution it finds, as well as a running evaluation showing the percentage of words and whole sentences that have been labeled correctly according to the ground truth. For example:

```
[djcran@raichu djc-sol]$ python label.py training_file testing_file
Learning model...
Loading test data...
Testing classifiers...
                          : Magnus   ab integro seclorum nascitur ordo .
 0. Ground truth (-143.52): noun   verb     adv     conj      noun noun .
   1. Simplified (-146.10): noun   verb     adv     conj       adv noun .
                          : 1.00   1.00    1.00     1.00     0.26 0.74 .
         2. HMM (-142.68): noun   verb     adv     conj       det noun .
      3. Complex (-142.70): noun   verb     adv     conj       det noun .
                          : 1.00   1.00    1.00     1.00     0.26 0.74 .

==> So far scored 1 sentences with 17 words.
                  Words correct:     Sentences correct:
   0. Ground truth:      100.00%              100.00%
```

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



(a) (b) (c)

Figure 2: Figure 1: (a) Sample input image, (b) corresponding edge strength map, and (c) highlighted ridge.

```
    1. Simplified:      83.33%              0.00%
          2. HMM:       83.33%              0.00%
       3. Complex:      83.33%              0.00%
----
```

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`.

Part 2: Mountain finding

Given a photo, how can you determine where on Earth it was taken? If the photo has GPS metadata, this is of course trivial, and if it's a photo of a major tourist attraction, it's relatively easy to use image matching to compare to a library of many tourist sites. But what if it's a photo like that of Figure 2a? It turns out that one relatively powerful cue is the shape of the mountains you see in the distance. The shape of these mountains potentially form a distinctive "fingerprint" that can be matched to a digital elevation map of the world, to determine (at least roughly) where the photographer was standing.

In this part, we'll consider the very first step of solving this process automatically: identifying the shape of mountains in a photo. In particular, we'll try to identify the ridgeline, i.e. the boundary between the sky and the mountain. We'll assume relatively clean images like that of Figure 2a, where the mountain is plainly visible, there are no other objects obstructing the mountain's ridgeline, the mountain takes up the full horizontal dimension of the image, and the sky is relatively clear. Under these assumptions, for each column of the image we need to estimate the row of the image corresponding to the boundary position. Plotting this estimated row for each column will give a ridgeline estimate.

We've given you some code that reads in an image file and produces an "edge strength map" that measures how strong the image gradient (local constrast) is at each point. We could assume that the stronger the image gradient, the higher the chance that the pixel lies along the ridgeline. So for an $m \times n$ image, this is a 2-d function that we'll call $I(x, y)$, measuring the strength at each pixel $(x, y) \in [1, m] \times [1, n]$ in the original image. Your goal is to estimate, for each column $x \in [1, m]$, the row s_x corresponding to the ridgeline. We can solve this problem using a Bayes net, where $s_1, s_2, ...s_m$ correspond to the hidden variables, and the gradient data are the observed variables (specifically $w_1, w_2, ...w_m$, where w_1 is a vector corresponding to column 1 of the gradient image).

1. Perhaps the simplest approach would be to use the Bayes Net in Figure 1b. Implement this technique, showing the result of the detected boundary with a red line in the output image.

2. A better approach would use the Bayes Net in Figure 1a. Use MCMC to do this, showing the result of the detected boundary with a blue line in the output image.

   (Hint: What should your transition probabilities look like? It's up to you, but you probably want to use a distribution that encourages "smoothness," i.e. that $P(s_{i+1}|s_i)$ is high if $s_{i+1} = s_i$ and is low if they are very different. How about the emission probabilities? Again, this is for you to decide, but intuitively a higher gradient should correspond to higher probability.)

3. A simple way of improving the results further would be to incorporate some human feedback in the process. Assume that a human gives you a single (x, y) point that is known to lie on the ridgeline. Modify your answer to step 2 to incorporate this additional information. (Hint: you can do this by just tweaking the HMM's probability distributions – no need to change the algorithms themselves!) Show the detected boundary in green.

Your program should be run like this:

```
python ./mountain.py input_file.jpg output_file.jpg row_coord col_coord
```

where *row_coord* and *col_coord* are the image row and column coordinates of the human-labeled pixel, and output_file.png should show the original image with the red, green, and blue lines superimposed.

Run your code on our sample images (and feel free to try other sample images of your own), and please commit the output images to your repo.

What to turn in

Turn in the file required above by simply putting the finished version (of the code with comments) on GitHub (remember to `add`, `commit`, `push`) — we'll grade whatever version you've put there as of 11:59PM on the due date. To make sure that the latest version of your work has been accepted by GitHub, you can log into the github.iu.edu website and browse the code online.