B551 Assignment 5: Probability

Fall 2015
Due: Sunday November 22, 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 in a real-world Natural Language Processing problem, by building a part-of-speech tagger using inference on Bayesian networks models. You'll get experience working with Bayesian networks, implementing the Viterbi and MCMC algorithms, and estimating Bayes nets parameters from labeled training data.

We've assigned you a team for this assignment, based on your teamwork preferences you expressed after Assignment 2. You should only submit **one** copy of the assignment for your team, through GitHub, as described below. Both people on the team will receive the same grade on the assignment, except in very unusual circumstances. Please read the instructions below carefully; we cannot accept any submissions that do follow the instructions given here. Most importantly: please **start early,** and ask questions on the OnCourse Forum or in office hours.

Academic integrity. You and your partner 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 partner must be your group's own work, which the two of you 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.

Background

Natural language processing (NLP) is an important research area in artificial intelligence, with a history dating back to at least the 1950's. The goal of NLP is to algorithmically understand and generate human language. Early work investigated rule-based systems that used linguistic knowledge of human languages to solve NLP, but over the last 10-20 years the most successful systems have used statistical and probabilistic approaches. These approaches do not use detailed linguistic models, but instead learn parameters of simple statistical models by analyzing large corpora of training data.

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

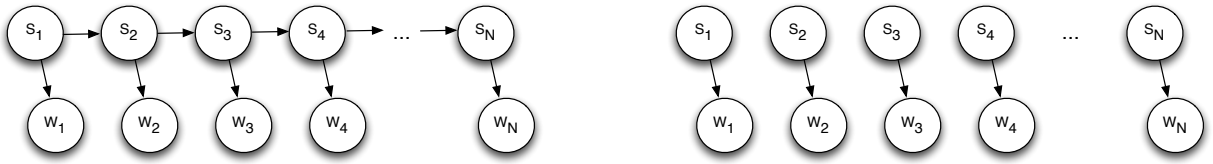
> Her position covers a number of daily tasks common to any social director.

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.

[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.



(a) (b)

Figure 1: Two Bayes Nets for part of speech tagging: (a) a full model, and (b) a simplified model.

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.

Writing your own part-of-speech tagger

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

Step 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 a pull-down menu. Select `cs-b551`. Then in the yellow box to the right, you should see a repository called *userid1-userid2*-a5, where the other user ID corresponds to your teammate. To get started, clone the github repository:

`git clone https://github.iu.edu/cs-b551/`*your-repo-name*`-a5`

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

Step 1: Learning. In the first step, 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. Use maximum-likelihood estimation.

[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.

Step 2: Naïve inference. 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. *Hint:* This is easy; if you don't see why, try running Variable Elimination by hand on the Bayes Net in Figure 1(b).

Step 3: Sampling. Now let's go back to the more sophisticated model of Figure 1(a). Write code that uses Gibbs (MCMC) Sampling to sample from the posterior distribution, $P(S|W)$. Use your code to display five samples. *Hint:* You'll want to have some warm-up period, i.e. throw away the first few samples.

Step 4: Approximate max-marginal inference. The MCMC sampling of step 3 allows you to (approximately) estimate marginal distributions, as we saw in class. Use your sampling code to estimate the most likely part of speech for each individual word, conditioned on all of the other words, e.g. display $s_i^* = \arg\max_{s_i} P(S_i = s_i|W)$, and also display the actual maximum probability, $P(S_i = s_i^*|W)$. Note that this probability is like a "confidence" score, indicating how certain the algorithm feels about each word's part of speech. *Hint:* To do this, use your MCMC code in Step 3, but generate several thousand samples instead of just five. Then for each individual word, check which part of speech has occurred most often.

Step 5: Exact maximum a posteriori inference. Next, implement the Viterbi algorithm to find the most likely sequence of state variables,

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

Step 6: Whatever works best! Finally, implement the best algorithm you can think of for this problem. This might be just one of the above algorithms you've already implemented, perhaps with some tuned parameter values, or it may be a different algorithm that you've found works better in practice. A small amount of your grade for this project (about 5% or less) will be based on how well your "best" algorithm works compared to other students' submissions. (You can use the bc.test file we've supplied to test your code, but be careful not to "overfit" to that test data, because we will use our own set of test data also.)

Program specifications. 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 6 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 bc.train bc.tiny.test
Learning model...
Loading test data...
Testing classifiers...
                       : poet twisted again and  nick's knuckles scraped on   the  air  tank ,    ripping off  the  skin .
 0. Ground truth (-143.52): noun verb    adv   conj noun   noun     verb    adp  det  noun noun .    verb    prt  det  noun .
       1. Naive (-146.10): noun verb    adv   conj adv    noun     verb    adp  det  noun noun .    verb    prt  det  noun .
     2. Sampler (-142.68): noun verb    adv   conj det    noun     verb    adp  det  noun noun .    verb    prt  det  noun .
                (-142.86): noun verb    adv   conj adj    noun     verb    adp  det  noun noun .    verb    adp  det  noun .
                (-142.85): noun verb    adv   conj adj    noun     verb    adp  det  noun noun .    verb    prt  det  noun .
                (-142.68): noun verb    adv   conj det    noun     verb    adp  det  noun noun .    verb    prt  det  noun .
                (-143.52): noun verb    adv   conj noun   noun     verb    adp  det  noun noun .    verb    prt  det  noun .
 3. Max marginal (-142.70): noun verb    adv   conj det    noun     verb    adp  det  noun noun .    verb    adp  det  noun .
                       : 1.00 1.00    1.00  1.00 0.26   0.74     1.00    0.98 1.00 1.00 1.00 1.00 1.00    0.50 1.00 1.00 1.00
         4. MAP (-142.68): noun verb    adv   conj det    noun     verb    adp  det  noun noun .    verb    prt  det  noun .
        5. Best (-142.68): noun verb    adv   conj det    noun     verb    adp  det  noun noun .    verb    prt  det  noun .

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

```
         1. Naive:       94.12%               0.00%
       2. Sampler:       94.12%               0.00%
   3. Max marginal:      88.24%               0.00%
            4. MAP:      94.12%               0.00%
           5. Best:      94.12%               0.00%
----
                         : desperately ,    nick flashed one  hand up   ,    catching poet's neck in   the  bend of   his  elbow .
 0. Ground truth (-135.66): adv         .    noun verb    num  noun prt  .    verb     noun   noun adp  det  noun adp  det  noun  .
       1. Naive (-137.37): adv         .    noun verb    num  noun prt  .    verb     noun   noun adp  det  verb adp  det  noun  .
     2. Sampler (-141.30): adv         .    noun verb    num  noun adp  .    noun     noun   noun adp  det  verb adp  det  noun  .
                (-137.37): adv         .    noun verb    num  noun prt  .    verb     noun   noun adp  det  verb adp  det  noun  .
                (-135.66): adv         .    noun verb    num  noun prt  .    verb     noun   noun adp  det  noun adp  det  noun  .
                (-138.76): adv         .    noun verb    num  noun adp  .    verb     noun   noun adp  det  noun adp  det  noun  .
                (-135.66): adv         .    noun verb    num  noun prt  .    verb     noun   noun adp  det  noun adp  det  noun  .
 3. Max marginal (-135.66): adv         .    noun verb    num  noun prt  .    verb     noun   noun adp  det  noun adp  det  noun  .
                         : 0.98        1.00 1.00 1.00    0.96 0.98 0.94 1.00 0.64     1.00   1.00 1.00 1.00 0.84 1.00 1.00 1.00  1.00
          4. MAP (-135.66): adv         .    noun verb    num  noun prt  .    verb     noun   noun adp  det  noun adp  det  noun  .
         5. Best (-135.66): adv         .    noun verb    num  noun prt  .    verb     noun   noun adp  det  noun adp  det  noun  .

==> So far scored 2 sentences with 35 words.
                   Words correct:     Sentences correct:
   0. Ground truth:      100.00%              100.00%
         1. Naive:        94.29%                0.00%
       2. Sampler:        88.57%                0.00%
   3. Max marginal:       94.29%               50.00%
            4. MAP:       97.14%               50.00%
           5. Best:       97.14%               50.00%
----
                         : the  air  hose was  free !    !
 0. Ground truth ( -50.10): det  noun noun verb adj  .    .
       1. Naive ( -50.10): det  noun noun verb adj  .    .
     2. Sampler ( -50.10): det  noun noun verb adj  .    .
                ( -51.67): det  noun noun verb adv  .    .
                ( -50.10): det  noun noun verb adj  .    .
                ( -50.10): det  noun noun verb adj  .    .
                ( -50.10): det  noun noun verb adj  .    .
 3. Max marginal ( -50.10): det  noun noun verb adj  .    .
                         : 1.00 1.00 1.00 1.00 0.80 1.00 1.00
          4. MAP ( -50.10): det  noun noun verb adj  .    .
         5. Best ( -50.10): det  noun noun verb adj  .    .

==> So far scored 3 sentences with 42 words.
                   Words correct:     Sentences correct:
   0. Ground truth:      100.00%              100.00%
         1. Naive:        95.24%               33.33%
       2. Sampler:        90.48%               33.33%
   3. Max marginal:       95.24%               66.67%
            4. MAP:       97.62%               66.67%
           5. Best:       97.62%               66.67%
----
```

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.

What to turn in

Turn in the file required above by simply putting the finished version (of the code with comments and PDF file for the first question) 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.

Please include a detailed comments section at the top of your code that describes: (1) a brief description of how your code works, (2) the results of the evaluation on the bc.test file (i.e. the percentage of correct words and sentences for each algorithm, as displayed by our scoring code), and (3) a discussion of any problems you faced, any assumptions, simplifications, and/or design decisions you made.