CS-5340/6340, Programming Assignment #2
Due: Tuesday, September 26, 2017 by 11:00pm

For this assignment, you should write a program that implements the Viterbi algorithm as applied to part-of-speech tagging. As we discussed in class, the Viterbi algorithm is an efficient way to find the most likely part-of-speech tag sequence for a sentence. As input, your program will need two files: (1) a file of probabilities, and (2) a file of sentences to process. Your program should read the names of the 2 input files from the command line in the following order:

```
viterbi <probabilities_file> <sentences_file>
```

For example, we should be able to run your program like so:

```
viterbi probs.txt sents.txt
```

(Including the programming language at the beginning is fine, for example:
`python viterbi probs.txt sents.txt`

---

1. The Probabilities File

This file will contain transition and emission probabilities. Each line will be formatted as:

X Y probability

which means that $P(X \mid Y) = \textit{probability}$.

If both X and Y are part-of-speech tags, then it is a transition probability. If X is a word and Y is a part-of-speech tag, then it is an emission probability. To keep things simple for this assignment, you should assume that there are only 4 possible part-of-speech tags: *noun verb inf prep*, plus a special tag *phi* to denote the beginning of a sentence. (If you see any of these five strings, you can assume that they are tags and not words.) A sample probabilities file might look like this:

```
bears noun .02
bears verb .02
fish verb .07
fish noun .08
noun phi .80
verb phi .10
noun verb .77
etc.
```

Use a default value of .0001 for all transition and emission probabilities that are not present in the probabilities file. Your program should be case insensitive, so "bears", "Bears", and "BEARS" should all be treated as the same token.

2. The Sentences File

This file will contain a set of sentences that you should run your program on. Each sentence will be on a separate line. There will not be any punctuation marks in this file. A sample file might look like this:

```
This is a sentence
And here is another sentence
```

Computing Sentence Probabilities

Since the probabilities will get very small, you should do the probability computations in log space (as discussed in class, also see the lecture slides). Your output values should be printed as logprobs (i.e., the logarithm of a probability) rather than the probability itself. **Please do these calculations using log base 2.** If your programming language uses a different log base, then you can use the formula on the lecture slides to convert between log bases.

When printing the logprob numbers, please print **exactly 4 digits** after the decimal point. For example, print -8.9753864210 as -8.9754. The programming language will have a mechanism for controlling the number of digits that are printed. If P(S) = 0, then the logarithm is not defined, so print logprob(S) = undefined.

OUTPUT FORMATTING

The output produced by your program should consist of 4 items for each sentence that is processed:

1. The sentence being processed.
2. The log probabilities associated with **every** node in the Viterbi network when the algorithm is finished.
3. The values associated with **every** node in the Backpointer network (i.e., the data structure that you use to reconstruct the final tag sequence) when the algorithm is finished. NOTE: this information could be stored in the Viterbi network itself or in a separate data structure. Either way, please print these values separately from the Viterbi probabilities.
4. The log probability and the part-of-speech tag assignments for the best tag sequence.

IMPORTANT: When you print this information, please format it exactly like the example in Figure 1 on the next page! Since you'll be printing a lot of numbers, this is crucial to ensure that we know exactly what your numbers and results correspond to.

Please print the nodes as if you were reading the sentence left-to-right (so all nodes for the leftmost word in the sentence should appear first) and print the nodes within a column ordered as "noun", "verb", "inf", and "prep". The trace files that we give you will follow this convention.

The final part-of-speech tag assignments should be printed starting with the last word in the sentence and ending with the first word. This order should be the easiest one for you to produce as you traverse the backpointer network.

```
PROCESSING SENTENCE: bears fish

FINAL VITERBI NETWORK
P(bears=noun) = -5.9658
P(bears=verb) = -8.9658
P(bears=inf) = -26.5754
P(bears=prep) = -26.5754
P(fish=noun) = -12.9867
P(fish=verb) = -10.4238
P(fish=inf) = -24.4379
P(fish=prep) = -20.9905

FINAL BACKPTR NETWORK
Backptr(fish=noun) = verb
Backptr(fish=verb) = noun
Backptr(fish=inf) = verb
Backptr(fish=prep) = noun

BEST TAG SEQUENCE HAS LOG PROBABILITY = -10.4238
fish -> verb
bears -> noun
```

Figure 1: Sample Output for Viterbi Program

FOR CS-6340 STUDENTS ONLY! (30 pts)

In addition to the implementation of the Viterbi algorithm, CS-6340 students should also implement the Forward Algorithm to compute the cumulative evidence that each word has a specific part-of-speech. The output should be the **normalized forward probabilities** produced in the last step of the Forward Algorithm.

For each test sentence, your program should print the results of the Viterbi algortihm followed by the results of the Forward Algorithm by showing the normalized forward probability computed for each node in the network. Please use the same printing conventions mentioned earlier regarding the order of the nodes.

However, for the Forward Algorithm **print the actual probability** for each node in the network (not a logprob). Please print exactly four digits after the decimal point. For example, your output should be formatted like this:

```
FORWARD ALGORITHM RESULTS
P(bears=noun) = 0.8889
P(bears=verb) = 0.1111
P(bears=inf) = 0.0000
P(bears=prep) = 0.0000
P(fish=noun) = 0.1448
P(fish=verb) = 0.8546
P(fish=inf) = 0.0001
P(fish=prep) = 0.0006
```

IMPORTANT you do NOT need to use logprobs when computing the Forward Algorithm because the summation does not lend itself so easily to doing all the computations in log space. We will keep the probabilities and examples small enough that you shouldn't have to worry about floating point issues.

GRADING CRITERIA

We will run your program on the files that we give you as well as new files to evaluate the generality and correctness of your code. **So please test your program thoroughly!** Even if your program works perfectly on the examples that we give you, that does not guarantee that it will work perfectly on different test cases.

Please make sure that your program conforms exactly to the input and output specifications, or a penalty will be deducted! Programs that do not conform to the specifications are a lot more difficult for us to grade.

SUBMISSION INSTRUCTIONS
(a.k.a. "What to turn in and how to do it")

Please use CANVAS to submit a gzipped tarball file named "viterbi.tar.gz". (This is an archived file in "tar" format and then compressed with gzip. Instructions appear below if you're not familiar with tar files.) Your tarball should contain the following 3 items:

1. The source code for your program. Be sure to include all files that are needed to compile and run your program!
2. A **README.txt** file that includes the following information:
   (a) what programming language and version you used (e.g., python2 or python3)
   (b) instructions on how to compile and run your code
   (c) which CADE machine you tested your program on
   (this info may be useful to us if we have trouble running your program)
   (d) any known bugs, problems, or limitations of your program

   REMINDER: your program *must* compile and run on the linux-based CADE machines! We will not grade programs that cannot be run on these CADE machines.
3. Submit a trace file called **viterbi-trace.txt** that shows the output of your program when applied to the sample input files. The sample input files are available in the Program #2 folder on CANVAS.

HELPFUL HINTS

TAR FILES: First, put all of the files that you want to submit in a directory called "viterbi". Then from the parent directory where the "viterbi" folder resides, issue the following command:

```
tar cvfz viterbi.tar.gz viterbi/
```

This will put everything inside the "viterbi/" directory into a single file called "viterbi.tar.gz". This file will be "archived" to preserve any structure (e.g., subdirectories) inside the "viterbi/" directory and then compressed with "gzip".

FYI, to unpack the gzipped tarball, move the viterbi.tar.gz to a new location and issue the command:

```
tar xvfz viterbi.tar.gz
```

This will create a new directory called "viterbi" and restore the original structure and contents.

For more general information on the "tar" command, this web site may be useful:
https://www.howtogeek.com/248780/how-to-compress-and-extract-files-using-the-tar-command-on-linux/

TRACE FILES: You can generate a trace file in (at least!) 3 different ways: (1) print your output to a file called `viterbi-trace.txt`, (2) print your output to standard output and then pipe it to a file (e.g., `viterbi probs.txt sents.txt > viterbi-trace.txt`), or (3) print your output to standard output and invoke the unix *script* command before running your program. The sequence of commands to use is:

```
script viterbi-trace.txt
viterbi probs.txt sents.txt
exit
```

This will save everything that printed to standard output during the session to a file called `viterbi-trace.txt`.