HOMEWORK 3

OVERVIEW[1]

In this assignment, you will implement the Forward Procedure and the Viterbi Algorithm for Hidden Markov Models. Your starting task will be to analyze the sentence “time flies quickly” using a Hidden Markov Model part of speech tagger. Specifically, you must implement the Forward Procedure and the Viterbi algorithm to determine:

i) what is the probability of observing this string of words (i.e. what is the probability of this **observed** sequence given your Hidden Markov model) and,

ii) what is the most probable sequence of part of speech tags for this observed string of words given this model? (i.e. what is the most probable **hidden** POS sequence that generated this string?)

For this assignment you will assume a very limited fragment of an HMM that could do POS tagging for English: to begin you will be limited to three words (*time*, *flies* and *quickly*) and three part of speech tags (***N***oun, ***V***erb and adve***R***b), plus the special start and stop ‘#’ symbol. This problem is not totally artificial however: you'll be using an HMM that is based on counts from the Brown corpus.

Here is the A matrix (the matrix of transition probabilities):

	N	*V*	*R*	#
N	0.54	0.23	0.08	0.15
V	0.62	0.17	0.11	0.10
R	0.17	0.68	0.10	0.05
#	0.70	0.20	0.10	0

Here is the B matrix (the matrix of emission probabilities):

	time	*flies*	*quickly*	#
N	0.98	0.015	0.005	0
V	0.33	0.64	0.03	0
R	0.01	0.01	0.98	0
#	0	0	0	1

PART 1

For part 1 of this assignment, write a function that uses the Forward Procedure to calculate the probability of any sequence of words given the HMM above.

Specifically, in your HMM.py script for this assignment, write a function *forward* that takes the two parameters of an HMM and a sequence of observations, and returns the probability of that sequence under that HMM. Inside the *forward* function, before returning the probability, call the *print_table()* function on the completed table to print the results out for inspection.

Within the above specifications, you may implement the *forward* Procedure as presented in class any way you see fit. Here is rough, but recommended pseudocode for a Forward Procedure function:

```
function: forward(test_sequence,A,B)
    prepend and postpend ‘#’ to test_sequence
    T is the length of the resulting test_sequence
    create table to store results
    Initialization: set forward probability of ‘#’ at time step 1 to 1 and all others to 0
    for time step t from 2 to T:
        for all the tags j in the tag set:
            the current forward probability for the tag j is sum over all tags i of:
                forward probability of tag i at time step t-1 *
                transition probability from tag i to tag j *
                emission probability of current word from tag j

    return the forward probability stored in the last column in the table for the tag ‘#’
```

[1] This assignment is a modified version of Brian Dillon’s tagging assignment used in previous LING 492 semesters.

Once you have written the *forward* function, you can uncomment the first print statement in the main code to run your *forward* function on the string 'time flies quickly' and print the result out to the standard output.

Tips

- The A and B matrices are stored as dictionaries at the top of the provided code stub for easy access
- Make sure you understand how the '#'s should be treated in the transitions and the emissions before starting to code!
- The simplest way to represent the table is as a list of lists, with each cell storing a float (the forward probability of that state). If you substantially deviate from this representation of the table, you will need to modify the *print_table()* function to work with whatever data structure you choose.

PART 2

Now you will implement the Viterbi algorithm for part-of-speech tagging!

Specifically, you should write a *viterbi* function that takes the two parameters of the HMM and the string of words to analyze as arguments, and returns the highest probability tag sequence together with its probability. As in Part 1, you should have your *viterbi* function print the completed table before returning the result.

Within the above specifications, you may implement the *viterbi* function as presented in class any way you see fit. Here is rough, but recommended pseudocode for a Viterbi function:

```
function: viterbi(test_sequence,A,B)
    prepend and postpend '#' to test_sequence
    T is the length of the resulting test_sequence
    create table v to store viterbi probabilities
    create table b to store backpointers
    Initialization: set viterbi probability of '#' at time step 1 to 1 and all others to 0
    for time step t from 2 to T:
        for all the tags j in the tag set:
            find the tag i that maximizes the following:
                viterbi probability of tag i at time step t-1 *
                transition probability from tag i to tag j *
                emission probability of current word from tag j
            viterbi probability of j at time t =
                viterbi probability of tag i at time step t-1 *
                transition probability from tag i to tag j *
                emission probability of current word from tag j
            backpointer from j at time t = i

    read and return the highest probability tag sequence and its probability
```

Once you implement the *viterbi* function, you should uncomment the second print statement in the main portion of the code to print out the best tags and their probability for the sequence 'time flies quickly'.

Tips

- Re-use as much of the forward code as possible
- I suggest that each cell of the table stores a tuple of a float and a string, where the float is the Viterbi probability for that state, and the string is the (partial) sequence of tags corresponding to that Viterbi probability. This way you keep track of the best solution as you go along without having to follow backpointers after completing the table.

QUESTIONS

After you've written the program, use it to examine the output you get and answer the following questions:

HOMEWORK 3

Q1
What is probability of ‘time flies quickly’? How many tag sequences are contributing to this probability (that is, how many non-zero ways of tagging ‘time flies quickly’ are there according to this HMM)?

Q2
What is the best tag sequence for ‘time flies quickly’? What meaning does this tag sequence correspond to?

Q3
What’s the probability of ‘quickly time flies’? What parameters would you need to change to make ‘quickly time flies’ more likely than ‘time flies quickly’ (don’t actually change the parameters in your code)? Be specific!

Q4
Modify matrix B to add a new word ‘swat’ to the lexicon. Assume that **‘swat’ can only be a verb**, and be sure your matrix defines proper conditional probability distributions. **Set your probabilities so that the most likely tag sequence for ‘swat flies quickly’ is ‘V N R’**. (hint: you will need to change probabilities in multiple rows). Show this matrix in your question write-up, and also encode it into the script as a dictionary ‘B2’. Once you define B2 at the top of your code, you can uncomment the third print statement in the main code to tag ‘swat flies quickly’ and make sure you have defined B2 correctly.

Q5
In your new HMM from Q4, ‘V N R’ is the most likely tag sequence for ‘swat flies quickly’. How is this probability different from the probability of the same tag sequence for ‘time flies quickly’ (i.e. what parameters are relevant to this comparison)? Which is higher? Explain why the answer to this question determines the answer to the following question:

- Which is higher P(‘N V’, ‘flies time’) or P(‘N V’, ‘flies swat’)

WHAT TO TURN IN
Turn in your final **HMM.py** program from Q4. Make sure your code is well commented. Submit your answers to the questions in a separate file (plain text or pdf).