# Assignment #2 NAME: XIANG LI, GWid: G47754486

**Instructions:**

✓ This assignment is due on **Tuesday October 8, 2019, by 11:59 pm**. Refer to course information about late submission policy. Late days are counted from 00:00 a.m. onwards.

✓ Submit your solution in a zipped folder through blackboard.

✓ All code must compile and run to receive credit for coding parts. We won't do any debugging. You can code in Perl, Python, Java, or C++. **Perl/Python** are recommended.

✓ Include brief instructions on how to run your scripts.

✓ Include citations for any online resources used or group discussions.

**Parts-of-Speech Tagging and Hidden Markov Models**

Q1. [20 points] Given the following formal description of a Hidden Markov Model, M={Q, Σ, A, O, $q_0$}

Q={$q_0$, $q_1$,$q_2$}

Start state: $q_0$

Σ = {a,b,c,d}

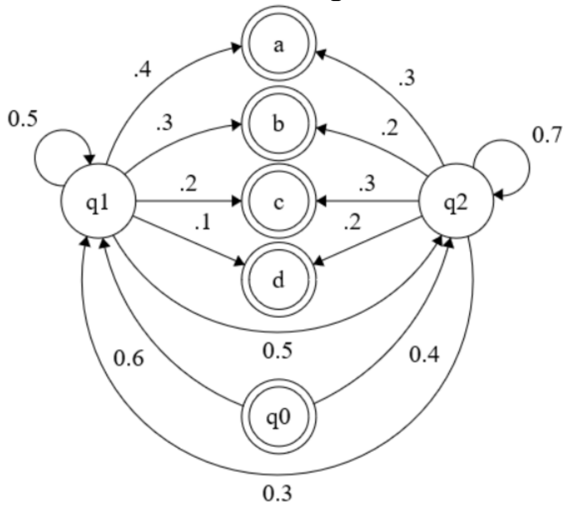
The transition probabilities matrix A:

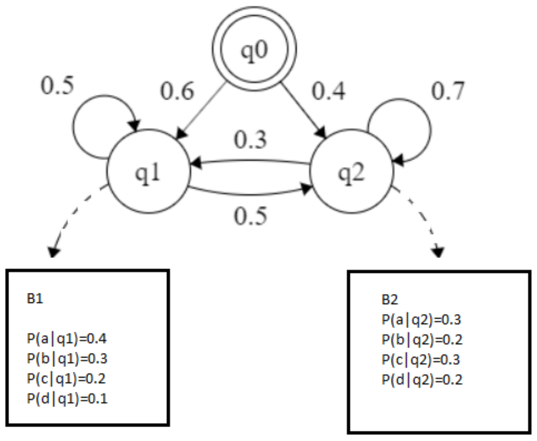
| | $q_1$ | $q_2$ |
|---|---|---|
| $q_0$ | 0.6 | 0.4 |
| $q_1$ | 0.5 | 0.5 |
| $q_2$ | 0.3 | 0.7 |

The output probabilities matrix O:

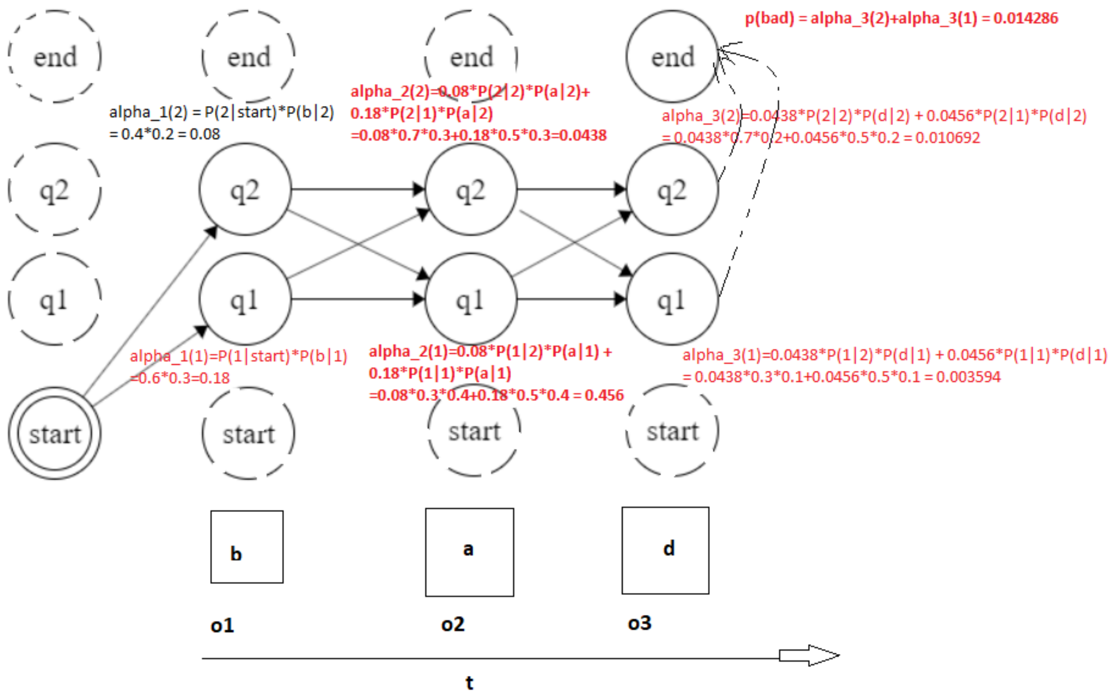
| | a | b | c | d |
|---|---|---|---|---|
| $q_1$ | 0.4 | 0.3 | 0.2 | 0.1 |
| $q_2$ | 0.3 | 0.2 | 0.3 | 0.2 |

a. Draw the transition diagram for M.





b. Use the forward algorithm to manually compute the probability of the string *bad*. Show all your work.



p(bad) = alpha_3(2)+alpha_3(1) = 0.014286

alpha_1(2) = P(2|start)*P(b|2)
= 0.4*0.2 = 0.08

alpha_2(2)=0.08*P(2|2)*P(a|2)+
0.18*P(2|1)*P(a|2)
=0.08*0.7*0.3+0.18*0.5*0.3=0.0438

alpha_3(2)=0.0438*P(2|2)*P(d|2) + 0.0456*P(2|1)*P(d|2)
= 0.0438*0.7*0.2+0.0456*0.5*0.2 = 0.010692

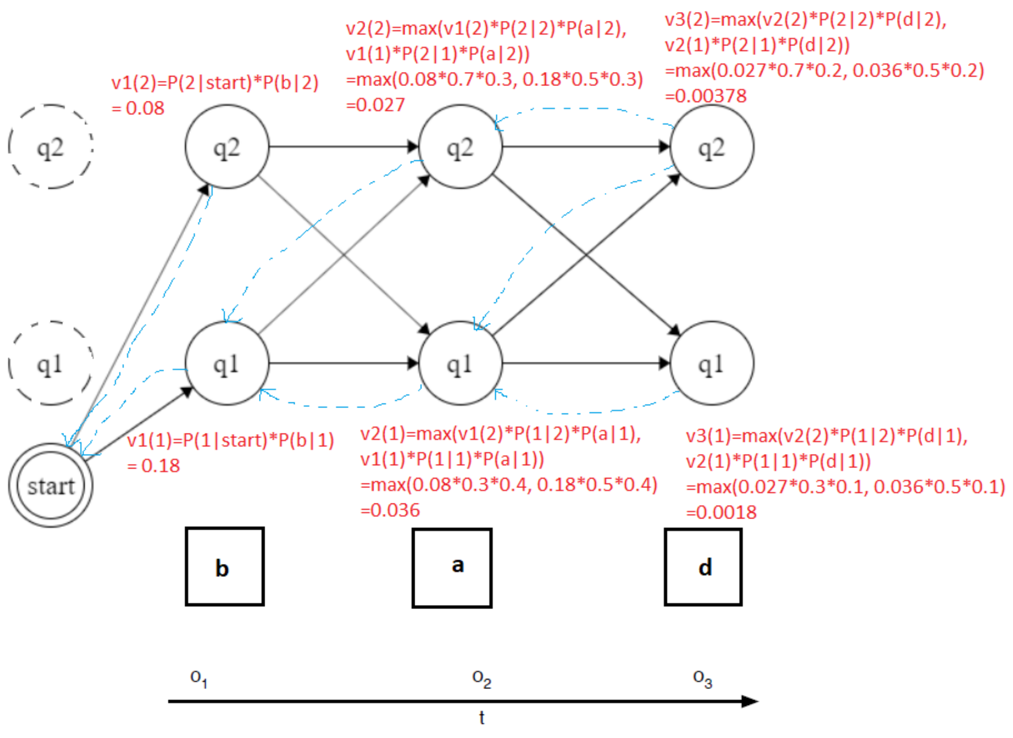
alpha_1(1)=P(1|start)*P(b|1)
=0.6*0.3=0.18

alpha_2(1)=0.08*P(1|2)*P(a|1) +
0.18*P(1|1)*P(a|1)
=0.08*0.3*0.4+0.18*0.5*0.4 = 0.456

alpha_3(1)=0.0438*P(1|2)*P(d|1) + 0.0456*P(1|1)*P(d|1)
= 0.0438*0.3*0.1+0.0456*0.5*0.1 = 0.003594

A: P(bad) = alpha_3(1)+alpha_3(2) = 0.014286

c. Use the Viterbi algorithm to manually compute the most likely hidden state sequence for the word *bad*. Show all your work, and report both the state sequence and its probability


v1(2)=P(2|start)*P(b|2)
= 0.08
v2(2)=max(v1(2)*P(2|2)*P(a|2),
v1(1)*P(2|1)*P(a|2))
=max(0.08*0.7*0.3, 0.18*0.5*0.3)
=0.027
v3(2)=max(v2(2)*P(2|2)*P(d|2),
v2(1)*P(2|1)*P(d|2))
=max(0.027*0.7*0.2, 0.036*0.5*0.2)
=0.00378
q2
q1
start
v1(1)=P(1|start)*P(b|1)
= 0.18
v2(1)=max(v1(2)*P(1|2)*P(a|1),
v1(1)*P(1|1)*P(a|1))
=max(0.08*0.3*0.4, 0.18*0.5*0.4)
=0.036
v3(1)=max(v2(2)*P(1|2)*P(d|1),
v2(1)*P(1|1)*P(d|1))
=max(0.027*0.3*0.1, 0.036*0.5*0.1)
=0.0018
b
a
d
o1
o2
o3
t

A: P* = max V3(qF) = max(v3(2), v3(1)) = 0.00378,

State Sequence: {q1,q2,q1} or {q1,q1,q2}

## Q2. Hidden Markov Models for parts-of-speech tagging:

**Note:** This part requires coding. Code must readily compile and run on SEAS Linux machines for full credit. Use the following files for training/testing:

`brown.train.tagged` : Tagged training data.
`brown.test.raw` : Raw test data that you need to tag.
`brown.test.tagged` : Tagged test data for evaluation.

For all the following scripts, to save your efforts to run them, I simply write the scripts without any input parameters.

IMPORTANT:

Please put all data and scripts in the same directory.
Python 3 was used and package numpy was imported.

All scripts in Q2:
a) and b) majority_class.py, c) HMM_Training.py, HMM_with_viterbi.py and HMM_with_viterbi_beam_search.py

a. [10 points] Implement a majority-class baseline for the Brown corpus. For each word in the test set, assign the tag that is most frequently associated with that word in the training corpus. Tag out-of-vocabulary words as NN. Report the overall accuracy of this baseline on the test set.

A: Using majority-class method as a baseline and train brown.train.tagged.txt for emission probability of HMM model. Apply trained emission probability matrix for test data, in here, there is one mismatch between "brown.test.tagged.txt" and "brown.test.txt". So that in order to avoid all potential trouble cost by that one mismatch. I used "brown.test.tagged.txt", and split every word in this test data by two parts, word and its tag. And comparing my model prediction to its real tag string to compute accuracy.
(PS: the inconsistence was found, { I/ppss 'll/md be/be damned/vnb ''/'' } > {I'll be damned '' , Ekstrohm said . } )

This result is from an emission probability matrix without using smoothing method.
This script is going to take around 1 mins to get the results.

| TEST DATA | Number of Total Words | Number of Correct Tag by majority class | Overall accuracy |
|---|---|---|---|
| "brown.test.tagged.txt" | 148,967 | 131,573 | 88.3236% |

b. [Graduate students only – 5 points] design and implement some transformation rules to improve the accuracy of the majority-class baseline. You should implement a minimum of 5 rules and no more than 15 rules. You may use the training set for tuning. For example, you may want to view the confusion matrix (of the training set) and think of rules that would fix the most common errors. You should **not** use the test set for tuning the rules. Report the overall accuracy of your model on the test set. (The submission with the highest accuracy in this part will receive 5 bonus points!)

At least Five Rules to improve accuracy:
1) Remove "" both in tags and vocabulary
2) Resolving ambiguity in tagging:
3) Eliminate VBN if VBD is an option when VBN|VBD follows "PRP"
4) Eliminate NN/RB/JJ if VB is an option following a "TO"
5) Eliminate VB if NN is an option following a "DT"
6) Transformation based learning algorithm:
    a. label every word with most likely tag
    b. Check every possible transformation & select the one that results in the most improved tagging accuracy
    c. Re-tag corpus applying this rule, and add rule to end of rule set
    d. Repeat b-c until accuracy is tend to be stable

c. [25 points] Implement an HMM Tagger for Parts-of-speech tagging.
i. You will need a training script that reads the training corpus and calculates transition and output probabilities. Note that output probabilities for OOV(Out of vocabulary) words will be zero for all tags, which means the algorithm will fail to tag any sentence that includes OOV words in the test set. You can decide how to handle such cases (one option is to set a fixed small count for OOV words given any tag).

**To run this scripts, please use cmd:**
python HMM_Training.py
This script simply read training data and computer transition and emission probability matrix and output them as a standard 2D matrix with index and column names.
(Approximately 24.1Mb for emission matrix Q2_HMM_emission_matrix_Model.txt",
and 135Kb for transition matrix, "Q2_HMM_transition_matrix_Model.txt")

A: Here, for OOV out of vocabulary words, it is a similar case of Good-Turing Smoothing, pretty much we can use the count of things we have seen once to estimate the OOV words. However, at this stage, I simply use a small count n=1/len(vocabulary) to represent OOV words.

ii. Implement the Viterbi algorithm that calculates the most likely tag sequence for each test sentence given a trained HMM model from part (i).
**To run this scripts, please use cmd:**
python HMM_with_viterbi.py

HMM with Viterbi Algorithm is going to take around 3 to 5 mins to finish.

iii. Evaluate the performance of your HMM tagger by reporting the overall accuracy on the test set.

| TEST DATA "brown.test.tagged.txt" | Tricks Apply | Number of Total Words: 148,967 | Overall Accuracy |
|---|---|---|---|
| Number of Correct Tag by majority class | Null | 131,573 | 88.3236% |
| Number of Correct Tag by HMM (first try) | Null | 102,324 | 68.6890% |
| Number of Correct Tag by HMM (Second try) | +Laplace-Smoothed for prob matrix | 137,656 | 92.4070% |
| Number of Correct Tag by HMM (Third try) | +Laplace-Smoothed for prob matrix + Good-Turing Smoo thing | 137,989 | 92.6305% |

After first try HMM model with Viterbi algorithm, it  doesn't return a very good results. Then I pay close attention to output, it looks like many predicted tags from HMM are a series of "". First try with remove "" as a tag, and apply several transformation rules. The most significant factor that could improve HMM model is to use proper smoothing method deal with sparse data matrix and also unknow word(OOV). Because using HMM model with Viterbi algorithm is going to calculate the maximum probability path.

d. [Graduate students only – 10 points] Implement beam search to speed up Viterbi decoding. Test your implementation with b=10 and report the accuracy.

**To run this scripts, please use cmd:**

```
python HMM_with_viterbi_beam_search.py
```

| TEST DATA "brown.test.tagged.txt" | Tricks Apply | Number of Total Words: 148,967 | Overall Accuracy |
|---|---|---|---|
| Number of Correct Tag by HMM (Third try) | +Laplace-Smoothed for prob matrix + Good-Turing Smoo thing | 137,989 | 92.6305% |
| Number of Correct Tag by HMM (Fourth try) | +Laplace-Smoothed for prob matrix + Good-Turing Smoo thing +Beam Search | 137,664 | 92.4124% |

*Reference:*
*1.* Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition Third Edition draft, and Second Edition
*2. Finite State Machine Designer (*http://madebyevan.com/fsm/)
*3.* Python, Documentation 3.7
*4. Numpy*
*5. Slides from Course*