**I am using 9 late hours on this set.**

1a. First, we factorize :

Repeat:

Since we stored each , we just need to account for the total stored per class. Since this grows with the binary nature of the input, we have the most complex input giving permutations. Thus, the big-O notation of this is , but we must consider this for each class. Thus, we need to consider parameters.

1b. For this situation, we just have to consider the discrete x-y combinations. This means we just have to consider storing all values of x and y in order to calculate for any combination. Thus, the input is a string of x’s of length D. For each x, we can either have a 0 or 1, since it is binary. Therefore, we have possibilities for the input. Then, we have an output for each class. Our complexity is then , which is the same as part a.

1c. We would use the Naïve Bayes in this case. Since we are working with a very small sample size, there is a higher probability of having holes in the data set. The full model is a discriminative model, so it cannot tolerate missing values, which loses all guarantees. The Naïve Bayes model can deal with these missing features, so it would give a lower test set error.

1d. We would use the full model to give a lower test set error. The full model performs better than the Naïve version when there are no holes because the Naïve model uses conditional probability while the full model uses fully dependent features.

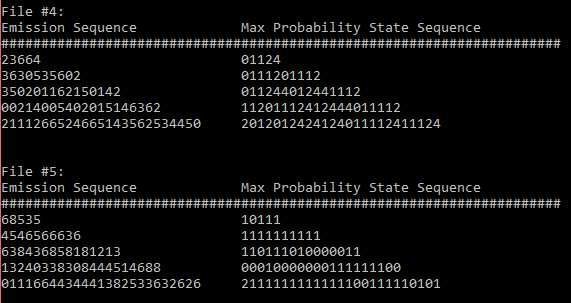
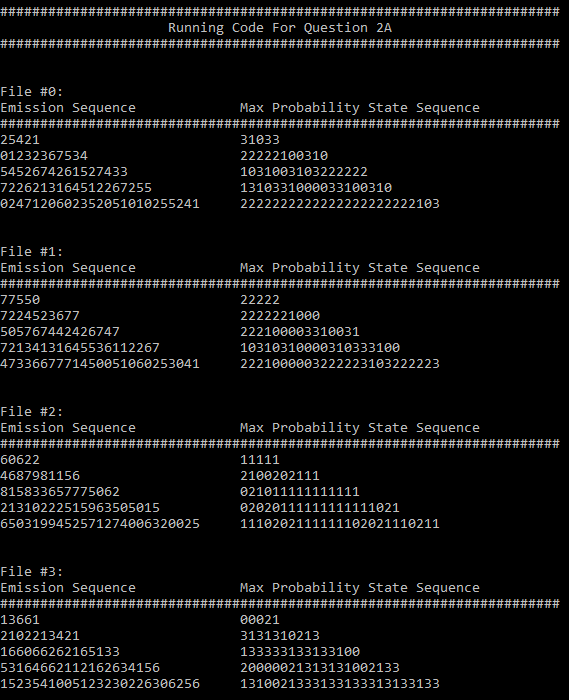
1e. For making a prediction using the Naïve model, we use

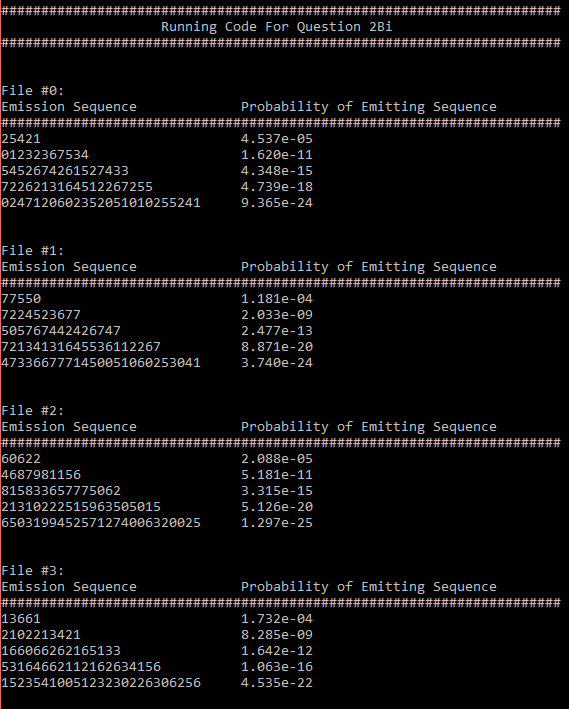
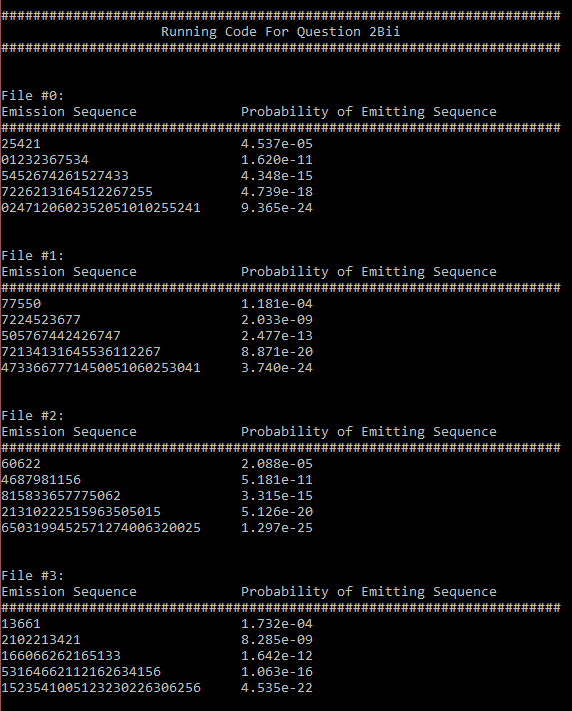
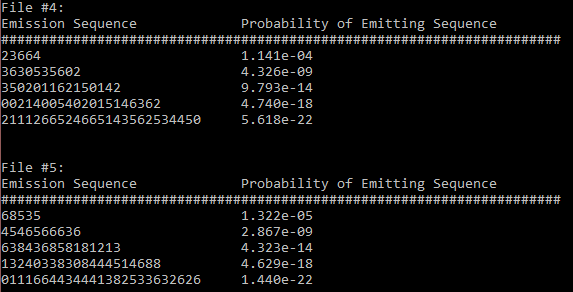
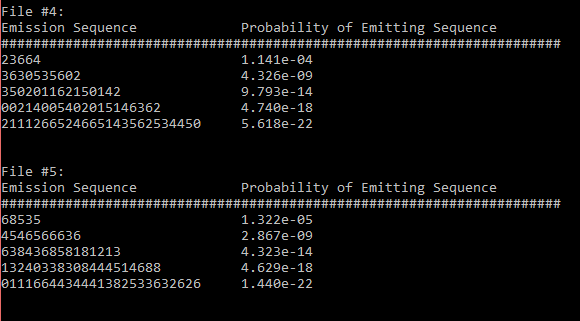
We can compute in time since we know y and there are D values for x. We also know is uniform, so it takes constant time to compute. Lastly, we have , which must be computed for each C. Thus, our overall complexity is .

For making a prediction using the full model, we use

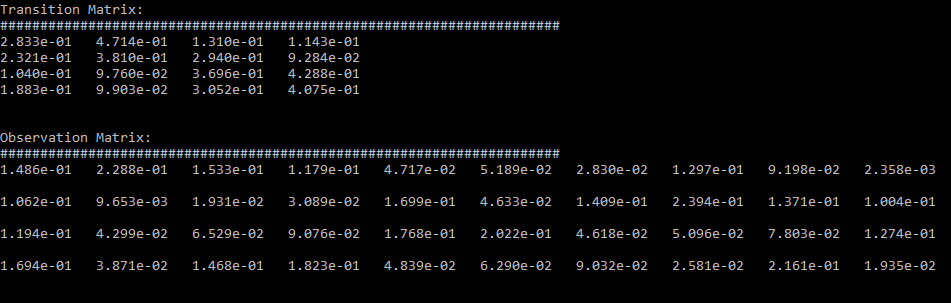
as computed in part a. Using the hint given in the problem, to compute the value we need for each C, we use the operation that was described. We then apply this to each class, so our total computational complexity for prediction is .

2a. My code generates:

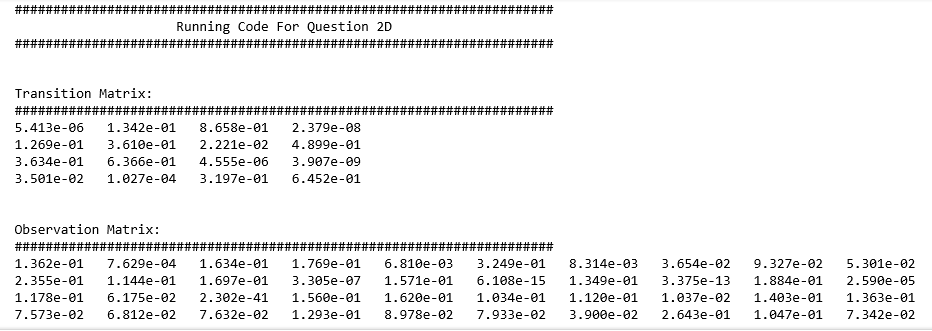


2b. Here are my results:

2c. My results:



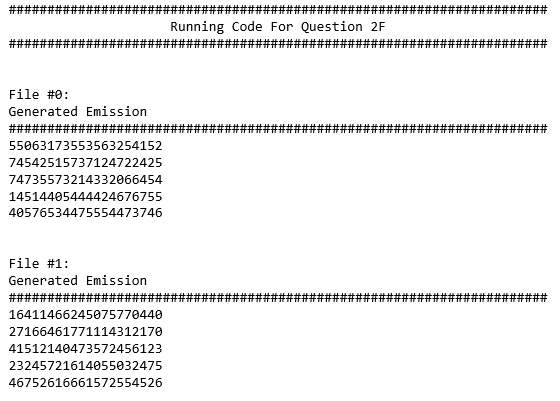
2d. My results (I had to switch to Jupyter because numpy was not working for some reason):

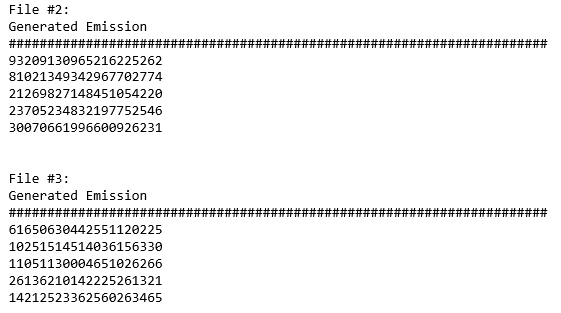


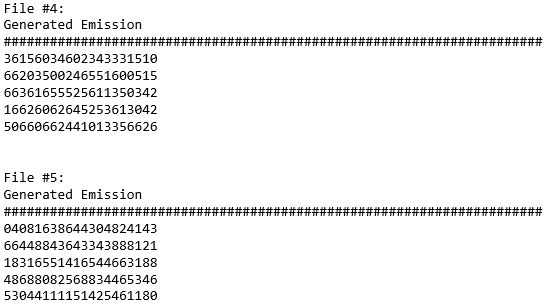
2e. It seems that the matrix from 2C provides a more accurate representation of Ron’s mood and how they affect his music choices. First off, 2C is trained with fully supervised data while 2D is trained with unsupervised data, which is half of the data in 2C. When we are given the moods of the training data, it would make sense that we could better predict on the moods than if we did not. Also, in the transition matrix, the values on the principal diagonal are mostly larger in 2C than in 2D. It would make sense that a mood is more likely to stay in the current state given any situation. Lastly, the observation matrix for 2D has odd probabilities. The 2C observation matrix has values that are similar in degree, which makes sense in the context of preferring songs. The 2D matrix has some values to the power of negative 14, which seems off for the context of preferring songs.

We could improve the unsupervised method by increasing the training data set by getting more observations on Ron.

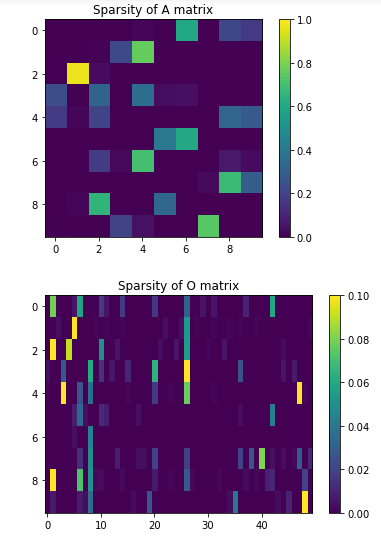
2f. My results:







2g. The graphs generated:

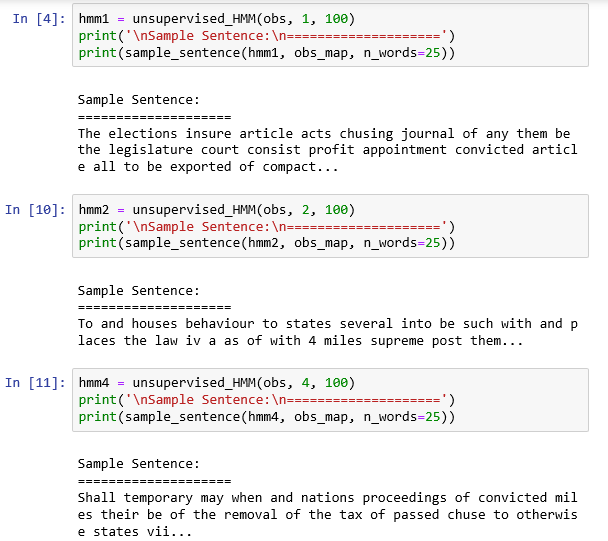


The sparsity of both matrices is pretty obvious where the values are very sparse. There are a lot of zero values, and a majority of the nonzero values are very small. The sparsity in the A or transition matrix allows for easy transition. Since there are only a few values that are nonzero in each row, it shows the transitions that will be made at each state. This is the same for the O or observation matrix, there are only a few concrete options at each given state. Therefore, the large sparsity of each matrix allows for a few behaviors at each state.

2h. The samples emission sentences become more and more coherent as the number of hidden states increases. With a small number of hidden states, the words look almost random at each spot in the sentence. As we increase the number of hidden states, we start to see more phrases that would appear in the Constitution. For example, at four hidden layers, we get the phrase “the removal of the tax,” which makes sense in the order of nouns, verbs, and articles in sentence structure.

When there is only one hidden state, it seems that the words are chosen at random. Some of the words match adjacent words, but this seems out of luck rather than being trained. In general, we can increase the training data likelihood by allowing more hidden states, but this likelihood eventually cannot be increases when the number of hidden states gets large. The results of the visualization show that chunks of phrases become more likely with more hidden states, but the creation of these phrases will eventually plateau with the fixed observation set.

Some examples from my code:



2i. I will pick state four because of its large presence of verbs. This state represents a transition from some of the other states. Most of the other states have a lot of nouns like “treason” or “state,” so a lot of the states lack the transition between these nouns. State four has a lot of words like “holding,” “will,” and “may” that act as connecting words. If I were to guess, the transition matrix values from the noun states to this state would be higher with models with higher numbers of hidden states. This is shown in the final animation as there are bolded arrows from states that include words like “congress” and “people”

