1 Class-Conditional Densities for Binary Data [25 Points]

This problem will test your understanding of probabilistic models, especially Naive Bayes. Consider a generative classifier for C classes, with class conditional density $p(x|y)$ and a uniform class prior $p(y)$. Suppose all the D features are binary, $x_j \in \{0,1\}$. If we assume all of the features are conditionally independent, as in Naive Bayes, we can write:

$$p(x \mid y = c) = \prod_{j=1}^{D} P(x_j \mid y = c)$$

This requires storing DC parameters.

Now consider a different model, which we will call the 'full' model, in which all the features are fully *dependent*.

Problem A [5 points]: Use the chain rule of probability to factorize $p(x \mid y)$, and let $\theta_{xjc} = P(x_j|x_{1,\ldots,j-1}, y = c)$. Assuming we store each θ_{xjc}, how many parameters are needed to represent this factorization? Use big-O notation.

Solution A.:

$$\begin{aligned}
p(x|y=c) &= p(x_1, x_2, ..., x_j|y=c) \\
&= p(x_j, x_{j-1}, ...x_1|y=c) \\
&= p(x_j|x_{j-1}, ..., x_1, y=c) \cdot p(x_{j-1}, ..., x_1|y=c) \\
&= \prod_{j=1}^{D} p(x_j|x_1, x_2, ...x_{j-1}, y=c) \\
&= \prod_{j=1}^{D} \theta_{xjc}
\end{aligned}$$

The number of parameters needed to represent this factorization is $O(C \cdot 2^D)$ because there are D binary features and C total classes.

Problem B [5 points]: Assume we did no such factorization, and just used the joint probability $p(x \mid y = c)$. How many parameters would we need to estimate in order be able to compute $p(x|y = c)$ for arbitrary x and c? How does this compare to your answer from the previous part? Again, use big-O notation.

Solution B.:

If we consider using the joint probability $p(x|y = c)$ without factorization, we would need to estimate the probability of each possible combination of feature values for each class. Since each feature can take on two values (0 or 1), for D features there are 2^D possible combinations of feature values. As there are C classes, this would result in $O(C \cdot 2^D)$ parameters (same as part (a)).

Problem C [2 points]: Assume the number of features D is fixed. Let there be N training cases. If the sample size N is very small, which model (Naive Bayes or full) is likely to give lower test set error, and why?

Solution C.:

If the sample size, N, is very small, the Naive Bayes model is likely to perform better and give lower test set error compared to the full model. As we saw in the previous problems, the Naive Bayes model only requires DC parameters, whereas the full model requires $O(C \cdot 2^D)$ parameters. Thus, with a small sample size, it is difficult to accurately estimate a large number of parameters as it leads to overfitting.

Problem D [2 points]: If the sample size N is very large, which model (Naive Bayes or full) is likely to give lower test set error, and why?

Solution D.:

With a large sample size, N, the fill model is more likely to give a lower test set error because it is less prone to overfitting in this case. Additionally, there is enough data to accurately estimate the many parameters required by the full model, capturing the dependencies between features that Naive Bayes ignores.

Problem E [11 points]: Assume all the parameter estimates have been computed. What is the computational complexity of making a prediction, i.e. computing $p(y \mid x)$, using Naive Bayes for a single test case? What is the computation complexity of making a prediction with the full model? For the full-model case, assume that converting a D-bit vector to an array index is an $O(D)$ operation. Also, recall that we have assumed a uniform class prior.

Solution E.:

For Naive Bayes, to compute $p(y|x)$, we use the formula $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$. Because we have a uniform class prior, $p(y)$ is constant and can be ignored for the purpose of prediction (it does not affect the ordering of probabilities across classes). Thus, we focus on computing $p(x|y)$, which is $\prod_{j=1}^{D} p(x_j|y)$. For each feature, we look up the probability of that feature given the class, which is an $O(1)$ operation. Since there are D features, this operation is repeated D times for each class. With C classes, the total operation is $O(CD)$.

For the full model, we need to consider every possible combination of feature values. Since we are assuming that converting a D-bit vector to an array index is an $O(D)$ operation and there are C classes, we would perform this $O(D)$ operation C times. However, because we need to compute the probability of the feature vector $p(x|y)$, and the number of possible feature vectors is 2^D, the storage required is $O(C \cdot 2^D)$. For each class, we would look up the probability of the entire feature vector, which, given the data is already computed and indexed, is an $O(1)$ operation after the $O(D)$ operation of converting the bit vector to an index. Thus, for C classes, the complexity would be $O(CD)$ for the lookups plus $O(D)$ for the index conversion for each class, leading to a total complexity of $O(CD)$.

2 Sequence Prediction [75 Points]

In this problem, we will explore some of the various algorithms associated with Hidden Markov Models (HMMs), as discussed in lecture. For these problems, make sure you are **using Python 3** to implement the algorithms. Please see the HMM notes posted on the course website—they will be helpful for this problem!

Sequence Prediction

These next few problems will require extensive coding, so be sure to start early! We provide you with eight different files:

- You will write all your code in `HMM.ipynb`, within the appropriate functions where indicated by ”TODO” in the comments. There is no need to modify anything else aside from what is indicated. There should be no need to write additional functions or use NumPy in your implementation, but feel free to do so if you would like.

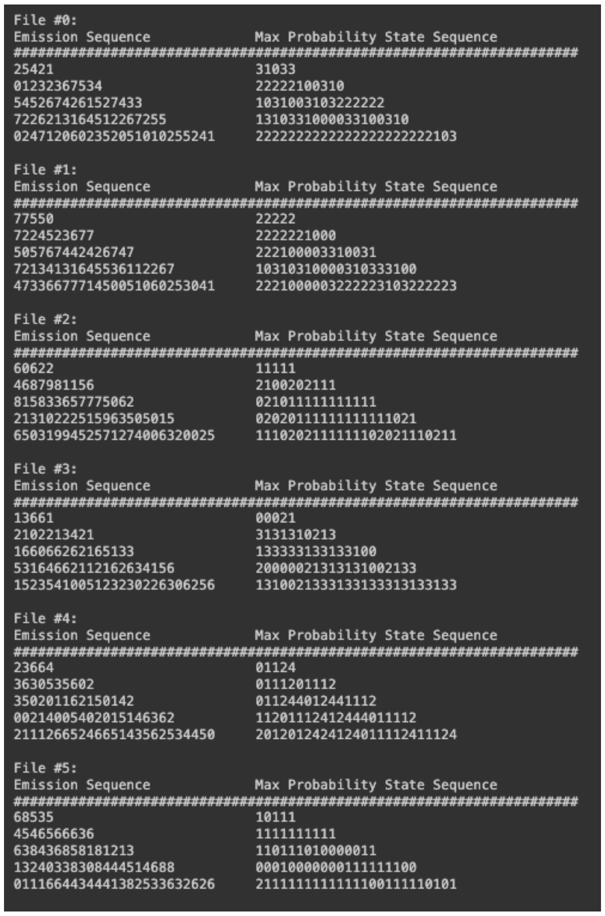
The supplementary data folder contains 6 files titled `sequence_data0.txt`, `sequence_data1.txt`, ... , `sequence_data5.txt`. Each file specifies a **trained** HMM. The first row contains two tab-delimited numbers: the number of states Y and the number of types of observations X (i.e. the observations are $0, 1, ..., X-1$). The next Y rows of Y tab-delimited floating-point numbers describe the state transition matrix. Each row represents the current state, each column represents a state to transition to, and each entry represents the probability of that transition occurring. The next Y rows of X tab-delimited floating-point numbers describe the output emission matrix, encoded analogously to the state transition matrix. The file ends with 5 possible emissions from that HMM.

The supplementary data folder also contains one additional file titled `ron.txt`. This is used in problems 2C and 2D and is explained in greater detail there.

Problem A [10 points]: For each of the six trained HMMs, find the max-probability state sequence for each of the five input sequences at the end of the corresponding file. To complete this problem, you will have to implement the Viterbi algorithm. Write your implementation well, as we will be reusing it in a later problem. See the end of problem 2B for a big hint!

Show your results on the 6 files. (Copy-pasting the results under the ”Part A” heading of `HMM.ipynb` suffices.)

Solution A.: *mantripragada_ishaan_Set6_HMM.ipynb*



```
File #0:
Emission Sequence             Max Probability State Sequence
######################################################################
25421                         31033
01232367534                   22222100310
5452674261527433              1031003103222222
7226213164512267255           1310331000033100310
0247120602352051010255241     2222222222222222222222103

File #1:
Emission Sequence             Max Probability State Sequence
######################################################################
77550                         22222
7224523677                    2222221000
505767442426747               222100003310031
72134131645536112267          10310310000310333100
4733667771450051060253041     2221000003222223103222223

File #2:
Emission Sequence             Max Probability State Sequence
######################################################################
60622                         11111
4687981156                    2100202111
815833657775062               021011111111111
21310222515963505015          02020111111111111021
6503199452571274006320025     1110202111111102021110211

File #3:
Emission Sequence             Max Probability State Sequence
######################################################################
13661                         00021
2102213421                    3131310213
166066262165133               133333133133100
53164662112162634156          20000021313131002133
1523541005123230226306256     1310021333133133313133133

File #4:
Emission Sequence             Max Probability State Sequence
######################################################################
23664                         01124
3630535602                    0111201112
350201162150142               011244012441112
00214005402015146362          11201112412444011112
2111266524665143562534450     2012012424124011112411124

File #5:
Emission Sequence             Max Probability State Sequence
######################################################################
68535                         10111
4546566636                    1111111111
638436858181213               110111010000011
13240338308444514688          00010000000111111100
0111664434441382533632626     2111111111111100111110101
```

Problem B [17 points]: For each of the six trained HMMs, find the probabilities of emitting the five input sequences at the end of the corresponding file. To complete this problem, you will have to implement the Forward algorithm and the Backward algorithm. You may assume that the initial state is randomly selected along a uniform distribution. Again, write your implementation well, as we will be reusing it in a later problem.

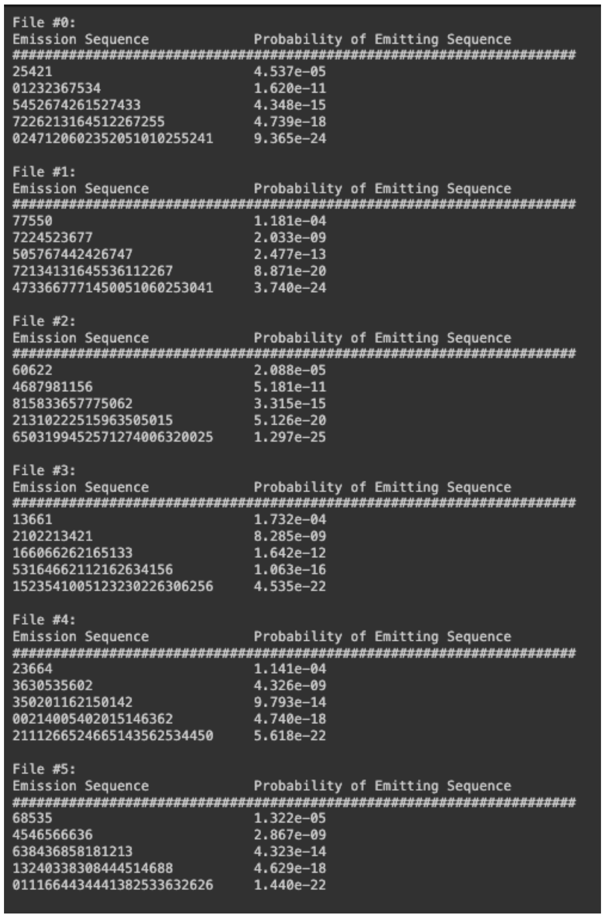
Note that the probability of emitting an input sequence can be found by using either the α vectors from the Forward algorithm or the β vectors from the Backward algorithm. You don't need to worry about this, as it is done for you in `probability_alphas()` and `probability_betas()`.

Implement the Forward algorithm. Show your results on the 6 files.
Implement the Backward algorithm. Show your results on the 6 files.

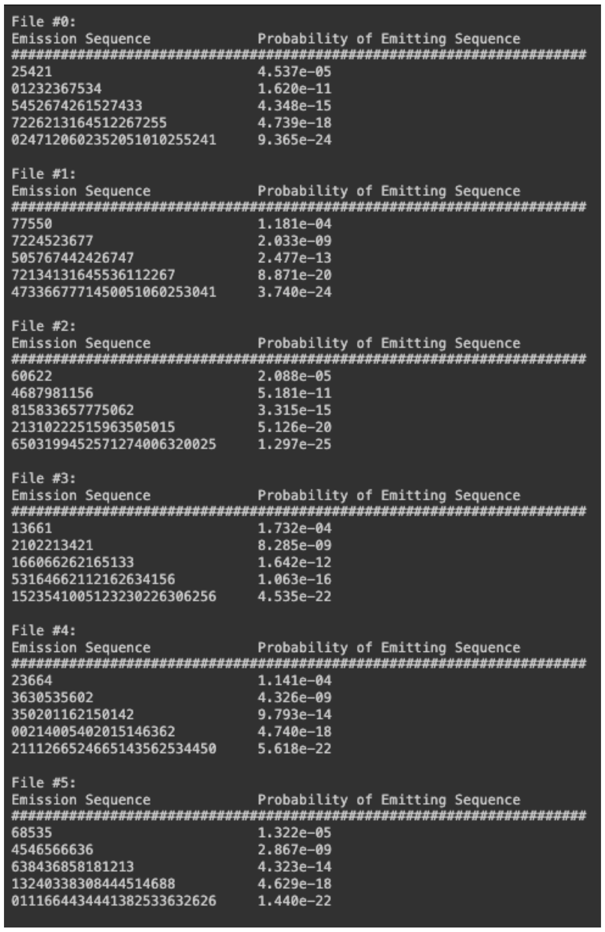
After you complete problems 2A and 2B, you can compare your results for the file titled `sequence_-data0.txt` with the values given in the table below:

Dataset	Emission Sequence	Max-probability State Sequence	Probability of Sequence
0	25421	31033	4.537e-05
0	01232367534	22222100310	1.620e-11
0	5452674261527433	1031003103222222	4.348e-15
0	7226213164512267255	1310331000033100310	4.739e-18
0	0247120602352051010255241	2222222222222222222222103	9.365e-24

Solution B:



```
File #0:
Emission Sequence             Probability of Emitting Sequence
######################################################################
25421                         4.537e-05
01232367534                   1.620e-11
5452674261527433              4.348e-15
7226213164512267255           4.739e-18
0247120602352051010255241     9.365e-24

File #1:
Emission Sequence             Probability of Emitting Sequence
######################################################################
77550                         1.181e-04
7224523677                    2.033e-09
505767442426747               2.477e-13
72134131645536112267          8.871e-20
4733667771450051060253041     3.740e-24

File #2:
Emission Sequence             Probability of Emitting Sequence
######################################################################
60622                         2.088e-05
4687981156                    5.181e-11
815833657775062               3.315e-15
21310222515963505015          5.126e-20
6503199452571274006320025     1.297e-25

File #3:
Emission Sequence             Probability of Emitting Sequence
######################################################################
13661                         1.732e-04
2102213421                    8.285e-09
166066262165133               1.642e-12
53164662112162634156          1.063e-16
1523541005123230226306256     4.535e-22

File #4:
Emission Sequence             Probability of Emitting Sequence
######################################################################
23664                         1.141e-04
3630535602                    4.326e-09
350201162150142               9.793e-14
00214005402015146362          4.740e-18
2111266524665143562534450     5.618e-22

File #5:
Emission Sequence             Probability of Emitting Sequence
######################################################################
68535                         1.322e-05
4546566636                    2.867e-09
638436858181213               4.323e-14
13240338308444514688          4.629e-18
0111664434441382533632626     1.440e-22
```



```
File #0:
Emission Sequence             Probability of Emitting Sequence
######################################################################
25421                         4.537e-05
01232367534                   1.620e-11
5452674261527433              4.348e-15
7226213164512267255           4.739e-18
0247120602352051010255241     9.365e-24

File #1:
Emission Sequence             Probability of Emitting Sequence
######################################################################
77550                         1.181e-04
7224523677                    2.033e-09
505767442426747               2.477e-13
72134131645536112267          8.871e-20
4733667771450051060253041     3.740e-24

File #2:
Emission Sequence             Probability of Emitting Sequence
######################################################################
60622                         2.088e-05
4687981156                    5.181e-11
815833657775062               3.315e-15
21310222515963505015          5.126e-20
6503199452571274006320025     1.297e-25

File #3:
Emission Sequence             Probability of Emitting Sequence
######################################################################
13661                         1.732e-04
2102213421                    8.285e-09
166066262165133               1.642e-12
53164662112162634156          1.063e-16
1523541005123230226306256     4.535e-22

File #4:
Emission Sequence             Probability of Emitting Sequence
######################################################################
23664                         1.141e-04
3630535602                    4.326e-09
350201162150142               9.793e-14
00214005402015146362          4.740e-18
2111266524665143562534450     5.618e-22

File #5:
Emission Sequence             Probability of Emitting Sequence
######################################################################
68535                         1.322e-05
4546566636                    2.867e-09
638436858181213               4.323e-14
13240338308444514688          4.629e-18
0111664434441382533632626     1.440e-22
```

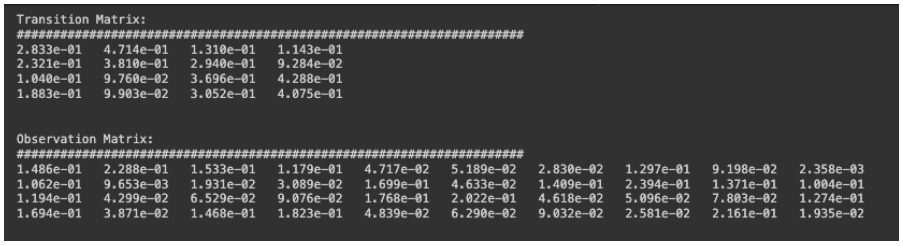
HMM Training

Ron is an avid music listener, and his genre preferences at any given time depend on his mood. Ron's possible moods are happy, mellow, sad, and angry. Ron experiences one mood per day (as humans are known to do) and chooses one of ten genres of music to listen to that day depending on his mood.

Ron's roommate, who is known to take to odd hobbies, is interested in how Ron's mood affects his music selection, and thus collects data on Ron's mood and music selection for six years (2190 data points). This data is contained in the supplementary file `ron.txt`. Each row contains two tab-delimited strings: Ron's mood and Ron's genre preference that day. The data is split into 12 sequences, each corresponding to half a year's worth of observations. The sequences are separated by a row containing only the character `-`.

Problem C [10 points]: Use a single M-step to train a supervised Hidden Markov Model on the data in `ron.txt`. What are the learned state transition and output emission matrices?

Solution C.:



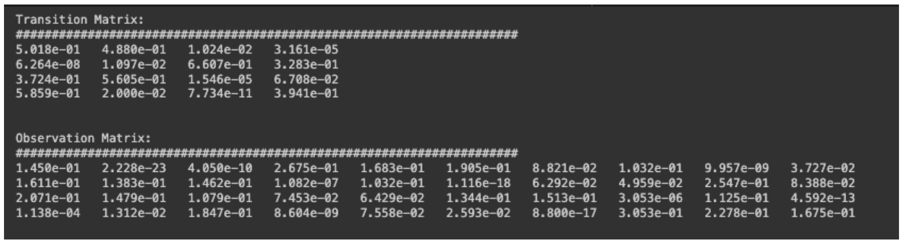
```
Transition Matrix:
######################################################################
2.833e-01   4.714e-01   1.310e-01   1.143e-01
2.321e-01   3.810e-01   2.940e-01   9.284e-02
1.040e-01   9.760e-02   3.696e-01   4.288e-01
1.883e-01   9.903e-02   3.052e-01   4.075e-01


Observation Matrix:
######################################################################
1.486e-01   2.288e-01   1.533e-01   1.179e-01   4.717e-02   5.189e-02   2.830e-02   1.297e-01   9.198e-02   2.358e-03
1.062e-01   9.653e-03   1.931e-02   3.089e-02   1.699e-01   4.633e-02   1.409e-01   2.394e-01   1.371e-01   1.004e-01
1.194e-01   4.299e-02   6.529e-02   9.076e-02   1.768e-01   2.022e-01   4.618e-02   5.096e-02   7.803e-02   1.274e-01
1.694e-01   3.871e-02   1.468e-01   1.823e-01   4.839e-02   6.290e-02   9.032e-02   2.581e-02   2.161e-01   1.935e-02
```

Problem D [15 points]: Now suppose that Ron has a third roommate who is also interested in how Ron's mood affects his music selection. This roommate is lazier than the other one, so he simply steals the first roommate's data. Unfortunately, he only manages to grab half the data, namely, Ron's choice of music for each of the 2190 days.

In this problem, we will train an unsupervised Hidden Markov Model on this data. Recall that unsupervised HMM training is done using the Baum-Welch algorithm and will require repeated EM steps. For this problem, we will use 4 hidden states and run the algorithm for 1000 iterations. The transition and observation matrices are initialized for you in the helper functions `supervised_learning()` and `unsupervised_learning()` such that they are random and normalized.

What are the learned state transition and output emission matrices? **(We will make a Piazza post about the seeds you should use. Please report the results with the specified seeds.)**

Solution D.:



```
Transition Matrix:
######################################################################
5.018e-01   4.880e-01   1.024e-02   3.161e-05
6.264e-08   1.097e-02   6.607e-01   3.283e-01
3.724e-01   5.605e-01   1.546e-05   6.708e-02
5.859e-01   2.000e-02   7.734e-11   3.941e-01


Observation Matrix:
######################################################################
1.450e-01   2.228e-23   4.050e-10   2.675e-01   1.683e-01   1.905e-01   8.821e-02   1.032e-01   9.957e-09   3.727e-02
1.611e-01   1.383e-01   1.462e-01   1.082e-07   1.032e-01   1.116e-18   6.292e-02   4.959e-02   2.547e-01   8.388e-02
2.071e-01   1.479e-01   1.079e-01   7.453e-02   6.429e-02   1.344e-01   1.513e-01   3.053e-06   1.125e-01   4.592e-13
1.138e-04   1.312e-02   1.847e-01   8.604e-09   7.558e-02   2.593e-02   8.800e-17   3.053e-01   2.278e-01   1.675e-01
```

Problem E [5 points]: How do the transition and emission matrices from 2C and 2D compare? Which do you think provides a more accurate representation of Ron's moods and how they affect his music choices? Justify your answer. Suggest one way that we may be able to improve the method (supervised or unsupervised) that you believe produces the less accurate representation.

Solution E.:

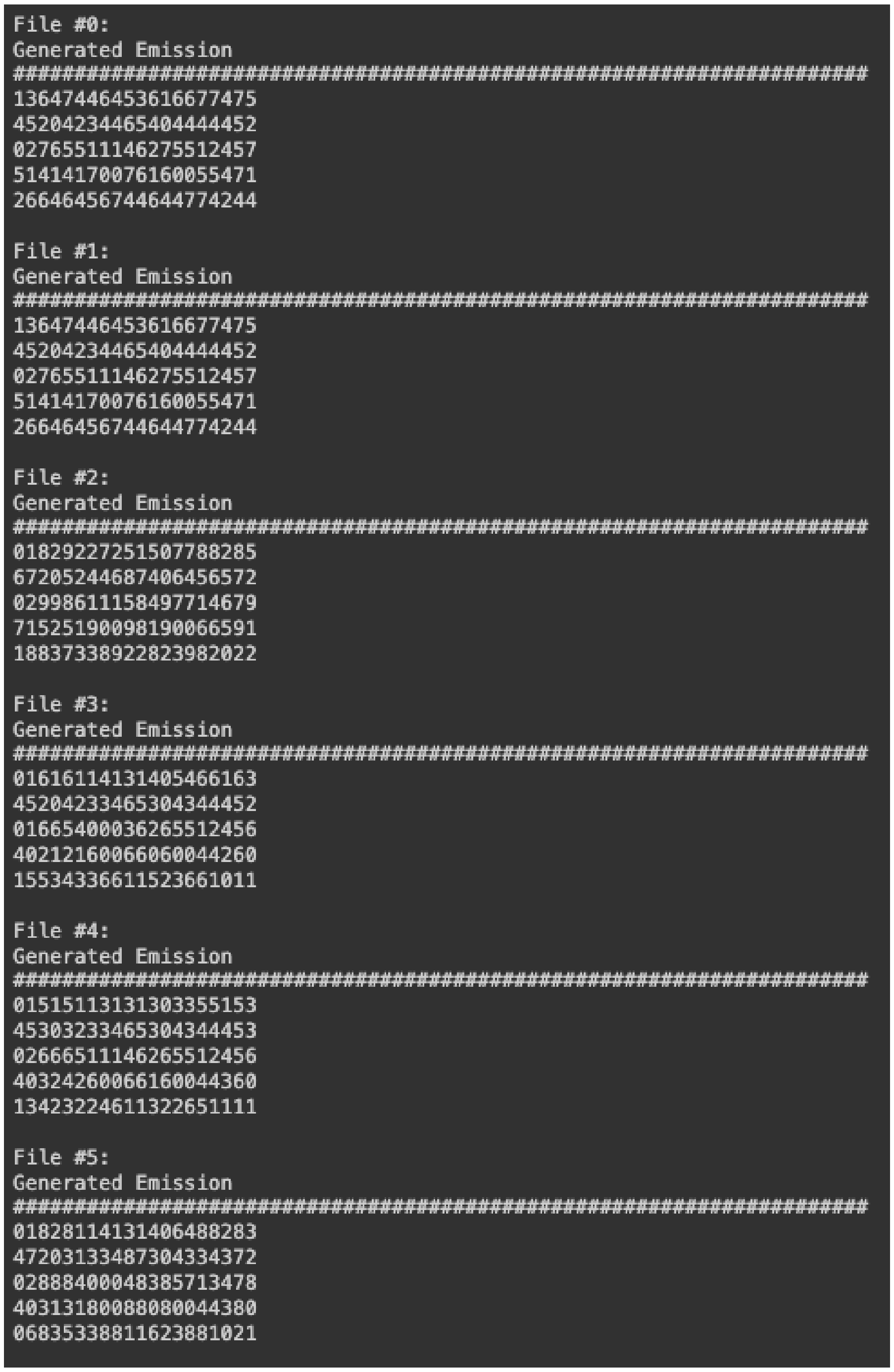
The transition matrix from 2C have values on the order of around 10^{-1} to 10^{-2}. On the other hand, the transition matrix from 2D are more variable, ranging from values around 10^{-1} to 10^{-11}. I would think that the matrix values from 2C would provide a more accurate representation because it does not make much sense to have values that are basically 0. One way that we may be able to improve the method in 2D is to provide more data to the model.

Sequence Generation

Hidden Markov Models fall under the umbrella of generative models and therefore can be used to not only predict sequential data, but also to generate it.

Problem F [5 points]: Load the trained HMMs from the files titled `sequence_data0.txt,...,sequence_-data5.txt`. Use the six models to probabilistically generate five sequences of emissions from each model, each of length 20. Show your results.

Solution F.:



```
File #0:
Generated Emission
######################################################################
13647446453616677475
45204234465404444452
02765511146275512457
51414170076160055471
26646456744644774244

File #1:
Generated Emission
######################################################################
13647446453616677475
45204234465404444452
02765511146275512457
51414170076160055471
26646456744644774244

File #2:
Generated Emission
######################################################################
01829227251507788285
67205244687406456572
02998611158497714679
71525190098190066591
18837338922823982022

File #3:
Generated Emission
######################################################################
01616114131405466163
45204233465304344452
01665400036265512456
40212160066060044260
15534336611523661011

File #4:
Generated Emission
######################################################################
01515113131303355153
45303233465304344453
02666511146265512456
40324260066160044360
13423224611322651111

File #5:
Generated Emission
######################################################################
01828114131406488283
47203133487304334372
02888400048385713478
40313180088080044380
06835338811623881021
```

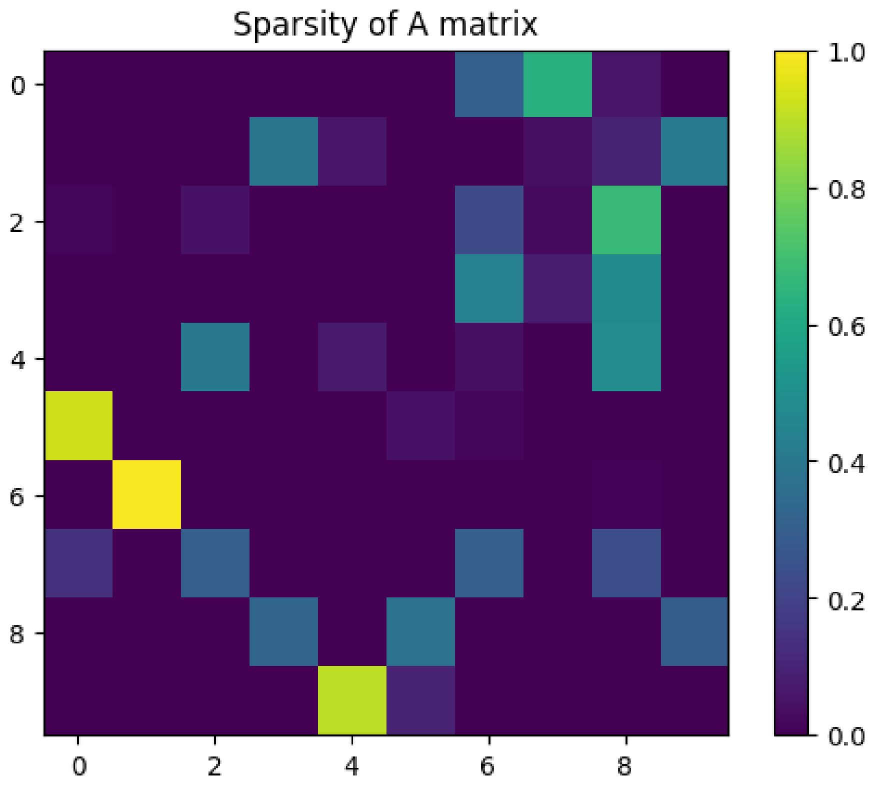
Visualization & Analysis

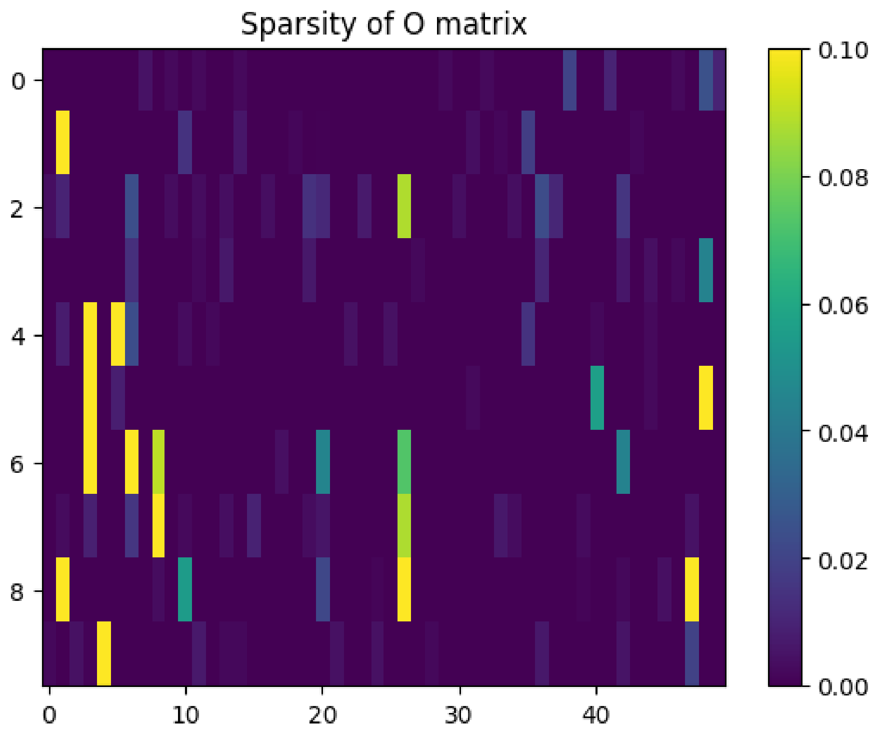
Once you have implemented the above, load and run `2_notebook.ipynb`. In this notebook, you will apply the HMM you have implemented to the Constitution. There is no coding required for this part, only analysis. To run the notebook, however, you will likely need to install the `wordcloud` package. Please refer to the provided installation instructions if you get an error when running `pip install wordcloud`.

Answer the following problems in the context of the visualizations in the notebook.

Problem G [3 points]: What can you say about the sparsity of the trained A and O matrices? How does this sparsity affect the transition and observation behaviour at each state?

Solution G.:





We can see from the figures that the sparsity of the trained A and O matrices are very high. Most of the matrices have values close to 0 (dark purple). The A matrix seems to be a little less sparse, showing more blue and green values than the O matrix. Overall, these matrices show that each state has a limited transition and observation behavior.

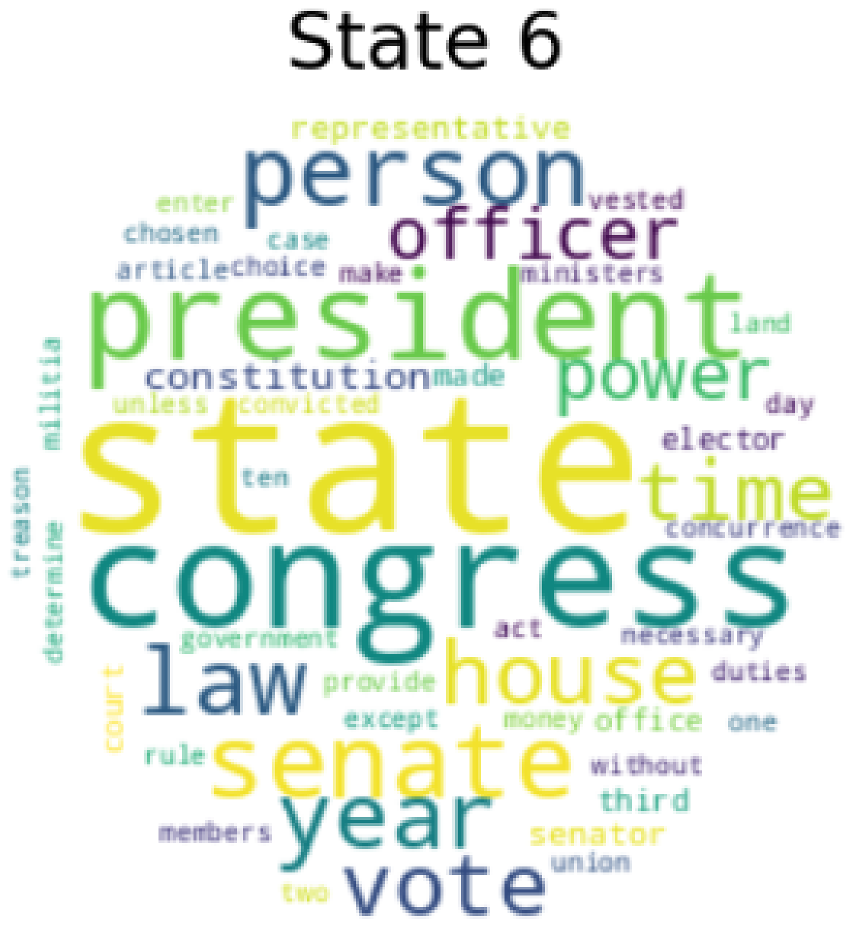
Problem H [5 points]: How do the sample emission sentences from the HMM change as the number of hidden states is increased? What happens in the special case where there is only one hidden state? In general, when the number of hidden states is unknown while training an HMM for a fixed observation set, can we increase the training data likelihood by allowing more hidden states?

Solution H.:

Looking at the sample emission sentences, as the number of hidden states increases, the sentence begins to make more sense. The start to sound like actual sentences both grammatically and logically. That being said, even the HMM with 16 hidden states is not able to produce a sentence that a human would make. In the case where there is only one hidden state, we see a string of random words. In general, when the number of hidden states is unknown while training an HMM, we can increase the training data likelihood by allowing more hidden states.

Problem I [5 points]: Pick a state that you find semantically meaningful, and analyze this state and its wordcloud. What does this state represent? How does this state differ from the other states? Back up your claim with a few key words from the wordcloud.

Solution I.:



I found state 6 to be semantically meaningful. In this state, we see many major terms in the constitution such as 'president', 'state', 'congress', 'law', etc. This state seems to represent a section in the constitution that deals with the roles and responsibilities of various positions in the government. It differs from other states as it has most of the words pertaining to the different parts of how the government is structured such as 'state', 'congress', 'law', and 'house'.