1 Class-Conditional Densities for Binary Data

Problem A:
We know that

$$p(x|y) = \prod_{i=1}^{D} P(x_j|x_{i,\dots,j-1}, y = c) = \prod_{j=1}^{D} \theta_{xjc} \tag{1}$$

and we want to use the chain rule of probability to factorize $p(x|y)$. If we store each θ_{xjc}, we know that the probability depends on $j-1$ binary features. So then we need 2^{j-1} parameters. Then we get that

$$\sum_{j=1}^{D} C2^{j-1} = C(2^D - 1) \tag{2}$$

which is the same thing as $O(C2^D)$.

Problem B:
If we assume there was no factorization and simply the joint probability $p(x|y = c)$ then we would need to store all possible combinations of $P(x|y = c)$ (as in, store all the estimations). We know that we have D x features and one class feature with C classes so there must be $C2^D$ potential combinations, meaning our answer would be the same as Problem A, $O(C2^D)$

Problem C:
We know that having a smaller training set means there wouldn't be enough data for a full model to cover the entirety of its probability estimates; thus, having a smaller training set would mean that the full model would be prone to overfitting and/or high testing error because of this. Contrastly, the naive model would most likely be more accurate since it doesn't need as much data; it only needs to have the probability of observing one feature and thus would be more likely to give lower test set error.

Problem D:
Because of what we mentioned in the previous problem, if we had a much larger training set then the more accurate model would be the full model for aforementioned reasons. The naive model would be less accurate in comparison to the full model because the naive model relies heavily on strong assumption on independence and thus, using it could lead to underfitting. Contrastly, now that the full model has enough data to cover the entirety of its probability estimates, it would perform much better than the other model would on the test set. Thus, the full model would be more likely to give lower test set error.

Problem E:
We want to find the computational complexity of making a prediction using the Naive Bayes model as well as the computation complexity of making a prediction with the full model. We will first examine the Naive Bayes:

We know that we have a uniform class prior, as mentioned in the question. This means that calculating $P(y)$ must then simply be the retrieval of a probability, which has a lookup time of $O(1)$; and we have already estimated our values for D number of probabilities. We also know

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{P(y)}{P(x)} \prod_{j=1}^{D} P(x_j|y = c) \tag{3}$$

and to find $\prod_{j=1}^{D} P(x_j|y = c)$ would be a complexity of $O(D)$. Given $P(x)$, the probability of observing some set of features would be the sum of the probability for observing that same set of features under each classification; we also know that, given some class, two things are equal: the probability of observing a set of features and the product of probability of observing each feature. So then given C class, our total complexity would be $O(CD)$. Now we will examine the full model.

For the full model we know that $P(y)$ has a complexity again of $O(1)$ (as mentioned above) and that $P(x|y)$ would again be $O(D)$. When we examine $P(x)$ we see that we only have to convert the D-bit vector into an array index and, from there, can simply look up the values needed from our table of probabilities (calculated previous), meaning we only need $C + D$ computations. Since $C < D$, we can see that the complexity of the calculation would be the sum of the previously mentioned parts, for a total of $O(D)$.

2 Sequence Prediction

Problem A:

```
                    Running Code For Question 2A
########################################################################


File #0:
Emission Sequence             Max Probability State Sequence
########################################################################
25421                         31033
01232367534                   22222100310
5452674261527433              1031003103222222
7226213164512267255           1310331000033100310
024712060235205101025524l     22222222222222222222103


File #1:
Emission Sequence             Max Probability State Sequence
########################################################################
77550                         22222
7224523677                    2222221000
505767442426747               222100003310031
72134131645536112267          10310310000310333100
473366777145005106025304l     22210000032222231032222 3


File #2:
Emission Sequence             Max Probability State Sequence
########################################################################
60622                         11111
4687981156                    2100202111
815833657775062               021011111111111
21310222515963505015          02020111111111111021
650319945257127400632002 5    11102021111111020211102 11


File #3:
Emission Sequence             Max Probability State Sequence
########################################################################
13661                         00021
2102213421                    3131310213
166066262165133               133333133133100
53164662112162634156          20000021313131002133
152354100512323022630625 6    13100213331331333131331 33


File #4:
Emission Sequence             Max Probability State Sequence
########################################################################
23664                         01124
3630535602                    0111201112
350201162150142               011244012441112
00214005402015146362          11201112412444011112
21112665246651435625344 50    20120124241240111124112 4


File #5:
Emission Sequence             Max Probability State Sequence
########################################################################
68535                         10111
4546566636                    1111111111
638436858181213               110111010000011
13240338308444514688          00010000000111111100
01116644344413825336326 26    21111111111111100111110 101
```

Problem B:
i.

```
######################################################################
                    Running Code For Question 2Bi
######################################################################


File #0:
Emission Sequence              Probability of Emitting Sequence
######################################################################
25421                          4.537e-05
01232367534                    1.620e-11
5452674261527433               4.348e-15
7226213164512267255            4.739e-18
024712060235205101025524l      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
473366777145005106025304l      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
650319945257127400632002l      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
152354100512323022630625l      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
211126652466514356253445l      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
011166443444138253363262l      1.440e-22
```

ii.

```
########################################################################
                  Running Code For Question 2Bii
########################################################################


File #0:
Emission Sequence             Probability of Emitting Sequence
########################################################################
25421                         4.537e-05
01232367534                   1.620e-11
545267426152743 3             4.348e-15
7226213164512267255           4.739e-18
024712060235205101025524 1    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
473366777145005106025304 1    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
650319945257127400632002 5    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
152354100512323022630625 6    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
211126652466514356253445 0    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
011166443444138253363262 6    1.440e-22
```

Problem C:

```
[########################################################################
                    Running Code For Question 2C
########################################################################


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


Melbas-MacBook-Pro:code melba$
```

Problem D:

```
######################################################################
                    Running Code For Question 2D
######################################################################


Transition Matrix:
######################################################################
8.029e-04   5.274e-01   8.376e-05   4.717e-01
1.856e-03   6.830e-01   2.922e-01   2.303e-02
6.214e-01   8.278e-13   3.747e-01   3.940e-03
2.051e-02   7.025e-01   2.042e-02   2.566e-01


Observation Matrix:
######################################################################
1.577e-01   4.863e-02   1.835e-01   4.863e-02   2.072e-01   3.686e-21   6.874e-02
1.120e-01   5.788e-02   1.132e-01   1.021e-01   1.309e-01   9.003e-02   8.507e-14
9.063e-02   7.887e-02   1.014e-16   1.937e-01   6.975e-02   2.191e-01   1.481e-01
3.079e-01   1.320e-01   9.116e-02   3.171e-02   5.488e-03   1.068e-06   2.835e-01


Melbas-MacBook-Pro:code melba$
```

Problem E:
Comparing parts 2C to 2D, we know that 2C trained with completely supervised data. Part 2D was trained with half of that data. This suggests that the matrices from 2C are more accurate. Additionally, 2C uses labelled data, which would make more sense since there's more information and therefore more accuracy with regards to the data. To make the unsupervised learning method more accurate, we could increase the size of the training data set or also implement regularization.

Problem F:

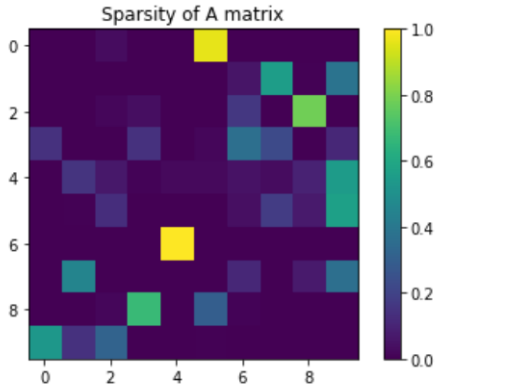
```
######################################################################
                    Running Code For Question 2F
######################################################################


File #0:
Generated Emission
######################################################################
77005277124641150552
77023225462354521711
36553710450573164057
44450325651465615456
50716577405257525776


File #1:
Generated Emission
######################################################################
75644574521045773052
57472440771501505561
74737027167362666242
55571745350552555370
75433253546557471274


File #2:
Generated Emission
######################################################################
508794655400519559 69
05960926914731758033
24633132722777240957
76712181825802926955
66034006144712667103


File #3:
Generated Emission
######################################################################
16666260252024632161
02260360402211513433
46643452121626365265
64143301156461210011
65640256621314420100


File #4:
Generated Emission
######################################################################
14556666016104436666
54426543156652444666
63601634426463633246
02345451004561602323
36366630306634361466


File #5:
Generated Emission
######################################################################
31345883461036024232
38533238655851443834
00641608015644338645
46586261043411444501
16600180480486480428


Melbas-MacBook-Pro:code melba$
```

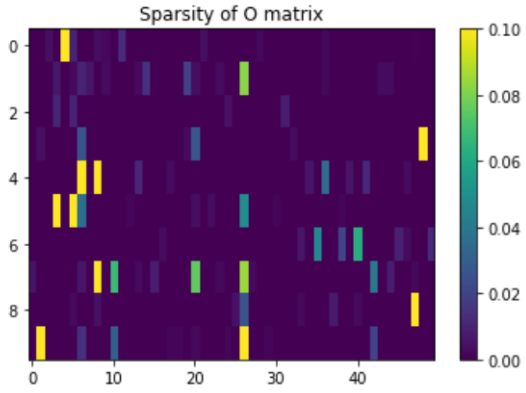
Problem G:
After visualizing the sparsities of the A and O matrices, we see that in the O matrix, sparsity increases along the x-axis, i.e. there is a higher concentration of yellow rectangles (value .10) closer to the inner corner of the O matrix.

Part G: Visualization of the sparsities of A and O

We can visualize the sparsities of the A and O matrices by treating the matrix entries as intensity values and showing them as images. Wh
notice?

```
In [43]: visualize_sparsities(hmm8, O_max_cols=50)
```





Problem H:

Part H: Using varying numbers of hidden states

Using different numbers of hidden states can lead to different behaviours in the HMMs. Below, we train several HMMs with 1, 2, 4, and 16 hidden states, respectively. What do you notice about their emissions? How do these emissions compare to the emission above?

In [61]:
```
hmm1 = unsupervised_HMM(obs, 1, 100)
print('\nSample Sentence:\n====================')
print(sample_sentence(hmm1, obs_map, n_words=25))
```

```
Sample Sentence:
====================
The union on its vice or office to laws made a and legislature the electors of but each on offices on first existing
be law...
```

In [62]:
```
hmm2 = unsupervised_HMM(obs, 2, 100)
print('\nSample Sentence:\n====================')
print(sample_sentence(hmm2, obs_map, n_words=25))
```

```
Sample Sentence:
====================
Third be constitution both he treason first any be so the department the power 1 public the following ii the senator
work but electors i...
```

In [63]:
```
hmm4 = unsupervised_HMM(obs, 4, 100)
print('\nSample Sentence:\n====================')
print(sample_sentence(hmm4, obs_map, n_words=25))
```

```
Sample Sentence:
====================
Title the receipts the direct sealed united another particular it and this senate one as states such enter may adjour
nment congress shall the it he...
```

In [64]:
```
hmm16 = unsupervised_HMM(obs, 16, 100)
print('\nSample Sentence:\n====================')
print(sample_sentence(hmm16, obs_map, n_words=25))
```

```
Sample Sentence:
====================
Such preserve before determined emolument class except or but the recess of exceeding state for a president shall be
holding order which which exported not...
```

We see that more states sentences make more sense; with 16 hidden states, the sentence generated generally follows proper grammar, whereas with only 1 hidden state, the sentence had articles in improper places. Increasing the number of states improves the fit in terms of likelihood, since more degrees of freedom improves the error rate. In general, when the number of hidden states is unknown, we can increase the training data likelihood by allowing for more hidden states, for reasons mentioned above.

Problem I:

Out of all of the states, I found State 6 to be the most semantically interesting. While most of the other states mentioned words such as 'constitution,' 'President,' and other structural nouns, this particular state included many words relating to individualism and rights: words such as 'entitled', 'inferior' and 'power', as well as 'vested' and 'attained.' This state mostly represents topics including inalienable rights and distinctions between classes of people (certain groups were considered 'inferior' and other groups had more 'power').

Problem 2

In this Jupyter notebook, we visualize how HMMs work. This visualization corresponds to problem 2 in set 6.

Assuming your HMM module is complete and saved at the correct location, you can simply run all cells in the notebook without modification.

In [37]:
```
import os
import numpy as np
from IPython.display import HTML

#from HMM import unsupervised_HMM
from HMM_helper import (
    text_to_wordcloud,
    states_to_wordclouds,
    parse_observations,
    sample_sentence,
    visualize_sparsities,
    animate_emission
)
```

For some reason, my HMM wouldn't import the updated version of my code so I copy-pasted it below. I removed a lot of the red method descriptions because my code wouldn't fit in otherwise.

In [ ]:
```
class HiddenMarkovModel:
    '''
    Class implementation of Hidden Markov Models.
    '''

    def __init__(self, A, O):
        self.L = len(A)
        self.D = len(O[0])
        self.A = A
        self.O = O
        self.A_start = [1. / self.L for _ in range(self.L)]


    def viterbi(self, x):
        M = len(x)      # Length of sequence.

        # The (i, j)^th elements of probs and seqs are the max probability
        # of the prefix of length i ending in state j and the prefix
        # that gives this probability, respectively.
        #
        # For instance, probs[1][0] is the probability of the prefix of
        # length 1 ending in state 0.
        probs = [[0. for _ in range(self.L)] for _ in range(M + 1)]
        seqs = [['' for _ in range(self.L)] for _ in range(M + 1)]
        t = []
        # initialize row 2 of probs and seqs as 0
        for i in range(self.L):
            probs[1][i] = self.A_start[i] * self.O[i][x[0]]

        # start w row 2 and use top-down approach
        for row in range(2, M+1):
            for col1 in range(self.L):
                temp = []
                for col2 in range(self.L):
                    temp.append(probs[row-1][col2] * self.O[col1][x[row-1]] *
                                self.A[col2] [col1])
                max_index, max_value = max(enumerate(temp), key=operator.itemgette
r(1))
                probs[row][col1] = max_value
                seqs[row][col1] = seqs[row-1][max_index] + str(max_index)

        max_index, max_value = max(enumerate(probs[M]), key=operator.itemgetter
(1))
        max_seq = seqs[len(probs)-1][max_index] + str(max_index)

        return max_seq
```

In [ ]:
```
    def forward(self, x, normalize=False):
        M = len(x)      # len of sequence

        if normalize:
            alphas = [[0. for i in range(self.L)] for j in range(M + 1)]

            # initialize row of alphas
            for i in range(self.L):
                alphas[1][i] = self.O[i][x[0]] * self.A_start[i]

            for row in range(2, M+1):
                for col in range(self.L):
                    temp = []
                    for col2 in range(self.L):
                        temp.append(alphas[row-1][col2] * self.O[col][x[row-1]] *
self.A[col2] [col])
                    alphas[row][col] = sum(temp)
                norm = sum(alphas[row])
                for nCol in range(self.L):
                    alphas[row][nCol] /= norm
            return alphas

        alphas = [[0. for i in range(self.L)] for j in range(M + 1)]

        # initialize first row of alphas
        for i in range(self.L):
            alphas[1][i] = self.O[i][x[0]] * self.A_start[i]

        for row in range(2, M+1):
            for col in range(self.L):
                temp = []
                for col2 in range(self.L):
                    temp.append(alphas[row-1][col2] * self.O[col][x[row-1]] *
                                self.A[col2] [col])
                alphas[row][col] = sum(temp)
        return alphas
```

In [ ]:
```
    def backward(self, x, normalize=False):
        M = len(x)      # len of sequence
        if normalize:
            betas = [[0. for i in range(self.L)] for j in range(M + 1)]

            for i in range(self.L):
                betas[M][i] = 1

            for row in range(M-1, 0, -1):
                for col in range(self.L):
                    for col2 in range(self.L):
                        betas[row][col] += (betas[row+1][col2] *
                                            self.O[col2][x[row]] * self.A[col] [co
l2])
                norm = sum(betas[row])
                for nCol in range(self.L):
                    betas[row][nCol] /= norm
            return betas

        betas = [[0. for i in range(self.L)] for j in range(M + 1)]

        for i in range(self.L):
            betas[M][i] = 1

        for row in range(M-1,0, -1):
            for col in range(self.L):
                for col2 in range(self.L):
                    betas[row][col] += (betas[row+1][col2] * self.O[col2][x[row]]
*
                                        self.A[col] [col2])
        return betas
```

In [ ]:
```
    def count_transitions(self, a, b, X, Y):
        '''
        y_i^j = b
        and y_i^(j-1) = a
        '''
        den = 0
        num = 0

        for i in range(len(X)):
            for j in range(1, len(X[i])):
                if Y[i][j-1] == a:
                    den += 1
                    if Y[i][j] == b:
                        num += 1

        return num, den

    def count_observations(self, w, a, X, Y):
        '''
        y_i^j = w
        and x_i^j = w
        '''
        den = 0
        num = 0

        for i in range(len(X)):
            for j in range(len(X[i])):
                if Y[i][j] == a:
                    den += 1
                    if X[i][j] == w:
                        num += 1
        return num, den
```

In [ ]:
```
    def supervised_learning(self, X, Y):
        for a in range(self.L):
            for b in range(self.L):
                num, den = self.count_transitions(a, b, X, Y)
                self.A[a][b] = num / den

        for a in range(len(self.O)):
            for w in range(len(self.O[0])):
                num, den = self.count_observations(w, a, X, Y)
                self.O[a][w] = num / den

        pass
```

In [ ]:
```
def unsupervised_learning(self, X, N_iters):
        for x in range(N_iters):
            A_den = [0. for i in range(self.L)]
            O_den = [0. for i in range(self.L)]

            A_num = [[0. for i in range(self.L)] for i in range(self.L)]
            O_num = [[0. for i in range(self.D)] for i in range(self.L)]

            for sequence in X:
                M = len(sequence)

                alpha = self.forward(sequence, normalize=True)
                beta = self.backward(sequence, normalize=True)

                for w in range(1, M+1):
                    temp = [0. for i in range(self.L)]
                    for z in range(self.L):
                        temp[z] = alpha[w][z] * beta[w][z]

                    # normalize
                    norm = sum(temp)

                    for z in range(len(temp)):
                        temp[z] /= norm

                    for z in range(self.L):
                        if w != M:
                            A_den[z] += temp[z]
                        O_num[z][sequence[w-1]] += temp[z]
                        O_den[z] += temp[z]

                for w in range(1, M):
                    norm = 0

                    temp = [[0. for _ in range(self.L)] for _ in range(self.L)]

                    for i in range(self.L):
                        for j in range(self.L):
                            temp[i][j] = alpha[w][i] * self.A[i][j]
                            * self.O[j][sequence[w]] * beta[w+1][j]

                    for row in temp:
                        norm += sum(row)

                    for i in range(self.L):
                        for j in range(self.L):
                            temp[i][j] /= norm

                    for i in range(self.L):
                        for j in range(self.L):
                            A_num[i][j] += temp[i][j]

            for i in range(self.L):
                for j in range(self.L):
                    self.A[i][j] = A_num[i][j] / A_den[i]

            for i in range(self.L):
                for j in range(self.D):
                    self.O[i][j] = O_num[i][j] / O_den[i]
```

In [ ]:

```
    def generate_emission(self, M):
        emission = []
        states = []
        state = random.choice(range(self.L))

        for _ in range(M):
            states.append(state)

            t = self.A[state]
            o = self.O[state]

            emission.append(int(np.random.choice(range(self.D), 1, p=o)))

            state = int(np.random.choice(range(self.L), 1, p=t))

        return emission, states


    def probability_alphas(self, x):
        # Calculate alpha vectors.
        alphas = self.forward(x)

        # alpha_j(M) gives the probability that the state sequence ends
        # in j. Summing this value over all possible states j gives the
        # total probability of x paired with any state sequence, i.e.
        # the probability of x.
        prob = sum(alphas[-1])
        return prob


    def probability_betas(self, x):
        betas = self.backward(x)

        # beta_j(1) gives the probability that the state sequence starts
        # with j. Summing this, multiplied by the starting transition
        # probability and the observation probability, over all states
        # gives the total probability of x paired with any state
        # sequence, i.e. the probability of x.
        prob = sum([betas[1][j] * self.A_start[j] * self.O[j][x[0]] \
                    for j in range(self.L)])

        return prob
```

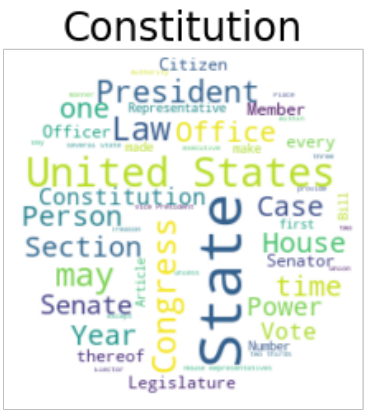
In [ ]:
```
def supervised_HMM(X, Y):
    # Make a set of observations.
    observations = set()
    for x in X:
        observations |= set(x)

    # Make a set of states.
    states = set()
    for y in Y:
        states |= set(y)

    # Compute L and D.
    L = len(states)
    D = len(observations)

    # Randomly initialize and normalize matrix A.
    A = [[random.random() for i in range(L)] for j in range(L)]

    for i in range(len(A)):
        norm = sum(A[i])
        for j in range(len(A[i])):
            A[i][j] /= norm

    # Randomly initialize and normalize matrix O.
    O = [[random.random() for i in range(D)] for j in range(L)]

    for i in range(len(O)):
        norm = sum(O[i])
        for j in range(len(O[i])):
            O[i][j] /= norm

    # Train an HMM with labeled data.
    HMM = HiddenMarkovModel(A, O)
    HMM.supervised_learning(X, Y)

    return HMM

def unsupervised_HMM(X, n_states, N_iters):
    # Make a set of observations.
    observations = set()
    for x in X:
        observations |= set(x)

    # Compute L and D.
    L = n_states
    D = len(observations)

    # Randomly initialize and normalize matrix A.
    random.seed(2020)
    A = [[random.random() for i in range(L)] for j in range(L)]

    for i in range(len(A)):
        norm = sum(A[i])
        for j in range(len(A[i])):
            A[i][j] /= norm

    # Randomly initialize and normalize matrix O.
    random.seed(155)
    O = [[random.random() for i in range(D)] for j in range(L)]

    for i in range(len(O)):
        norm = sum(O[i])
        for j in range(len(O[i])):
            O[i][j] /= norm
```

In [ ]:

Visualization of the dataset

We will be using the Constitution as our dataset. First, we visualize the entirety of the Constitution as a wordcloud:

In [40]:
```
text = open(os.path.join(os.getcwd(), 'data/constitution.txt')).read()
wordcloud = text_to_wordcloud(text, title='Constitution')
```



Training an HMM

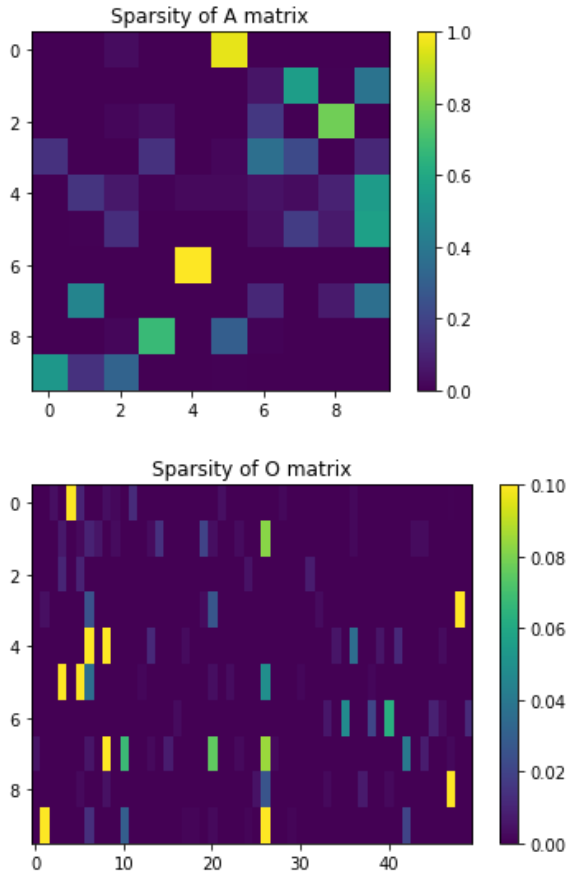
Now we train an HMM on our dataset. We use 10 hidden states and train over 100 iterations:

In [59]:
```
obs, obs_map = parse_observations(text)
hmm8 = unsupervised_HMM(obs, 10, 100)
```

Part G: Visualization of the sparsities of A and O

We can visualize the sparsities of the A and O matrices by treating the matrix entries as intensity values and showing them as images. What patterns do you notice?

In [43]:
```
visualize_sparsities(hmm8, O_max_cols=50)
```



Generating a sample sentence

As you have already seen, an HMM can be used to generate sample sequences based on the given dataset. Run the cell below to show a sample sentence based on the Constitution.

In [60]:
```
print('Sample Sentence:\n====================')
print(sample_sentence(hmm8, obs_map, n_words=25))
```

```
Sample Sentence:
====================
And state to georgia for seventeenth ambassadors period court the such power rel
igious the levying all the respective of necessary the number crimes their titl
e...
```

Part H: Using varying numbers of hidden states

Using different numbers of hidden states can lead to different behaviours in the HMMs. Below, we train several HMMs with 1, 2, 4, and 16 hidden states, respectively. What do you notice about their emissions? How do these emissions compare to the emission above?

In [61]:
```
hmm1 = unsupervised_HMM(obs, 1, 100)
print('\nSample Sentence:\n====================')
print(sample_sentence(hmm1, obs_map, n_words=25))
```

```
Sample Sentence:
====================
The union on its vice or office to laws made a and legislature the electors of b
ut each on offices on first existing be law...
```

In [62]:
```
hmm2 = unsupervised_HMM(obs, 2, 100)
print('\nSample Sentence:\n====================')
print(sample_sentence(hmm2, obs_map, n_words=25))
```

```
Sample Sentence:
====================
Third be constitution both he treason first any be so the department the power 1
public the following ii the senator work but electors i...
```

In [63]:
```
hmm4 = unsupervised_HMM(obs, 4, 100)
print('\nSample Sentence:\n====================')
print(sample_sentence(hmm4, obs_map, n_words=25))
```

```
Sample Sentence:
====================
Title the receipts the direct sealed united another particular it and this senat
e one as states such enter may adjournment congress shall the it he...
```

In [64]:
```
hmm16 = unsupervised_HMM(obs, 16, 100)
print('\nSample Sentence:\n====================')
print(sample_sentence(hmm16, obs_map, n_words=25))
```

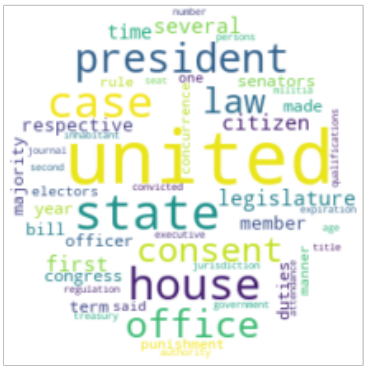
```
Sample Sentence:
====================
Such preserve before determined emolument class except or but the recess of exce
eding state for a president shall be holding order which which exported not...
```

Part I: Visualizing the wordcloud of each state

Below, we visualize each state as a wordcloud by sampling a large emission from the state:

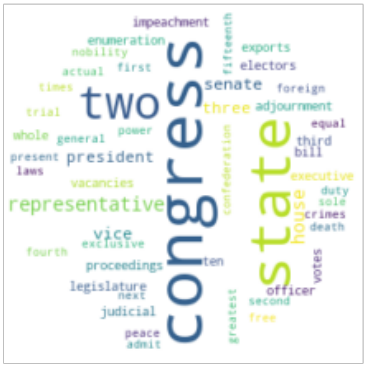
In [65]:
```
wordclouds = states_to_wordclouds(hmm8, obs_map)
```

State 0

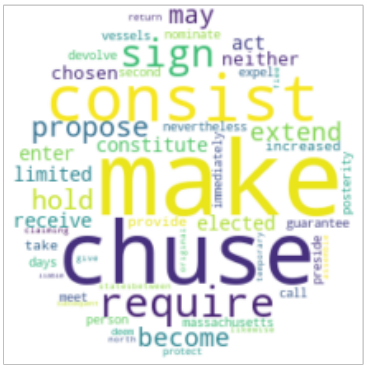


State 1

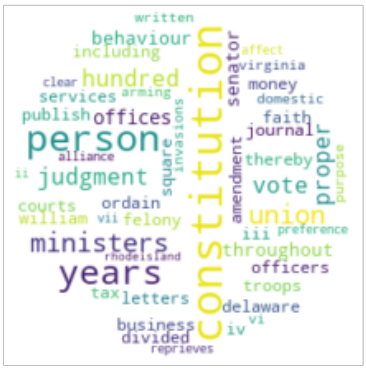
State 2



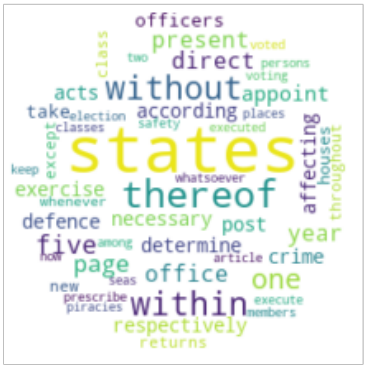
State 3



State 4

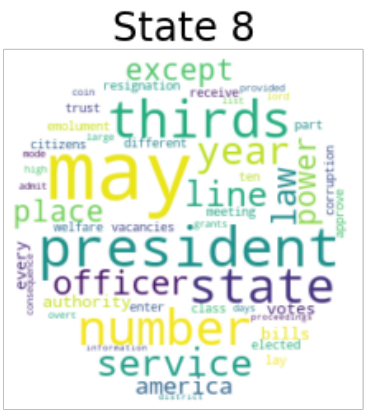


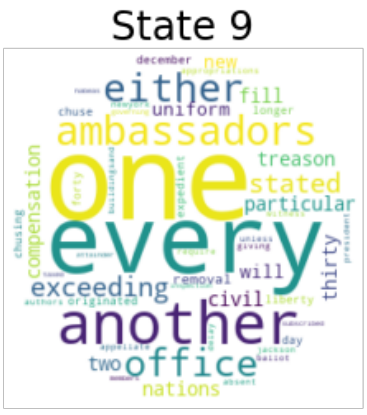
State 5



State 6

State 7





Visualizing the process of an HMM generating an emission

The visualization below shows how an HMM generates an emission. Each state is shown as a wordcloud on the plot, and transition probabilities between the states are shown as arrows. The darker an arrow, the higher the transition probability.

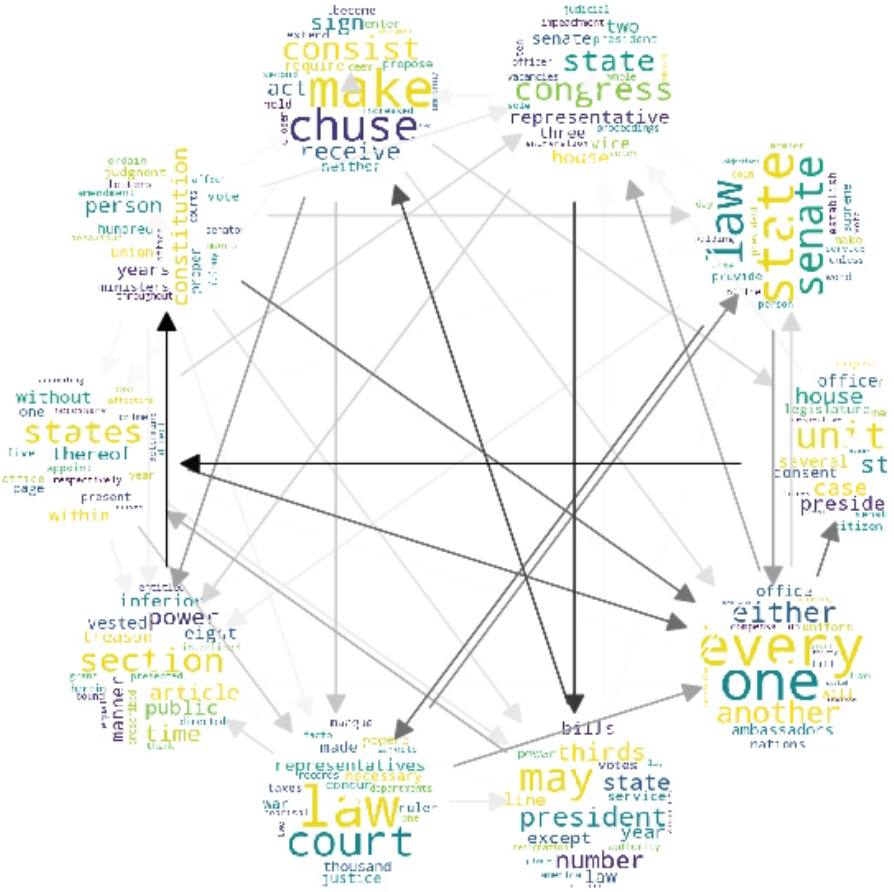
At every frame, a transition is taken and an observation is emitted from the new state. A red arrow indicates that the transition was just taken. If a transition stays at the same state, it is represented as an arrowhead on top of that state.

Use fullscreen for a better view of the process.

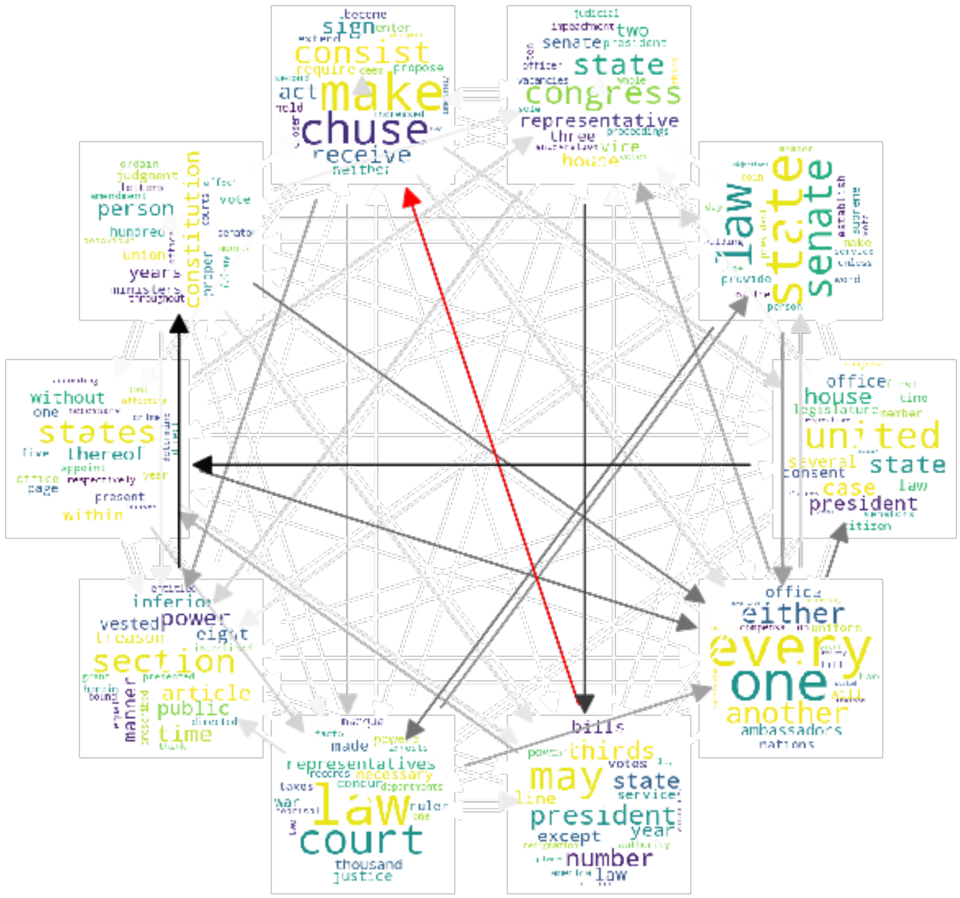
In [66]:
```
anim = animate_emission(hmm8, obs_map, M=8)
HTML(anim.to_html5_video())
```

```
Animating...
```

Out[66]:



Citizen of executing as will congress shall consist



In [ ]: