Machine Learning & Data Mining
Homework 6

Caltech CS/CNS/EE 155
February 2024

---

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.:

Notice that the number of classes will stay the same, so we will have the answer on the scale of $C * f(D)$ where $f(D)$ is sum term related to the D features. Now notice for the dependent model:

$$P(x|y) = P(x_j|x_{1,\ldots,j-1}, y = c) = P(x_j|x_{1,\ldots,j-1}, y = c) * P(x_{1,\ldots,j-1}, y = c)$$

Where

$$P(x_{1,\ldots,j-1}, y = c)$$

can be written out using the same recursive nature in terms of

$$P(j - 1|x_{1,\ldots,j-2}, y = c)$$

This will go on until $j = 1$ so we will get D summative products given by:

$$\prod_{j=1}^{D} P(x_j|x_{1,\ldots,j-1}, y = c)$$

Since we are storing each of these terms (since equivalent to θ_{xjc}) and each probability term needs the previous terms calculated using all possible permutations of each binary x_j for the case $j = D$, we will need parameters for each permutation of $D - 1$ features which is 2^{D-1} meaning that the parameters needed to represent the factorization are on the scale of $O(2^D)$ for each of the C classes, leading to the conclusion that to represent this factorization we will need on the scale of $O(C \cdot 2^D)$ parameters.

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.:

For an arbitrary x we would need to compute parameters for each setting of x which given D features and each x_j is binary would be on the scale of 2^D, this arbitrary x parameters would need to be computed for each class for a total of C classes, creating an overall computational complexity for estimating parameters of $O(C \cdot 2^D)$ which is the same as the answer from part A, which makes sense as for the parameters on x both methods need to compute parameters for all combinations of binary values of size $O(D)$.

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.:

The full model will have on the scale of $O(C \cdot 2^D)$ parameters to train on where as the Naive Bayes model will only have $O(CD)$, meaning the full model is much more complex. As a result, on smaller training data N, the more complex model is likely to overfit and create higher testing error, where as a simpler model would be favored when less training data is available meaning the Naive Bayes model would give a lower test set error.

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.:

The full model will have on the scale of $O(C \cdot 2^D)$ parameters to train on where as the Naive Bayes model will only have $O(CD)$, meaning the full model is much more complex. As a result, on larger training data N, the simpler model is likely to underfit and create higher testing error, where as the more complex model would be favored when larger set of training data is available meaning the full model would give a lower test set error.

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.:

We know that $p(y|x)$ for Naive Bayes on a single test case is computed as $\frac{P(x,y)}{P(x)}$ where $P(x,y)$ gives a scale of D computations for each feature x by C computations of each class to determine the likelihoods in the numerator since we need to compute only since we have an assumed uniform class prior. (the denominator can be simply computed looking at the data in linear time). Thus, we get that for a single test case of Naive Bayes, we only need $O(CD)$ computations. On the other hand for the full model, since all combinations of parameters are stored already, we only need to access the needed parameters by accessing the parameters at the necessary index, which is done by taking the D-bit vector x and converting it to an array in $O(D)$ but then accessing the parameters only takes $O(1)$ (since stored by the full model in training) yielding a total complexity of $O(D)$ for each of the D features, and multiplied by each of the C classes to find the likelihoods of each class for the full model giving a computational complexity of $O(CD)$ as well. So we see that making predictions with the full model is computationally the same expense as Naive Bayes, meaning that our decision on what model to use in training does not have to depend on complexity of the testing as well, since the testing complexity is the same for both models.

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.:

Code to solution

```
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
47336677714500510602530 41    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    13100213331331333133133 3

File #4:
Emission Sequence             Max Probability State Sequence
########################################################################
23664                         01124
3630535602                    0111201112
350201162150142               011244012441112
00214005402015146362          11201112412444011112
21112665246651435625345 0     20120124241240111124112 4

File #5:
Emission Sequence             Max Probability State Sequence
########################################################################
68535                         10111
4546566636                    1111111111
638436858181213               110111010000011
13240338308444514688          00010000000111111100
01116644344413825336326 6     21111111111111100111110 1
```

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	22222222222222222222222103	9.365e-24

Solution B:

Code to solution

Forward Algorithm followed by Backward Algorithm results:

```
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
721341316455361122 67         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
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
132403383084445146 88         4.629e-18
011166443444138253363262 6    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
```

Dataset	*Sequence*	*Max-probability state sequence*	*Probability of sequence*
1	*77550*	*22222*	*1.181e-04*
1	*7224523677*	*2222221000*	*2.033e-09*
1	*505767442426747*	*222100003310031*	*2.477e-13*
1	*72134131645536112267*	*10310310000310333100*	*8.871e-20*
1	*4733667771450051060253041*	*22210000032222231032222223*	*3.740e-24*
2	*60622*	*11111*	*2.088e-05*
2	*4687981156*	*2100202111*	*5.181e-11*
2	*815833657775062*	*021011111111111*	*3.315e-15*
2	*21310222515963505015*	*020201111111111111021*	*5.126e-20*
2	*6503199452571274006320025*	*11102021111111102021110211*	*1.297e-25*
3	*13661*	*00021*	*1.732e-04*
3	*2102213421*	*3131310213*	*8.285e-09*
3	*166066262165133*	*133333133133100*	*1.642e-12*
3	*53164662112162634156*	*20000021313131002133*	*1.063e-16*
3	*1523541005123230226306256*	*13100213331333133313133133*	*4.535e-22*
4	*23664*	*01124*	*1.141e-04*
4	*3630535602*	*0111201112*	*4.326e-09*
4	*350201162150142*	*011244012441112*	*9.793e-14*
4	*00214005402015146362*	*11201112412444011112*	*4.740e-18*
4	*2111266524665143562534450*	*20120124241240111124112411124*	*5.618e-22*
5	*68535*	*10111*	*1.322e-05*
5	*4546566636*	*1111111111*	*2.867e-09*
5	*638436858181213*	*110111010000011*	*4.323e-14*
5	*13240338308444514688*	*00010000000111111100*	*4.629e-18*
5	*0111664434441382533632626*	*21111111111111100111110101*	*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.:

Code to solution

```
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.:

Code to solution

```
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.:

Code to solution

Looking at the transition matrix of moods as well as the observation matrix of genres for both models, we notice that the unsupervised model pushes values in both matrices to near 0 where as the supervised learning model does not and retains some significant probability of each transition and observation. In a logical sense the supervised learning models results make more sense in that it is statistically unlikely that some of the transition and observation probabilities for the unsupervised learning model are so close and essentially zero that the observation or transition is basically non existent. Given only 4 moods in the state-space this makes even less sense as across 2190 data points each of the 16 transitions is very likely to be observed at least once. Thus, these observations suggest to conclude that the supervised learning model provides better accuracy of representation whereas the unsupervised model tends to be 'overly confident' in some of its probabilities. With unsupervised learning we can always improve with more data as it will allow us to generalize better and identify general trends/ create stronger intuition on the true probabilities of transitions and observations.

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.:

Code to solution

```
File #0:
Generated Emission
########################################################################
13616214141517467165
47413442477305255552
02776511157475644657
50325170077360166471
26646325712742762241

File #1:
Generated Emission
########################################################################
13616214141517467165
47413442477305255552
02776511157475644657
50325170077360166471
26646325712742762241

File #2:
Generated Emission
########################################################################
028292261515066882 75
682052346865064555 73
039766122573978236 69
723251900981900675 91
188474569228347720 21

File #3:
Generated Emission
########################################################################
016262141213054661 62
452021324552042243 51
016664110462645124 36
402121600661600444 61
155343456113236610 11

File #4:
Generated Emission
########################################################################
016262141404054662 63
462031224662042332 62
026665001363645134 66
403232600650600443 60
164343256223226621 12

File #5:
Generated Emission
########################################################################
018363253414065683 63
482031345864154443 63
038885011482766134 68
413233810881610554 80
068353468236236620 23
```

Visualization & Analysis

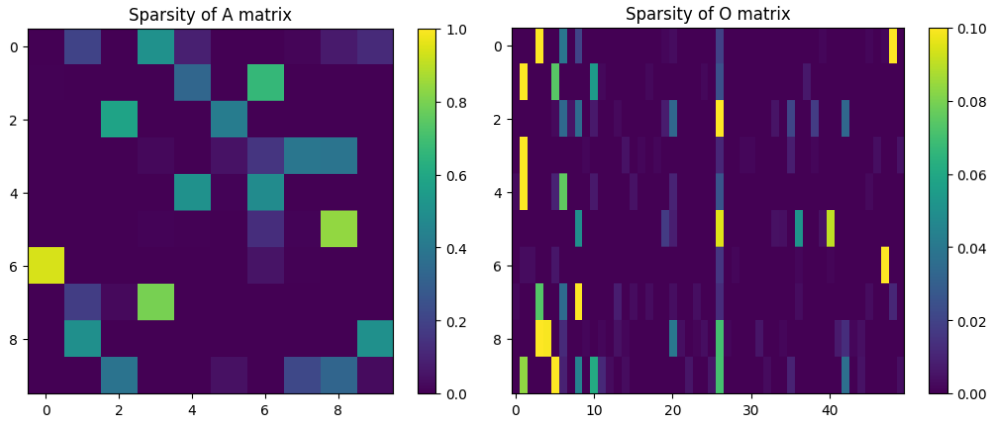
Once you have implemented the above, load and run `2_notebook.ipynb`. In this notebook, you will apply the HMM you have implemented to the Constitution. There is no coding required for this part, only analysis. To run the notebook, however, you will likely need to install the `wordcloud` package. Please 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.:

Code to solution



We can see from the graph that the sparsity of trained matrices A and O are relatively sparse with what seems to be over 50% of the entries being 0 or very close to 0. This would restrict the transition and observation space in that there are less possibilities of transitions and observations that can actually happen/ be generated compared to the entire theoretically possible size of the transition and observation space (since most entries and therefore probabilities of occurring are 0).

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.:

Code to solution

As the number of states increases for the HMM the emission sample sentence increases in its grammatical correctness and how much sense the sentence makes. For the special case of only one hidden state we notice that the type of word being generated at each consecutive step does not depend on the previous word or have any restriction on what type of word it is as all words generated are coming from the same state, this results in double verbs or nouns as well as repeat words that are grammatically incorrect and make little/ no sense. In general, allowing more hidden states can increase the training data likelihood because it allows for more parameters to be trained on and thus more flexibility in the model to maximize the likelihood of the training data beyond models with lesser hidden states (more degrees of flexibility to fit the data, note this can also risk overfitting).

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.:

Code to solution

State 4 has a strong concentration of adverbs and verbs compared to other states, such as execute, granted, provided, entitled, passed, deprived, presented, convicted, admit, faithfully, votes, respectively, regulate, will, etc. Notice these are all general action terms used in writing law, and are much more concentrated in this state compared to other states which are largely nouns. This suggests that state 4 represents verbs/ adverbs used in sentences of the Constitution specifically with describing laws and government structure as many of state 4's words would make a lot of sense in those contexts.