Policies

- Due 9 PM, March 2nd, via Moodle.
- You are free to collaborate on all of the problems, subject to the collaboration policy stated in the syllabus.
- You should submit all code used in the homework. We ask that you use Python 3 and sklearn version 0.19 for your code, and that you comment your code such that the TAs can follow along and run it without any issues.

Submission Instructions

Please submit your assignment as a .zip archive with filename `LastnameFirstname.zip` (replacing `Lastname` with your last name and `Firstname` with your first name), containing a PDF of your assignment writeup **in the main directory** with filename `LastnameFirstname_Set6.pdf` and your code files **in a directory named `LastnameFirstname`**. Failure to do so will result in a **2 point deduction**. Submit your code as Jupyter notebook `.ipynb` files or `.py` files, and **include any images generated by your code along with your answers in the solution .pdf file.**

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.

Problem A [10 points]: Now consider a different model, which we will call the 'full' model, in which all the features are fully *dependent*.

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

Solution A: *Given that there are D features, we get by the chain rule of probability*

$$p(x \mid y = c) = p(x_1 \mid y = c)p(x_2 \mid x_1, y = c)...p(x_D \mid x_{1:D-1}, y = c) = \theta_{x1c}\theta_{x2c} \dots \theta_{xDc}$$

as our representation of the full model. We would need to store all the θ_{xjc}'s. For each $j \in \{1, ..., D\}$ we need to store $C2^{j-1}$ parameters. Thus the total number would be

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

ii. [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 A: *The number of parameters needed here would be equivalent to storing C tables, each holding all possible D-dimensional bit vectors x and for each, the estimated value of $P(x \mid y = c)$. This means the number of parameters would be $O(C2^D)$. This is the same as the number of parameters we would be required to store if we used the factorized distribution, so there's no point in factoring it at all :D*

Problem B [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 B: *For small N, the naive Bayes model will outperform the full model, because the full model's parameter estimates will be inaccurate, and the model will overfit to the small dataset. Naive Bayes, being a simpler model, does not overfit as easily.*

Problem C [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 C: *For very large N, the opposite happens. The strong hypothesis of Naive Bayes will cause the model to underfit, and the full model, having more accurate estimates for its parameters, will be able to better perform on test data.*

Problem D [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 D: *For the Naive Bayes case, given a sample x, we need to compute*

$$p(y \mid x) = \frac{P(y)}{P(x)} \prod_{d=1}^{D} P(x_d \mid y)$$

Computing $P(y)$ is an $O(1)$ operation since we assume a uniform class prior. Computing $P(x_d \mid y)$ is also an $O(1)$ operation since we have stored these parameter estimates. Thus, computing $\prod_d P(x_d \mid y)$ is an $O(D)$ operation. However,

$$P(x) = \sum_{i=1}^{C} P(x \mid y = c_i) = \sum_{i=1}^{C} \prod_{d=1}^{D} P(x_d \mid y = c_i)$$

and the complexity of computing this probability is $O(CD)$.

The full model case is faster. We will have computed the class conditional probability of every string $p(x \mid y = c)$, and so we will just need to do a table lookup for the class, and we have to convert the D-bit vector to an array index first. The complexity thus will be $O(D)$.

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.py`, within the appropriate functions where 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.
- You can (and should!) use the helper files `2A.py`, `2Bi.py`, `2Bii.py`, `2C.py`, `2D.py`, and `2F.py`. These are scripts that can be used to run and check your implementations for each of the corresponding problems. The scripts provide useful output in an easy-to-read format. There is no need to modify these files.
- Lastly, `Utility.py` contains some functions used for loading data. There is no need to modify this file.

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 of `2A.py` suffices.)

Solution A: *See the solution code.*

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()`.

i. [10 points]: Implement the Forward algorithm. Show your results on the 6 files.

ii. [7 points]: 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:

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*	*2221000003222223103222223*	*3.740e-24*
2	*60622*	*11111*	*2.088e-05*
2	*4687981156*	*2100202111*	*5.181e-11*
2	*815833657775062*	*021011111111111*	*3.315e-15*
2	*21310222515963505015*	*02020111111111111021*	*5.126e-20*
2	*6503199452571274006320025*	*1110202111111102021110211*	*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*	*1310021333133133313133133*	*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*	*2012012424124011112411124*	*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*	*2111111111111100111110101*	*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?

Solution D: *Results may vary depending on your initialization. Some things to watch out for to check that your solution is correct:*

- *The rows of your matrices sum to 1.*
- *Your matrices don't change drastically every iteration.*
- *After many iterations, your matrices converge.*

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 matrices should be quite different - unsupervised learning is unstable and highly depends on the initialization. Supervised learning is preferred in this case as the model is explicitly given the states it needs to learn, whereas in unsupervised learning, the hidden states could be anything. Unsupervised learning can be improved in the following ways:*

- *Use semi-supervised learning*
- *Initialize the matrices with the results of supervised learning on a smaller dataset that you can label, or by projecting probabilities*
- *Alter the number of hidden states with some form of validation*

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: *Results may vary.*

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: *Most elements of A and O are very close to 0. For the A matrix, this implies that at each state, there are only a few states to which the HMM will likely transition. For the O matrix, this implies that at each state, a few key observations are likely to be emitted. Usually, these correspond to frequently occurring words like "the," "and," "a," etc.*

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: *The sample emission sentences get increasingly more semantically meaningful as the number of hidden states increases. In the special case where there is only one hidden state, there is no transition behaviour and the HMM simply emits observations at random without regard to the previous observations, at probabilities equal to the observations' frequencies in the dataset.*

It is true that we can increase the training data likelihood by allowing more hidden states. To illustrate this, we can consider this case: we have a sequence of length n, and we choose to model this with n hidden states (this is the max number of hidden states that we can use to model this). So, each element in the sequence has its own state, and therefore every transition and output emission only happens once. For example, throughout the training data, for any given state j, it will only ever transition to a state k (the state whose corresponding element in the sequence follows j), and for any given state j, it will always emit the same observation x_j. So, the transition and output emission matrices will be all 0s and 1s, and the likelihood of the training data will be 1. This will obviously generalize badly, so we see a tradeoff between generalization and likelihood of the training data based on the chosen number of hidden states. The idea is the same for a smaller increase in the number of hidden states. It is analogous to model complexity, and allows the model to fit the given data with higher accuracy *to a larger number of model parameters, but can lead to over fitting if the number of hidden states is too large, or under fitting if the number of hidden states is too small.*

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: *Answers may vary. States may be a collection of nouns, verbs, prepositions, etc.*