

Policies

- Due 9 PM, March 6th, via Gradescope.
- 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 3.6+ 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 report as a single .pdf file to Gradescope (entry code 9YJGEX), under "Problem Set 4". **In the report, include any images generated by your code along with your answers to the questions.** For instructions specifically pertaining to the Gradescope submission process, see https://www.gradescope.com/get_started#student-submission.
- Since we are only using Gradescope this year, we ask that you upload both your code and solution pdf together as a single pdf and mark each page with the problem number. There are various free online tools to combine pdfs. To download your code as a pdf from Jupyter notebook, you can navigate to File -> Download as -> "PDF via Latex" or alternatively download the Latex file and compile it yourself.

TA Office Hours

- **Eric Zhao [Question 1 and Question 2.A, 2.B]**
 - Tuesday, 3/3: 5:00 pm - 7:00 pm
 - Annenberg 104
- **Shuyue (Erika) Yu [Question 2.C - 2.I]**
 - Thursday, 3/5: 4:00 pm - 6:00 pm
 - Annenberg 104

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:

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

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 $p(x | y)$, and let $\theta_{xjc} = P(x_j | x_1, \dots, x_{j-1}, y = c)$. Assuming we store each θ_{xjc} , how many parameters are needed to represent this factorization? Use big-O notation.

Problem B [5 points]: Assume we did no such factorization, and just used the joint probability $p(x | y = c)$. 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.

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?

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?

Problem E [11 points]: Assume all the parameter estimates have been computed. What is the computational complexity of making a prediction, i.e. computing $p(y | x)$, 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.

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 $0, 1, \dots, X - 1$). 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.)

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 | 2222222222222222222103 | 9.365e-24 |

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?

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.)**

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.

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.

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?

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?

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.