CS440: Intro to Artificial Intelligence, Fall 2017, Homework 3

Name: Nestor Alejandro Bermudez Sarmiento (nab6)

Worked individually

Assignment 3 focuses on the Naive Bayes classification method. This report shows the performance of such method against different datasets and different feature extraction methods. I'm using Python 3.6 for my implementation.

Part 1

In this section we will take ASCII representations of digits to train our model and then evaluate its performance with a different of examples in the same format.

Implementation details

My code is split in the following pieces:

Parser: this class takes care of interfacing with the text file. It takes the paths for the files that contain the data and labels and generates tuples formed with a matrix representation of an example and its corresponding label. The class itself is a Python generator to prevent blowing up the memory (not that it will happen with the given dataset).

Feature Extractors: I created multiple of these classes that I call 'Feature extractors', they basically take an item and generate an array of features that will then be fed into the classifier. For Part 1.1 I created the **SinglePixelFeatureExtractor** which simply maps every pixel into its value (0, 1). For Part 1.2 I created the **PixelGroupFeatureExtractor** that accepts arguments to specify the dimensions of the group and whether or not they should be disjoint.

Visualizations: I use matplotlib[1] for the color maps. The implementation is rather simple and can be found in the **HeatMap** class. I use the *pcolormesh* function with a *jet* color schema because it seems to be the closest to the colors shown in the sample visualizations.

Probability Distributions: this is generic class that counts how many times a given value has appeared and provides methods to retrieve the probability and the log of the probability of a given value. It also takes care of implementing Laplacian smoothing [2].

Classifier: this class has a couple of data structures to hold the different probability distributions per class and per feature. It also provides methods to train the model and

[1] https://matplotlib.org/

[2] https://en.wikipedia.org/wiki/Laplacian_smoothing

exposes functions to classify a new example or evaluate the model performance given a sequence of examples. It also implements **Maximum a posteriori**[3] to make the predictions. Finally, it provides methods that calculate the confusion, class likelihood and log odd ratios matrices. For log odd ratios I use numpy[4] to take two matrices, divide them and then take the log of the resulting matrix.
Util: under this class you will find helpful methods and functions to print out matrices, examples and find the pair of classes we need to use for the visualizations (high confusion rates).

Part 1.1.

I tried different smoothing constants between 0.2 and 10 with increments of 0.2. The maximum accuracy was achieved using 0.2 as the constant and it seems to be inversely proportional: as the smoothing constant increased the accuracy overall decreased. It did have a local maximum around 0.8-1.0. For the sake of being pragmatic the following table does not include all the intervals but the remaining can be found in the zip file accompanying this report.

Classifier accuracy	
0.2	77.3%
0.4	77.2%
0.6	77.0%
0.8	77.1%
1	77.1%
2	76.6%
3	76.3%
4	76.2%
5	76.1%
6	76.0%

Table 1: Accuracy for different smoothing constants.

Confusion Matrix

As expected, the classifier was not perfect. Some of the examples were misclassified. The statistics of how the model behaved follows.

[3] https://en.wikipedia.org/wiki/Maximum_a_posteriori_estimation
[4] https://docs.scipy.org/doc/numpy-1.13.0/

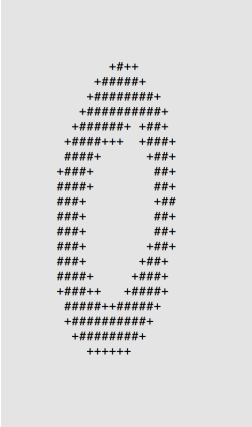
		Predicted class									
		0	1	2	3	4	5	6	7	8	9
Class	0	84.44	0	1.11	0	1.11	5.56	3.33	0	4.44	0
	1	0	96.3	0.93	0	0	1.85	0.93	0	0	0
	2	0.97	2.97	77.67	3.88	1.94	0	6.8	0.97	4.85	0
	3	0	1	0	80	0	3	2	7	1	6
	4	0	0	0.93	0	75.7	0.93	2.8	0.93	1.87	16.82
	5	2.17	1.09	1.09	13.04	3.26	68.48	1.09	1.09	2.17	6.52
	6	1.1	4.4	4.4	0	4.4	6.59	76.92	0	2.2	0
	7	0	5.66	2.83	0	2.83	0	0	72.64	2.83	13.21
	8	0.97	0.97	2.91	13.59	2.91	7.77	0	0.97	60.19	9.71
	9	1	1	0	3	10	2	0	2	1	80

Table 2: Confusion matrix using smoothing constant 0.2. Values are percentages.

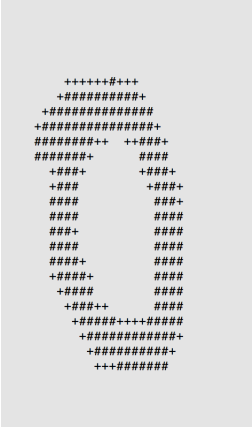
The classification rates are simply the values in the diagonal of the previous table.
Overall accuracy achieved: **77.3%**

Examples with highest and lowest probability

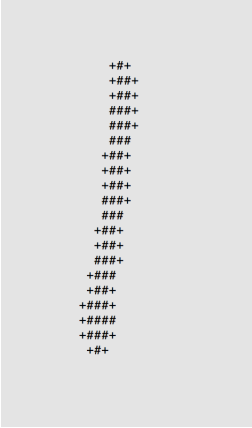
Class 0 (Highest)



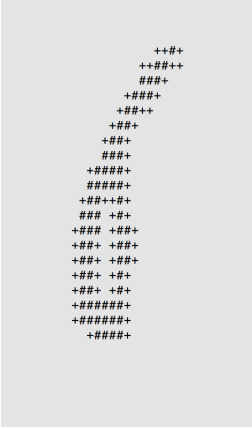
Class 0 (Lowest)



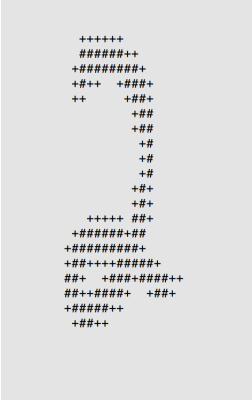
Class 1 (Highest)



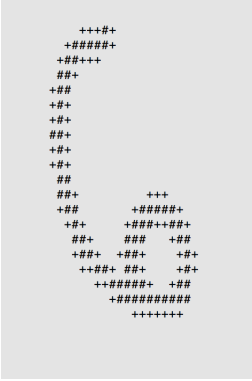
Class 1 (Lowest)



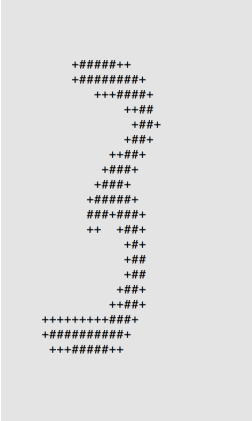
Class 2 (Highest)



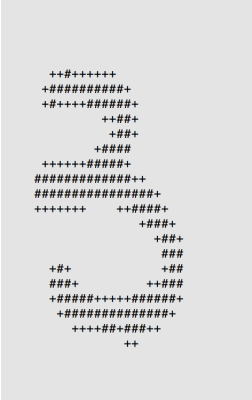
Class 2 (Lowest)



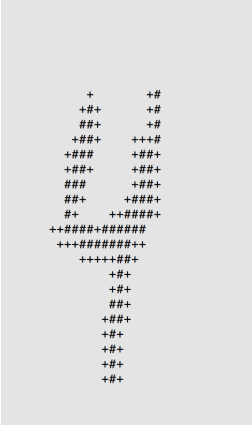
Class 3 (Highest)



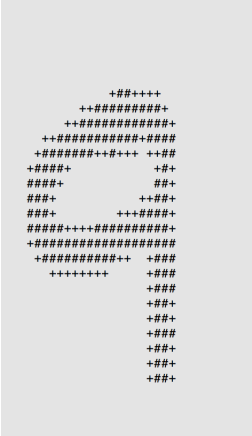
Class 3 (Lowest)



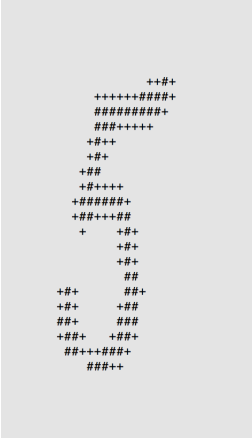
Class 4 (Highest)



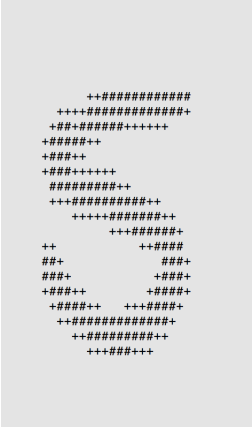
Class 4 (Lowest)



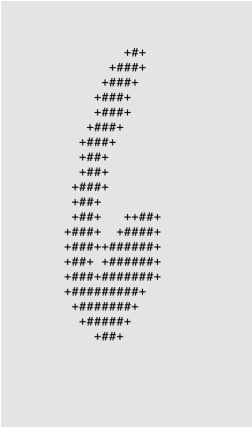
Class 5 (Highest)



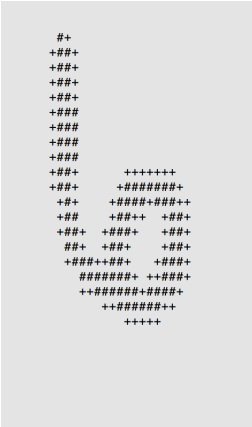
Class 5 (Lowest)



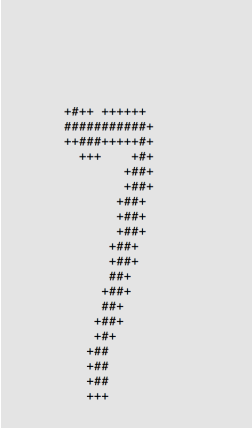
Class 6 (Highest)



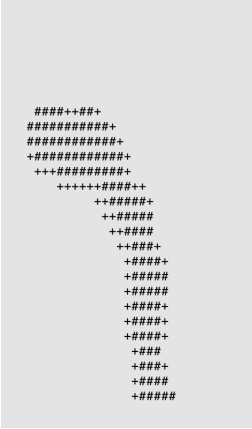
Class 6 (Lowest)



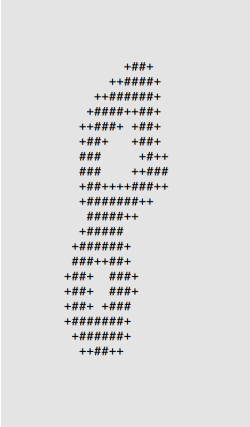
Class 7 (Highest)



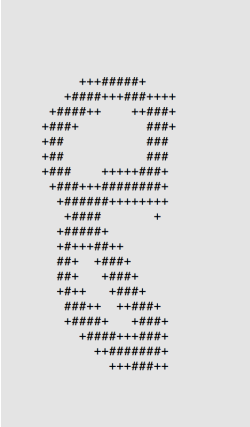
Class 7 (Lowest)



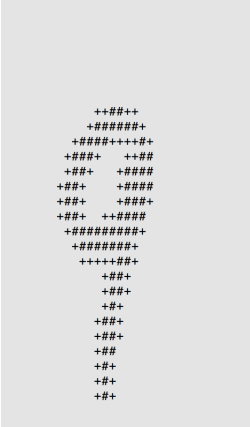
Class 8 (Highest)



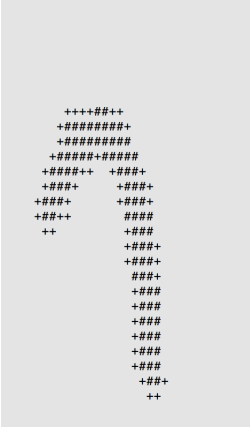
Class 8 (Lowest)



Class 9 (Highest)



Class 9 (Lowest)



Model likelihood and log odd ratios

Now lets look at the model likelihood for each of the classes with highest confusion rates.

Class 4 vs Class 9

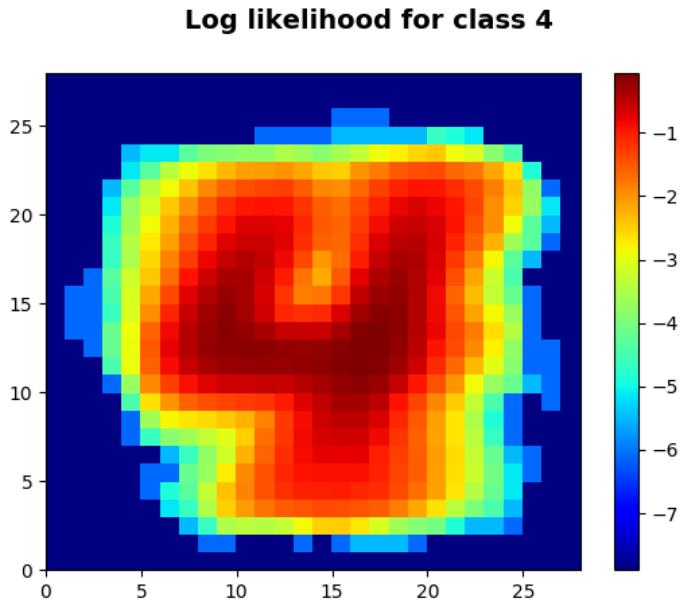


Figure 1: Log likelihood of class 4

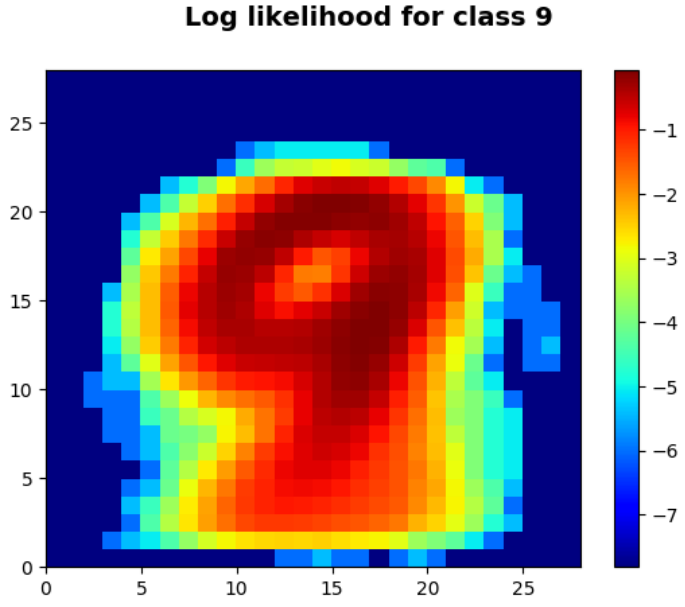


Figure 2: Log likelihood of class 9

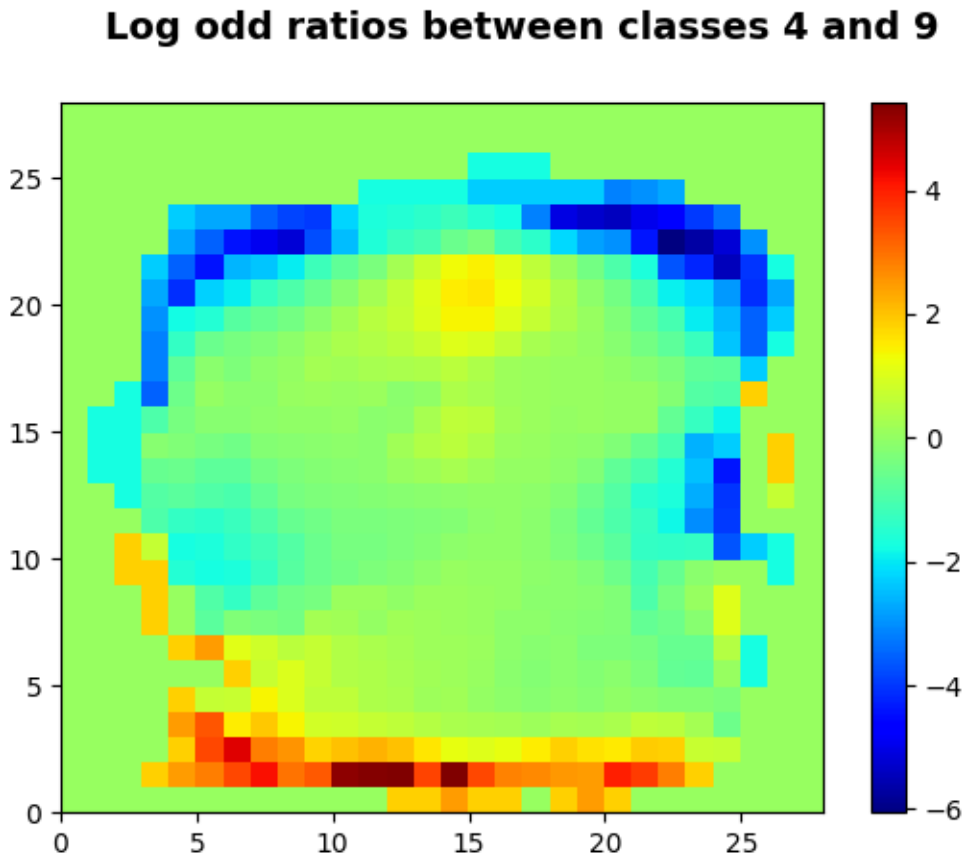


Figure 3: Log odd ratio of 4 over 9

Class 5 vs Class 3

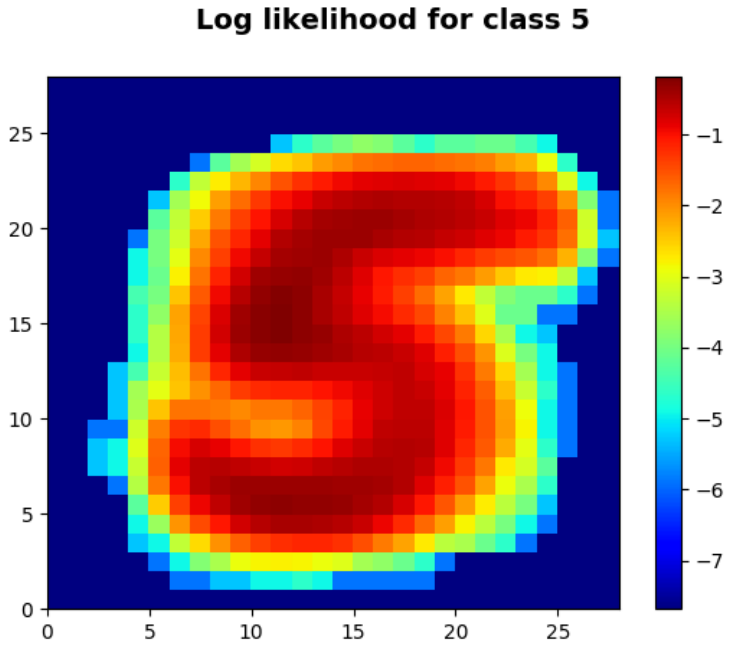


Figure 4: Log likelihood of class 5

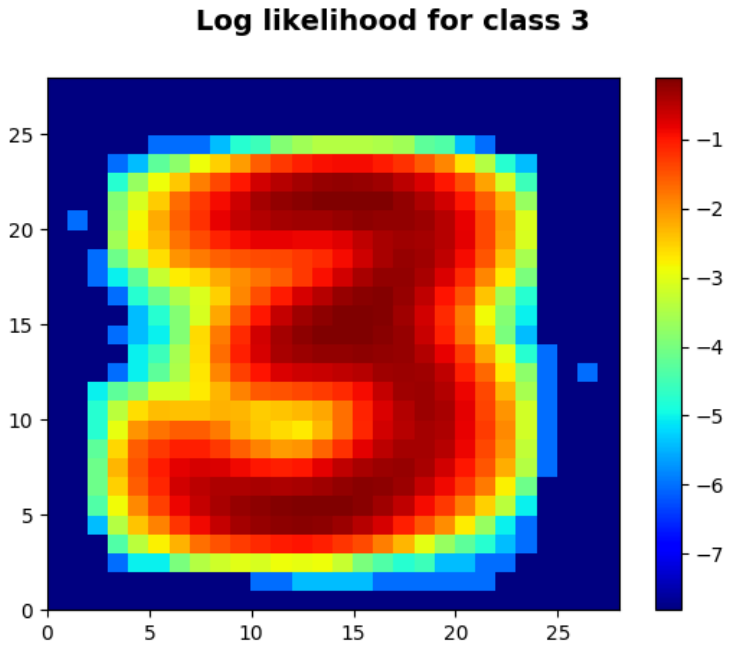


Figure 5: Log likelihood of class 3

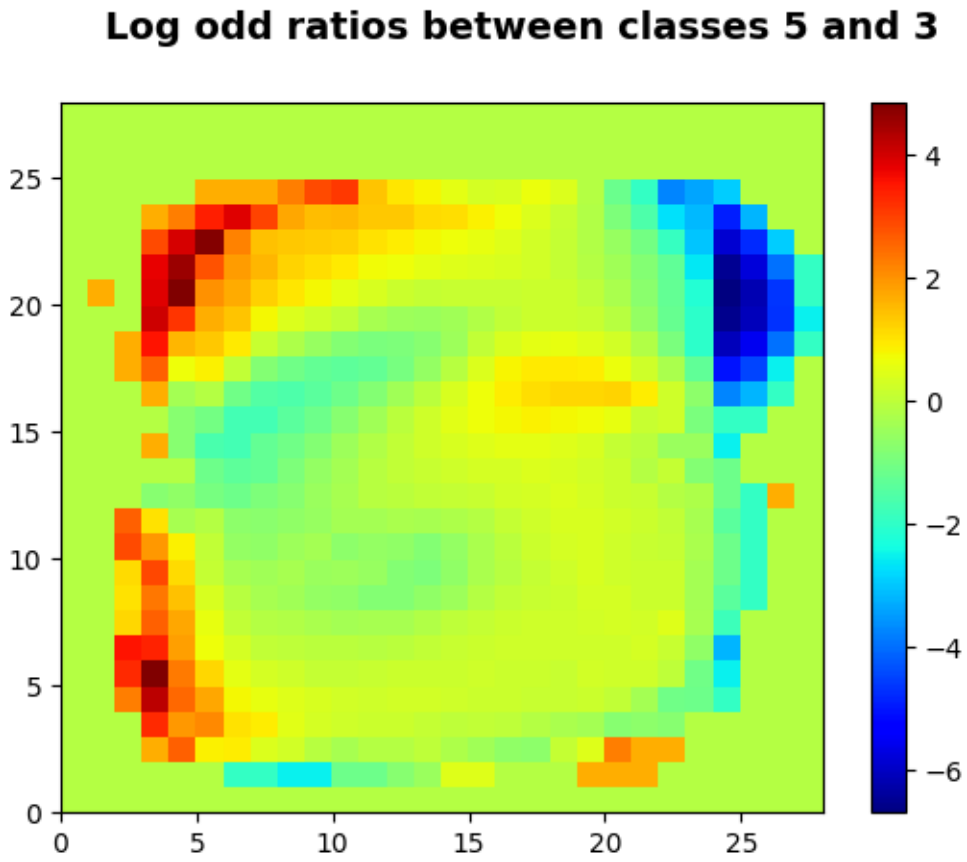


Figure 6: Log odd ratio of 5 over 3

Class 7 vs Class 9

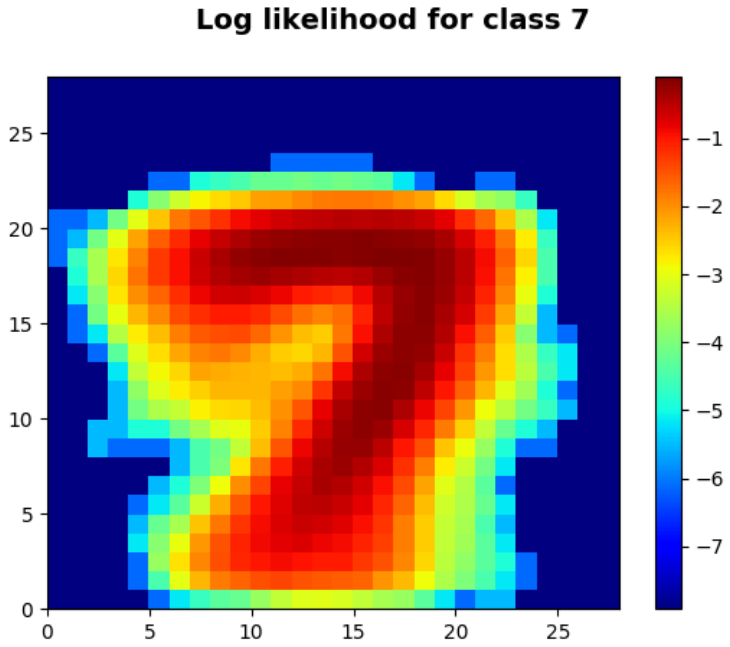


Figure 7: Log likelihood of class 7

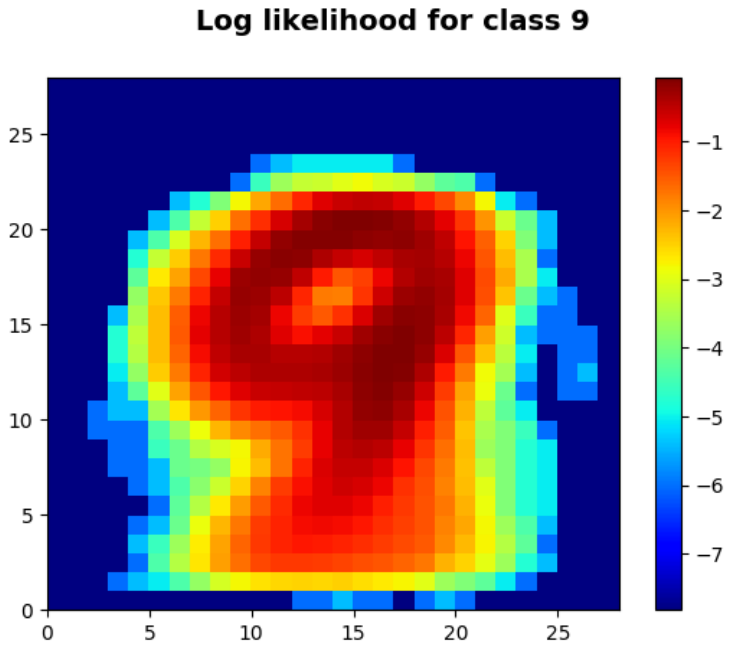


Figure 8: Log likelihood of class 9

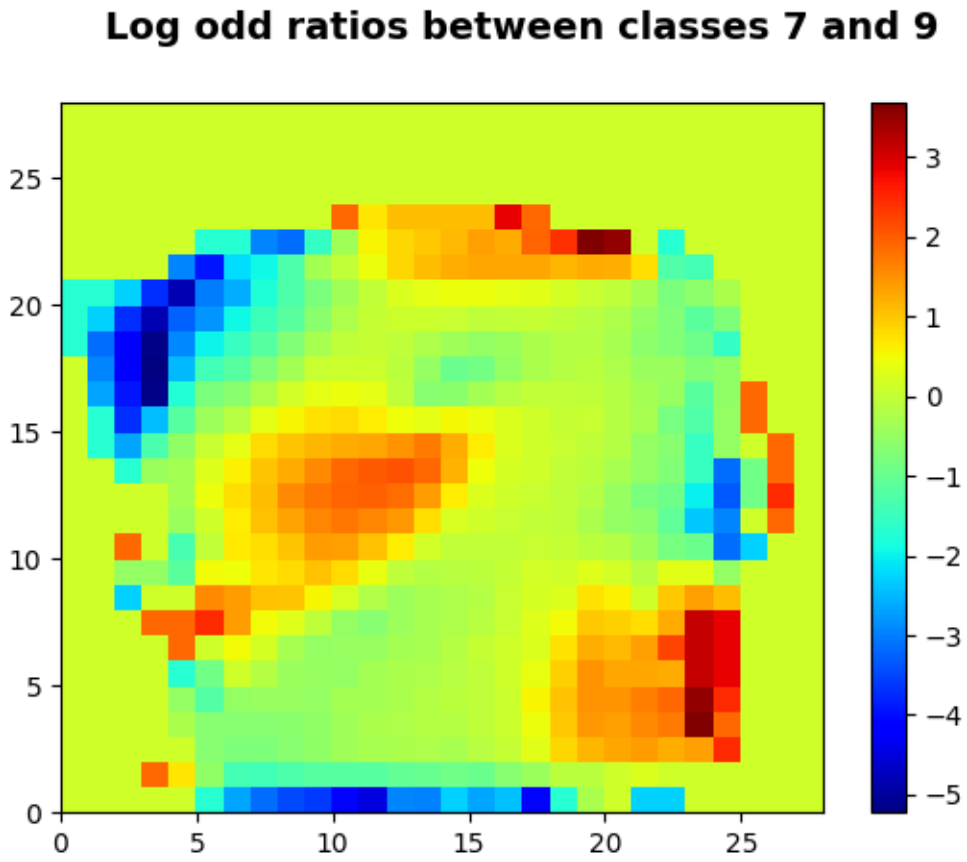


Figure 9: Log odd ratio of 7 over 9

Class 8 vs Class 3

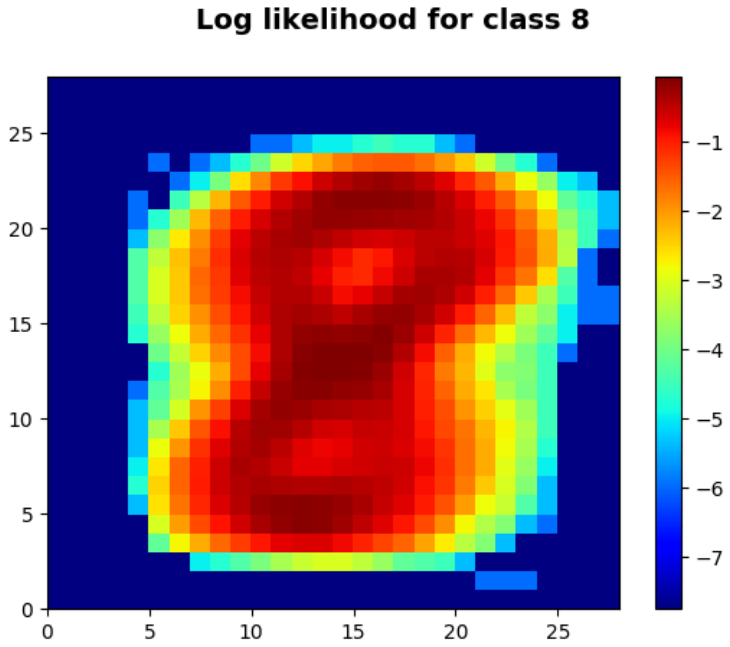


Figure 10: Log likelihood of class 8

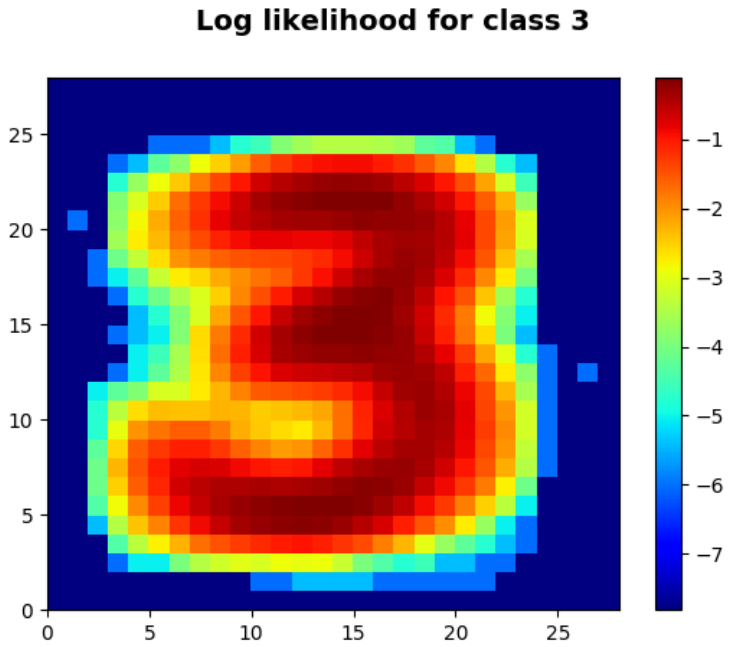


Figure 11: Log likelihood of class 3

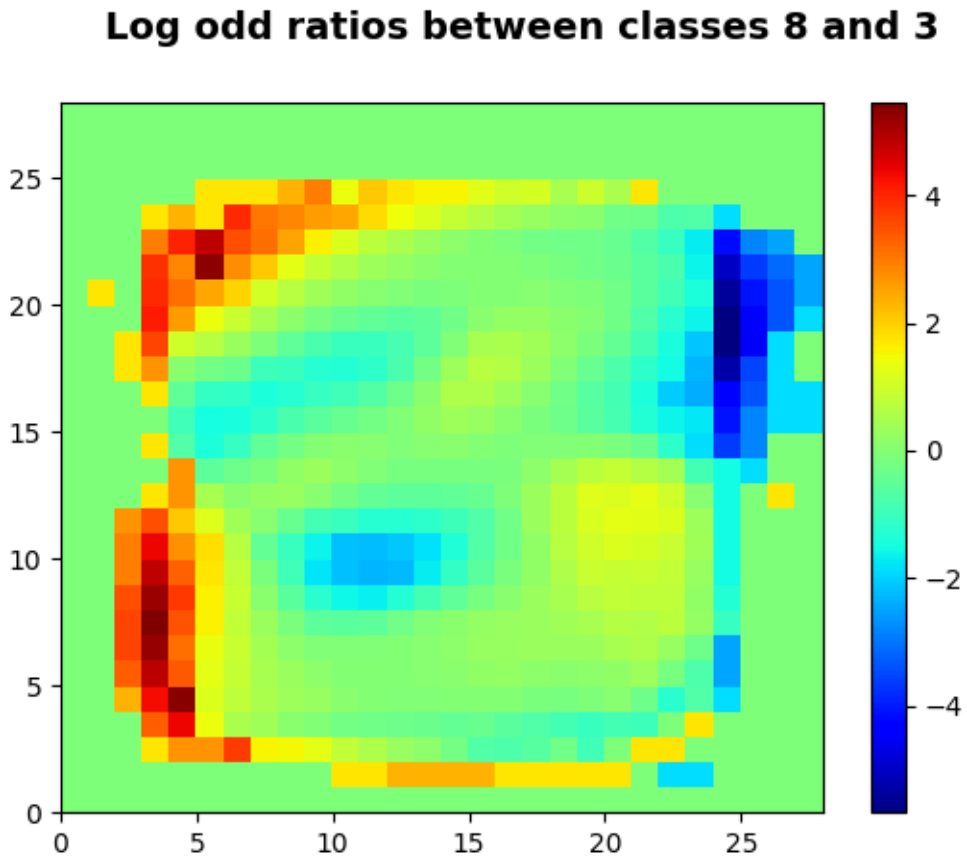


Figure 12: Log odd ratio of 8 over 3

Part 1.2.

Extra Credit

Part 2

Part 2.1

Part 2.2

Extra Credit