I have implemented everything in C++.

# Part 1

I have implemented the objects for estimating the likelihood P(x|c) using a 2D array in which each entry stores a pair (# of Fij = 1 for this class, # of total Fij tokens for this class). This array is updated by simply going through the training data set, and whenever a there is a token Fij for this class, the denominator is incremented. And similarly, if Fij = 1 for an accurance seen, the numerator is incremented.

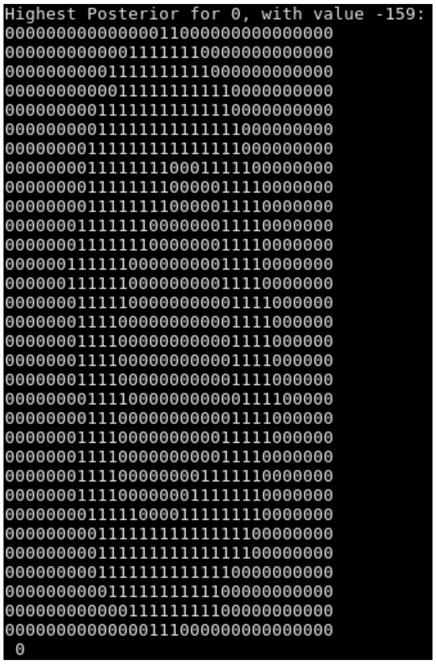
The final posterior calculation used in the testing phase is exactly the same as the on described in class. The results are shown below:

```
number of training samples: 2436
class: 0, fit: 0.972222
class: 1, fit: 0.933333
class: 2, fit: 0.853659
class: 3, fit: 0.909091
class: 4, fit: 0.881356
class: 5, fit: 0.932203
class: 6, fit: 0.976744
class: 7, fit: 1
class: 8, fit: 1
class: 9, fit: 0.952381
Overall acuracy: 0.939326
```

Figure1: Overall accuracy breakdown.

| | | | | | | Guess | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Class | 0 | 0.972222 | 0 | 0 | 0 | 0.0277778 | 0 | 0 | 0 | 0 | 0 |
| | 1 | 0 | 0.933333 | 0 | 0 | 0 | 0 | 0 | 0.0222222 | 0.0222222 | 0.0222222 |
| | 2 | 0 | 0 | 0.853659 | 0 | 0 | 0 | 0 | 0 | 0.121951 | 0.0243902 |
| | 3 | 0 | 0 | 0 | 0.909091 | 0 | 0 | 0 | 0.030303 | 0 | 0.060601 |
| | 4 | 0 | 0 | 0 | 0 | 0.881356 | 0 | 0 | 0.0677966 | 0.0508475 | 0 |
| | 5 | 0 | 0 | 0 | 0 | 0 | 0.932203 | 0 | 0 | 0 | 0.0677966 |
| | 6 | 0 | 0 | 0 | 0 | 0.0232558 | 0 | 0.976744 | 0 | 0 | 0 |
| | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| | 9 | 0 | 0 | 0 | 0.0238095 | 0 | 0 | 0 | 0.0238095 | 0 | 0.952381 |

Figure2: Confusion matrix.



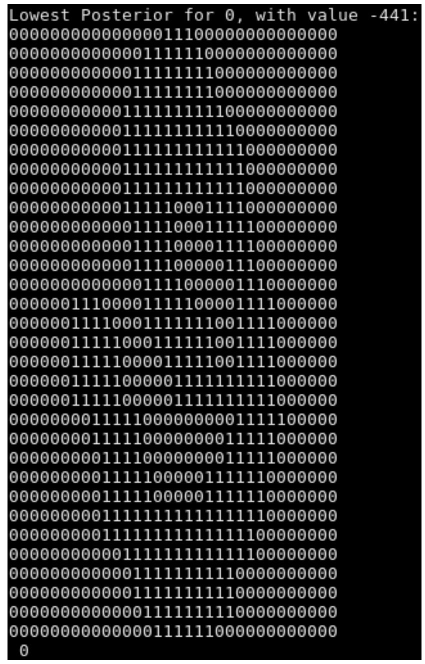
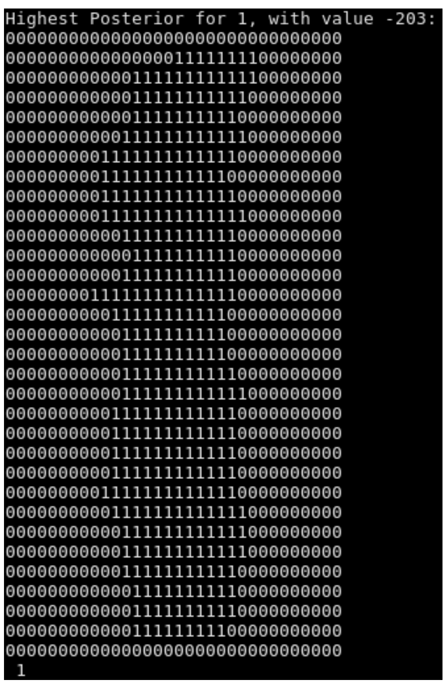




Figure3: Best posterior for 0 on the left, and worst on the right.



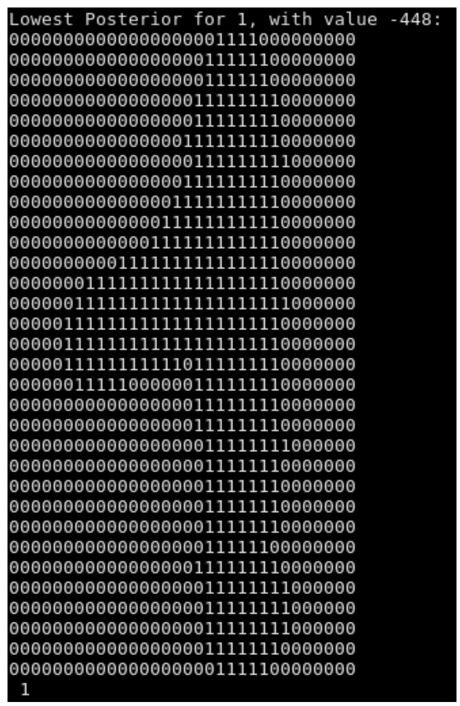
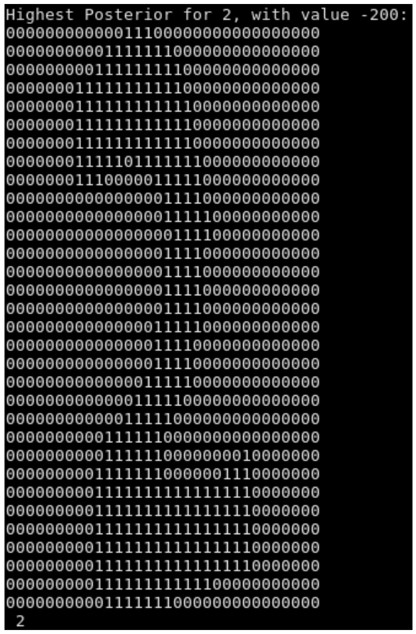




Figure4: Best posterior for 1 on the left, and worst on the right.



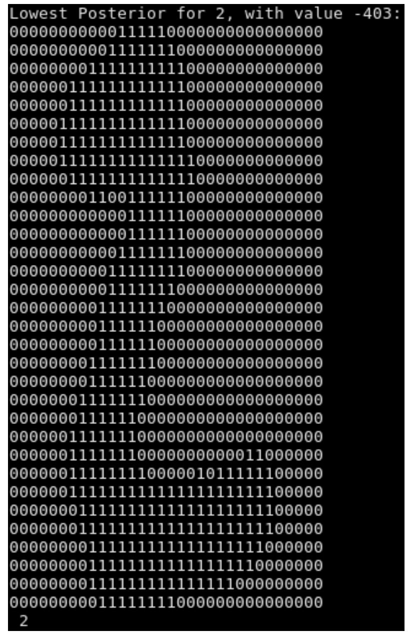




Figure5: Best posterior for 2 on the left, and worst on the right.



Figure6: Best posterior for 3 on the left, and worst on the right.

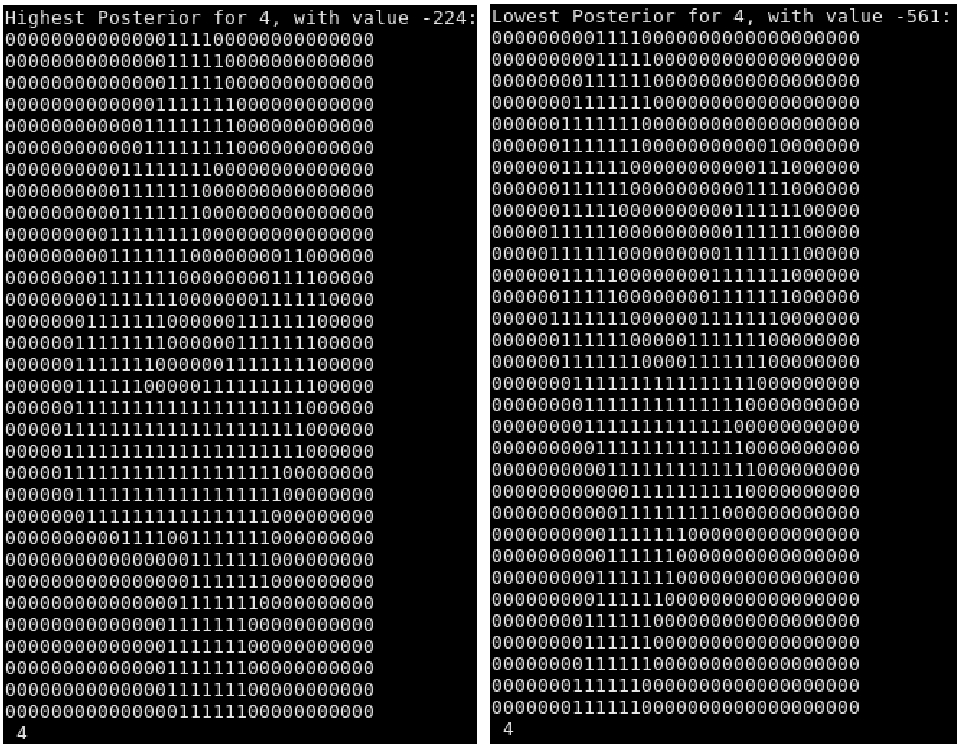



Figure7: Best posterior for 4 on the left, and worst on the right.

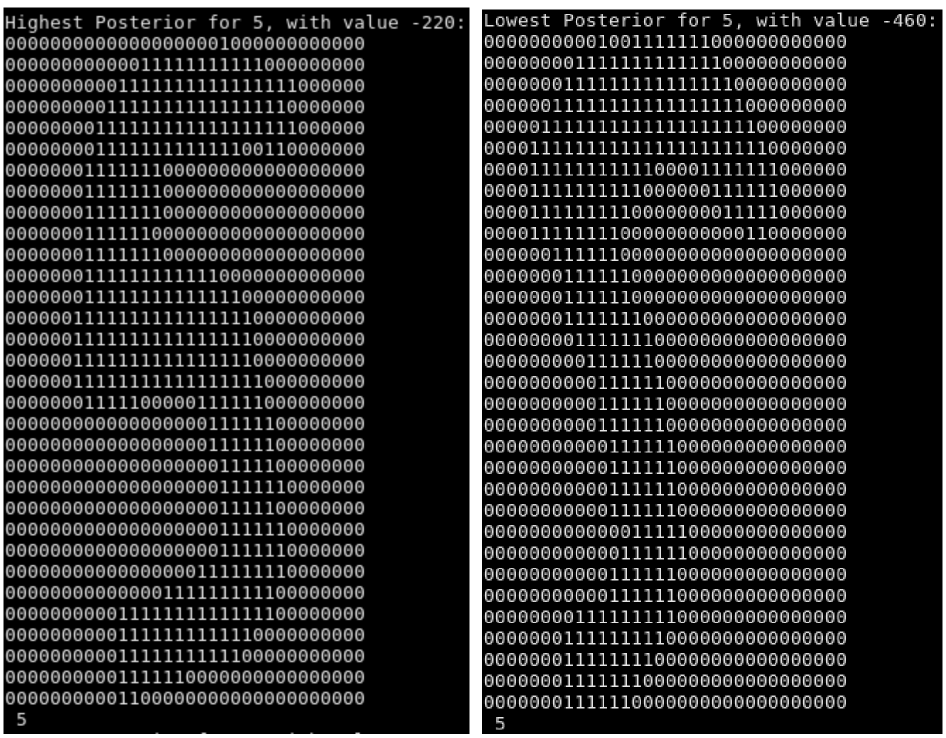



Figure8: Best posterior for 5 on the left, and worst on the right.

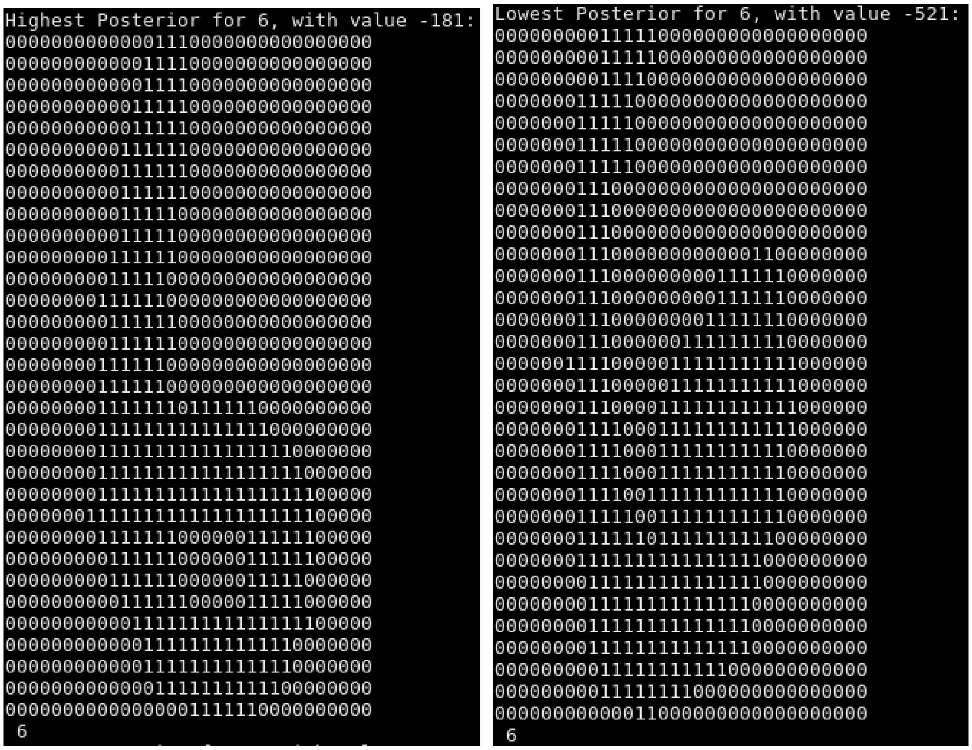



Figure9: Best posterior for 6 on the left, and worst on the right.

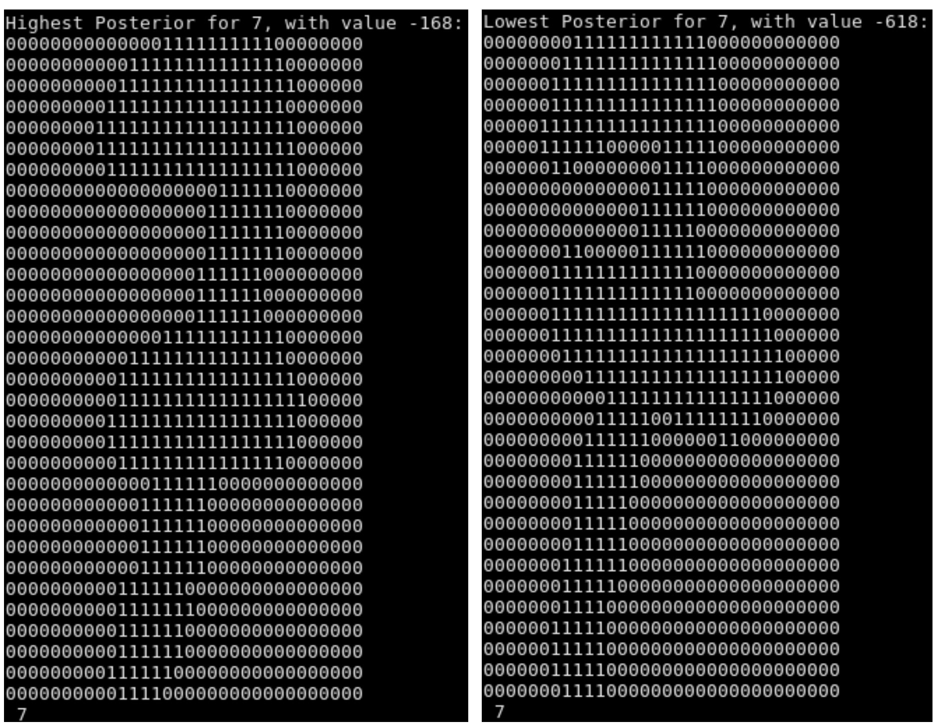
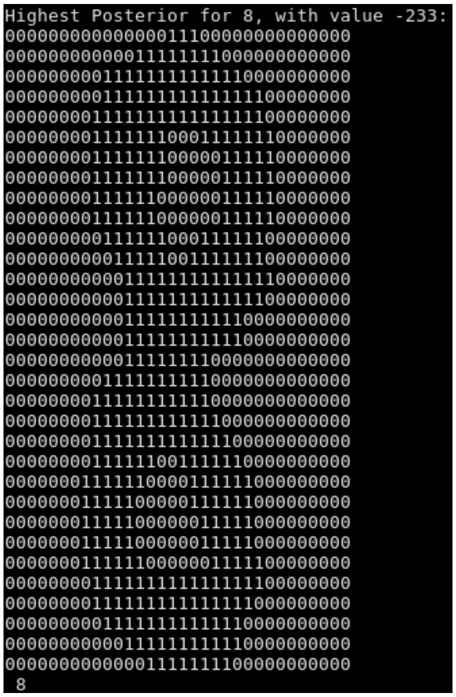



Figure10: Best posterior for 7 on the left, and worst on the right.



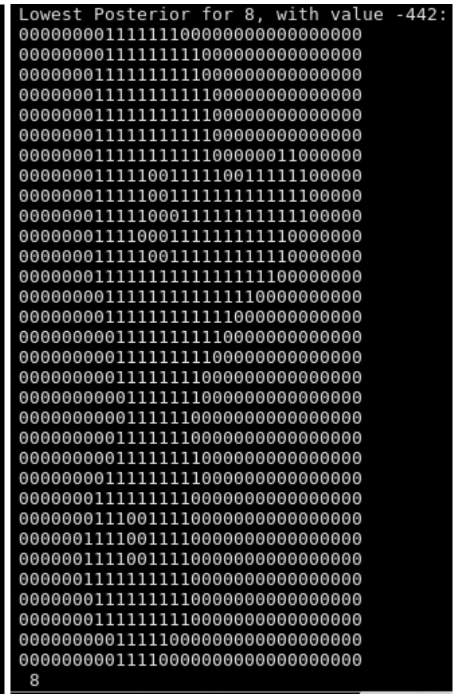
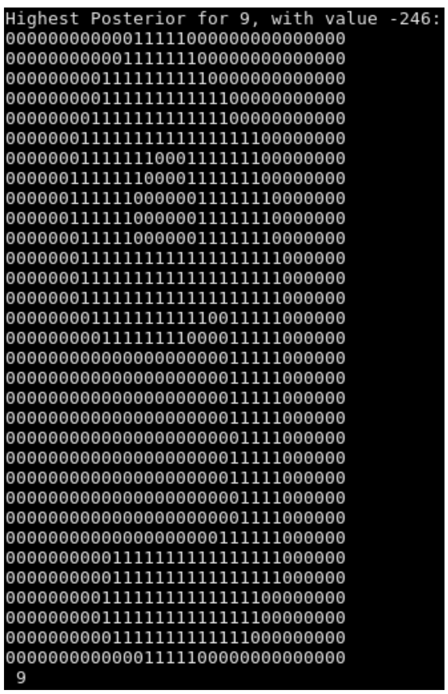



Figure11: Best posterior for 8 on the left, and worst on the right.



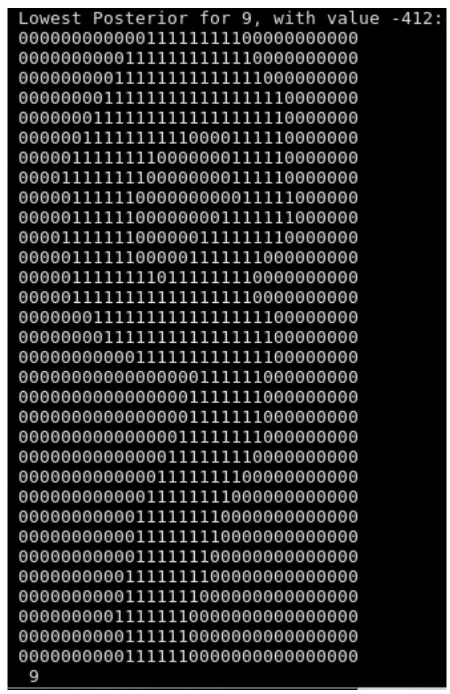



Figure12: Best posterior for 9 on the left, and worst on the right.

From the confusion matrix, I have found that the 4 pairs of number which have the highest confusion rates are (2, 8), (3, 9), (9, 5) and (4, 7). The "heatmap" illustration of their likelihood and Odds ratio can be seen below.

For likelihood plots, I have adopted the convention such that:

- If likelihood > 0.75, then it is represented as "+".
- If likelihood > 0.5, then it is represented as " ".
- Otherwise, it is represented as "-".

For Odds ratio plots, I have adopted the convention such that:

- If Log(ratio) > 0.5, then it is represented as "+".
- If Log(ratio) > -0.5, then it is represented as " ".
- Otherwise, it is represented as "-".

The plots for all 4 pairs are shown below:

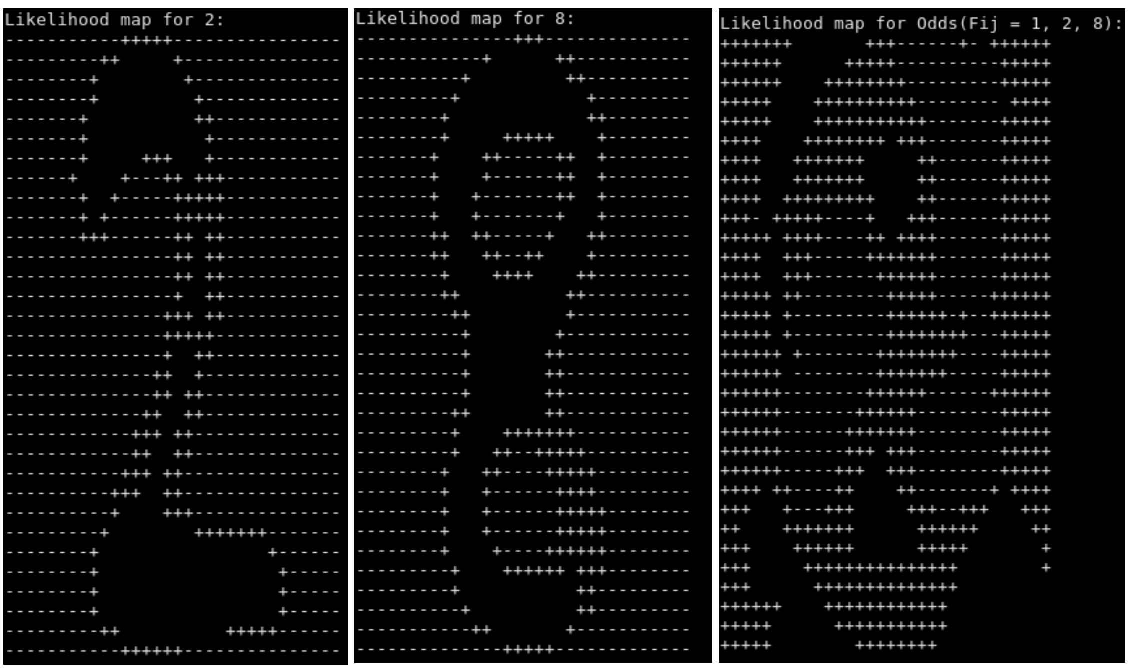



Figure13: Likelihood maps for 2 & 8 on the left, and Odds(Fij = 1, 2, 8) on the right.

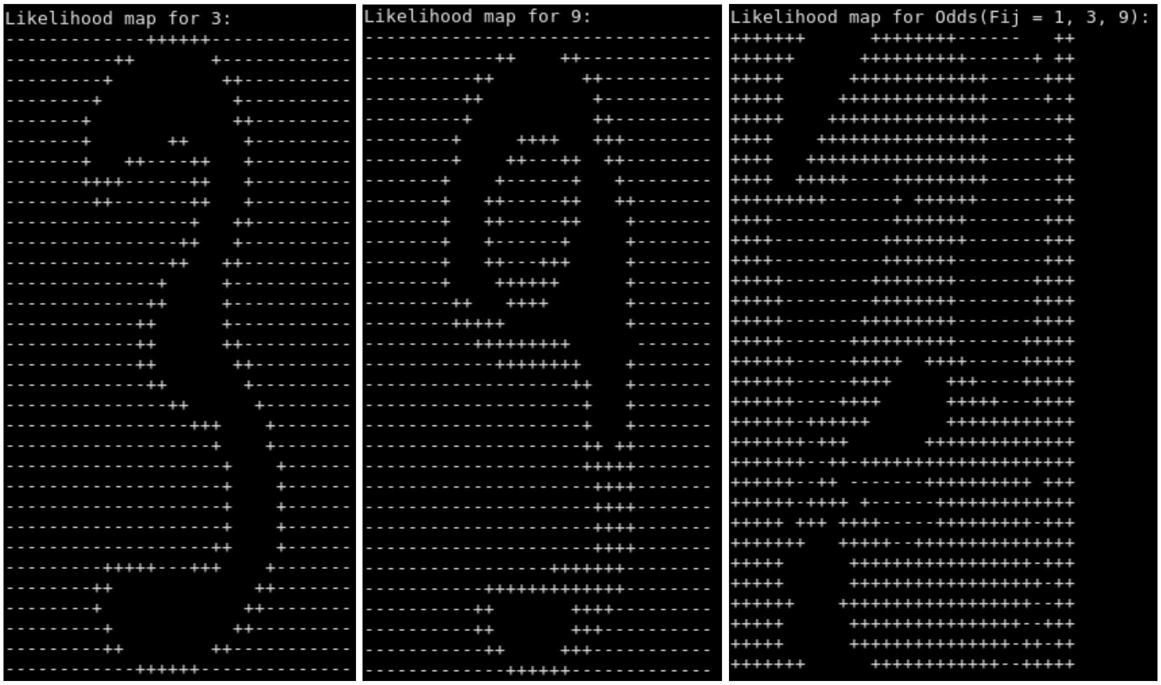



Figure14: Likelihood maps for 3 & 9 on the left, and Odds(Fij = 1, 3, 9) on the right.


Figure15: Likelihood maps for 4 & 7 on the left, and Odds(Fij = 1, 4, 7) on the right.

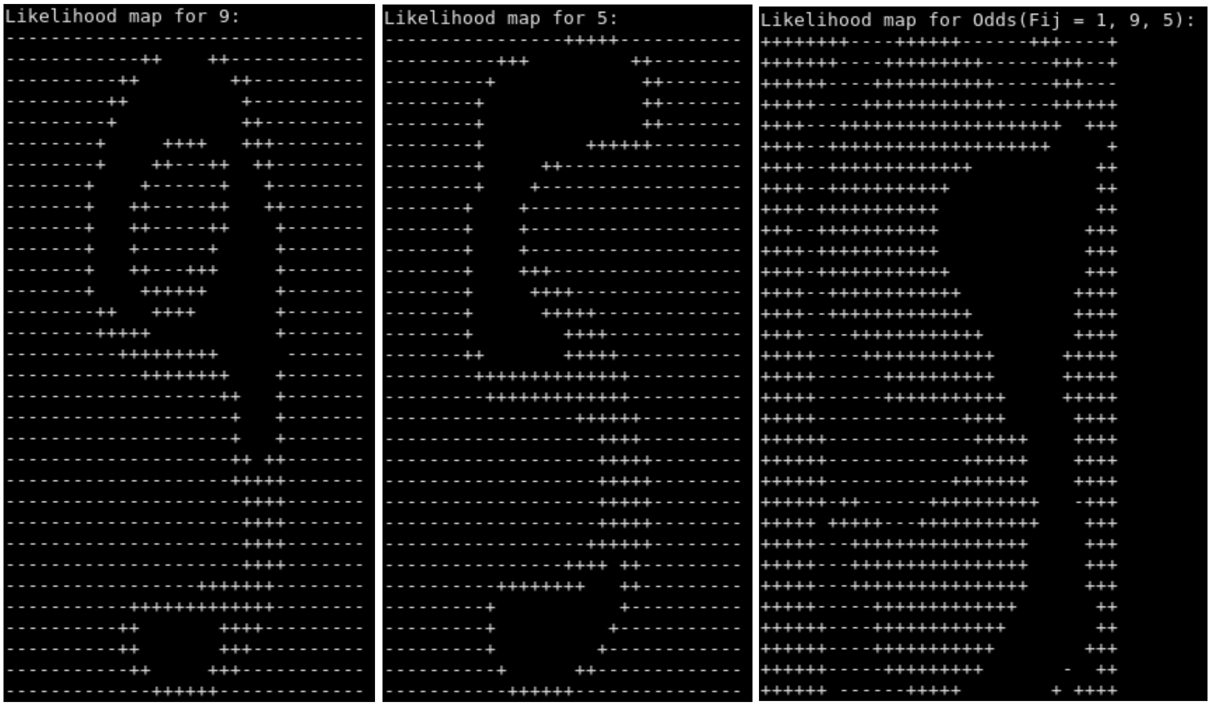



Figure16: Likelihood maps for 9 & 5 on the left, and Odds(Fij = 1, 9, 5) on the right.

## Part 2

For determining how I train the perceptrons, I have played around with some variables:

- Learning rate decay function is needed for the perceptrons to converge (dataset is not linearizeable). So, I have use Eta = 1/n, where n is the epoch number being currently ran through. This has helped it to converge within 15 epochs.
- I have chosen to use a bias. Without bias, the overall accuracy is only about 0.93, whereas with a bias, 0.94 can be achieved.
- Initialization of weights didn't really do much since it depends on how well a particular random weight is formed. Thus, I stuck to having it all initialize to 0.
- Similar to random initialization, order of training examples depended on how lucky we were. Sometimes it converges faster as compared to a fixed sequence (best I have seen is 11 rounds). But most of the time it takes around 15. For the illustration of how fast it converges, I have used a fixed sequence.
- The number of epochs is 15 since I can verify that it definitely converges after 15 rounds.

The convergence table can be seen below:

| Epoch | Overall Accuracy |
|---|---|
| 1 | 0.876923 |
| 2 | 0.925275 |
| 3 | 0.927473 |
| 4 | 0.931868 |
| 5 | 0.934066 |
| 6 | 0.938642 |
| 7 | 0.936264 |
| 8 | 0.936264 |
| 9 | 0.940659 |
| 10 | 0.945055 |
| 11 | 0.942857 |
| 12 | 0.940659 |
| 13 | 0.942857 |
| 14 | 0.942857 |
| 15 | 0.942857 |

Figure17: Accuracy vs Epoch table as the perceptrons are trained.