MP3 Report

Dennis J. McWherter, Jr. (dmcwhe2) 4 credit
Joe Fetsch (jfetsch2) 4 credit
Revanth Reddy (rreddy4) 4 credit

November 7, 2016

1 Part 1.1

This section discusses our approach and results to part 1.1 for assignment 3.

1.1 Implementation

Our implementation takes the standard form for a Naive Bayes classifier. We first train our model using the training data and training labels. With the training set, we are capable of computing the full table for our *likelihood* estimates:

$$\mathrm{P}(F_{ij} = k|digit) \tag{1}$$

where $k \in \{0, 1\}$, $digit \in \{0, 1, .., 9\}$, and the F_{ij} denotes the value of the pixel at position (i, j) in a 28 x 28 pixel grid. More specifically, for each distinct *digit*, we compute this probability for each pixel by taking the frequency of occurrence of a specific value (i.e. either 0 or 1) across the entire subset of test data that is labeled with the *digit* class and the dividing by the total number of test images in that class.

Additionally, we use *Laplace smoothing* when estimating our likelihoods. By using this technique, we can more effectively account for rare or unseen values in our model. After some experimentation we found that lower values (i.e. 0.1 vs 10) for our Laplace constant, k, provide better overall accuracy for our classifier. This result is logical if we assume our training sample is representative of the real world. A lower constant k produces a lower overall probability for unseen values. Conversely, a large constant k will produce a higher probability in our model for an unseen value. Plainly, if a result is truly very rare, the lower probability is likely more inline with the data the classifier will actually see when put into practice.

Similarly, we can estimate our *priors* by counting the frequency of each *digit* in our training set and dividing by the total number of training samples. After we have estimated both our *priors* and *likelihoods*, our model is trained and ready to be used for testing.

When testing our model, we use the test data **without** our test labels. Specifically, we treat our test sample as though it is an unlabeled sample. We then use the **MAP** algorithm in conjunction with our model to classify each input image from our test sample. That is, we choose the class (i.e. *digit*) that presents the highest overall likelihood given all of the observed pixels in the test image. However, to combine likelihoods, we use the **log likelihood** variant of MAP to avoid underflow.

Finally, to perform evaluation we take our classified results and compare them against the test labels. This is the first and only time that test labels are used throughout our analysis. In

our evaluation, we can breakdown the overall accuracy of our classifier, the accuracy per class, and, finally, compute a **confusion matrix** to provide better insight into our misclassifications.

1.2 Accuracy and Confusion Matrix

For part 1.1, we used a Laplace smoothing constant of 0.1 resulting in an *overall accuracy* of **77.3%** for our classifier.

The per-digit accuracy breakdown is:

Digit	Accuracy
0	84.44%
1	96.30%
2	78.64%
3	80.00%
4	74.77%
5	68.48%
6	76.92%
7	72.64%
8	60.19%
9	80.00%

Figure 1: Per-digit accuracy on part 1.1 classifier

The confusion matrix is as follows. Rows and columns are labeled 0-9 from top-bottom and left-right, respectively. Moreover, this matrix displays values as *percentages* rather than decimal places. Therefore, each row is out of 100 rather than 1 (with some small error for rounding).

	0	1	2	3	4	5	6	7	8	9
0	84.44	0.00	1.11	0.00	1.11	5.56	3.33	0.00	4.44	0.00
1	0.00	96.30	0.93	0.00	0.00	1.85	0.93	0.00	0.00	0.00
2	0.97	2.91	78.64	3.88	1.94	0.00	5.83	0.97	4.85	0.00
3	0.00	1.00	0.00	80.00	0.00	3.00	2.00	7.00	1.00	6.00
4	0.00	0.00	0.93	0.00	74.77	0.93	3.74	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	1.10	4.40	4.40	0.00	4.40	6.59	76.92	0.00	2.20	0.00
7	0.00	4.72	3.77	0.00	2.83	0.00	0.00	72.64	2.83	13.21
8	0.97	0.97	2.91	13.59	2.91	7.77	0.00	0.97	60.19	9.71
9	1.00	1.00	0.00	3.00	10.00	2.00	0.00	2.00	1.00	80.00

Figure 2: Part 1.1 confusion matrix

For the remaining ASCII result images required for the report (i.e. prototype figures), please refer to section **Appendix A** (section 5.1).

1.3 Odds Ratio Plots

This section describes our odds ratio calculation for the 4 highest pairs of misclassified digits. Just as in the assignment write-up, all plots are done to the log-scale. Similarly, these plots are

looking for pixels where the value is 1.

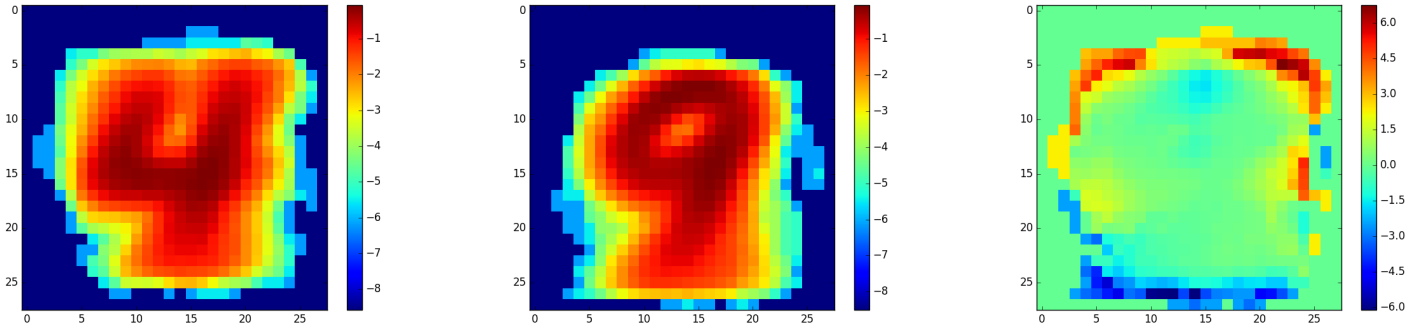


Figure 3: Log likelihood plots for 4 and 9. Log of odds ratio for 4 / 9.

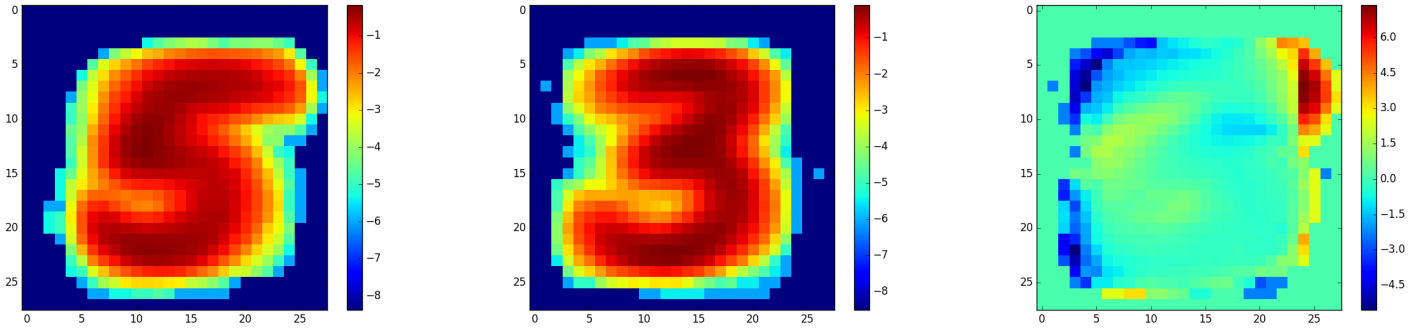


Figure 4: Log likelihood plots for 5 and 3. Log of odds ratio for 5 / 3.

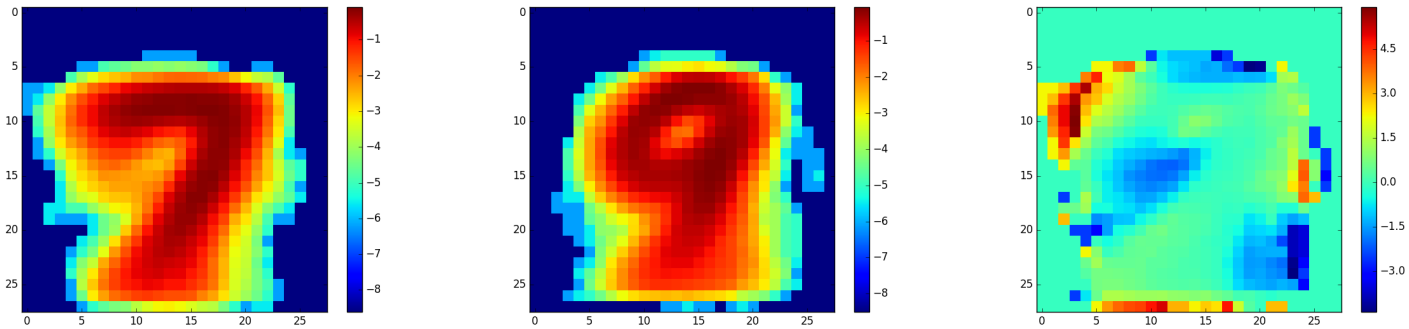


Figure 5: Log likelihood plots for 7 and 9. Log of odds ratio for 7 / 9.

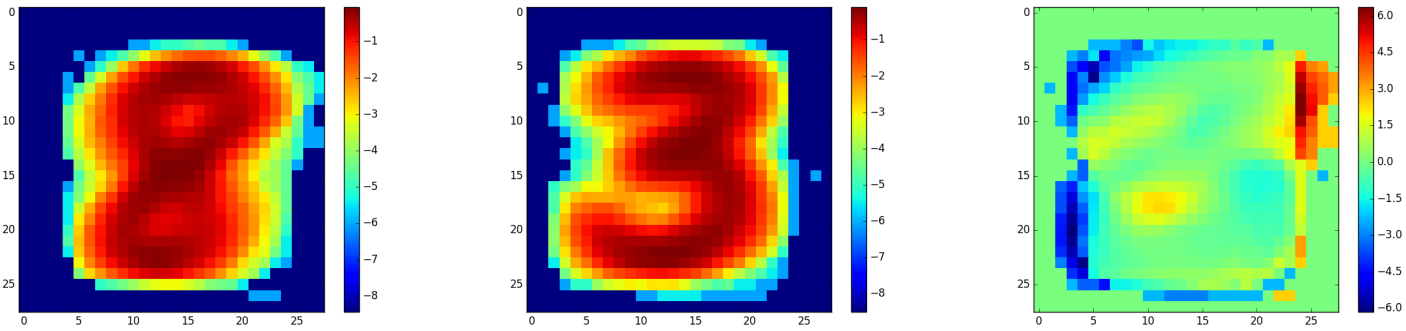


Figure 6: Log likelihood plots for 8 and 3. Log of odds ratio for 8 / 3.

2 Part 1.2

This section describes our approach, modifications, and results to part 1.2 for assignment 3.

2.1 Implementation

Our implementation was slightly modified from part 1.1. Primarily, we changed our calculation for our likelihoods to contain any combination of values provided within our pixel group. That is, instead of the valid values being only $\{0, 1\}$, they were now 2^n where n is the number of pixels in the group. More concretely, if you had a pixel group of size 2 (i.e. 1 x 2 or 2 x 1), the possible values would be $\{00, 01, 10, 11\}$. Similarly, we modified our implementation for a configurable grouping type; namely, overlapping and non-overlapping. In the case of non-overlapping, all values pixels in the group were computed with the same probability for that sequence. In the overlapping section, each pixel would be computed separately where it was considered the *starting* pixel in the group. Edge pixels were appropriately wrapped around modulo 28 (i.e. the size of the square grid representing a digit image).

2.2 Solutions

This section presents the solutions for part 1.2. Accuracy in the charts below is the total percentage of accurately classified images over the whole test set.

Dimensions	Accuracy (%)	Training Time (s)	Classifying Time (s)
1 x 1	77.3	6.89	13.89
2 x 2	83.5	13.15	25.63
2 x 4	78.2	19.22	34.74
4 x 2	72.7	22.94	41.62
4 x 4	56.6	848.38	2614.19

Figure 7: Solution table for part 1.2 containing data for **disjoint** pixel groups of specified dimension.

Dimensions	Accuracy (%)	Training Time (s)	Classifying Time (s)
1 x 1	77.3	6.00	13.69
2 x 2	87.0	12.26	25.50
2 x 4	89.7	18.02	34.14
4 x 2	89.7	22.14	41.92
4 x 4	86.1	918.99	2638.24
2 x 3	88.8	14.52	29.36
3 x 2	89.9	16.07	33.24
3 x 3	89.9	22.86	39.42

Figure 8: Solution table for part 1.2 containing data for **overlapping** pixel groups of specified dimension.

2.3 Discussion

The **accuracy** of the disjoint vs. overlapping pixel group classifiers was generally better in the overlapping case. The exception is when running these two schemes on a 1x1 group (i.e. single

pixel), the result is the same for both (as well as for part 1.1) since 1x1 is simply a special case for our general solution. Moreover, the groups of pixels generally (with few exceptions, see charts above) perform better than observing only a single pixel. This is most likely due to the fact that groups of pixels take into account a greater amount of information than a single pixel. In the cases where groups of pixels perform worse, this result could be from capturing insignificant features. The amount of *significant* features captured by the model depends highly on the shape and size of the groups.

The **running times** of training and testing are highly correlated to the number of pixels in a group. The running times were always better on smaller groups of pixels. Since the number of features is $2^{rowDim*colDim}$, we can see that the computation must increase with larger pixel groups. For example, on a 1x1 pixel group, there are only $2^{1*1} = 2$ possible values; however, on a 4x4 pixel group there are $2^{4*4} = 2^{16} = 65536$ possible values for each pixel. This exponentiation in values can also be seen in our running times. If you consider a 3x3 pixel group you'll observe that there are $2^9 = 512$ different possible values. This is *roughly* two orders of magnitude fewer values than for a 4x4 pixel group. Likewise, when you compare running times of either training or testing, you'll see that the same two orders of magnitude are approximately reflected there in computation time.

3 Part 2.1

This section discusses the implementation and results of part 2.

3.1 Implementation

Our implementation was quite similar to part 1. The major differences lie in our data parsing with the new format and in the likelihood estimates for each classifier. Since the Multinomial and Bernoulli classifier had different likelihood estimates, those were modified accordingly to meet their specification as described in the assignment write-up.

3.2 Multinomial Results

The Multinomial classifier had an overall accuracy of **73.00%** on the movie ratings dataset and **91.84%** on the Fisher 2-topic dataset. Below we present several figures to describe model accuracy. These include the confusion matrices as well as per-class accuracy tables. Moreover, we provide the top 10 words by greatest likelihood *per class* and also, the top 10 words by greatest odds ratio.

Class	Accuracy
-1	73.20%
1	72.80%

(a) Movie ratings

Class	Accuracy
-1	93.88%
1	89.80%

(b) Fisher 2-topic

Figure 9: Multinomial naive bayes model accuracy by class.

	−1	1
−1	73.20	26.80
1	27.20	72.80

(a) Movie-ratings

	−1	1
−1	93.88	6.12
1	10.20	89.80

(b) Fisher 2-topic

Figure 10: Confusion matrices for Multinomial naive bayes model (values in percent)

Class -1	Class 1
movie	film
film	movie
like	–
one	one
–	like
bad	story
story	good
much	comedy
time	way
even	even

(a) Movie ratings

Class -1	Class 1
know	know
yeah	yeah
uh	like
like	uh
um	um
right	right
just	don
think	think
oh	just
don	oh

(b) Fisher 2-topic

Figure 11: Top 10 words by descending likelihood for Multinomial model per class.

flat
disturbing
stale
tired
mediocre
plain
refreshingly
engrossing
haunting
grief

(a) Movie ratings

compatibility
friendship
attracted
inflation
waitresses
attraction
waitress
retail
walmart
assistance

(b) Fisher 2-topic

Figure 12: Top 10 words based on odds ratio for Multinomial naive bayes model.

3.3 Bernoulli Results

The Bernoulli classifier had an overall accuracy of **73.20%** on the movie ratings dataset and **94.90%** on the Fisher 2-topic dataset. Below we present several figures to describe model accuracy. These include the confusion matrices as well as per-class accuracy tables. Moreover, we provide the top 10 words by greatest likelihood *per class* and also, the top 10 words by greatest odds ratio.

Class	Accuracy
-1	71.80%
1	74.60%

(a) Movie ratings

Class	Accuracy
-1	91.84%
1	97.96%

(b) Fisher 2-topic

Figure 13: Bernoulli naive bayes model accuracy by class.

	−1	1
−1	71.80	28.20
1	25.40	74.60

(a) Movie-ratings

	−1	1
−1	91.84	8.16
1	2.04	97.96

(b) Fisher 2-topic

Figure 14: Confusion matrices for Bernoulli naive bayes model (values in percent)

Class -1	Class 1
movie	film
film	movie
like	one
one	like
story	–
much	story
–	comedy
bad	way
time	even
even	good

(a) Movie ratings

Class -1	Class 1
like	um
know	don
just	just
yeah	think
don	like
think	know
um	people
right	yeah
oh	oh
really	right

(b) Fisher 2-topic

Figure 15: Top 10 words by descending likelihood for Bernoulli model per class.

flat
stale
disturbing
tired
mediocre
refreshingly
engrossing
haunting
plain
grief

(a) Movie ratings

attracted
compatibility
inflation
waitresses
waitress
attraction
friendship
retail
assistance
hourly

(b) Fisher 2-topic

Figure 16: Top 10 words based on odds ratio for Bernoulli naive bayes model.

3.4 Result Discussion

When looking at the top 10 words based on likelihood for movie ratings for the Bernoulli model, the majority of the words appear to have a more *neutral* sentiment in typical English vernacular. Contrarily, when ranking based on odds ratio, it can be observed that the words have a far strong sentiment attached to them. Likewise, a similar phenomenon can be observed when comparing the results of the Fisher 2-topic results.

4 Part 2.2

This section provides our results and discussion for part 2.2.

4.1 Multinomial Results

The overall accuracy for our classifier using the Bernoulli model on the full 40-topic Fisher dataset is **86.22%**. Presented below is the confusion matrix. Indices are sorted top to bottom and left to right in ascending numerical order.

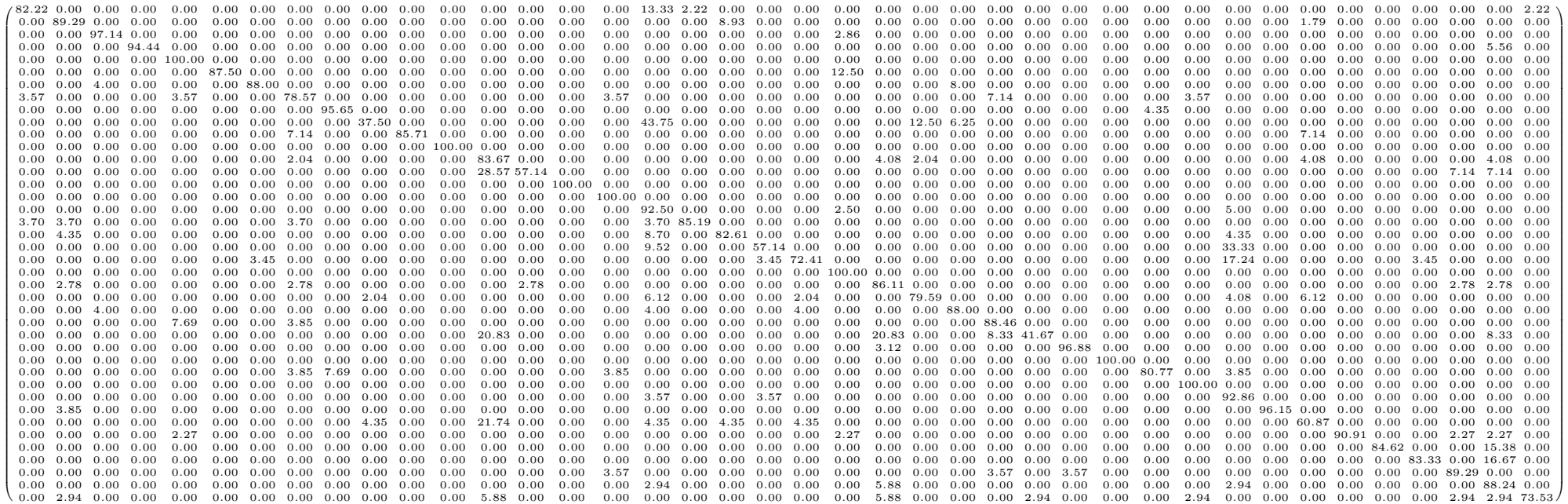


Figure 17: Confusion matrix for Fisher 40-topic Multinomial model.

4.2 Bernoulli Results

The overall accuracy for our classifier using the Bernoulli model on the full 40-topic Fisher dataset is **83.97%**. Presented below is the confusion matrix. Indices are sorted top to bottom and left to right in ascending numerical order.

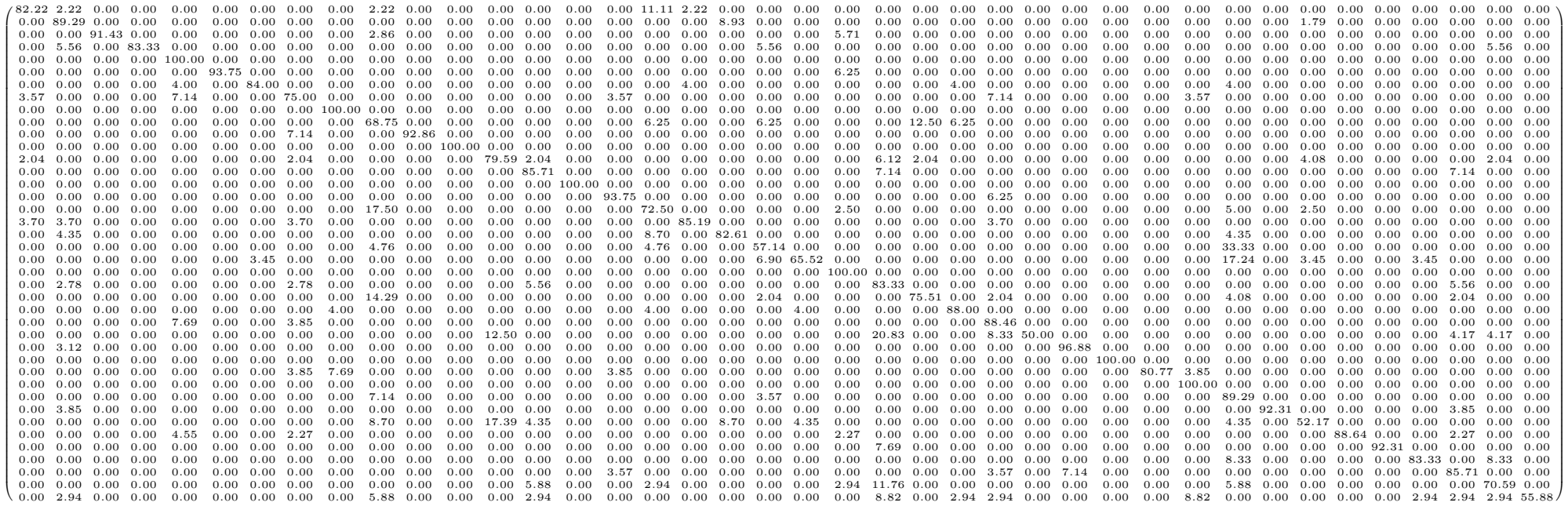


Figure 18: Confusion matrix for Fisher 40-topic Bernoulli model.

4.3 Misclassification

Since our multinomial classifier performed better on the fisher's 40-topic dataset, we charted the highest confusion rates for each category. These rates are outlined in the table below. If there were ties, the lowest numerical topic id was selected. In the case of values with 0% rates, this means that there were no misclassifications for that particular topic.

Actual	Confused	Percent
1	17	13.33
2	19	8.93
3	22	2.86
4	39	5.56
5	1	0
6	22	12.5
7	25	8.0
8	26	7.14
9	30	4.35
10	17	43.75
11	8	7.14
12	1	0
13	23	4.08
14	13	28.57
15	1	0
16	1	0
17	32	5.0
18	1	3.7
19	17	8.7
20	32	33.33
21	32	17.24
22	1	0
23	2	2.78
24	17	6.12
25	3	4.0
26	5	7.69
27	13	20.83
28	23	3.12
29	1	0
30	9	7.69
31	1	0
32	17	3.57
33	2	3.85
34	13	21.74
35	5	2.27
36	39	15.38
37	39	16.67
38	16	3.57
39	23	5.88
40	13	5.88

Figure 19: Misclassification rates per-topic for Fisher 40 dataset with Multinomial classifier

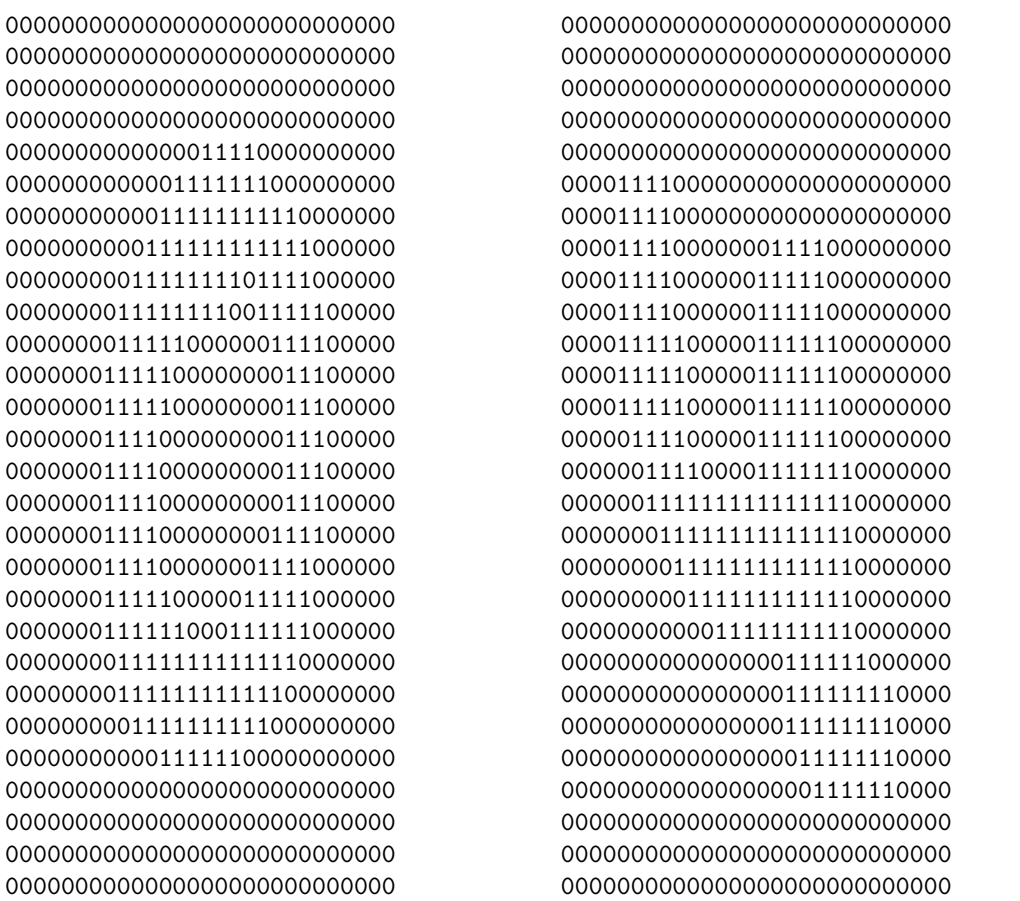
4.4 Discussion

The Multinomial and Bernoulli classifiers both performed better than 80% on the Fisher 40 dataset. However, the Multinomial classifier resulted in a better overall classification than the Bernoulli classifier. This could be due to the amount of data used to train our models. Since

Bernoulli takes a broader approach by looking at the entire document, it has the opportunity to better estimate the shape of the classification with fewer data points and fewer classes. However, when more granular data is available and a greater number of classes, the Multinomial distribution is capable of better capturing the more subtle nuances among classes since it operates at a token level. Since word counts and redundancy are taken into account for the Multinomial distribution, it allows for greater overall differentiation.

5 Appendix

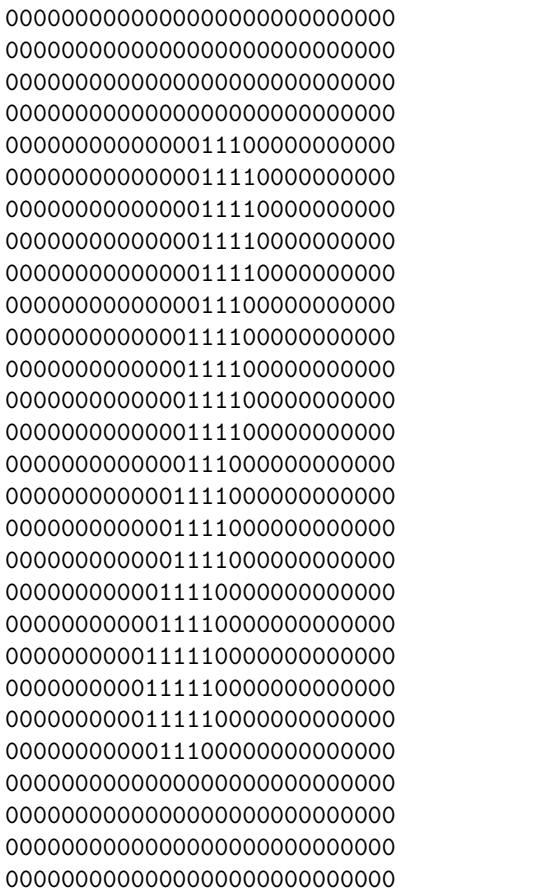
5.1 A



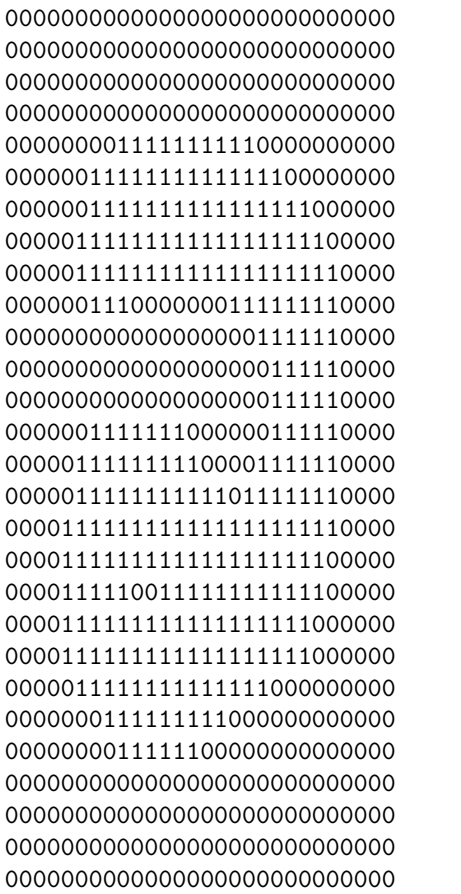
(a) Most prototypical for digit 0

(b) Least prototypical for digit 0

Figure 20: Most and least prototypical figures for digit 0

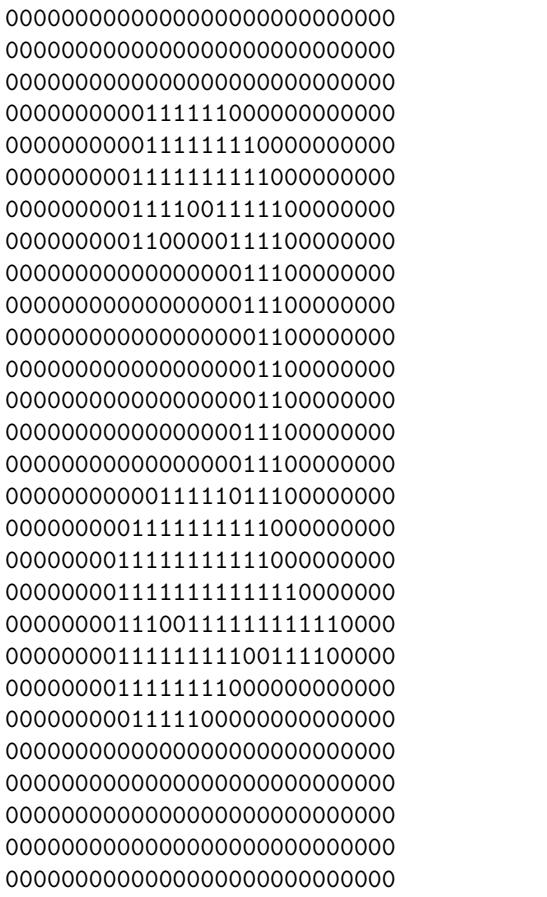


(a) Most prototypical for digit 1

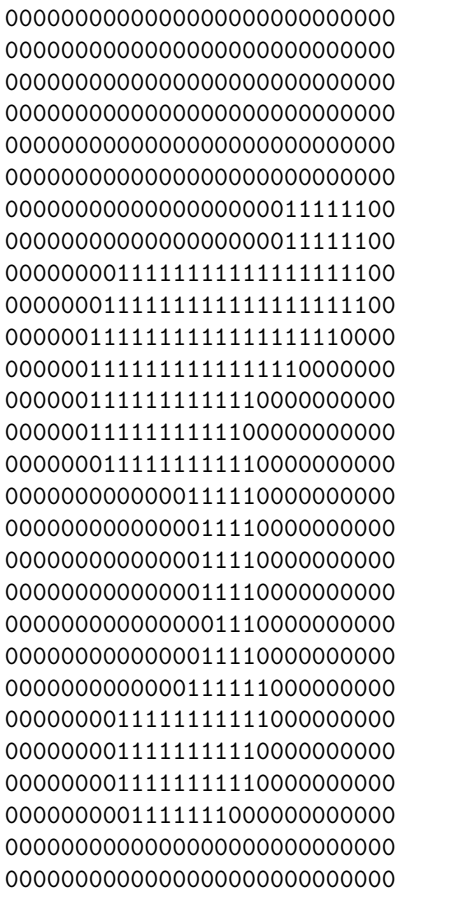


(b) Least prototypical for digit 1

Figure 21: Most and least prototypical figures for digit 1

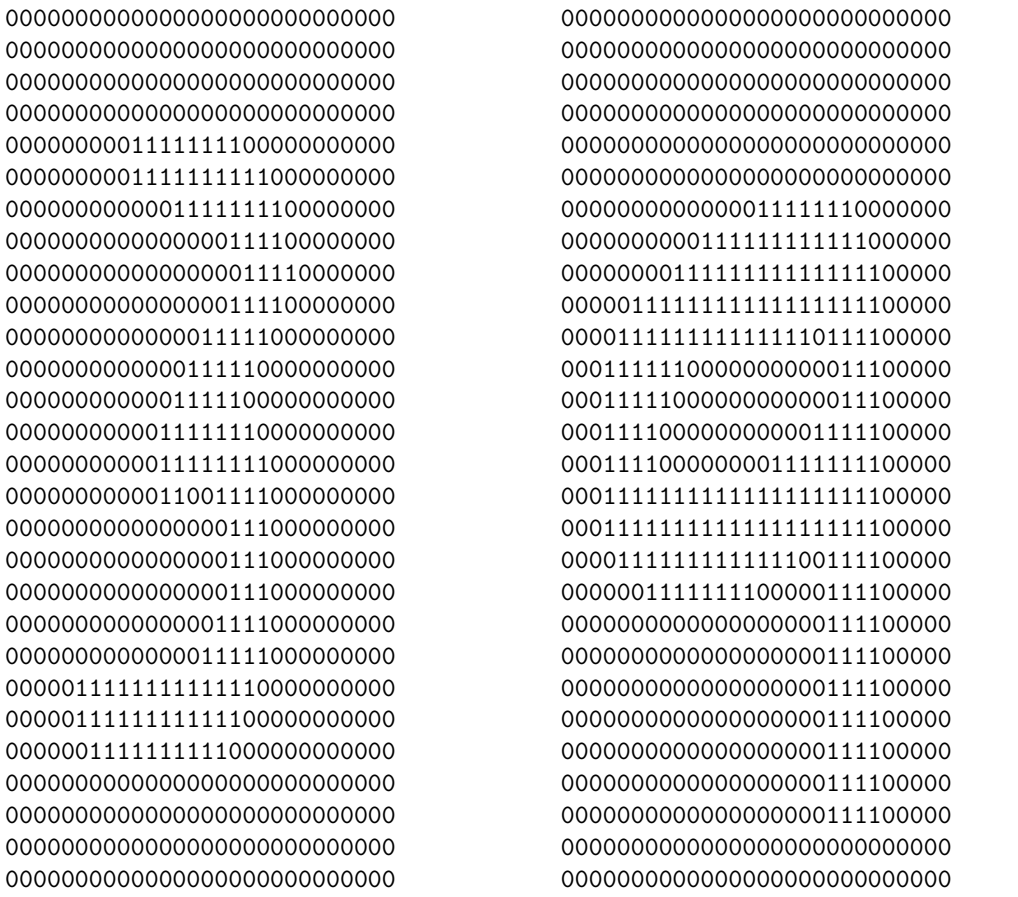


(a) Most prototypical for digit 2



(b) Least prototypical for digit 2

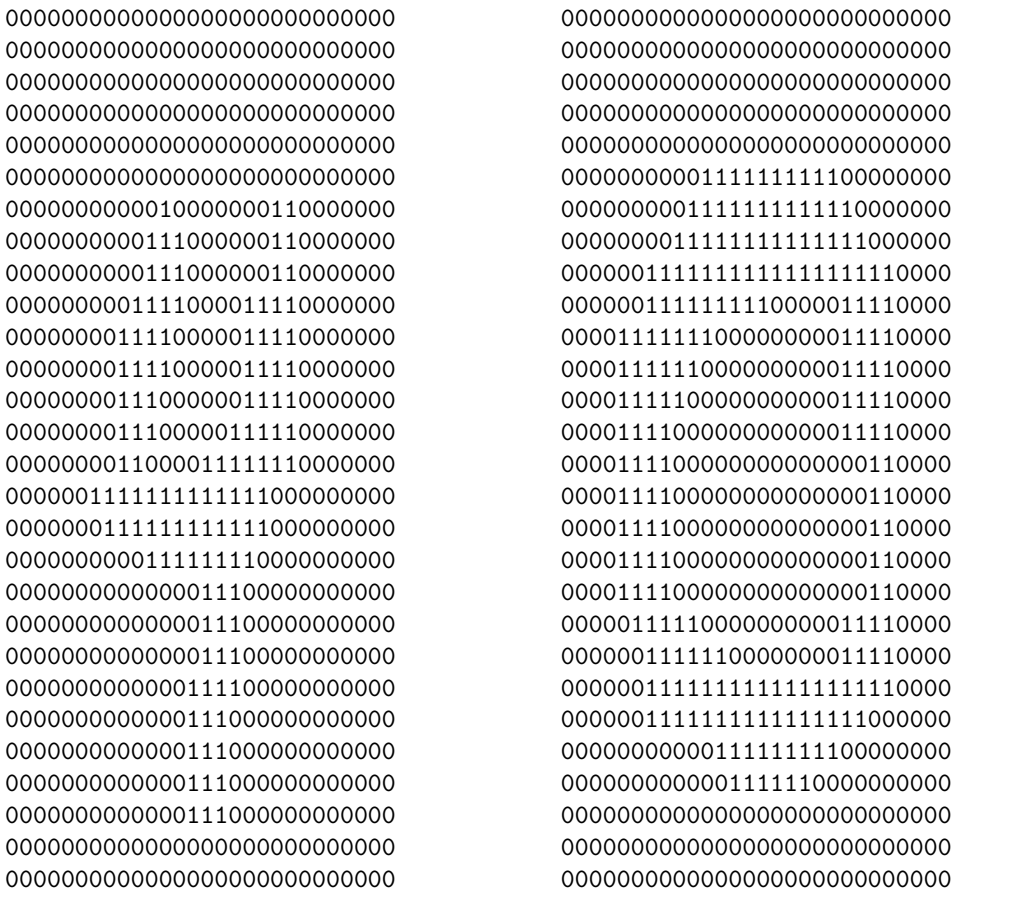
Figure 22: Most and least prototypical figures for digit 2



(a) Most prototypical for digit 3

(b) Least prototypical for digit 3

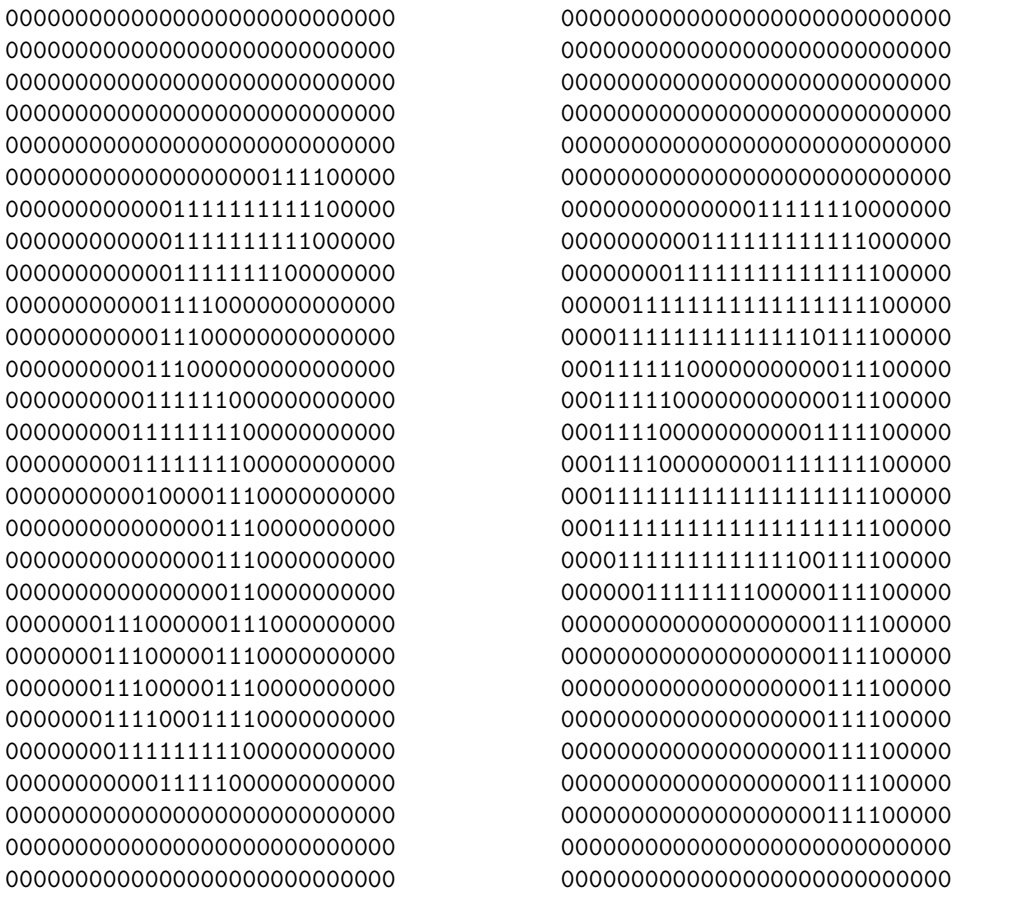
Figure 23: Most and least prototypical figures for digit 3



(a) Most prototypical for digit 4

(b) Least prototypical for digit 4

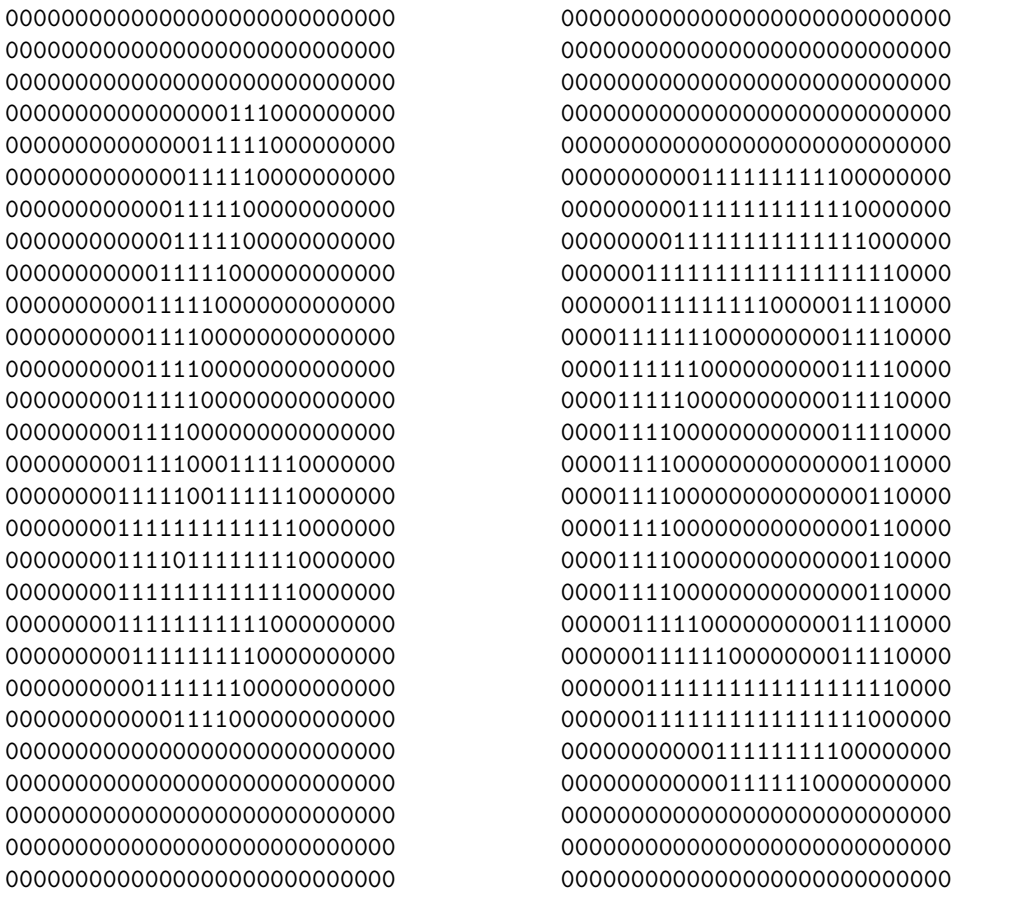
Figure 24: Most and least prototypical figures for digit 4



(a) Most prototypical for digit 5

(b) Least prototypical for digit 5

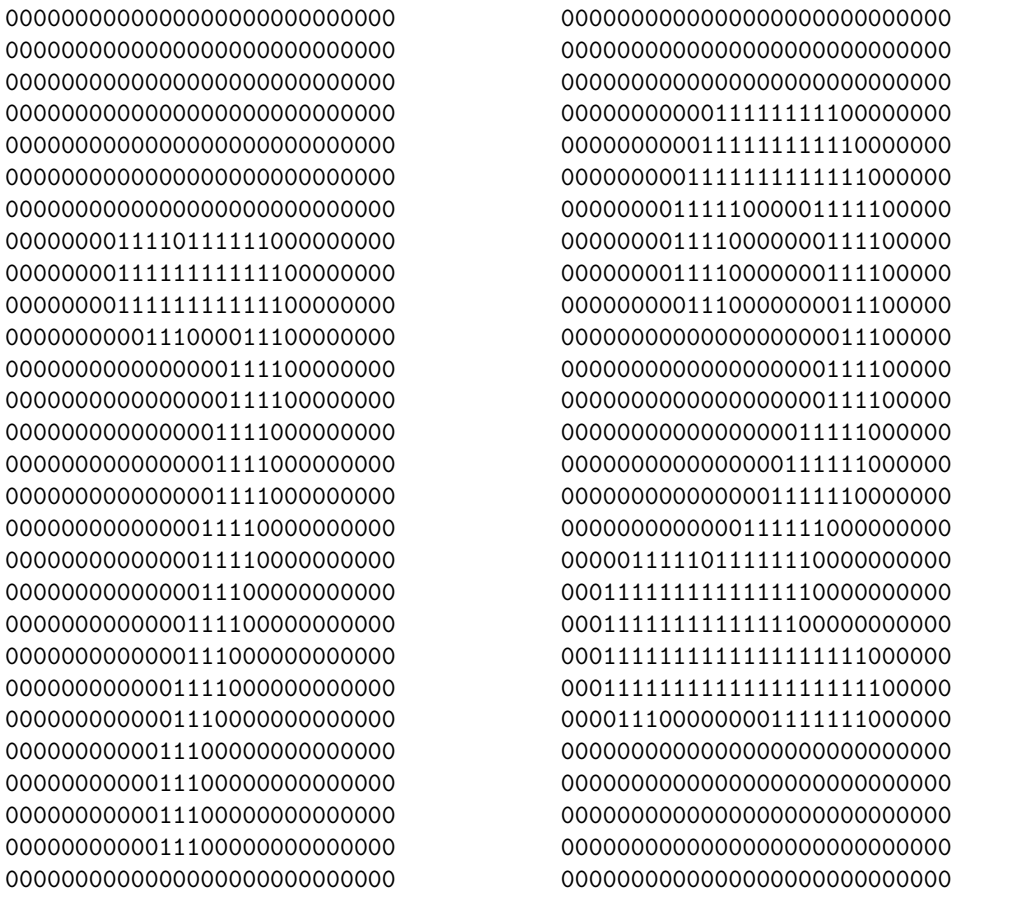
Figure 25: Most and least prototypical figures for digit 5



(a) Most prototypical for digit 6

(b) Least prototypical for digit 6

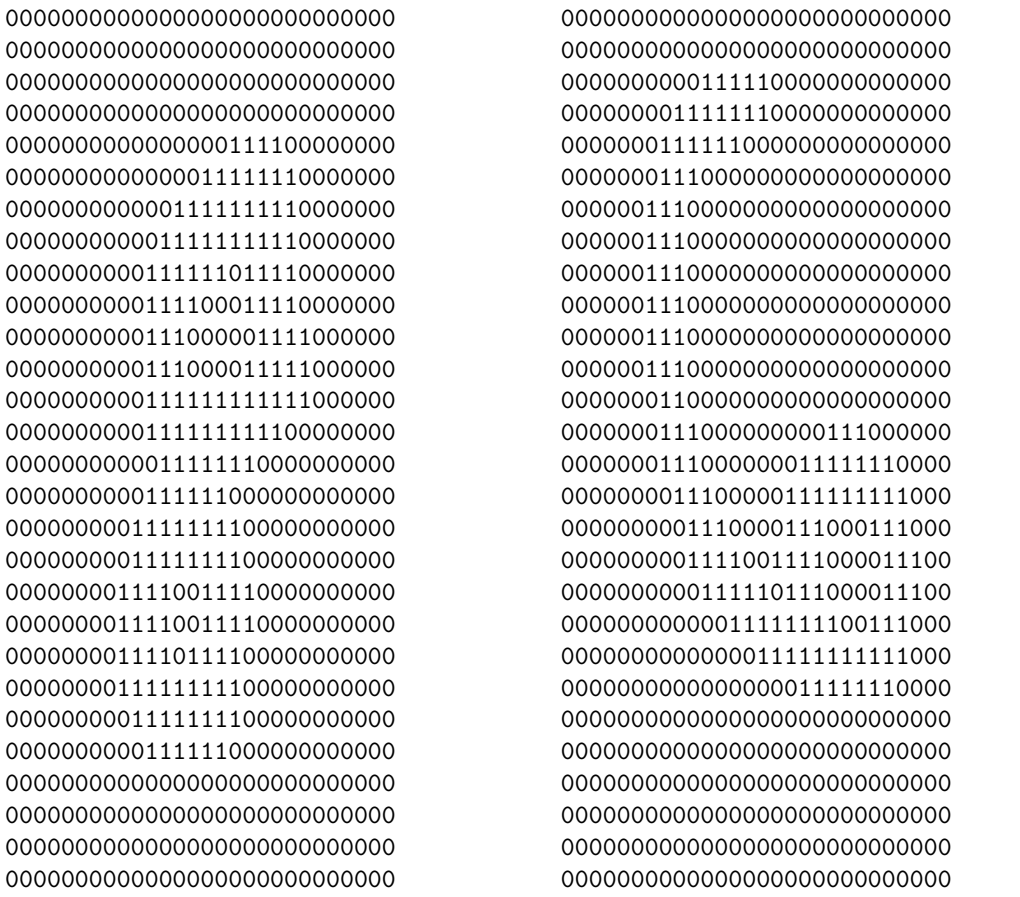
Figure 26: Most and least prototypical figures for digit 6



(a) Most prototypical for digit 7

(b) Least prototypical for digit 7

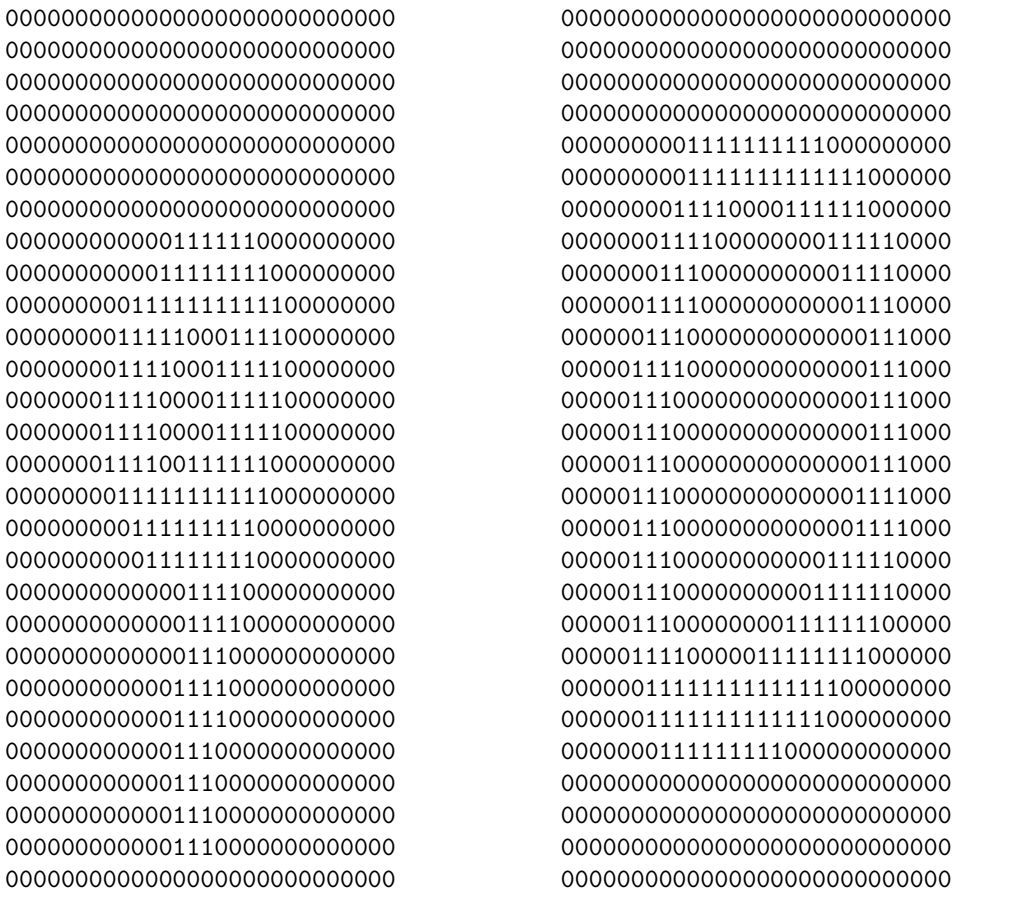
Figure 27: Most and least prototypical figures for digit 7



(a) Most prototypical for digit 8

(b) Least prototypical for digit 8

Figure 28: Most and least prototypical figures for digit 8



(a) Most prototypical for digit 9

(b) Least prototypical for digit 9

Figure 29: Most and least prototypical figures for digit 9