CS 440 Assignment 3

R section 3 credits

Group members:
Chendi Lin, Zhuoyue Wang, Zhaoyu Wu

1. Overview

In this assignment, we try to implement Naive Bayes classifiers to classify visual patterns and audio documents by given training and test data.

In the first part Digit Classification, we build the classifier to distinguish the handwritten digits. Firstly, we take some operation to parse the raw training and test data. Then we create a probability table to record the possibility for each pixel in the specific digit. After that, we train and test all of data by computing posteriori, odds ratio and confusion matrix. In conclusion, our classification has 77% Accuracy.

For second, we need to utilize Naive Bayes classifiers to classify audio signals like Hebrew words of “yes” and “no” and 5 digits. The energy levels are presented by “%”. Since the optimization of k value is not needed, by several manual trials, we picked k = 3 for 3-credit, 4-credit and extra assignments, and they all performed pretty well. The detailed implementations will be discussed in later sections.

2. Work distribution

Chendi Lin: audio classification, report
Zhuoyue Wang: digit classification, report
Zhaoyu Wu: digit classification, report

3. Section 1: Digit classification

3.1.

Part 1.1 Single pixels as features

For smoothing constant, if we set k=1, the accuracy rate is 77.1%; if k=0.7, the accuracy rate is 77.01%; if k=0.5, accuracy rate is 76.92%; if k=0.3, the accuracy rate is 77.23%; if we set k=0.1, the accuracy rate is 77.24%, which is the best accuracy rate we got. Consequently, we decided to set k=0.1.

(1) Classification rates for each digit:

0: 84.44%

1: 96.3%

2: 78.64%

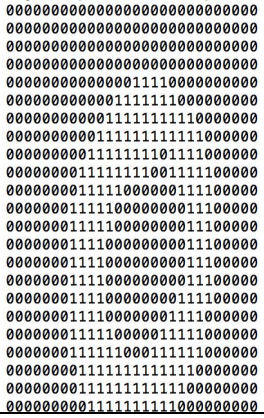
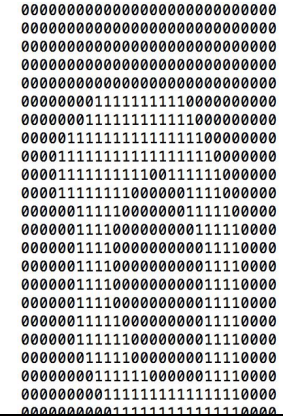
3: 80%
4: 74.77%
5: 68.48%
6: 76.92%
7: 72.64%
8: 60.19%
9: 80%

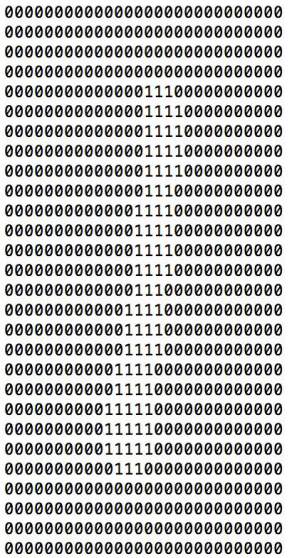
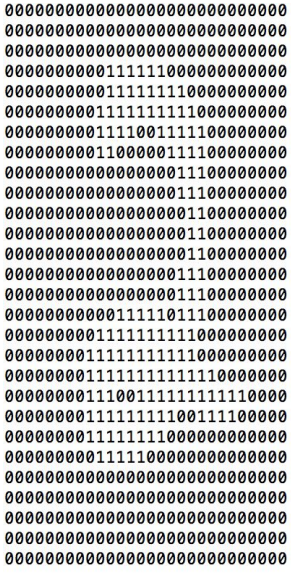
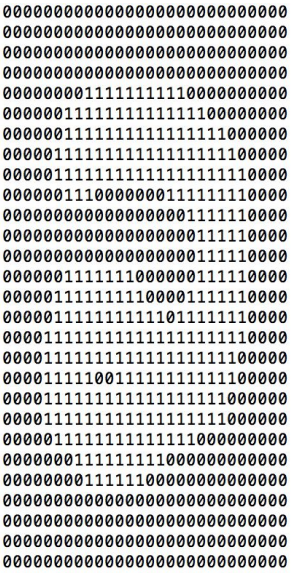
(2) Confusion Matrix:

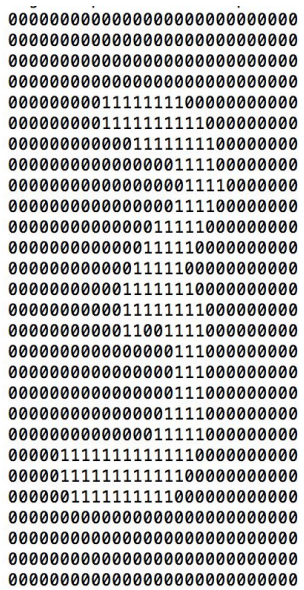
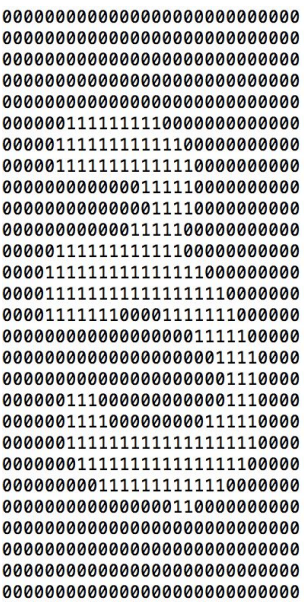
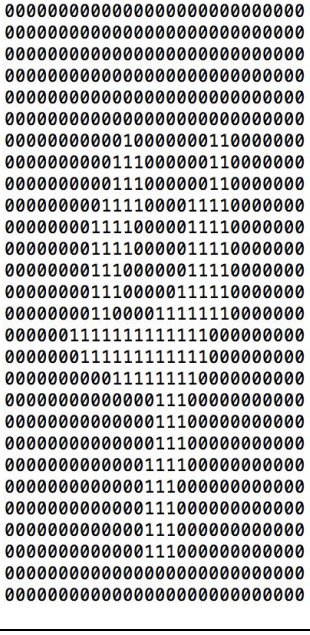
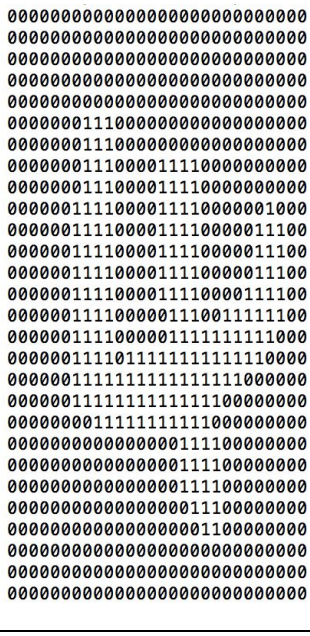
	0	1	2	3	4	5	6	7	8	9
0	**84.44%**	0%	1.11%	0%	1.11%	5.56%	3.33%	0%	4.44%	0%
1	0%	**96.3%**	0.93%	0%	0%	1.85%	0.93%	0%	0%	0%
2	0.97%	2.91%	**78.64%**	3.88%	1.94%	0%	5.83%	0.97%	4.85%	0%
3	0%	1%	0%	**80%**	0%	3%	2%	7%	1%	6%
4	0%	0%	0.93%	0%	**74.77%**	0.93%	3.74%	0.93%	1.87%	16.82%
5	2.17%	1.09%	1.09%	13.04%	3.26%	**68.48%**	1.09%	1.09%	2.17%	6.52%
6	1.1%	4.4%	4.4%	0%	4.4%	6.59%	**76.92%**	0%	2.2%	0%
7	0%	4.72%	3.77%	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%**

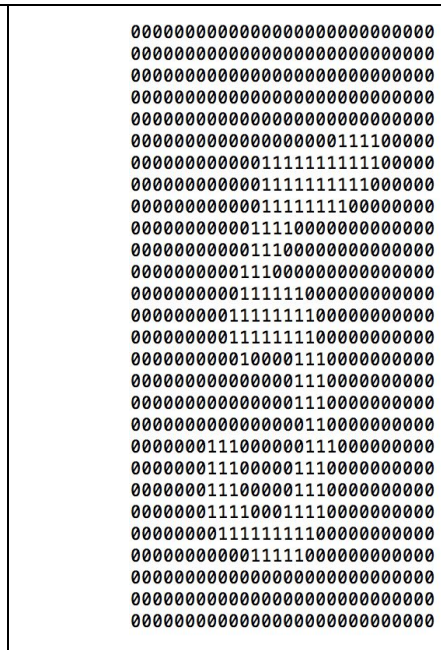
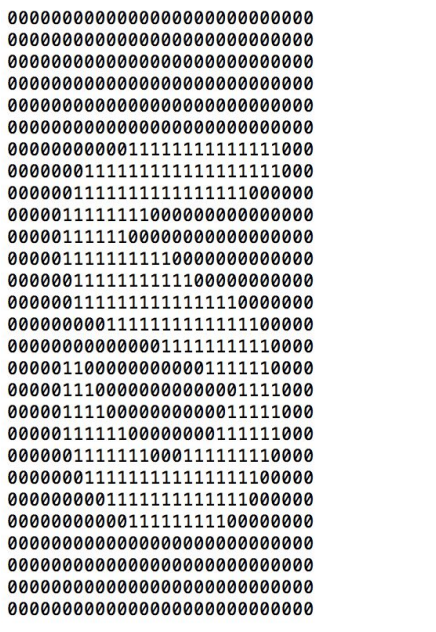
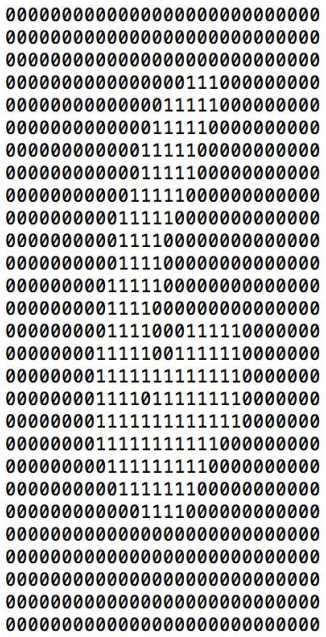
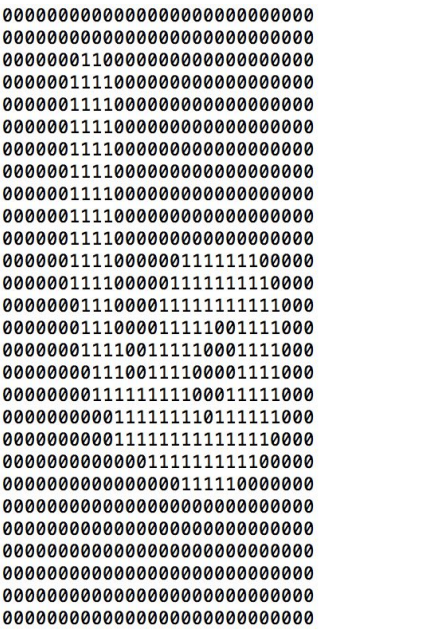
(3) Overall accuracy: 77.24%

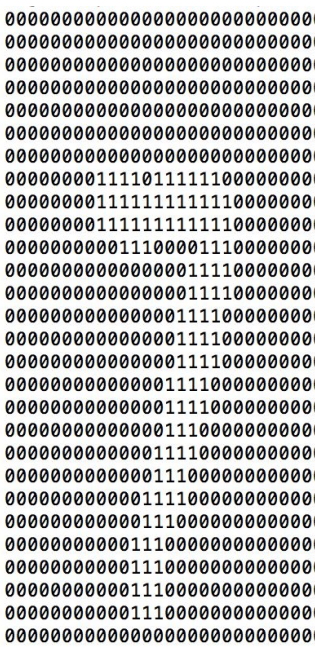
(4) Test examples of highest and lowest posterior probabilities

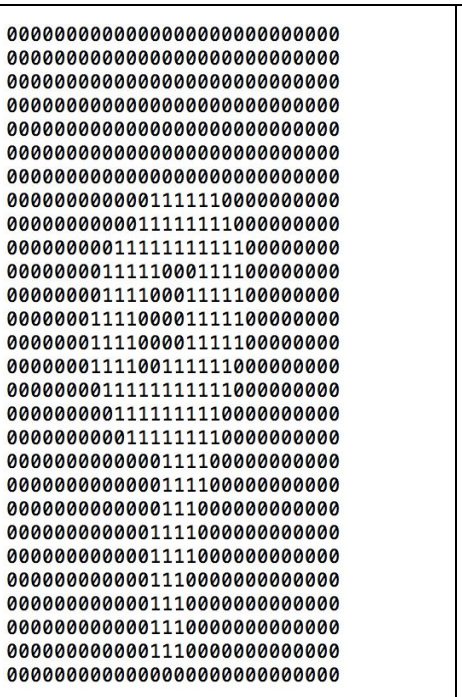
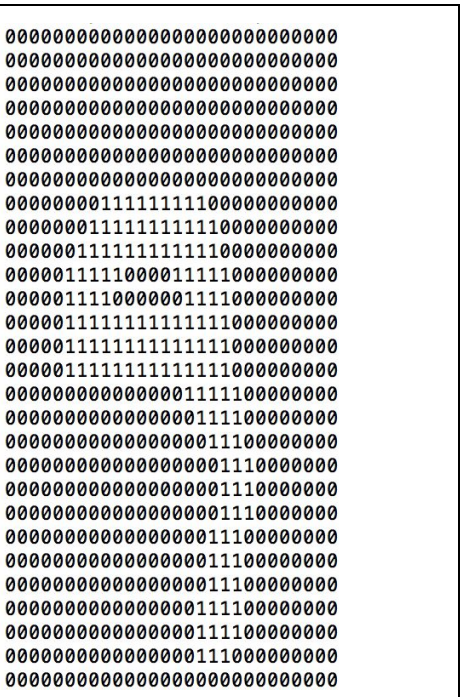
Digit	Highest posterior example	Lowest posterior example
0		

1		
2		

3		
4		

5		
6		

7		
8		

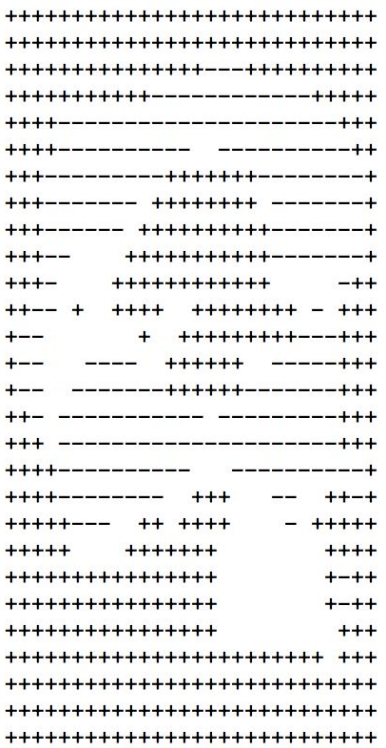
9		

(5) Four highest pairs of confusion rates and odd ratio

1. 4 vs 9

   Confusion rate: 16.82%

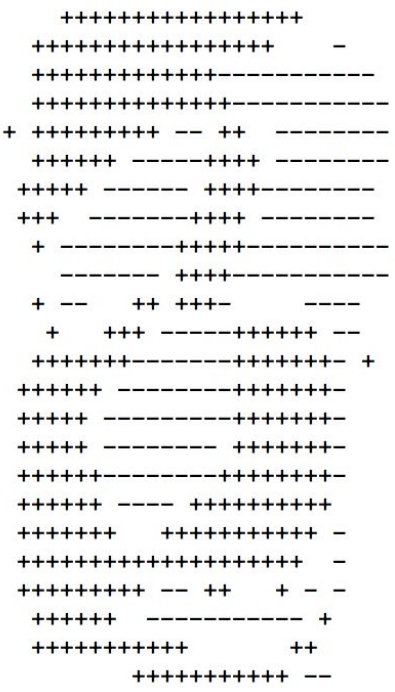
   Feature likelihoods and Odd radio:



2. 8 vs 3

   Confusion rate: 13.59%

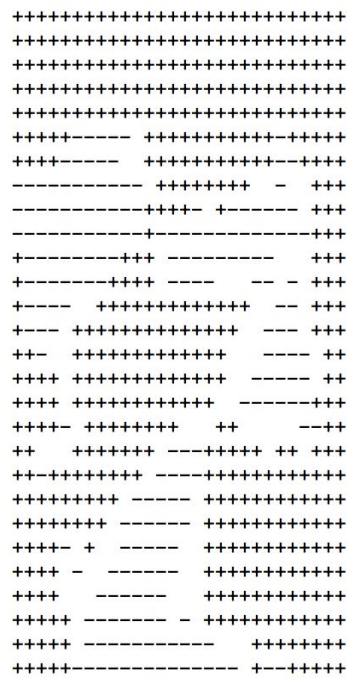
   Feature likelihoods and Odd radio:



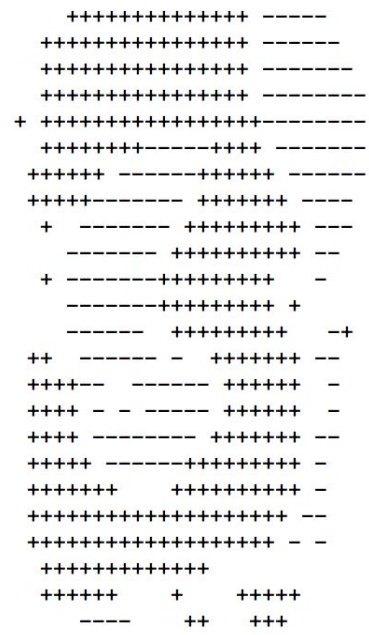
3. 7 vs 9

   Confusion rate: 13.21%

   Feature likelihoods and Odd radio:



4. 5 vs 3
   Confusion rate: 13.04%
   Feature likelihoods and Odd radio:



Part 1.2 Pixel groups as features (4-credit)

3.2. Extra credit: Ternary Features

In this part, we continue using k=0.1 as smoothing constant, which gives rise to the best accuracy.

(1) Classification rates for each digit:

0: 83.33%

1: 95.37%

2: 76.7%

3: 81%

4: 76.64%

5: 68.48%

6: 81.32%

7: 73.58%

8: 60.19%

9: 81%

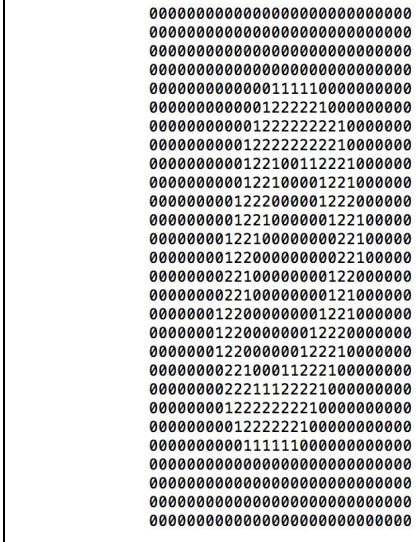
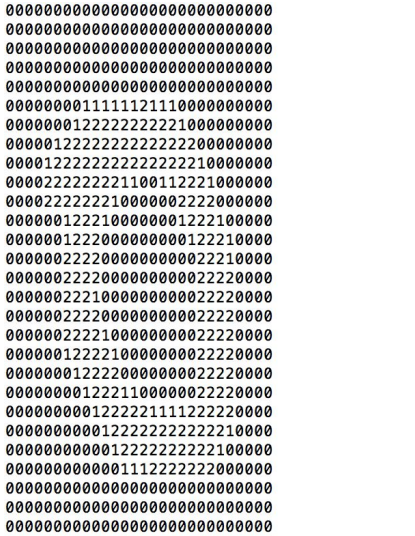
(2) Confusion Matrix:

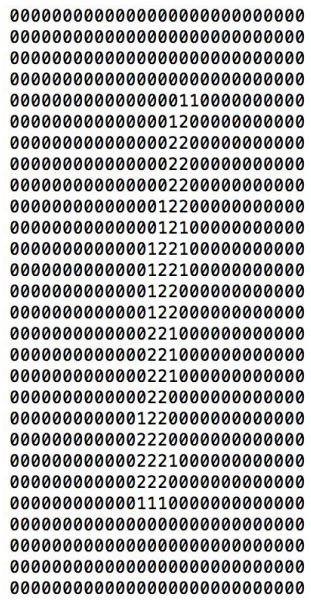
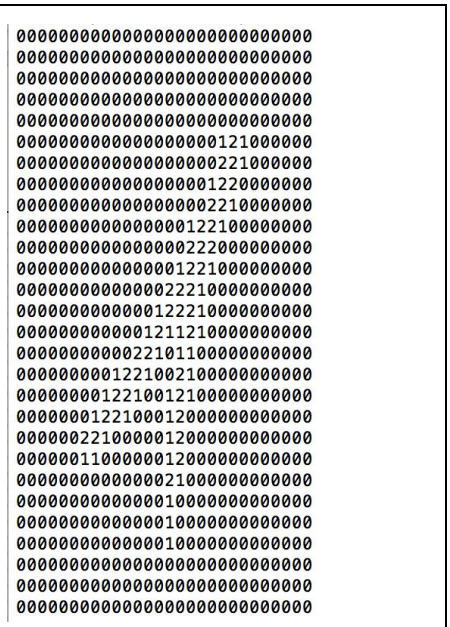
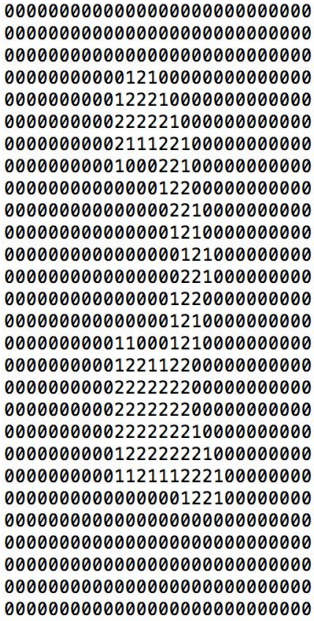
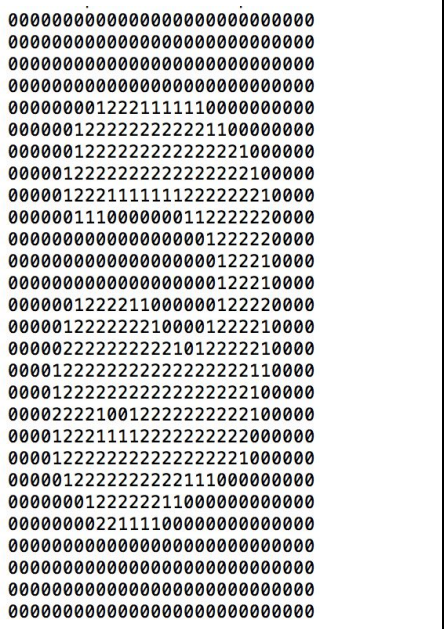
	0	1	2	3	4	5	6	7	8	9

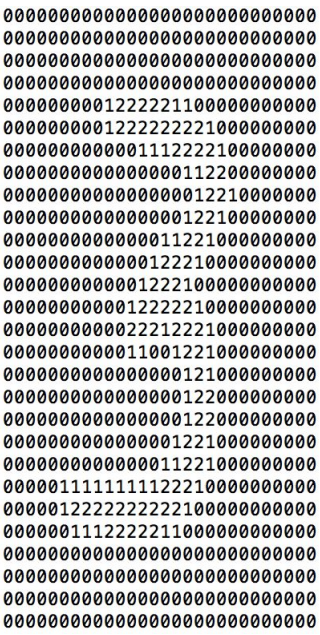
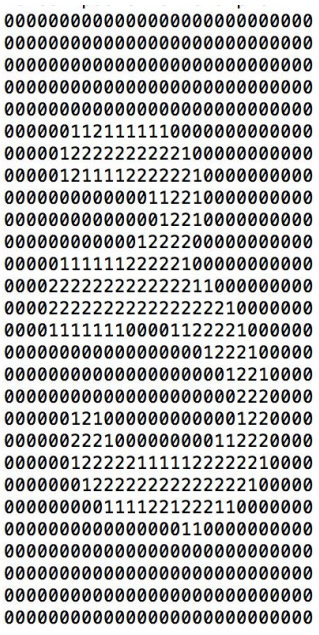
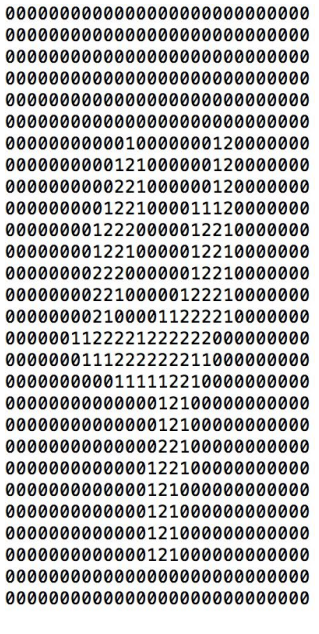
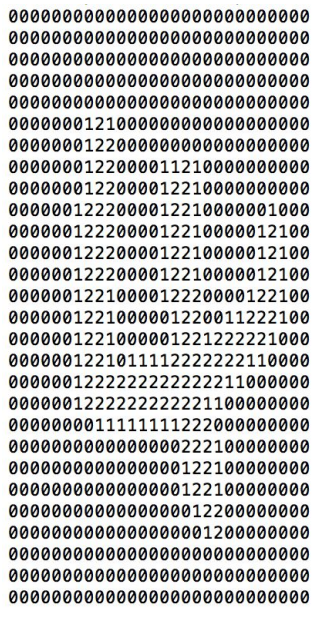
0	**83.33%**	0%	1.11%	0%	0%	6.67%	3.33%	0%	5.56%	0%
1	0%	**95.37%**	0%	0%	0%	1.85%	0.93%	0%	1.85%	0%
2	0.97%	2.91%	**76.7%**	3.88%	0.97%	0.97%	6.8%	1.94%	4.85%	0%
3	0%	1%	0%	**81%**	0%	3%	2%	6%	2%	5%
4	0%	0%	0%	0%	**76.64%**	0.93%	2.8%	0.93%	1.87%	16.82%
5	2.17%	1.09%	1.09%	13.04%	3.26%	**68.48%**	1.09%	1.09%	2.17%	6.52%
6	0%	3.3%	3.3%	0%	4.4%	5.49%	**81.32%**	0%	2.2%	0%
7	0%	5.66%	2.83%	0%	2.83%	0%	0%	**73.58%**	2.83%	12.26%
8	0.97%	0.97%	2.91%	11.65%	2.91%	8.74%	0&	0.97%	**60.19%**	10.68%
9	1%	0%	0%	2%	10%	2%	0%	2%	2%	**81%**

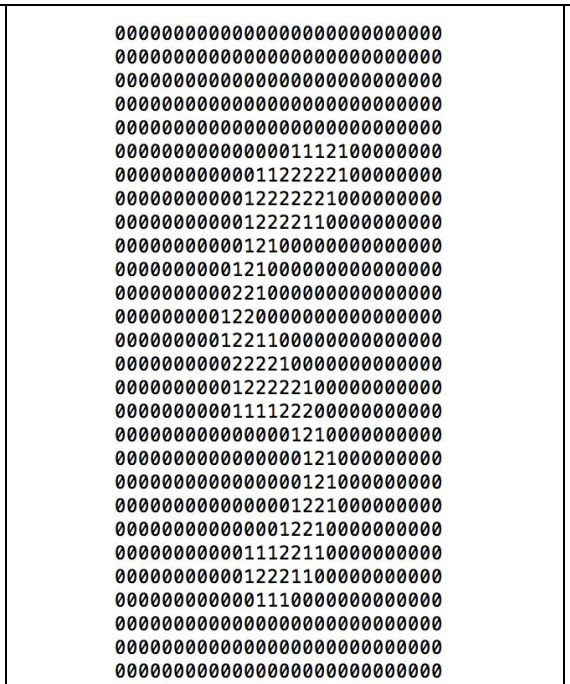
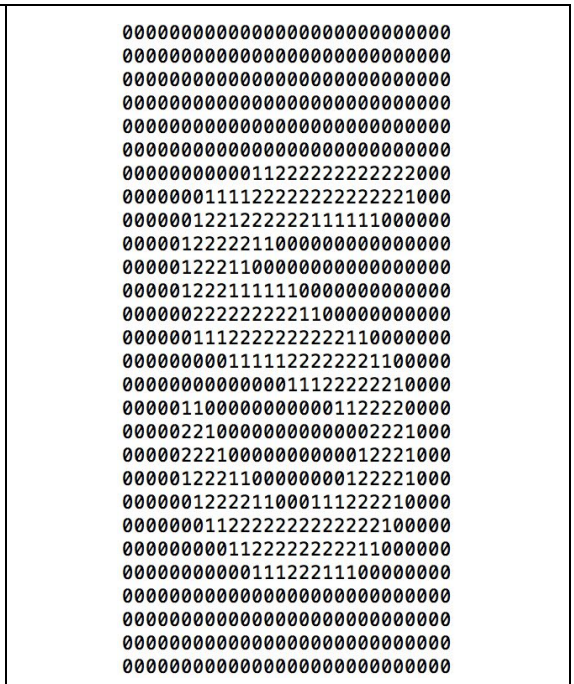
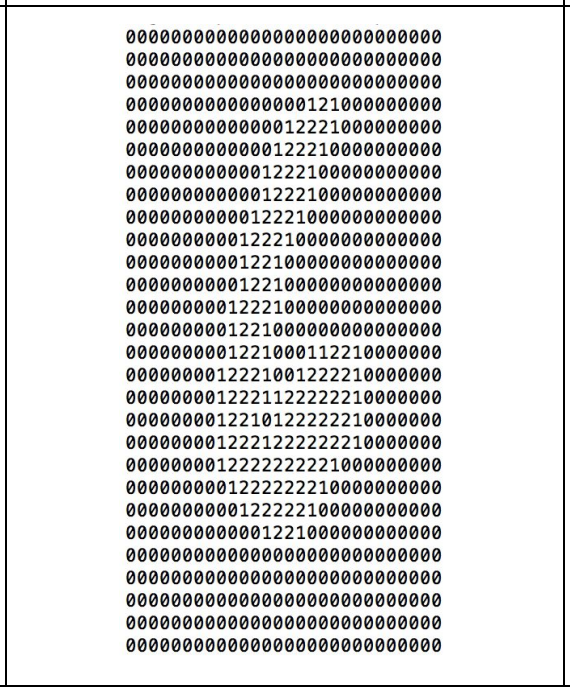
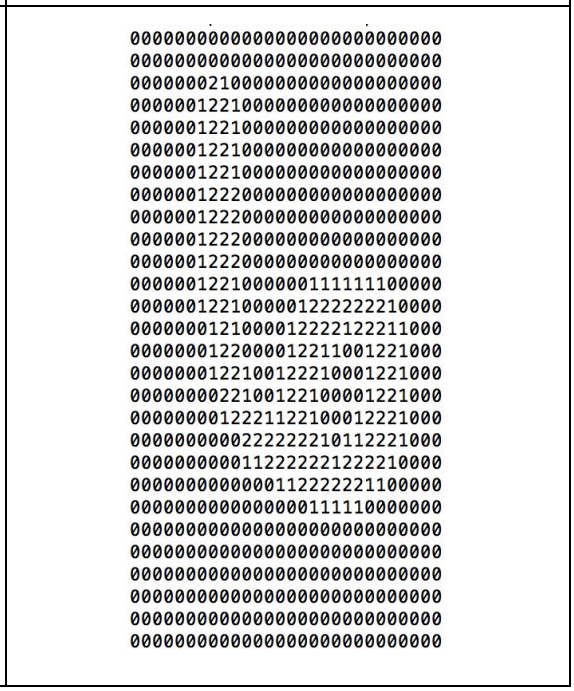
(3) Overall accuracy: 77.76%

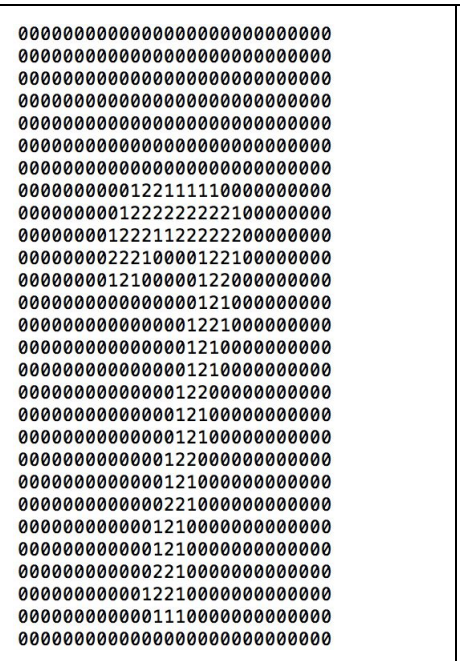
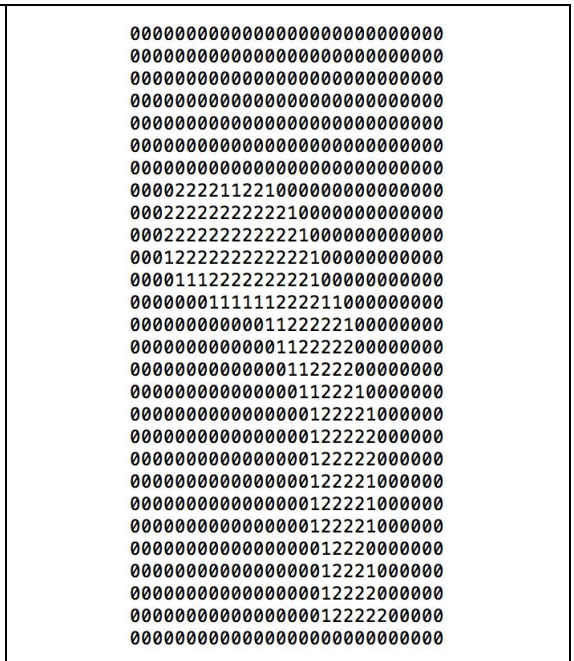
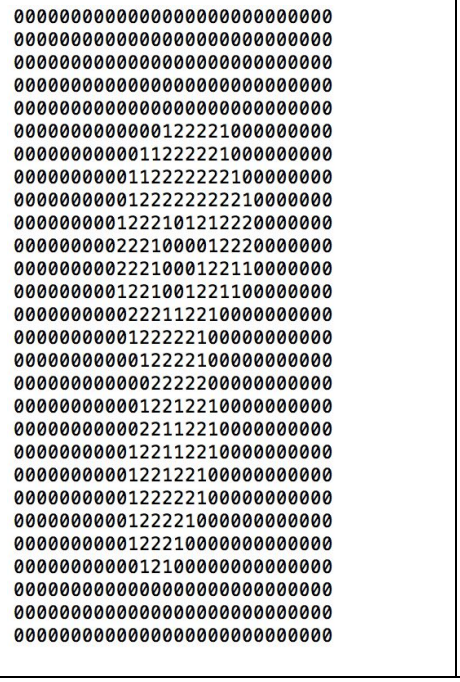
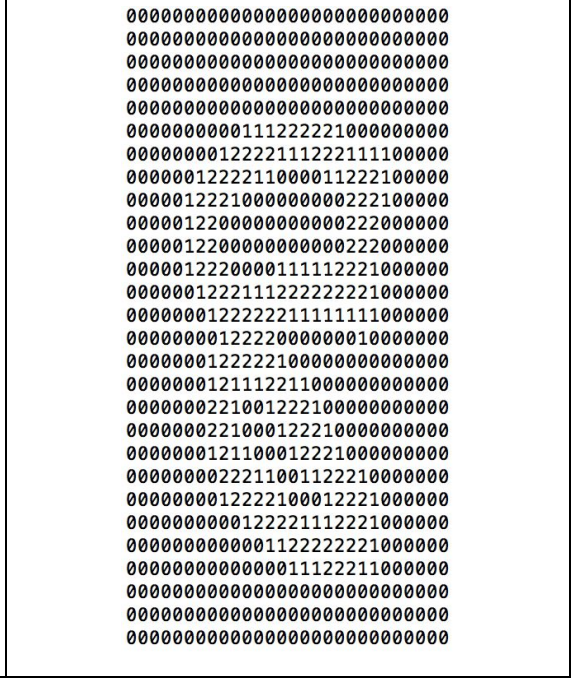
(4) Test examples of highest and lowest posterior probabilities

Digit	Highest posterior example	Lowest posterior example
0		

1		
2		

3		
4		

5		
6		

7		
8		

9	00000000000000000000000000000 00000000000000000000000000000 00000000000000000000000000000 00000000000000000000000000000 00000000000000000000000000000 00000000000000000000000000000 00000000000001121100000000000 00000000000112222200000000000 00000000001222112210000000000 00000000012221001210000000000 00000000012200001210000000000 00000000122000012100000000000 00000000121000001220000000000 00000000122000012210000000000 00000000122122222000000000000 00000000001222222100000000000 00000000000001122000000000000 00000000000001210000000000000 00000000000002221000000000000 00000000000012220000000000000 00000000000122100000000000000 00000000000122000000000000000 00000000000221000000000000000 00000000000122000000000000000 00000000000121000000000000000 00000000000121000000000000000 00000000000000000000000000000 00000000000000000000000000000	00000000000000000000000000000 00000000000000000000000000000 00000000000000000000000000000 00000000000000000000000000000 00000000000000000000000000000 00000000000000000000000000000 00000000000000000000000000000 00000001122222210000000000 00000012222222221000000000 00000122221112221000000000 00001222100001122100000000 00001221000000112100000000 00002221111111112100000000 00001222222222221000000000 00001122221122222100000000 00000000000000122210000000 00000000000000012220000000 00000000000000001220000000 00000000000000002210000000 00000000000000002210000000 00000000000000002210000000 00000000000000012200000000 00000000000000012200000000 00000000000000022200000000 00000000000000122200000000 00000000000000222100000000 00000000000000122000000000 00000000000000000000000000000

3.3. Extra credit: Face data

(1) Classification rate:

Class 0: 89.61%

Class 1: 86.3%

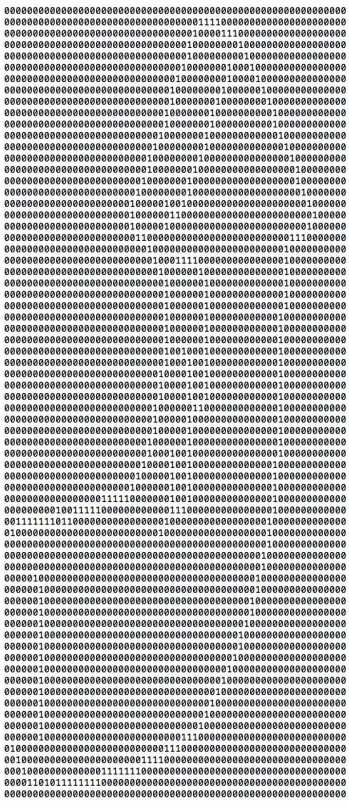
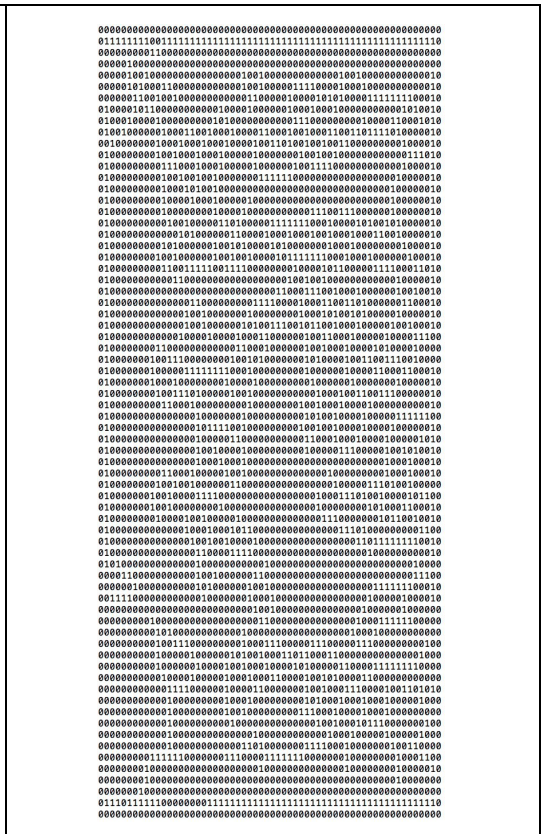
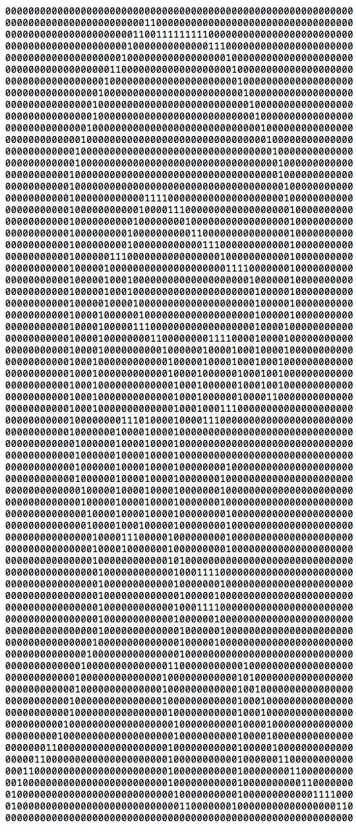
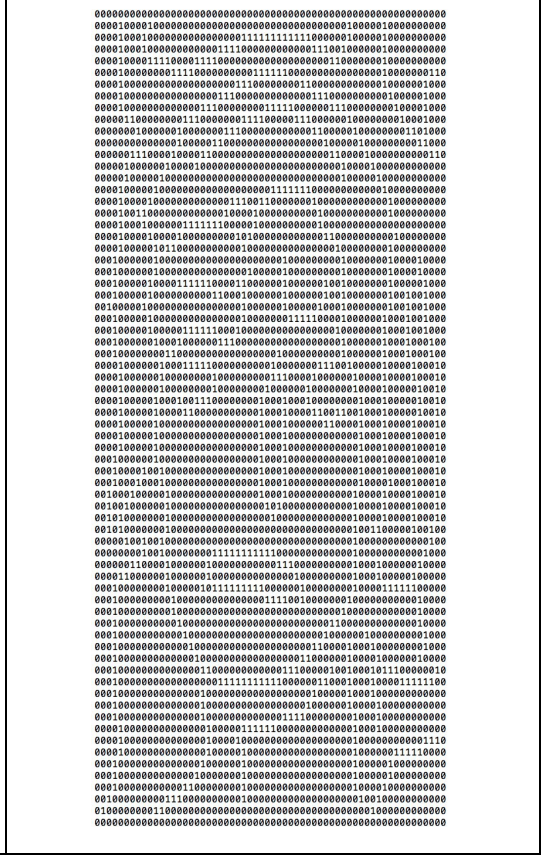
(2) Confusion Matrix:

	0	1
0	**89.61%**	10.39%
1	13.7%	**86.3%**

(3) Overall accuracy: 87.96%

(4) Test examples of highest and lowest posterior probabilities:

	Highest posterior example	Lowest posterior example

0		
1		

4. Part 2: Audio Classification

4.1. 3 credit

The implementations of this assignment are fairly trivial. Firstly, we read all the training data, and convert the "%" to be 1, " " to be 0. Since we only have two classes here, "yes" and "no", we can set up two probability tables for these two classes. Each entry in the probability tables records the frequency of "%" at the ij-th block of the spectrogram for each class.

To avoid zero counts, we applied Laplace smoothing here. So now, each entry of the probability table of the given class is calculated as:

$P(F_{ij} = f \mid class)$ = (# of times pixel (i,j) has value f in training examples from this class + k) / (Total # of training examples from this class + k*V)

Where k is a constant ranging from 0.1 to 10, and V is the number of possible values that the feature can take on, which is 2 in this case. Since optimization of k value is not required in this assignment, we randomly picked a well-performed value k = 3 for all the tasks in part 2.

To perform maximum a posteriori (MAP) classification, we need to maximize P(class | evidence). Since P(class | evidence) is proportional to P(evidence | class) P(class), assuming each feature is conditionally independent given the class, we can calculate the maximum posteriori by finding the maximum of the product of the prior and the likelihood. The prior of each class is easy to find. It can be calculated by the empirical frequencies of different classes in the training set. To avoid underflow, we calculate the likelihood and prior in log scale. Thus, instead of calculating $P(class) * \Pi P(f_{i,j} \mid class)$, we calculated $log\, P(class) + \Sigma\, log\, P(f_{i,j} \mid class)$, and pick the class with the largest calculated value.

The results along with the confusion matrix are shown below:

```
percentage of correctness for yes_test = 0.98
percentage of correctness for no_test = 0.96
the overall correctness = 0.97
the confusion matrix is
yes    [0.98, 0.02]
no     [0.04, 0.96]
time taken:  1.3944408893585205
```

The overall accuracy is 97%, which is pretty good. The classification of “yes” cases is slightly better than “no” cases.

It took 1.39 s for the classification. This is used for comparison in the extra credit part.

4.2. 4 credit (extra credit for 3-credit students)

In this assignment, we need to classify audio digits 1-5 spoken by four different speakers. The implementations are very similar to the previous one, just with more classes (five here).

The results along with the confusion matrix are shown below:

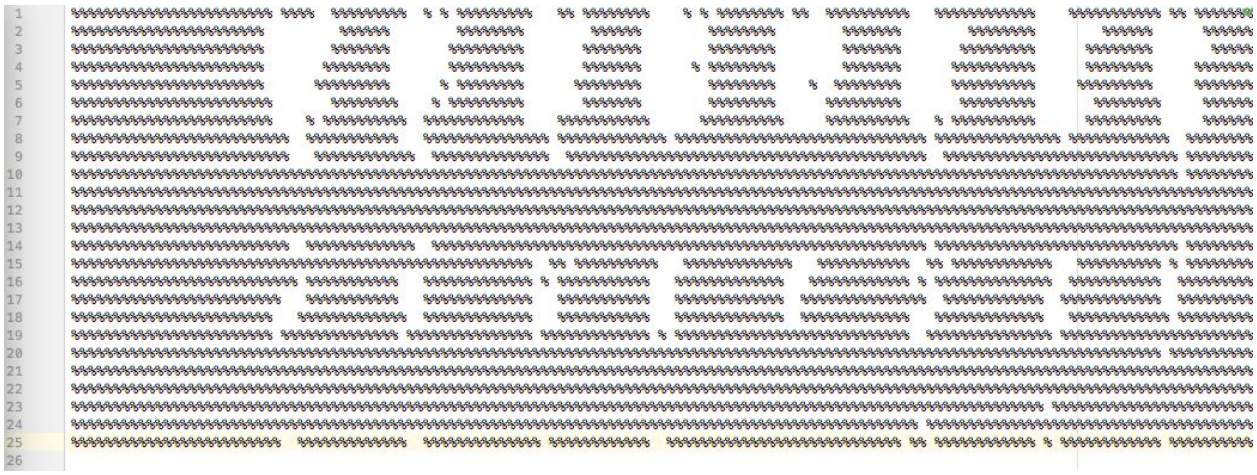
```
The labels of the testing data are
[1, 1, 1, 1, 4, 1, 1, 3]
[2, 2, 2, 2, 2, 2, 2, 2]
[3, 3, 3, 3, 3, 3, 3, 3]
[2, 4, 4, 2, 4, 4, 4, 2]
[3, 5, 5, 5, 5, 5, 5, 5]
percentage of correctness for testing = 0.85
the confusion matrix is
[[ 0.75   0.     0.125  0.125  0.   ]
 [ 0.     1.     0.     0.     0.   ]
 [ 0.     0.     1.     0.     0.   ]
 [ 0.     0.375  0.     0.625  0.   ]
 [ 0.     0.     0.125  0.     0.875]]
```

The classification is not as accurate as part 1, with 85% accuracy. It is reasonable because it has more classes and the problem is more complicated.

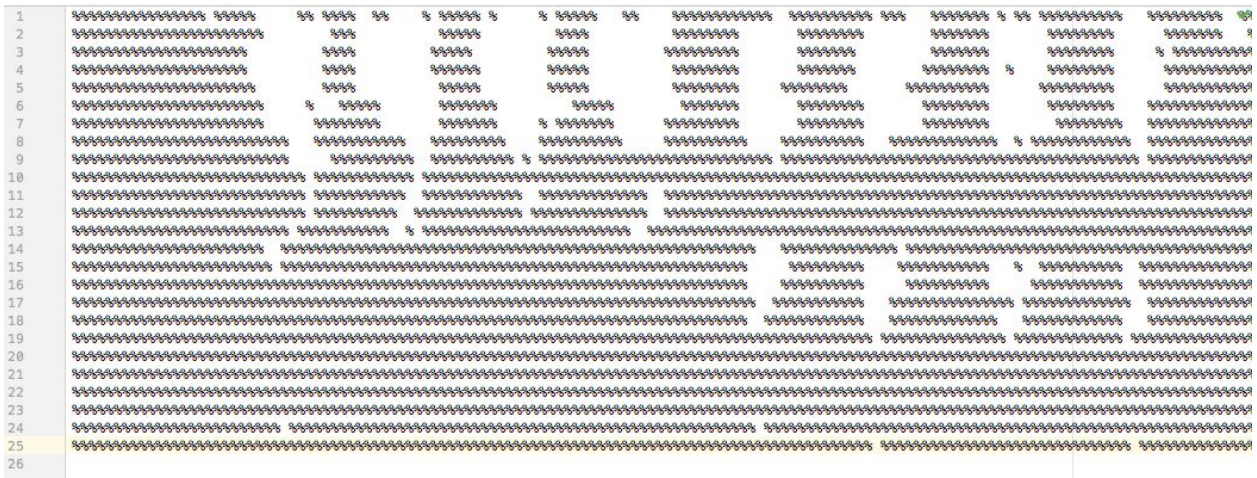
4.3. Extra credit 1: Unsegmented Training Data

In this assignment, we need to train the classifier using the unsegmented version of the same data. We still used exactly the same way as the 3-credit assignment to implement the classifier. The only challenge here is how to divide the training data into individual words. By reading the filename, we can obtain the order of the sequence of yes’s and no’s. Each training data is a txt file with 25 rows and 151 columns. Since each word should have only 25 by 10 characters, there must be redundant characters included in the training data. We used a naive but useful method to filter them. By reading the training data manually, especially “1_1_1_1_1_1_1_1.txt” and “0_0_0_0_1_1_1_1”, it is easy to tell where “yes” and “no” should start. We figured out that, for a sequence starting

with “yes”, usually the first 24 columns are noises, and for a sequence starting with “no”, usually the first 29 columns are noises. For example, “1_1_1_1_1_1_1_1.txt” looks like:



And “0_0_0_0_1_1_1_1.txt” is like:



The results and confusion matrix shown below proved our ideas.

```
percentage of correctness for yes_test = 0.98
percentage of correctness for no_test = 0.92
the overall correctness = 0.95
the confusion matrix is
yes    [0.98, 0.02]
no     [0.08, 0.92]
time taken:  1.3575551509857178
```

The results are actually pretty good, a little bit worse than the one with segmented training data, but still satisfactory. We still have 98% accuracy for

“yes” label, but only 92% accuracy for “no” label. The overall accuracy is 95%. The confusion matrix is also attached above.

4.4. Extra credit 3: Average Column

Here, instead of every element of the test image as a feature, we compute the average of each row first. Then we can decrease the number of features from i*j to i, which can save the computational cost a lot, but the number of values that a feature can take becomes 11.

The results along with the confusion matrix are shown below:

```
percentage of correctness for yes_test = 0.98
percentage of correctness for no_test = 0.92
the overall correctness = 0.95
the confusion matrix is
yes    [0.98, 0.02]
no     [0.08, 0.92]
time taken:  0.39024782180786133
```

We can tell from the results above that, by sacrificing a little bit accuracy (mainly from the classification of “no” data), we gained a huge improvement in computational efficiency.