ECE448 MP3 report 4 credits students

Yu\_Fan Fang, Chieh Hsu, Yu\_ming Huang

## Part 1: Naive Bayes Classifiers on Digit classification

### 1-1:  Single pixels as features

**Implementation:**

We first read the data and store them as ArrayList<ArrayList<String>> in the variable “dataset”. We then store each digit figure with an array with 1024 elements. We create a 3 dimensional array(10\*32\*32) called “probability” to store the figure into the corresponding digit(0~9). We also create a 2 dimension array called “count feature” and store the number of occurrence of the features(i,jth value is 1) in it, and then we normalize it with the total size of the dataset. Thus, we can get the likelihood function. We also calculate the prior probability as the number of the occurrence of each digit divided by the total size of the training set.

For the test set, we read the data into an ArrayList<String> as testset, where the size of the string is 1024(32\*32). And we use the logarithm of the prior probability calculated and sum it with other 1024 log likelihood function and get the final max apriori probabioity. We then decide that which number is most likely to be the result based on it.

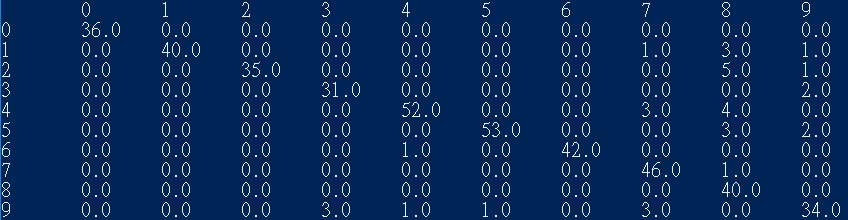
With the smoothing factor chosen as 0.1, we are able to get the test set accuracy as 92.12%. We found that with no smoothing factor, the accuracy is around 85%, however, when the smoothing factor is larger than 0.1, the accuracy also keeps decreasing.

For the confusion matrix, we calculate all the test tokens and assign the real input as rows, and the classify result as the columns. With the confusion matrix, we are able to calculate the “precision” and “recall”.

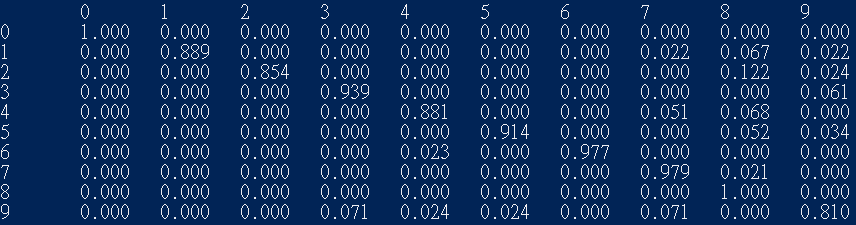
For the odd ratio problem, we calculate the ratio of the conditional probability of “given class 1 and the feature value i,j is 1” and “given class 2 and the feature value i,j is 1”. With such method, we are able to tell that which symbols best separate the 2 similar tokens.

**Picture 1**

**Confusion matrix total count:**

****

**Confusion matrix:**

****

**Defining precision as , and recall as**

**Then for each digit:**

0:precision:100%, recall:100%

1:precision:100%, recall:88.9%

2:precision:100%, recall:85.3%

3:precision:91.1%, recall:93.9%

4:precision:96.3%, recall:88.1%

5:precision:98.1%, recall:91.4%

6:precision:100%, recall:97.7%

7:precision:86.8%, recall:97.9%

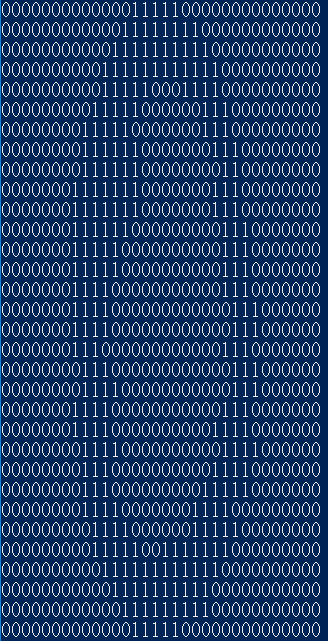
8:precision:71.4%, recall:100%

9:precision:85%, recall:81%

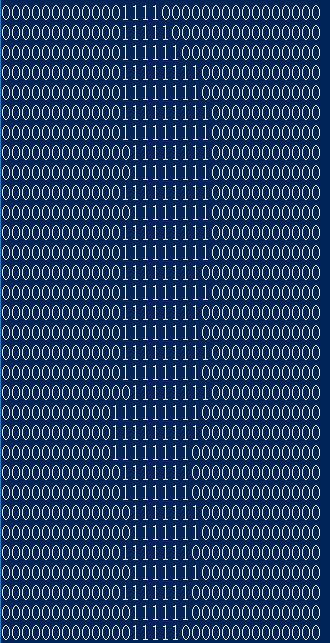
**Test tokens with the highest/the lowest probability for according to the classifier:**

**Highest:**

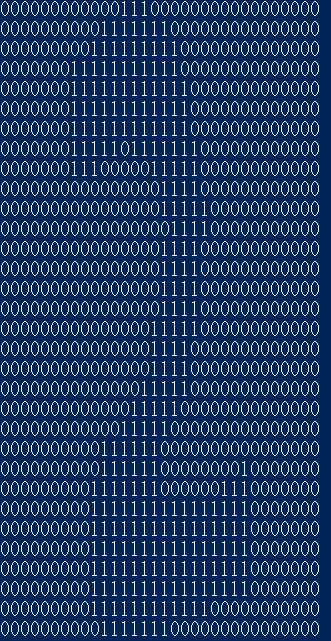
**0:**

****

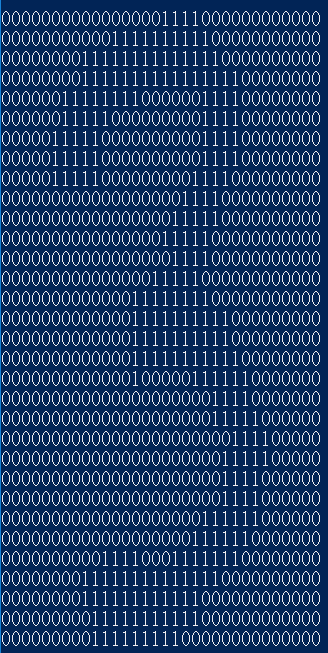
**1.**

****

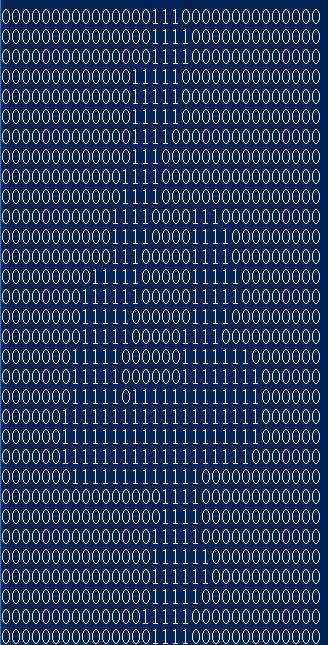
**2.**

****

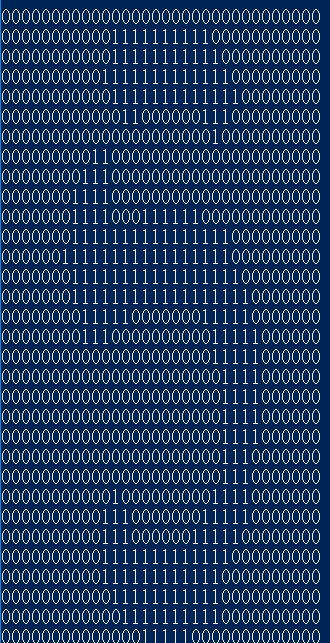
**3.**

****

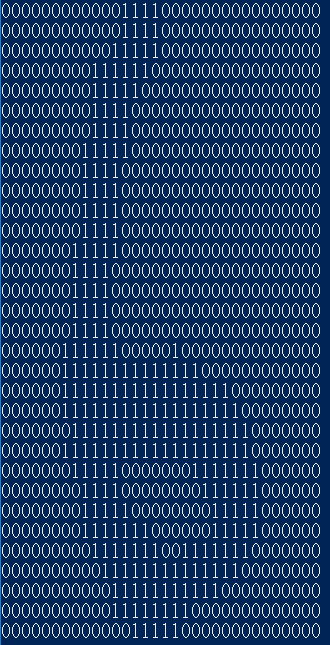
**4.**

****

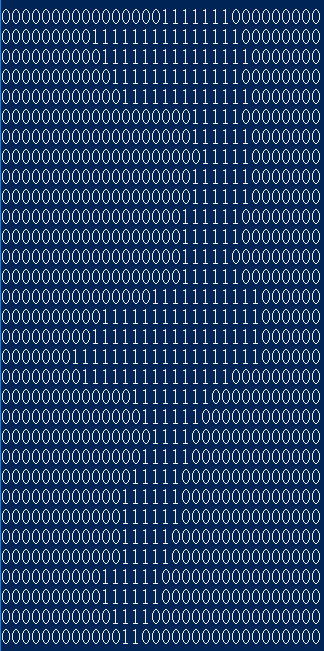
**5.**

****

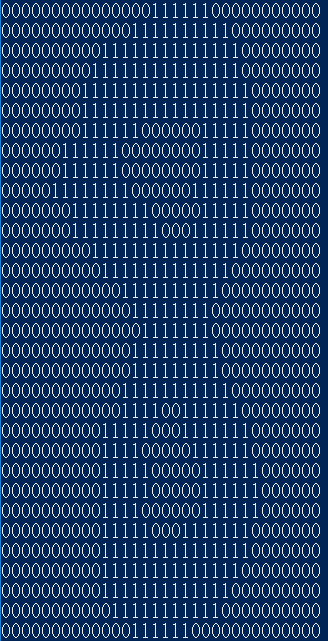
**6.**

****

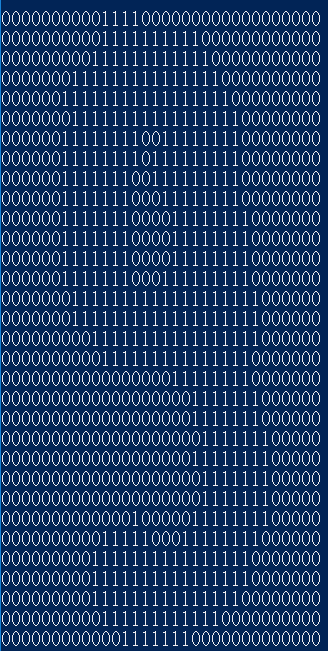
**7.**

****

**8.**

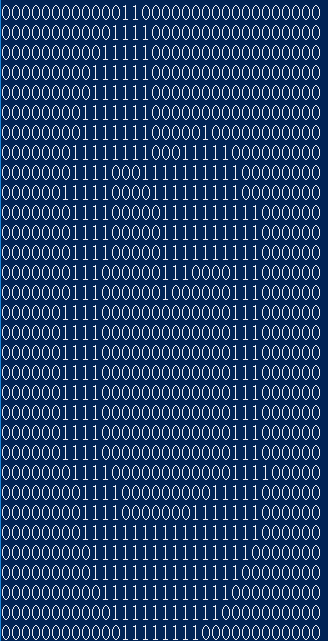
****

**9.**

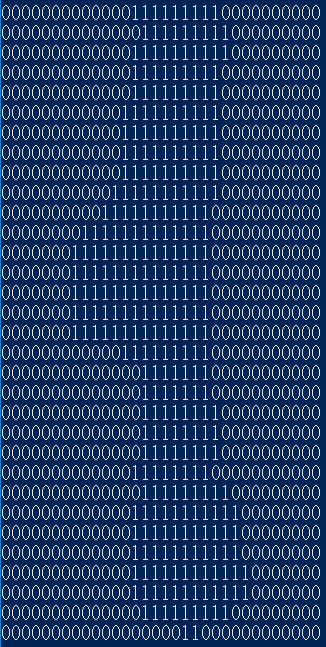
****

**Lowest:**

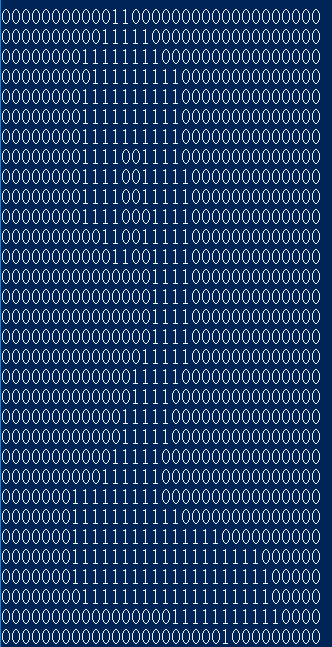
**0.**

****

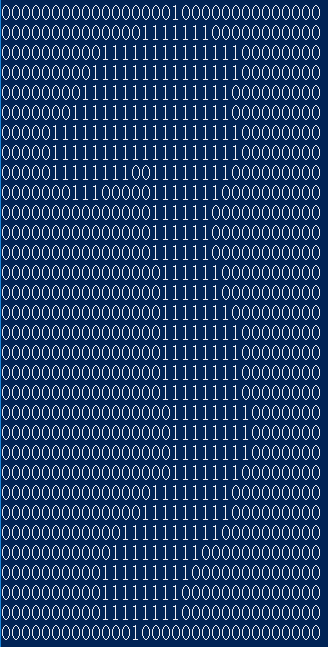
**1.**

****

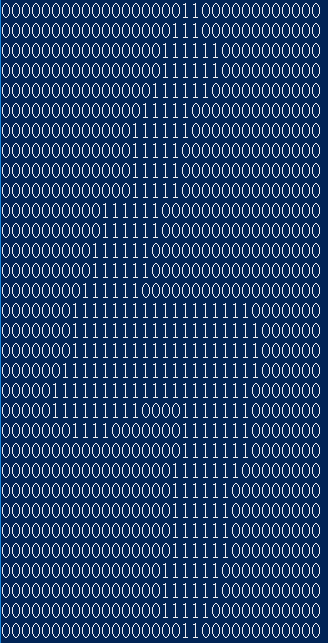
**2.**

****

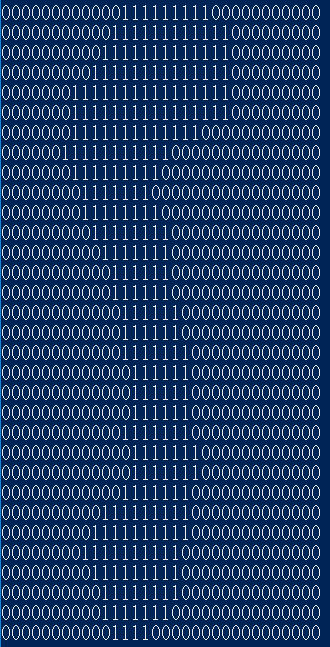
**3.**

****

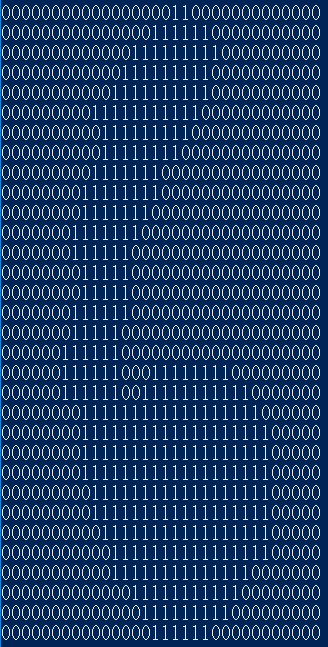
**4.**

****

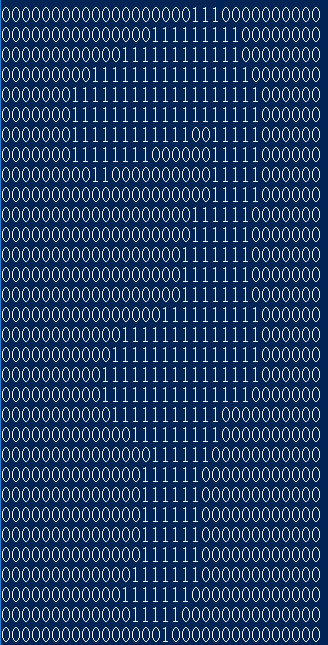
**5.**

****

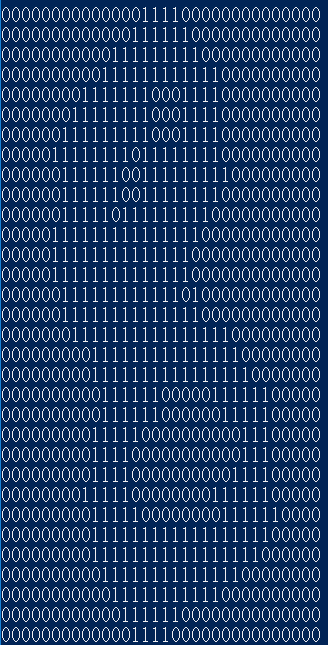
**6.**

****

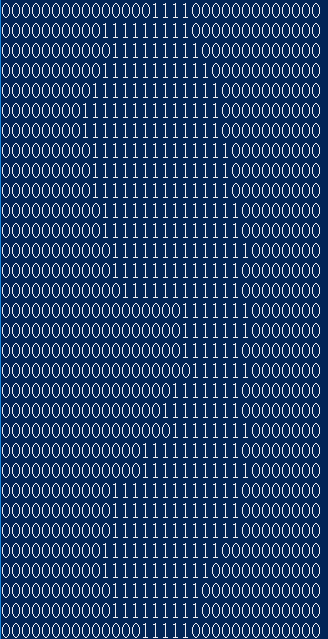
**7.**

****

**8.**

****

**9.**

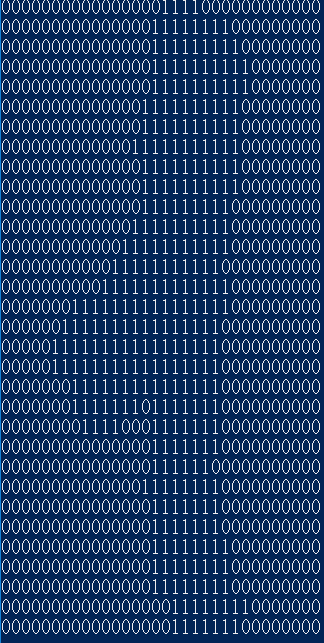
****

**4 pairs of digit types that have the highest confusion rates:**

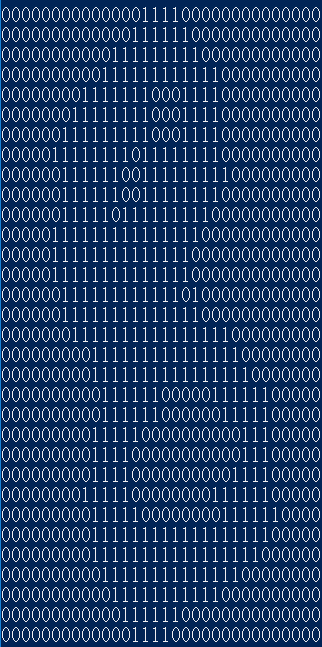
1. **1 and 8**
2. **2 and 8**
3. **3 and 9**
4. **7 and 9**

**(1).1 and 8**

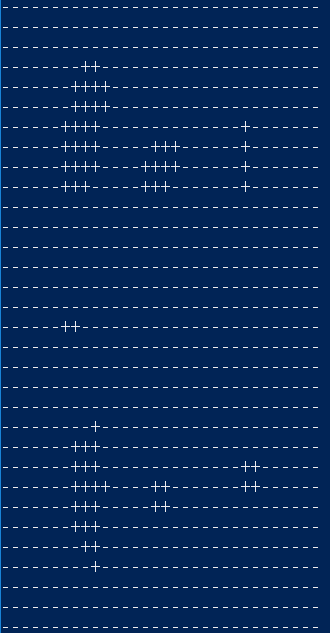
**1.**

****

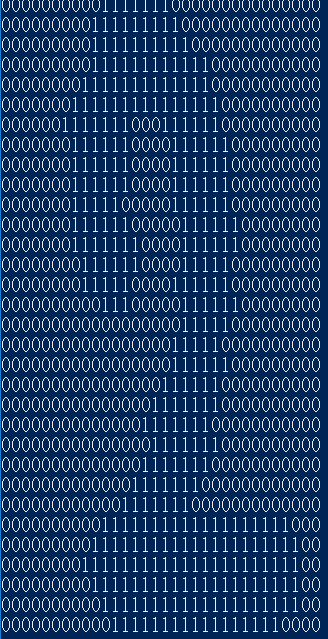
**8.**

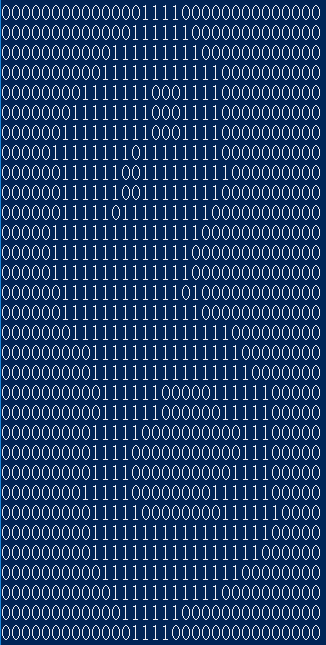
****

**Log odd ratio:**

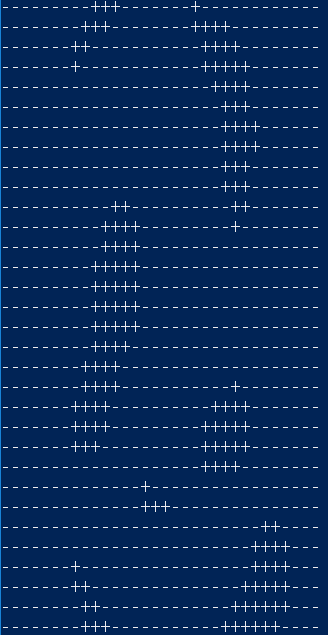
****

**(2).2 and 8**



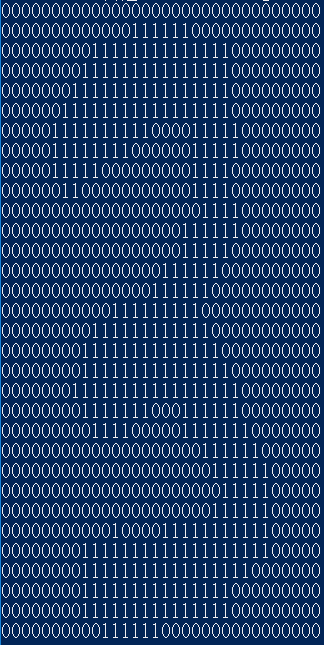


**Log odd ratio:**

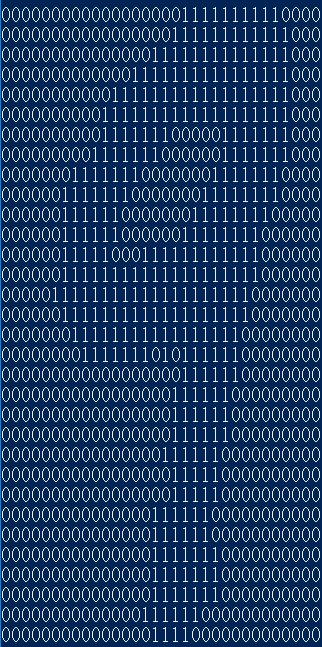


**(3)3 and 9**

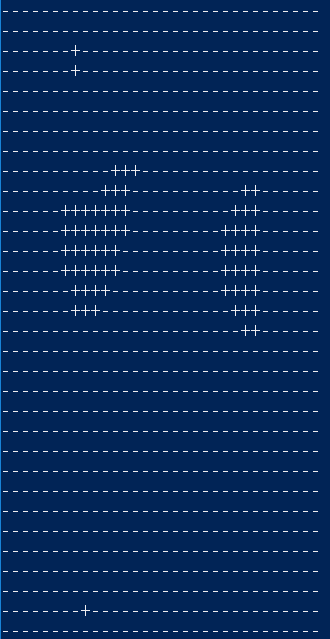
**3.**

****

**9.**

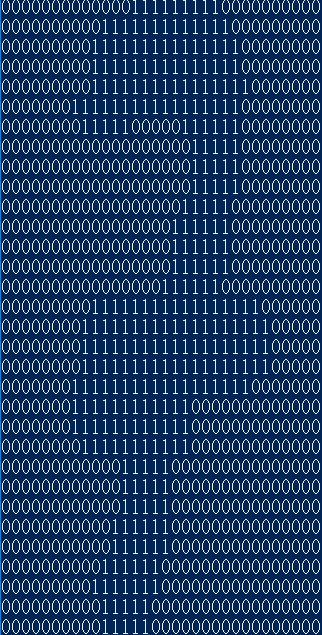
****

**Log odd ratio:**

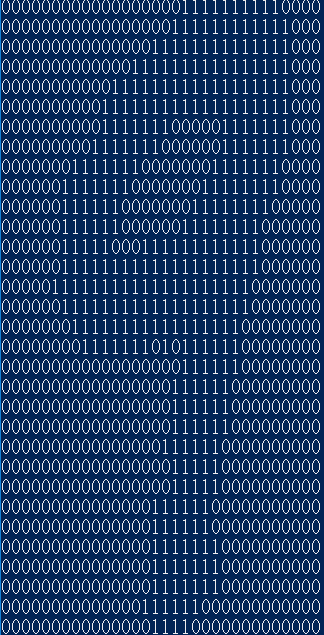
****

**(4).7 and 9**

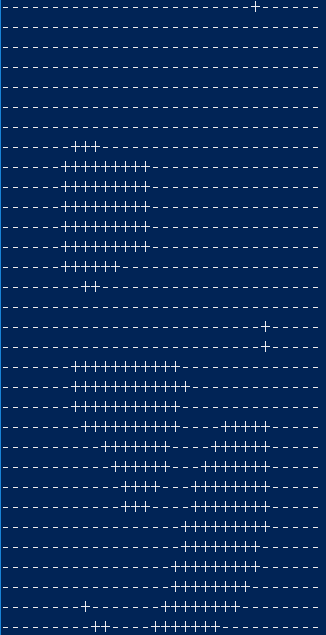
**7.**

****

**9.**

****

**Log odd ratio:**

****

**1.2**

**Implementation:**

For the pixel groups problem, we first store all the data in the format of ArrayList<ArrayList<String>>, and we then group the pixels. We select the smoothing factor as 0.1 and create a 5 dimensional array to store all the feature-value pairs.

For example, an 2\*2 disjoint patch, we will use a 10\*16\*16\*2\*2 array to store all the features(where 10 is corresponding to the ground truth digits), and 16 is the row and column, coming from 32/2(since it is disjoint), while 2\*2 is the distinct pixel level information corresponding to each feature. Note that for the 2\*2 distinct pixel level information, we can have 2^(2\*2) distinct values, which is a large overhead for storage space if the exponent gets larger.

And for the overlap 3\*3 patch, we will use a 10\*30\*30\*3\*3 array to store all the features(where 10 is corresponding to the ground truth digits), and 30 is the row and column, coming from 32-3+1(since it is overlapped), while 3\*3 is the distinct pixel level information corresponding to each feature. Note that for the 3\*3 distinct pixel level information, we can have 2^(3\*3) distinct values, which is a large overhead for storage space if the exponent gets larger.

disjoint patches:

(1)2\*2:

Correct count:409



Accuracy:92.117%

Picture 20

(2)2\*4:(same with 2\*2 coincidentally)

Correct count:409



Accuracy:92.117%

Picture 20

(3)4\*2:

Correct count:411

**Picture 37**

Accuracy:92.57 %

Picture 38

(4)4\*4: (same with 2\*2 and 2\*4 coincidentally)

Correct count:409

圖片 61

Accuracy:92.117%

Picture 20

overlapping patches:

(1)2\*2:

Correct count:410

**Picture 41**

Accuracy:92.34%

Picture 43

(2)2\*4:(same with 2\*2 coincidentally)

Correct count:410

**Picture 41**

Accuracy:92.34%

Picture 43

(3)4\*2: (same with 2\*2 and 2\*4 coincidentally)

Correct count:410

**Picture 41**

Accuracy:92.34%

Picture 43

(4)4\*4: (same with 2\*2 and 2\*4 and 4\*2 coincidentally)

Correct count:410

**Picture 41**

Accuracy:92.34%

Picture 43

(5) 2\*3: (same with 2\*2 and 2\*4 and 4\*2 and 4\*4 coincidentally)

Correct count:410

**Picture 41**

Accuracy:92.34%

Picture 43

(6)3\*2:

Correct count:409

圖片 61

Accuracy:92.117%

Picture 20

(7)3\*3: (same with 3\*2 coincidentally)

Correct count:409

圖片 61

Accuracy:92.117%

Picture 20

Discussion:

The trends we have observed for the different feature sets (including single pixels), in particular, why certain features work better than others for this task.

We found that with overlapping pixel groups, we can generally get better prediction accuracy. And we think that it is because the pixel of the digits are not only of single pixel levels, but a cluster of pixels. The pixel is highly correlated with its neighbors, so taking the nearby pixels into consider can generate better results, hence, the pixel groups generally have better performance than the single pixel.

What’s more, the overlapping patches utilize the windows with a smaller stride(the stride is one) compared to the disjoint one, which means that the overlapping one can generate more features. It also takes more information into consider. With the experiment results, we found that the overlapping patches can yield better results than disjoint ones, corresponding well with the our inference.

Running time comparison:

The running time for single pixel is the fastest, since there are only 1024 features with 2 values, so there are only 2048 different variable-value for each digit. And the second fastest is the disjoint patches, the 2\*2 one is faster than the rest, since it has only 16\*16\*2^(2\*2)=4096(space) different variable-value pair and 16\*16\*2\*2=1024 assignments should be made for each digit. The 4\*4 one is almost the same(slightly slower), since it has 8\*8\*2^(4\*4)=4194304 different variable-value pair and 8\*8\*4\*4=1024 assignments for each digit.

However, the overlapping patches take a longer time. Since there are more features and generally more variable-value pairs. For the 2\*2 one, it has 31\*31\*2^(2\*2)=15376(space) different variable-value pair and 31\*31\*2\*2=3844 assignments for each digit; while the 4\*4 one has 29\*29\*2^(4\*4)=55115776 different variable-value pairs and 29\*29\*4\*4=13456 assignments for each pair, which takes a longer time to execute.

With the calculation above, we found that if the size of the patches grow, then the execute time is going to grow linearly(almost proportional to the feature set size). However, the space needed grows exponentially.

**1-3 Extra credits:**

**Implementation:**

The extra credit problem is similar to the previous one, however, the number of the classes to be classified is reduced from 10 to 2. And the row number is changed from 32 to 70, while the column number is changed from 32 to 60. And with the parameter fine tune, we found that the smoothing factor as 0.07 yields the best result for test set accuracy.

We use the same method as naïve Bayes to calculate the prior probability with the training set first, and then use the result to get the MAP for the prediction of the test set.

We’ve tested with single pixel, and also use the features as overlapping features. Since the size of the face dataset is 70\*60, which is not as easy to be divided as 32\*32 does. And with the results from previous sections, we know that overlapping patches can yield better results than disjoint patches, so we use the various kinds of overlapping patches besides single pixel one. The result is as follows:

**Single pixel:**

Total correct count out of 150 test cases:131



The accuracy:87.33%

Picture 53

**Overlapping patches:**

**(1)2\*2**

Total correct count out of 150 test cases:129

**Picture 55**

The accuracy:86%



**(2)2\*4**(same with 2\*2 coincidentally)

Total correct count out of 150 test cases:129

**Picture 55**

The accuracy:86%

Picture 56

**(3)4\*2**

Total correct count out of 150 test cases:128

**Picture 66**

The accuracy:85.33%

Picture 67

**(4)4\*4**

Total correct count out of 150 test cases:130

**Picture 68**

The accuracy:86.67%

Picture 69

**(5)2\*3**(same with 4\*4 coincidentally)

Total correct count out of 150 test cases:130

**Picture 68**

The accuracy:86.67%

Picture 69

**(6)3\*2**(same with 4\*2 coincidentally)

Total correct count out of 150 test cases:128

**Picture 66**

The accuracy:85.33%

Picture 67

**(7)3\*3**(same with 4\*2 and 3\*2 coincidentally)

Total correct count out of 150 test cases:128

**Picture 66**

The accuracy:85.33%

Picture 67

## Part 2: Digit Classification using Discriminative Machine Learning Methods

**2.1**

**Implementation:**

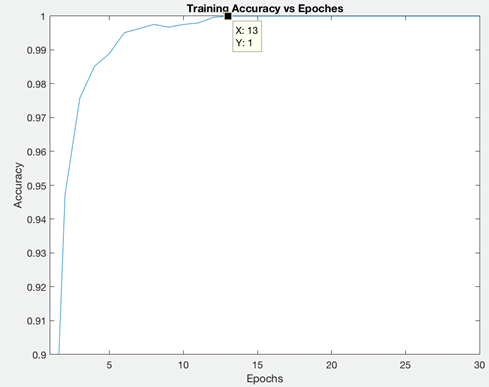
There are 10 classes(0~9), so we create 10 perceptrons. Each of them includes 1024(32x32) weights for each pixels in the picture. We read each images as 1D string array consisting of 32x32 characters and create a integer array to save the corresponding label of each images.

In the training phase, we take the data of one image and make dot product with the weights of 10 perceptrons to get 10 outputs. The owner of maximum outputs is the prediction class of this image. If it misclassifies “ c ” as “ c’ ”, we increase weights of “ c “ and decrease weights of “ c’ “. If the prediction is the same as the label, the training accuracy increases. After running through all the input, we do the above things again for next epoch. Stop training until finishing all the epochs.

In the testing phase, we do the same thing as training phase did but do not update weights. Meanwhile, calculate the testing accuracy and make the confusion matrix. Finally, print the confusion matrix and write the training accuracy of each epochs to text file for plotting in Matlab.

**Training set accuracy vs epochs:**

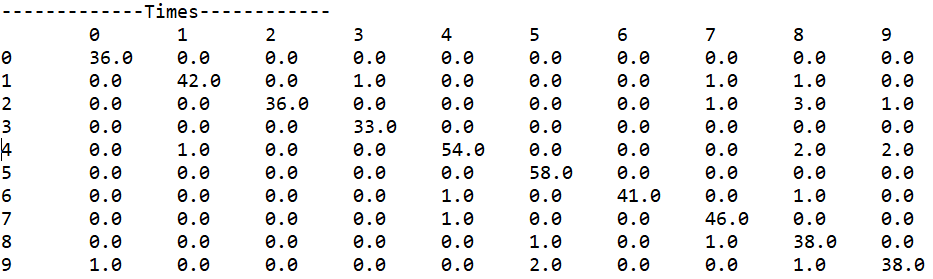
**The accuracy is about to converge at the 13th epochs.**



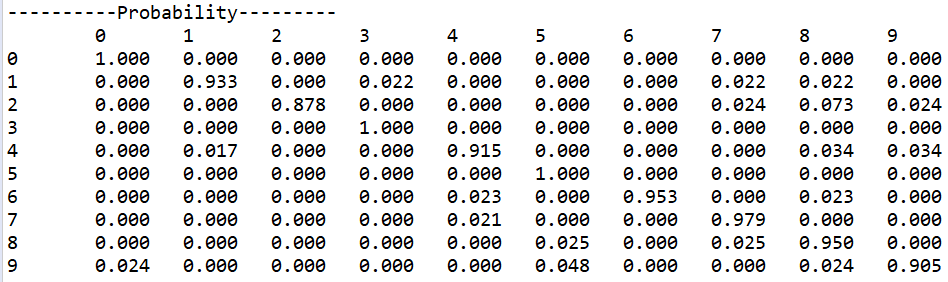
**Testing set accuracy: 95.04%**



**Confusion matrix total count:**



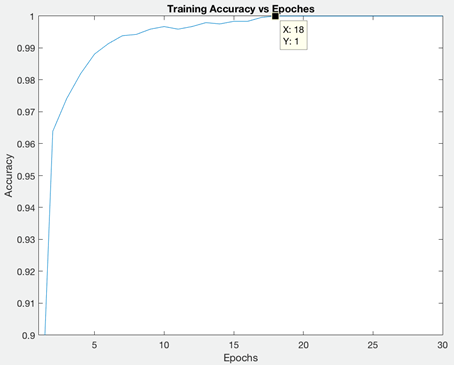
**Confusion matrix:**



**Parameters tuning:**

**-** Learning rate decay function   
We set the decay function for to decrease the learning rate while the training times increase. The training accuracy with decay function converges faster than training accuracy without decay function. It’s reasonable that after a period of training, the weights should be changed more slightly.

With decay function :

Without decay function :   
- Bias v.s. No bias  
I set input bias as 1 and weight bias as 1 and run the algorithm again. There is no change after adding bias to the original algorithm, because it affects less when comparing to the output value.

- Initialization of weights  
I set weights randomly from 0 to 0.1 at the beginning and the result is the same as setting those to 0. The accuracy is bad if the weights are set from 0 to 1 because the learning rate is too small to recover some wrong weights.  
  
- Ordering of training examples  
With random ordering of training examples in every epoch, the training accuracy sometimes converge faster than the fixed order one.

- Number of epochs  
Actually, the training accuracy converges from 10 to 20 epochs according to other parameters. It converges to 1 so tuning other parameters only accelerates the speed of converging instead of making the accuracy better.

**2.2**

**Implementation:**

The distance/similarity function we choose is the sum of the square distance between each feature points(a total of 1024) for each data point, if the pixels are different, then the distance is increased by one;if they are the same, then we make no change to the distance. We take the smallest k(k from1~25) distance between the test case and the data points in the training set into consider, and find the digit they represent, and add count to the corresponding digit. We then classify the test case into the digit with the highest count. This gives great result.

|  |  |  |
| --- | --- | --- |
| K | Correct count | Accuracy |
| 1 | 442 | 0.9954954954954955 |
| 2 | 442 | 0.9954954954954955 |
| 3 | 442 | 0.9954954954954955 |
| 4 | 442 | 0.9954954954954955 |
| 5 | 442 | 0.9954954954954955 |
| 6 | 442 | 0.9954954954954955 |
| 7 | 442 | 0.9954954954954955 |
| 8 | 442 | 0.9954954954954955 |
| 9 | 442 | 0.9954954954954955 |
| 10 | 442 | 0.9954954954954955 |
| 11 | 442 | 0.9954954954954955 |
| 12 | 442 | 0.9954954954954955 |
| 13 | 442 | 0.9954954954954955 |
| 14 | 442 | 0.9954954954954955 |
| 15 | 442 | 0.9954954954954955 |
| 16 | 442 | 0.9954954954954955 |
| 17 | 442 | 0.9954954954954955 |
| 18 | 442 | 0.9954954954954955 |
| 19 | 442 | 0.9954954954954955 |
| 20 | 442 | 0.9954954954954955 |
| 21 | 442 | 0.9954954954954955 |
| 22 | 442 | 0.9954954954954955 |
| 23 | 442 | 0.9954954954954955 |
| 24 | 442 | 0.9954954954954955 |
| 25 | 442 | 0.9954954954954955 |

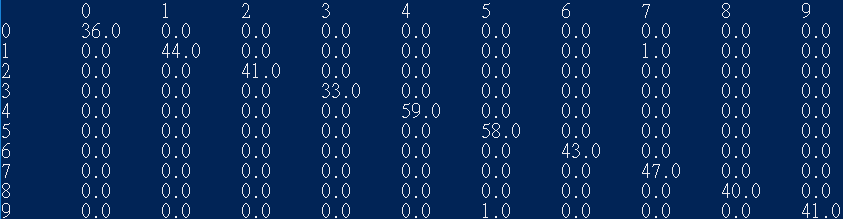
The classification accuracy for k=1~25 are all 99.55%, which is much higher than the naïve bayes accuracy:92.12% and perceptrons:96.17%. The difference/similarity function we use is quite accurate, so the result for the KNN classification is high for k=1~25.

As a baseline, the running time for a single query (classify a single instance in the test dataset) by using brute force is 291ms. We can optimize the performance by removing the feature that is at the edges, which is very unlikely to be 1.

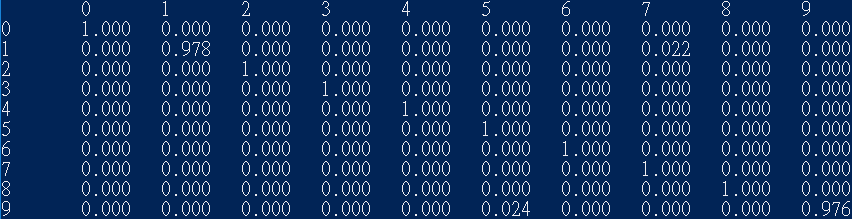


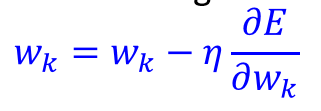
Confusion matrix for k=7:

Correct count:



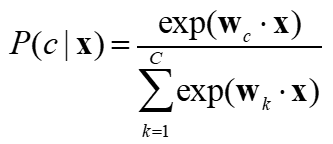
Accuracy:



**2.extra\_b**

**Implementation:**

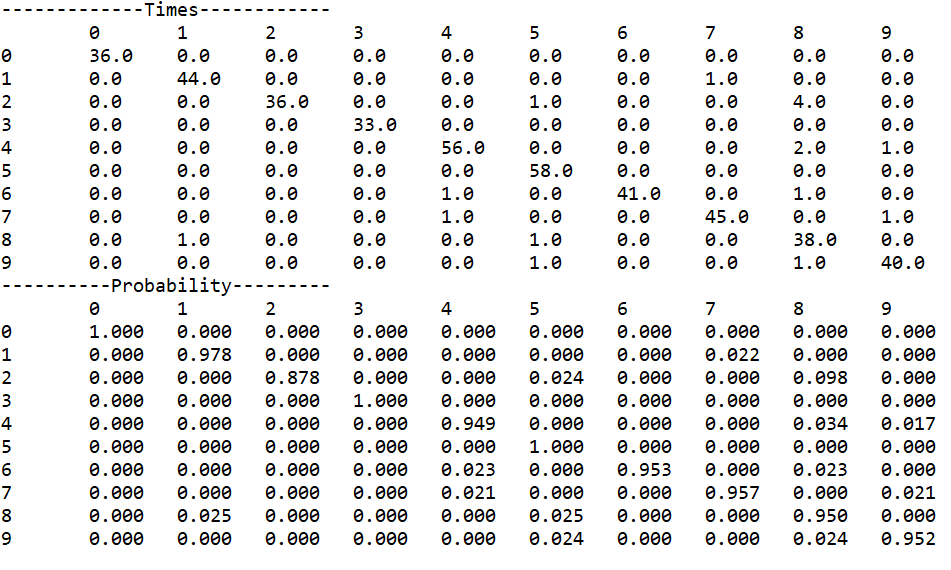
From 2.1, we use the same way to decide which classes the pictures are. However, we change the error function to differential function. To do so, we should change the argmax decision to softmax decision so that we can differentiate E which represents .

In the training phase, if it misclassifies “ c ” as “ c’ ”, we upgrade all the weights by the error function. On contrast, if misclassifies the prediction is the same as the label, the training accuracy increases.

**Testing set accuracy: 96.17%**



**Confusion matrix total count and Confusion matrix:**



Comparing to the non-differential one, we can find that the accuracy has increasing a little. It is reasonable because we update all weights instead of two weights by differentiate the error. Also, the running time is only a bit longer than the non-differential one.

**2.extra\_c**

**Implementation:**

We focus on face detection and we use convolutional neural network to classify cats and dogs faces with Keras. We use a 3\*3 kernel for the 2D convolution and the activation function is relu, concatenated with an 2\*2 max pooling. We then flatten the data, and use a fully connection hidden layer to construct the model, there is a single output(cat or dog). Since it is a classification problem, so binary\_entropy is used for the loss function, and we use adam as the optimizer. We also rescale the data during the preprocessing process to make the prediction more efficient. The accuracy of the classification of cat and dog is 84.6%.

