Class Name: CS5785 Applied Machine Learning Homework 1

Team member:

Zexi Liu

(zl558@cornell.edu)

Howard Chen

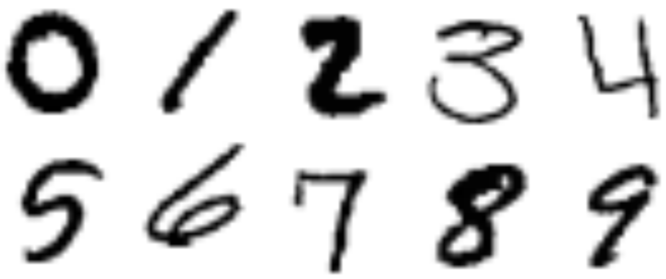
(hc839@cornell.edu)

K-Nearest Neighbor

(a) Join the Digit Recognizer competition on Kaggle. Download the training and test data. The competition page describes how these files are formatted.

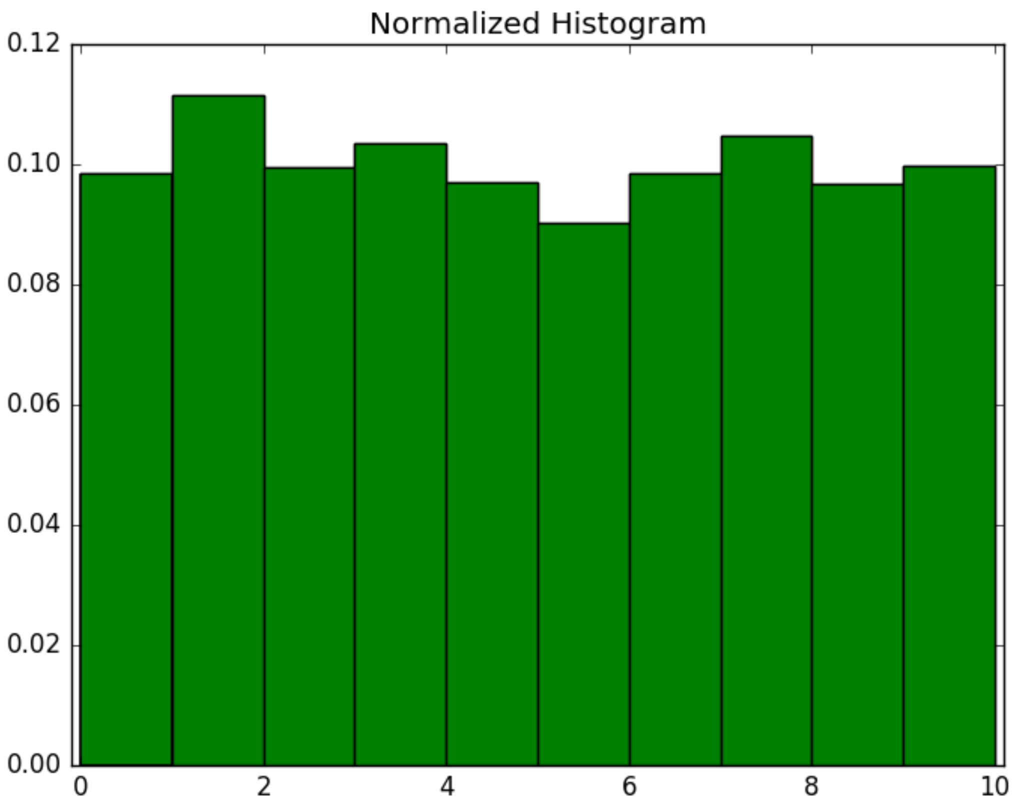
In the preprocessing of data, I took out the label column and created an extra list to store different digits separately.

(b) Write a function to display anMNIST digit. Display one of each digit.



I reshape the 1*784 array into a 28*28 array and then print them out.

(c) Examine the prior probability of the classes in the training data. Is it uniform across the digits? Display a normalized histogram of digit counts. Is it even?



’0’: 0.098, ’1’: 0.011, ’2’: 0.099, ’3’: 0.104, ’4’: 0.097

’5’: 0.090, ’6’: 0.099, ’7’: 0.105, ’8’: 0.097, ’9’: 0.100

It is hard to say the digits are uniform distributed. However, in general, we can say all the probability of digits are around 0.1 and the difference is trivial.

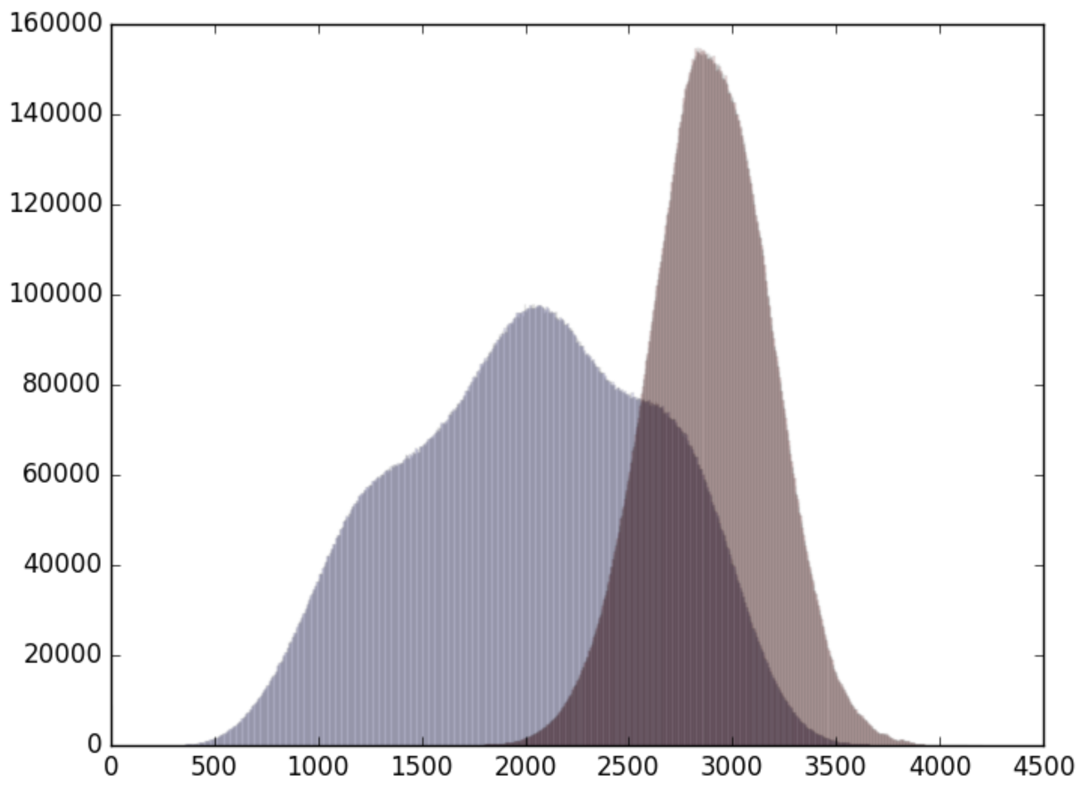
(d) Pick one example of each digit from your training data. Then, for each sample digit, compute and show the best match (nearest neighbor) between your chosen sample and the rest of the training data. Use L2 distance between the two images’ pixel values as the metric. This probably won’t be perfect, so add an asterisk next to the erroneous examples.

Sample is 0, result is 0.	Sample is 1, result is 1.
Sample is 2, result is 2.	Sample is 3, result is 3.
Sample is 4, result is 4.	Sample is 5, result is 5.
Sample is 6, result is 6.	Sample is 7, result is 7.

Sample is 8, result is 8. Sample is 9, result is 9.

This is actually KNN method when K=1. The result is surprisingly good. All my samples are correctly recognized.

(e) Consider the case of binary comparison between the digits 0 and 1. Ignoring all the other digits, compute the pairwise distances for all genuine matches and all impostor matches, again using the L2 norm. Plot histograms of the genuine and impostor distances on the same set of axes.

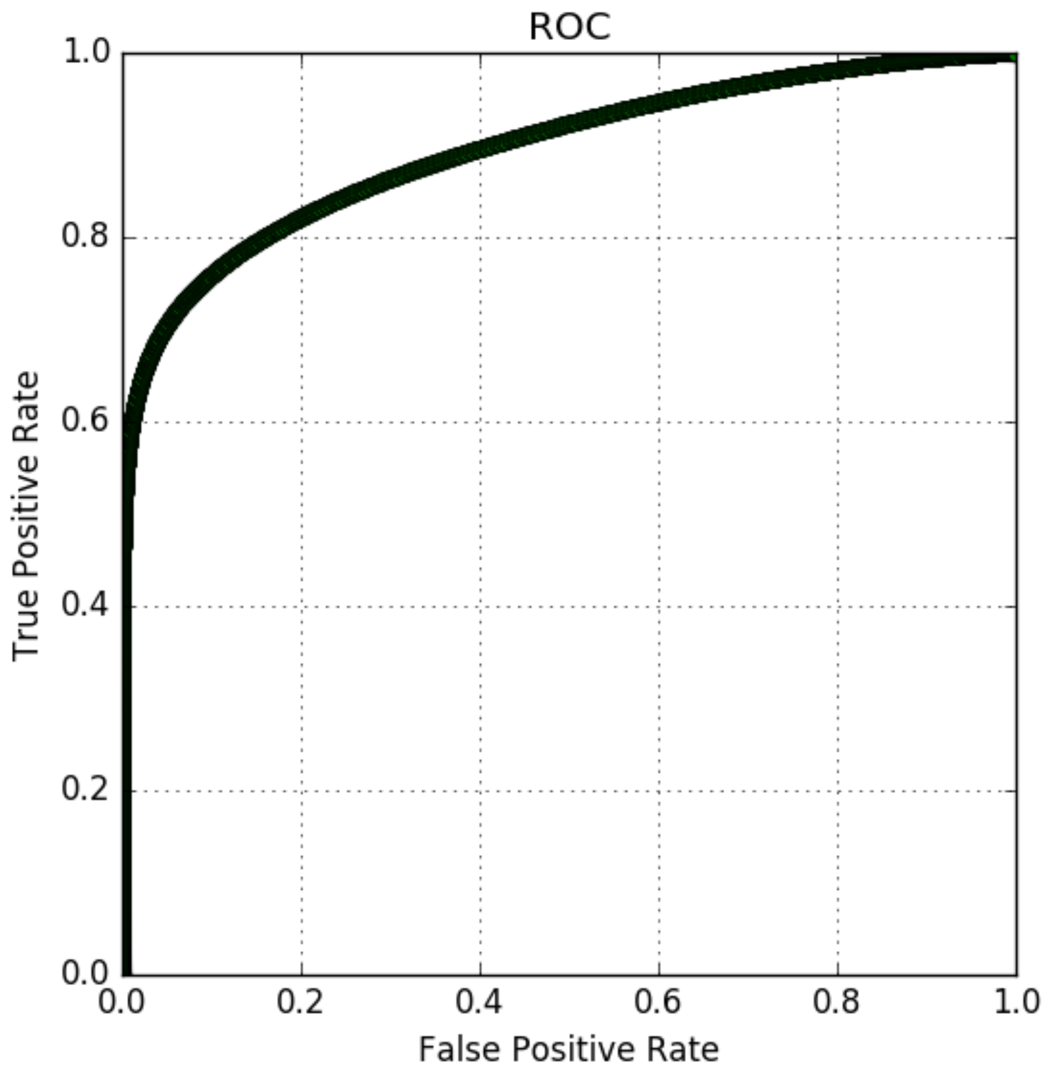


Genuine Matches : Within 0 and within 1.

Impostor Matches : Sample 0 recognized to be 1 and sample 1 recognized to be 0.

I don't plot the normalized histogram because I want to capture the number of different matches. It seems there is a big overlap between genuine and impostor matches.

(f) Generate an ROC curve from the above sets of distances. What is the equal error rate? What is the error rate of a classifier that simply guesses randomly?



EER is: 0.18557241682

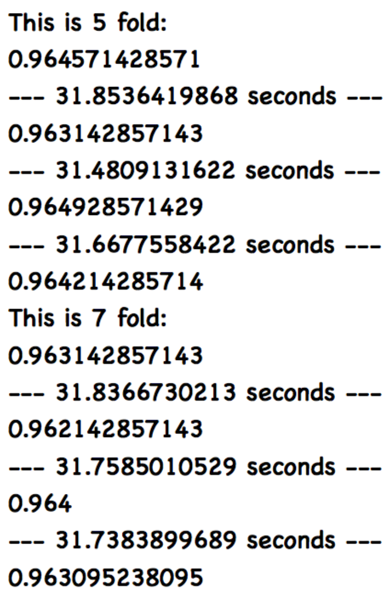
Threshold is 2633

To find EER, we have to find the point where False Positive Rate (FPR) equals to False Negative Rate (FNR). Note that FNR = 1 - TPR, so FPR = FNR = 1 - TPR. The point of interest is actually the point where ROC line and FPR + TPR = 1 line intersect.

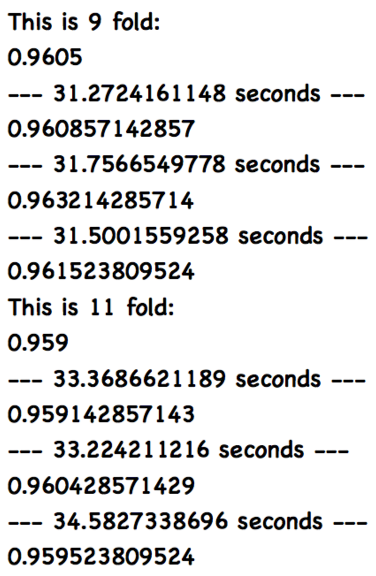
(g) Implement a K-NN classifier.

At first, I write my L2 norm function to calculate the distance matrix, however, it is so slow that the program barely get the complete result on the whole dataset. Later, I used the sklearn.metrics.pairwise euclidean_distances, which tremendously shorten the runtime of my program. After I read the document on sklearn, I know that they use a special data structure called KD Tree, which calculate the distance matrix in O(logn) time. Amazing!

(h) Using the training data for all digits, perform 3 fold cross-validation on your K-NN classifier and report your average accuracy.



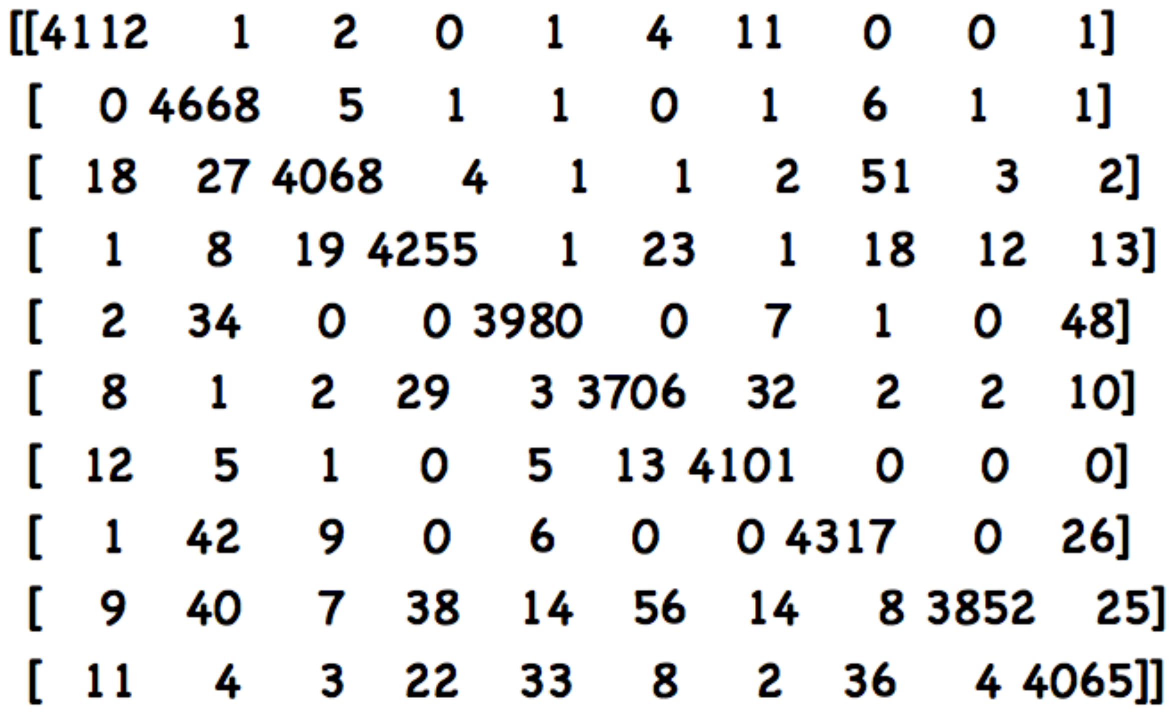
```
This is 5 fold:
0.964571428571
--- 31.8536419868 seconds ---
0.963142857143
--- 31.4809131622 seconds ---
0.964928571429
--- 31.6677558422 seconds ---
0.964214285714
This is 7 fold:
0.963142857143
--- 31.8366730213 seconds ---
0.962142857143
--- 31.7585010529 seconds ---
0.964
--- 31.7383899689 seconds ---
0.963095238095
```



```
This is 9 fold:
0.9605
--- 31.2724161148 seconds ---
0.960857142857
--- 31.7566549778 seconds ---
0.963214285714
--- 31.5001559258 seconds ---
0.961523809524
This is 11 fold:
0.959
--- 33.3686621189 seconds ---
0.959142857143
--- 33.224211216 seconds ---
0.960428571429
--- 34.5827338696 seconds ---
0.959523809524
```

I used K=[5,7,9,11] to do the 3 fold cross validation. As you can see, when K=5, the average accuracy is the best. So I choose K=5 as my model.

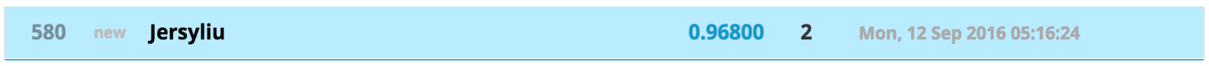
(i) Generate a confusion matrix (of size 10 £ 10) from your results. Which digits are particularly tricky to classify?



```
[[4112    1    2    0    1    4   11    0    0    1]
 [   0 4668    5    1    1    0    1    6    1    1]
 [  18   27 4068    4    1    1    2   51    3    2]
 [   1    8   19 4255    1   23    1   18   12   13]
 [   2   34    0    0 3980    0    7    1    0   48]
 [   8    1    2   29    3 3706   32    2    2   10]
 [  12    5    1    0    5   13 4101    0    0    0]
 [   1   42    9    0    6    0    0 4317    0   26]
 [   9   40    7   38   14   56   14    8 3852   25]
 [  11    4    3   22   33    8    2   36    4 4065]]
```

This is a 10*10 matrix. We can see digit 8 is the most tricky one to classify. Following digit 8 is digit 4.

(j) Train your classifier with all of the training data, and test your classifier with the test data. Submit your results to Kaggle.



580 new Jersyliu 0.96800 2 Mon, 12 Sep 2016 05:16:24

With the help from sklearn, running the KNN method on the whole dataset is a no more than 100 seconds thing. So basically everyone can get a 96.8% or above correct rate.