Assignment 3

Johnnie Oldfield

October 2019

1 Problem Definition

The purpose of this assignment is to implement a memory-based deterministic classifier using nearest neighbor. While you can use a k-nearest neighbor(KNN) approach to better predict the class, my test only cares about the closest (1-NN). The goal is to create a classifier to identify handwritten digits similar to that of the United States Postal Service. I will be using the Modified National Institute of Standards and Technology(MNIST) data set of handwritten numbers.

2 Decisions

I start off by initializing 5 empty arrays. x to help plot the number of training samples. ey and cy are used to plot the accuracy's for their respective distance function used. ey for euclidean and cy for cosine. Similarly I have edistances and cdistances to keep track of distances. I use tsize to define the fixed number of test samples for my tests.

I decided not to shuffle the data set for this assignment. I was more interested in the how adjusting the fixed test samples would affect the nearest neighbor classification and digit confusion.

Listing 1: Getting Distances

```python
for i in range(1, 5000, 150):
  ec, cc = 0, 0
  for tstx in range(tsize):
    edistances, cdistances = [], []
    for trnx in range(i):
      edistances.append( euclidean_distance(
        trainX[trnx, :], testX[tstx, :] ) )
      cdistances.append( cosine_distance(
        trainX[trnx,:], testX[tstx,:] ) )
```

Listing 1 shows the beginning of the nested loop I used to populate those arrays. Starting from 1 to 5000 testing sample by intervals of 150. I did this not only because going by every 1 test sample would take a substantial amount of time but also it is not important to consider for the sake of the test. The next to nested loop go through the test indexes(trnx) in range of the fixed sample size(tsize) and the training indexes(trnx) in range of the number of test sample(i). I use those indexes to calculate all of the euclidean and cosine distances and append them to the arrays.

Listing 2: Distance functions

```python
def euclidean_distance(x, y):
  return np.sqrt(np.sum(np.power(x-y, 2)))

def cosine_distance(x, y):
  return np.dot(x,y) / (np.sqrt(np.dot(x,x))
    * np.sqrt(np.dot(y,y)))
```

Listing 2 shows the functions that I use to get the Euclidean and cosine distances. Euclidean distance for n dimensions can be found by the following equation [1]:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Listing 3: Storing Accuracy's and Confusion matrix

```
    if(trainy[0, np.argmin(edistances)] == testy[0, tstx]):
          ec += 1
        else:
          conmat[trainy[0, np.argmin(edistances)],
            testy[0, tstx]] += 1

    if(trainy[0, np.argmax(cdistances)] == testy[0, tstx]):
         cc += 1
```

The second part of the nested for loop contains some conditional statements. The purpose of them is to increment my counter variables to help plot the distance functions accuracy's and the confusion matrix. The counters are divided by the tsize and then appended to their respective arrays. The min and max functions grab the indexes in their respective distance arrays to classify the digits.

3 Evaluation

For the assignment I ran four different tests. Each time changing the fixed testing samples to the outcome of the accuracy's and the confusion of digits. I ran a test for 50, 200, 500, and 1000 samples.

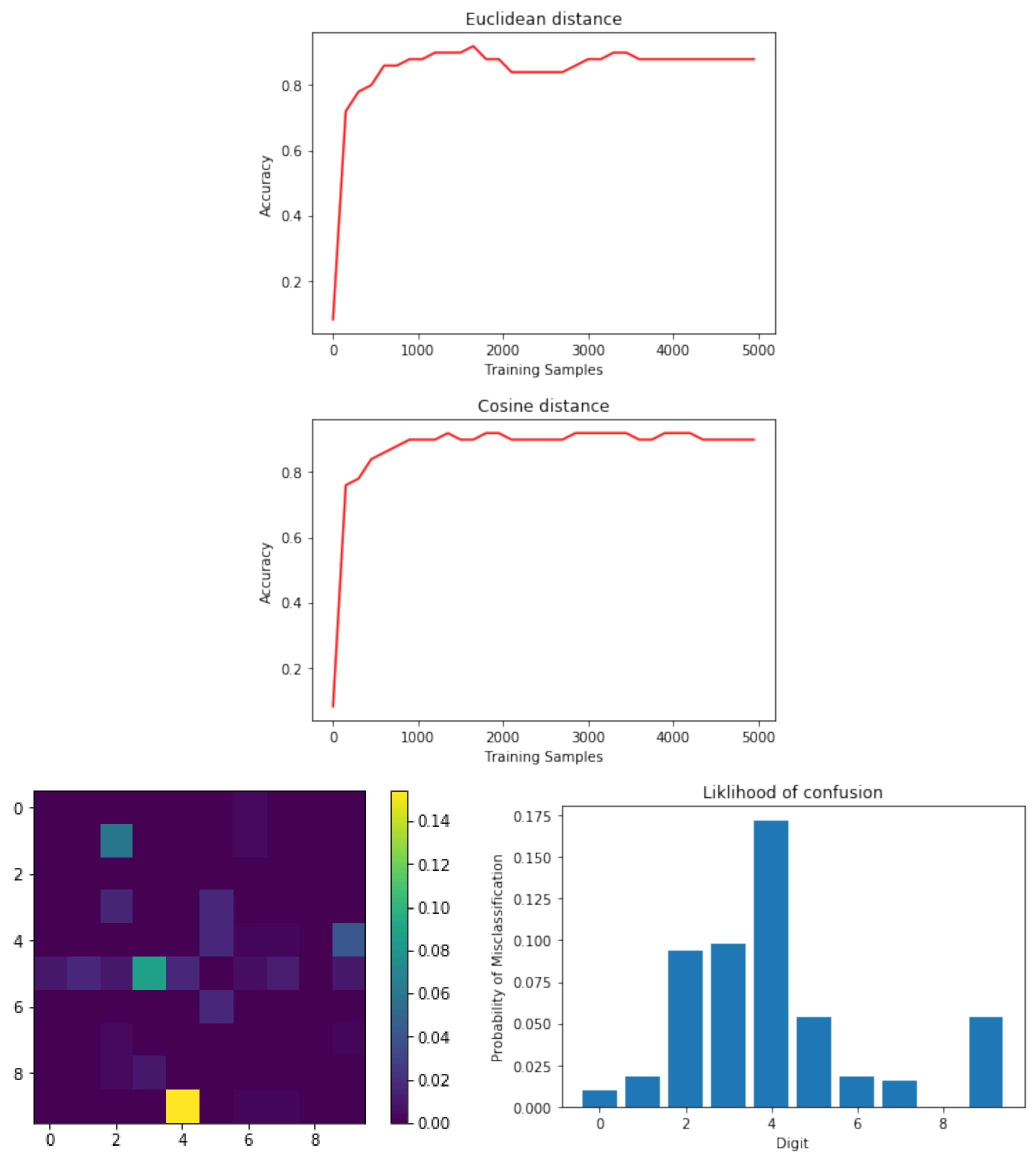

Figure 1: Results of setting fixed testing samples to 50.

Figure 1 shows the resulting plots and confusion matrix for a test with tsize set to 50. It plateaus around 1500 before dropping and plateauing again. Since there is so few samples the confusion of digits is rather low. Surprising the digit 8 did not get confused in that run.

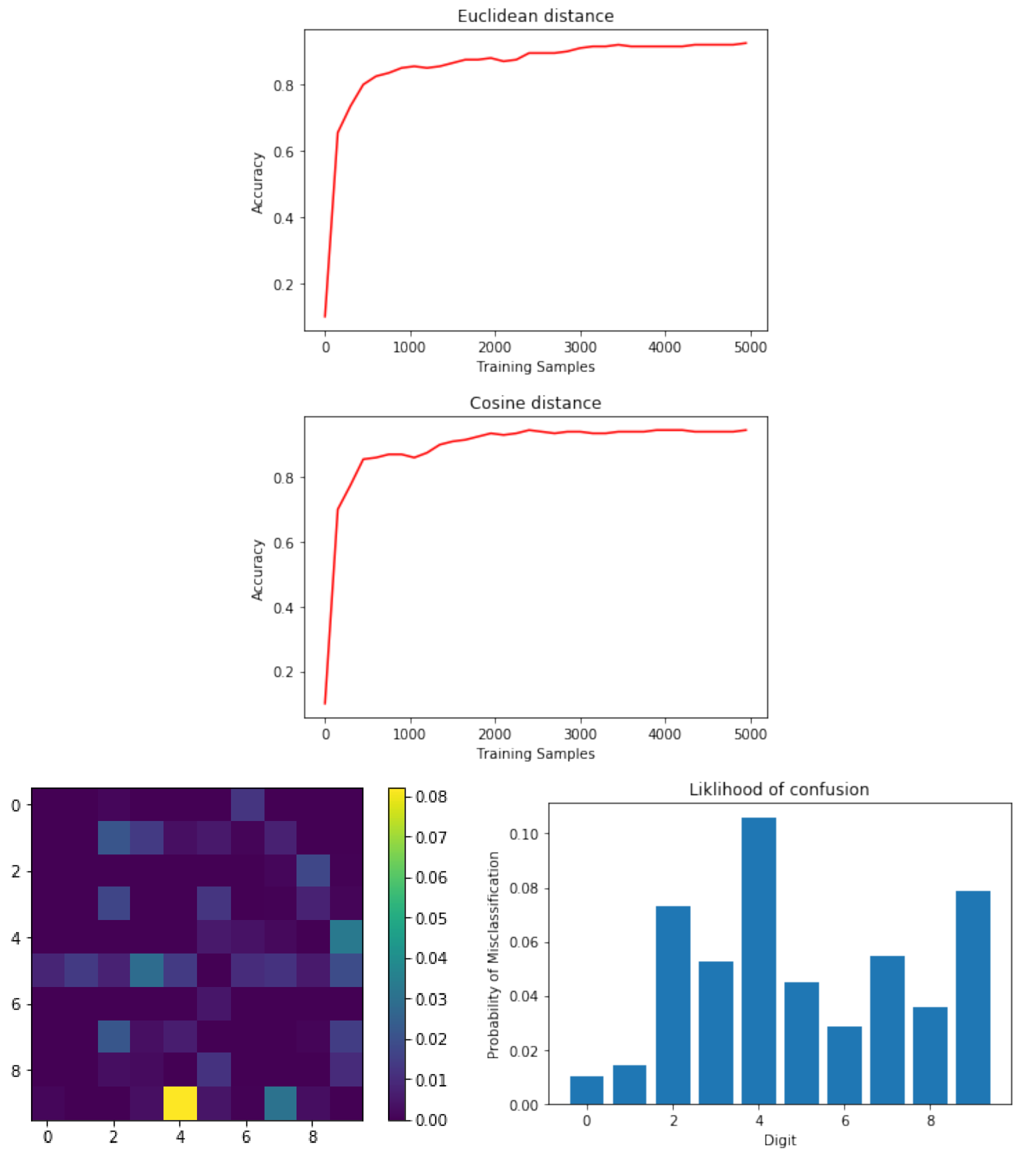

Figure 2: Results of setting fixed testing samples to 200.

With 150 more samples for the test the likelihood of confusion increases. Digit 8 is now being mistaken other digits but digits 0 and 1 are still relatively low. The Euclidean accuracy no longer dips after 2000 training samples and both get more accurate.

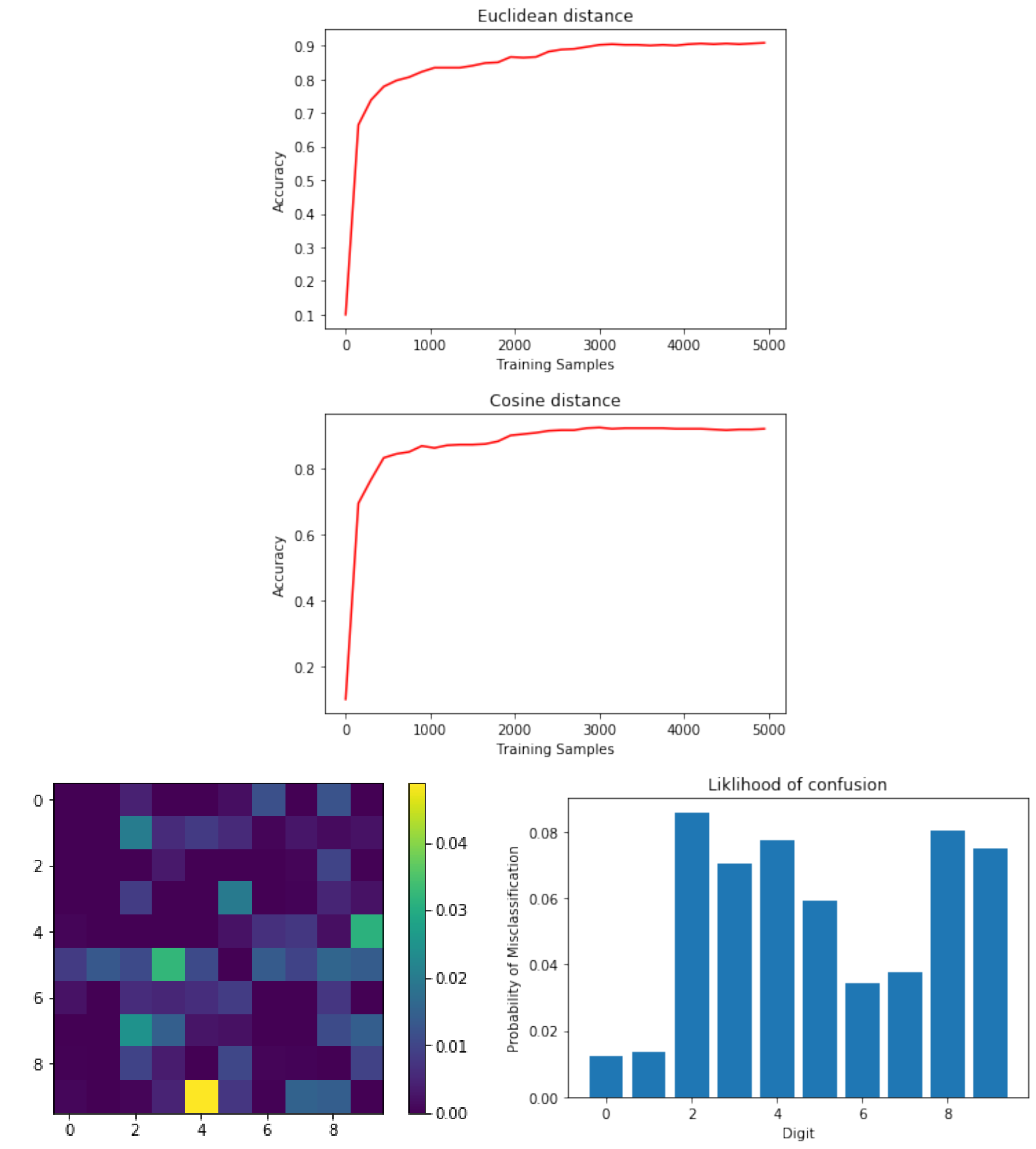

Figure 3: Results of setting fixed testing samples to 500.

At 500 test samples the accuracy seems to have reached a peak. Both distances keep a very steady accuracy after they plateau. Most digits have a much higher likelihood of confusion at this point. 0 and 1 still have low misclassificaiton.

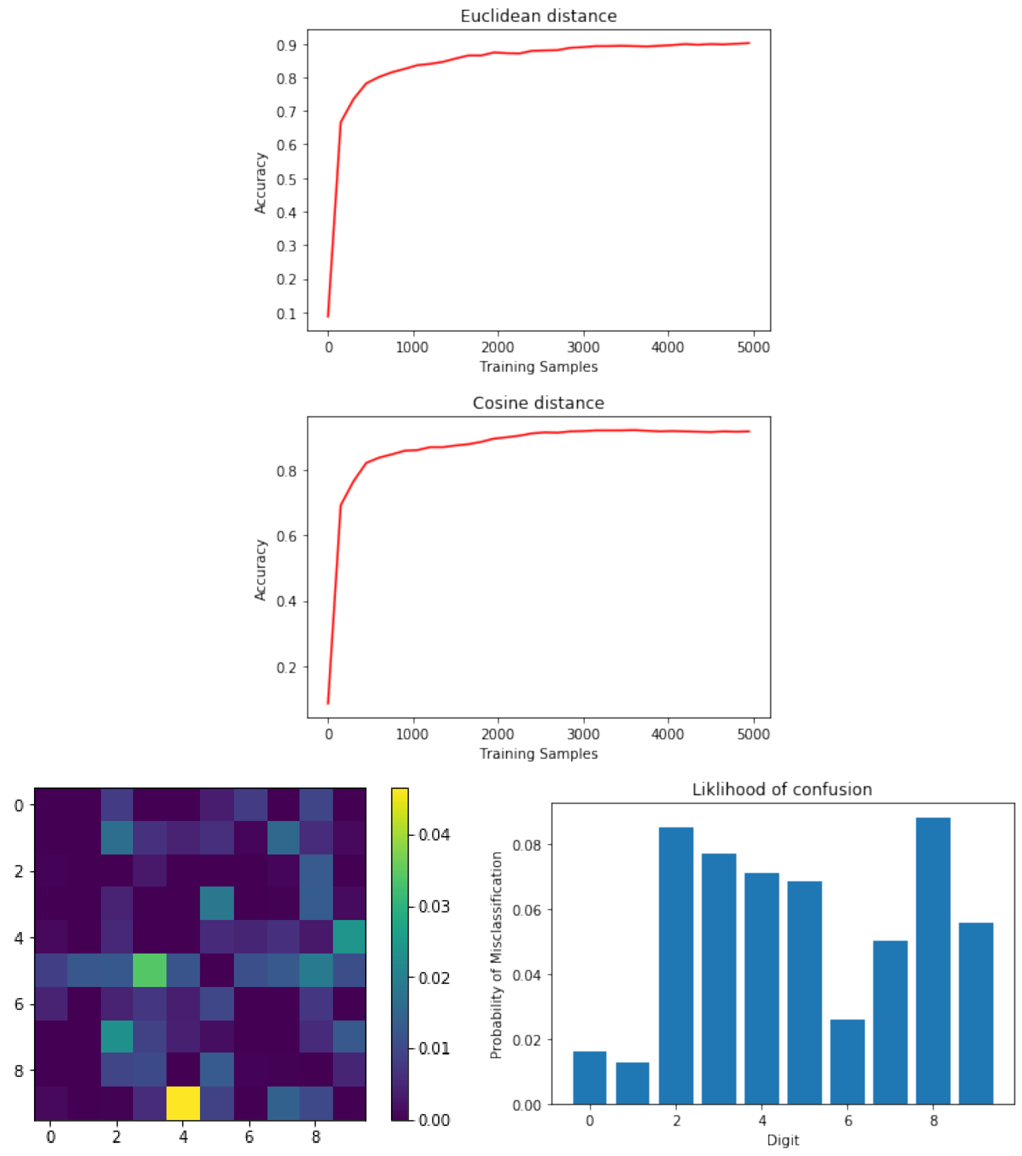

Figure 4: Results of setting fixed testing samples to 1000.

Not much changes from 500 to 1000 test samples. Accuracy's are about the same and confusion has only slightly increased.

As expected the more training samples there are the better the accuracy will be for the distance formulas. Also, the more digits there are there more likely they are to get confused.

4 What I Learned

I learned how to make a 1-Nearest Neighbour classifier. Although it takes a while and can be more efficient it does what it needs to for the tests. Also I now have a better understanding on the multidimensional spaces and different distances points in that those spaces can have (e.g Euclidean and Cosine)

Like last assignment I thought making the confusion matrix would be harder than it was. This time it is a 10x10 instead of a 2x2 and filling it was straight forward. [2]

It was interesting to see how the distance functions can peak at round 500 samples. Adding more did not affect the functions much at all but it did increase the digit confusion. It was also surprising to see digits 0 and 1 with so little confusion.

5 Future Work

My current classifier if very slow and inefficient, I would like to improve that. The 1-NN classifier can be improved or changed to handle k-NN neighbors. I spend too much time trying to get 1-NN to work and was not able to get k-NN working for this assignment.

References

[1] `https://en.wikipedia.org/wiki/Euclidean_distance`.

[2] `https://en.wikipedia.org/wiki/Confusion_matrix`.