# K-Nearest Neighbors

We have tried 2 distance functions, Euclidean and Manhattan. For each of these distance functions we used 2 versions of labeling procedure. One selects the label of test data from plurality of the neighbors, another uses a weighted distance methodology, where number of neighbors with same label were divided by total distance of them from query point and then the label with maximum average was taken. Following is a comparison table:

| K | Distance Function | Accuracy (percent) | Is Weighted? | Approx. Execution Time (in sec) |
|---|---|---|---|---|
| 1 | Euclid | 67.2322375398 | No | 1900.777 |
| 1 | Euclid | 67.2322375398 | Yes | 1906.822 |
| 1 | Manhattan | 68.0805938494 | No | 1900.170 |
| 1 | Manhattan | 68.0805938494 | Yes | 1906.432 |
| 2 | Euclid | 65.2173913043 | No | 1900.412 |
| 2 | Euclid | 67.2322375398 | Yes | 1906.827 |
| 2 | Manhattan | 65.3234358431 | No | 1900.027 |
| 2 | Manhattan | 68.0805938494 | Yes | 1906.867 |
| 3 | Euclid | 68.7168610817 | No | 1900.827 |
| 3 | Euclid | 65.1113467656 | Yes | 1906.877 |
| 3 | Manhattan | 69.0349946978 | No | 1900.821 |
| 3 | Manhattan | 67.1261930011 | Yes | 1906.980 |
| 4 | Euclid | 69.5652173913 | No | 1900.210 |
| 4 | Euclid | 65.4294803818 | Yes | 1906.750 |
| 4 | Manhattan | 68.2926829268 | No | 1900.098 |
| 4 | Manhattan | 66.1717921527 | Yes | 1906.123 |
| 5 | Euclid | 69.1410392365 | No | 1900.728 |
| 5 | Euclid | 64.1569459173 | Yes | 1906.827 |
| 5 | Manhattan | 69.67126193 | No | 1900.086 |
| 5 | Manhattan | 64.262990456 | Yes | 1906.827 |
| 6 | Euclid | 69.5652173913 | No | 1900.223 |
| 6 | Euclid | 63.0965005302 | Yes | 1906.827 |
| 6 | Manhattan | 69.0349946978 | No | 1900.224 |
| 6 | Manhattan | 62.6723223754 | Yes | 1906.765 |
| 7 | Euclid | 69.5652173913 | No | 1900.827 |
| 7 | Euclid | 61.2937433722 | Yes | 1906.765 |
| 7 | Manhattan | 69.7773064687 | No | 1900.827 |
| 7 | Manhattan | 62.3541887593 | Yes | 1906.765 |
| 8 | Euclid | 69.4591728526 | No | 1900.827 |
| 8 | Euclid | 60.3393425239 | Yes | 1906.765 |
| 8 | Manhattan | 67.868504772 | No | 1900.827 |
| 8 | Manhattan | 60.65747614 | Yes | 1906. 765 |
| **9** | **Euclid** | **70.7317073171** | **No** | **1900.600** |
| 9 | Euclid | 59.8091198303 | Yes | 1906.832 |
| **9** | **Manhattan** | **70.5196182397** | **No** | **1900.600** |
| 9 | Manhattan | 60.3393425239 | Yes | 1906.827 |
| 10 | Euclid | 70.3075291622 | No | 1900.240 |
| 10 | Euclid | 59.1728525981 | Yes | 1906.998 |
| 10 | Manhattan | 70.7317073171 | No | 1900.190 |
| 10 | Manhattan | 58.854718982 | Yes | 1906.981 |

So it gives best result (70.7%) when k=9, for both Euclidean and Manhattan distance functions. For every value KNN takes roughly the same amount of time (~ 31 min or ~ 1900 seconds), so the graph can be imagined as a straight line.

# Neural Network

We have used input bias, input weights, different values of learning rate (alpha) in back-propagation, different values for multiple iterations (epoch) in training. We have used simple Tanh function as our activation function because it tends to perform (and more like to help system converge) better than other alternatives like sigmoid and bipolar-sigmoid. Following is a comparison table:

| **Hidden Nodes Count** | **Alpha (Learning Rate in Back-Propagation)** | **Epoch (Number of iterations in training)** | **Accuracy (percent)** | **Approximate (Train + Test) Execution Time (in sec)** |
|---|---|---|---|---|
| 11 | 0.1 | 10 | 32.9798515376 | 600.876 |
| 11 | 0.1 | 20 | 18.0275715801 | 602.009 |
| 11 | 3.0 | 10 | 27.5715800636 | 602.021 |
| 19 | 0.1 | 20 | 29.7083775186 | 900.856 |
| 19 | 3.0 | 20 | 33.3297985154 | 900.856 |
| 23 | 0.08 | 10 | 25.1325556734 | 977.996 |
| 37 | 1.0 | 20 | 38.0869565217 | 1200.001 |

The model has low accuracy compared to KNN counterpart. We think that for some reason the model is not converging correctly, and the final weights seems to go outside of solution space.

Possible reason for this low accuracy can be that the initial random weights were too much different than the expected correct weights that the mode did not converge even after iterating up to 100 times through all the training data during accuracy test. We tried varied range for alpha, from 0.01 to 3, and reducing alpha by 0.9 after every 5 to 10 iterations. But the model doesn't seem to converge properly.

# Answers to other questions asked in the assignment…

## Which classifiers and which parameters would you recommend to a potential client?

The choice of classifier depends on the situation:

- **For better overall accuracy**, we will be recommending **K-Nearest Neighbors**. In our implementations, it has been observed that K-Nearest Neighbors tend to perform consistently better.
- **For faster execution time**, we will be recommending **Neural Network**, with already tuned weights (present into *model_file.mj*), so that it can just execute feed-forward and classify the test values.

## How does performance vary depending on the training dataset size, i.e. if you use just a fraction of the training data?

**K-Nearest Neighbors:**

Larger the training set size, more time it will take to classify an input test record.
Let N is the training set size. Let D be the dimension of each record in Training and Test set. So for each test set record with D dimensions it will compute Euclidean Distance against all the N training set records; and then choosing the top K closest Train Set Records for voting. In terms of accuracy, the most important thing is to find 1/2 training examples that are really closest to the query point. So amount of training data is not crucial for high accuracy, rather presence of actually related/close training data is more important.

Lesser the training data, there will be less chance of having more accuracy because of sparseness in the training data point. Statistical methods requires more training data for better accuracy.

**Neural Network:**

*Training and Tuning*: Larger the training set size, more time it will take to train and tune the Neural Network. Back-Propagation method will have to adjust weights every time for the new train set record. This process gets computationally expensive as the train set size increases. The accuracy varies a lot with varying training dataset size, if the training dataset is much smaller then it gives almost no correct output.

Time consumed by Neural Net also depends on several other factors like –

Number of Layers: More layers more computation complexity will happen, in our case it is 3 layers total.

Number of Nodes in each layer: More nodes more computation complexity will happen; besides bias nodes, in our case Input Layer has 192 nodes, output layer has 4 nodes, and the hidden layer has variable number of nodes. More the number of nodes in the hidden layer, more time it takes to compute the result.

Number of iterations used for tuning the Neural Network: More iterations, more time it will take to tune the network.

*Evaluating the Test Record with model_file loaded*: Here it is assumed that we have completed the training and tuning of neural net, followed by saving the model. Whenever we have to use the model, we just load the mode_file.mj from the disk and use it as our Neural Net Model. For classifying each test records, it is the feed-forward method that will take some fixed amount of time per input.

Time consumed by Neural Net also depends on several other factors like –

Number of Layers: More layers more computation complexity will happen, in our case it is 3 layers total.

Number of Nodes in each layer: More nodes more computation complexity will happen; besides bias nodes, in our case

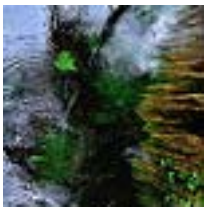
Input Layer has 192 nodes, output layer has 4 nodes, and the hidden layer has variable number of nodes. More the number of nodes in the hidden layer, more time it takes to compute the result.

Further, accuracy in NNet will not help if the model is not converging properly. In our case, we suspect model did not converge properly. If the model converges correctly then we think more train data will help the system achieve higher accuracy. Statistical methods requires more training data for better accuracy if the underlying model is converging properly.

## Show a few sample images that were classified correctly and incorrectly. Do you see any patterns to the errors?

Below are 5 images that are classified **incorrectly** by **KNN**:

 Predicted: 0 Actual: 270

 Predicted: 0 Actual: 90

 Predicted: 0 Actual: 180

 Predicted: 90 Actual: 0

 Predicted: 180 Actual: 0

Below are 5 images that are classified **correctly** by **KNN**:

 Predicted: 90 Actual: 90

 Predicted: 0 Actual: 0

 Predicted: 180 Actual: 180

 Predicted: 270 Actual: 270

 Predicted: 180 Actual: 180

There are some wrong outputs by KNN algorithm (with k=9), we consider these incorrectly classified samples as outliers in the local space where the training data points are present with different rotation-label values. Among 9 nearest neighbors (based on Euclidean distance), majority of these data points voted these test images anything but the actual value. It is not always necessary that certain RGB patterns of an image will be sufficient to correctly classify the image. It doesn't matter how much RBG pattern based training set we use, there will be some incorrectly classified images when testing the performance of KNN Model. RGB patterns of images are not sufficient to always correctly classify the other images.

Below are 5 images that are classified **incorrectly** by **NNET**:

 Predicted: 180 Actual: 0

 Predicted: 270 Actual: 0

 Predicted: 270 Actual: 180

 Predicted: 180 Actual: 0

 Predicted: 270 Actual: 180

Below are 5 images that are classified **correctly** by **NNET**:

 Predicted: 270 Actual: 270

 Predicted: 0 Actual: 0

 Predicted: 90 Actual: 90

 Predicted: 90 Actual: 90

 Predicted: 0 Actual: 0

There are mostly the wrong outputs by NNET system (with alpha=0.01 and epoch=37), we consider these incorrectly classified samples as a result of sub-optimal values of input weights. We suspect that for some reason the model is not converging correctly, and the final weights seems to go outside of solution space.

Possible reason for this low accuracy can be that the initial random weights were too much different than the expected correct weights that the mode did not converge even after iterating up to 100 times through all the training data during accuracy test. We tried varied range for alpha, from 0.01 to 3, and reducing alpha by 0.9 after every 5 to 10 iterations. But the model doesn't seem to converge properly. Debugging the NNET system wasted our 9 days in total.