Assignment 5: CSCI B 551

1. KNN:

KNN working flow :-

Loop through all the test points and calculate the distance between each test point with all the train points. Create a dictionary which keeps track of all the distances calculated. Variate k to find the best accuracy

We used the np.linalg.norm() function to calculate the distance which also normalizes the data. We tried various distance functions Euclidean, Manhattan,etc but
Euclidean distance gave the best result so we tried to optimize this further.

Limitations : -

Computation time. It is taking around 20-30 mins just to calculate all the possible distances and then compute the predicted class of the test point. We checked the different between the accuracies given by normalized as well as unnormalized data and it was around the same. Normalized data is a little faster for computation.

Getting an accuracy of 64 when k is 77

K	Accuracy
3	59
5	61
31	61.18
77	64.79
91	64.79

Note: Computation takes around 20 mins.

2. Adaboost:

This implementation of Adaboost uses Decision stumps as classifiers. We select the best of out of 10 decision stumps
based on which gives least error rate for each stump that needs to selected. This is done by selecting 2 random indexes(index1 and index2).

It is positive if value at index1 > value at index2 and negative otherwise. Each classifier is weighed based on the perfomance it
gave(based on error rate). Lesser the error rate more its weight. Finally, the adaboost outputs a list of classifiers for each class,
with a corresponding weight for each of them.

alpha calculation:

alpha = 1/2 ln((1 - err)/err)

data = data*e^alpha for misclassified
data = data*e^-alpha for correctly classified

final_ensemble_dict contains the list of classifiers for each degree/orientation with the index details.

test_and_classify() uses this final_ensemble_dict to check what is the majority weight to assign a class for each.

For Decision Stump count=5:
Accuracy:
0.575821845175
Confusion Matrix:

	0	90	180	270
0	145	49	28	17
90	49	140	20	15
180	40	27	140	29
270	46	46	34	118

For Decision Stump count=10:
Accuracy:
0.612937433722
Confusion Matrix:

	0	90	180	270
0	145	42	39	13
90	27	149	27	21
180	28	31	163	14
270	59	36	28	121

For Decision Stump count=15:
Accuracy:
0.640509013786
Confusion Matrix:

	0	90	180	270
0	171	28	22	18
90	41	150	13	20
180	44	45	118	29
270	39	29	11	165

3. Neural Networks:

Steps involved:

1. Train the network:
a. Initialize network: Build a neural network with the hidden neuron count.
b. Forward Propagation: This calculates the output of each neuron in each layer as a sum of weights * inputs for that neuron.
c. Back Propagation: This is a way in which the error at each layer is calculated for each neuron.
d. Update Weights: This is calculated considering the learning rate, delta value and the previous received at each layer and neuron.
d. Activation: Calculation of a output which quantifies the value at that position.

2. Predict the data:
Taking the test data and running it using the forward propagation will give a output value at the end.
The output neuron for a class which throws the maximum probability is the predicted class for the data row.

Explanation of different functions:

Sigmoid function: 1.0 / (1.0 + math.exp(-1.0 * x))
Activation funtion: sum of weight * input
Transfer Derivative: output * (1.0 - output)
Error function: for output layer: (expected_output - actual_output) * transfer_derivative,
for hidden layer: (weight*error)*transfer_derivative.
Learning rate: It is a parameter which controls how much faster the weights can converge onto the final set.