MACHINE LEARNING

Assignment 5

Waquar Shamsi (MT20073)

Reshan Faraz (Phd19006)

20th November 2020

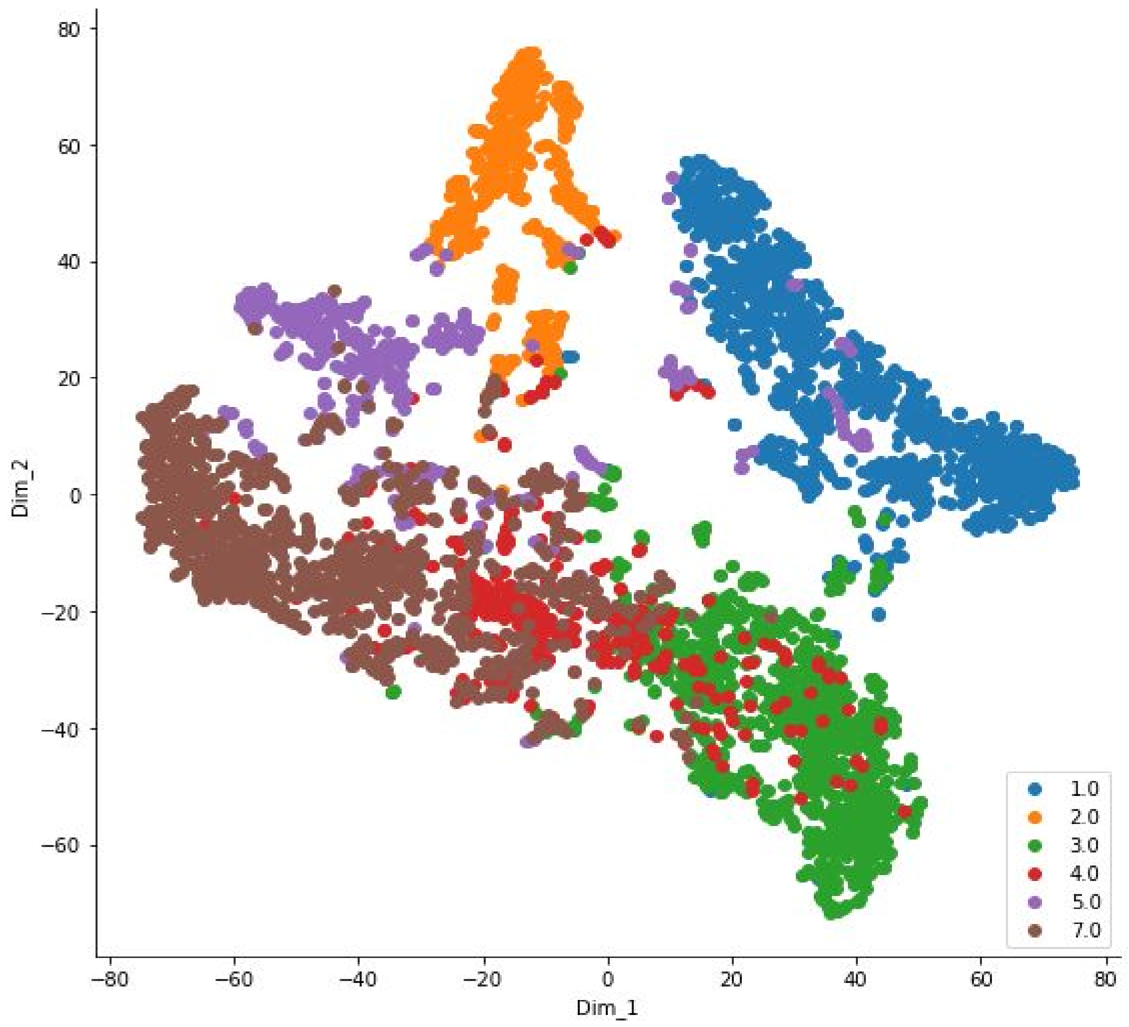
Question 1: kNN Algorithm

Answer to Question 1 :

Preprocessing of Data : We load the data using pandas library and stored as dataframe .

The columns are space separated. There are two data sets, one for training and the other for testing. We checked for any null values or NaN in the column before using the data set.

Answer 1-1 : Here is the plot visualization of the data set with t-SNE. We reduced the features to two dimensions for better visualization. Following are the parameter used in t-SNE : iteration = 1000 and perplexity : 30



Answer 1-2 : we implemented the kNN from scratch. For this we implement a class userkNN with Euclidean distance as a metric algorithm. We take the point and based on the hyperparameter *k* we find the Euclidean distance between the k nearest point. the `knn_distances()` method of class userkNN will return the top k closet distance and the `knn_predictions()` method returns the predicted value.

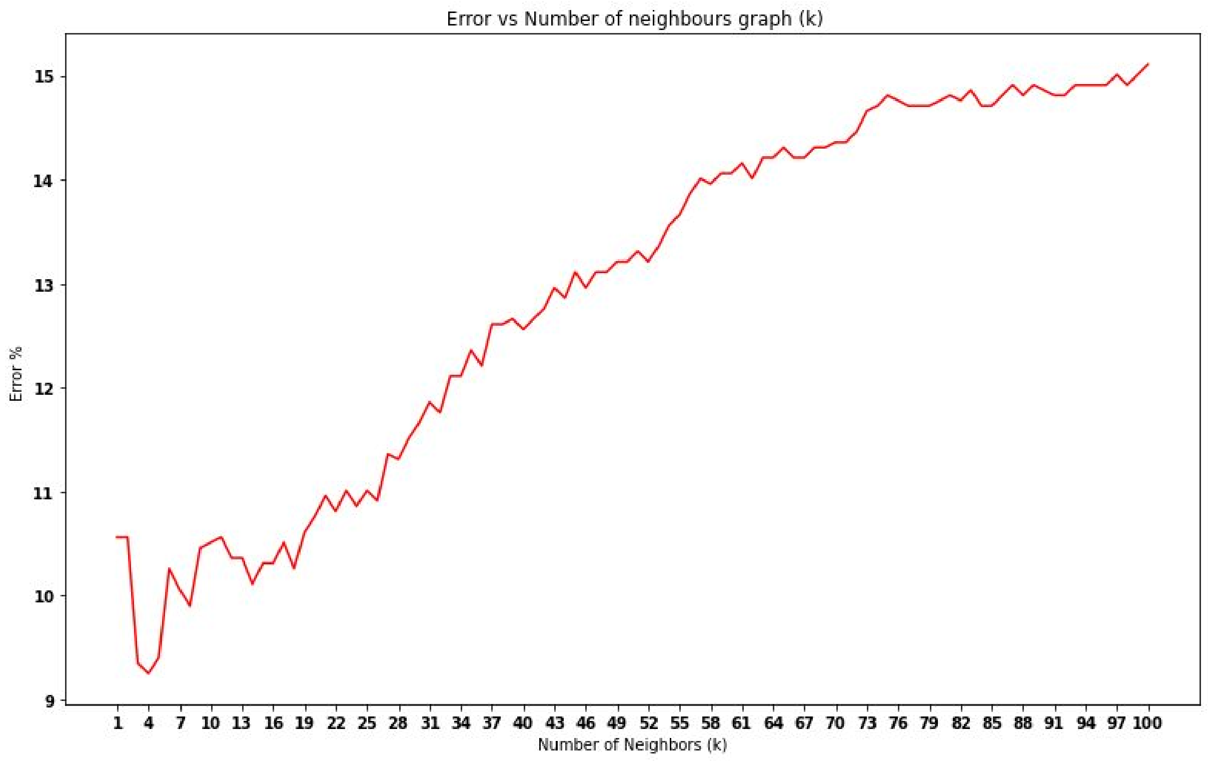
We used grid search to find the optimal value of k among 100 values starting from *k =1 to k=100* .

Optimal value of k is **4**

Maximum accuracy at optimal value is **90.75%**

We also show the error for each value of K (1-100) :

We also show the error vs number of neighbour (k) table. Here is the graph for error vs number of neighbours graph (k).



Answer 1-3 : Here is the result for user defined KNN and sklearn KNN

Validation Accuracy user defined KNN: **90.75 %**

Training Accuracy user defined KNN: **96.82 %**

Validation Accuracy using sklearn: **90.25 %**

Validation Accuracy using sklearn: **93.98 %**

For Validation Accuracy there is 0.50% difference between the user defined KNN algorithm and sklearn . For Training Accuracy there is about 2.84%. This might be a different implementation from sklearn . Also algorithm parameter is auto in sklearn KNN hence it will select based value passed to fit(). And in the case of user defined KNN we calculate the Euclidean distance based on matrix form.

All the above will be evaluated for the optimal value of the number *of neighbour (k)* obtained in Question 1-2 which is 4.

Question 2: Neural Networks

Use the MNIST subset data for this question.

(1) Split the data into a train and test set with 80:20 (use seed 42). The test set should be held out. [5 Points]

DATASET DESCRIPTION:

Dataset provided is a subset of the MNIST digit classification dataset. It is used for handwritten digit identification/classification.

The dataset consisted of 14251 samples, each with two dimensions (28, 28) and there was 14251 number of labels one for each sample.

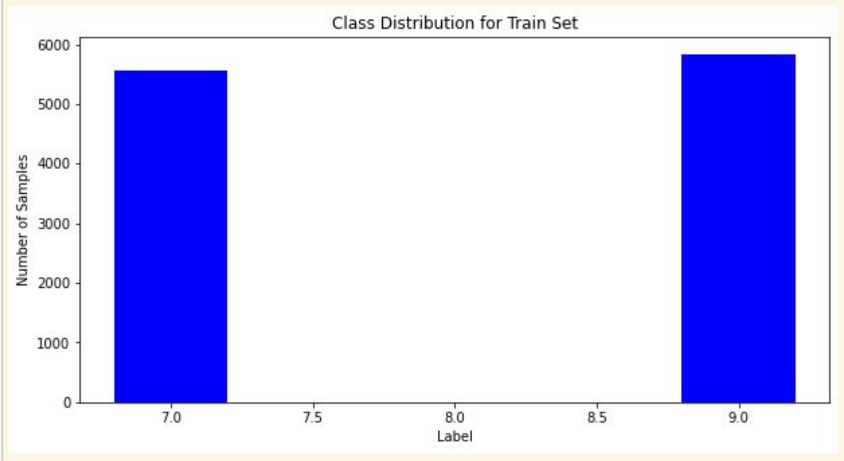
Distinct Class Labels:

```
DISTINCT CLASS LABELS: {9, 7}
```

There were two distinct class labels i.e, 9 and 7.

Check for class Imbalance:

```
Count of samples from each class:  Counter({7: 5833, 9: 5567})
```



It can be observed that there is an almost equal number of labels of each class. So there is no class imbalance problem here.

APPROACH:

Expanded the 28*28 into 784 features.

Then used sklearn to split the dataset into 80:20 ratio with seed value 42.

Stored the train and test set using pickle for further use.

---

(2) Implement a NN architecture using sklearn with 3 hidden layers - [100, 50, 50]. Assume a Sigmoid activation function in each layer. Report the accuracy and loss. [15 Points]

APPROACH:

Feature Scaling: As mentioned in sklearn's documentation page for MLPClassifier, 'Multi-layer Perceptron is sensitive to feature scaling' so performed feature scaling prior to applying the neural network using sklearn's StandardScaler.

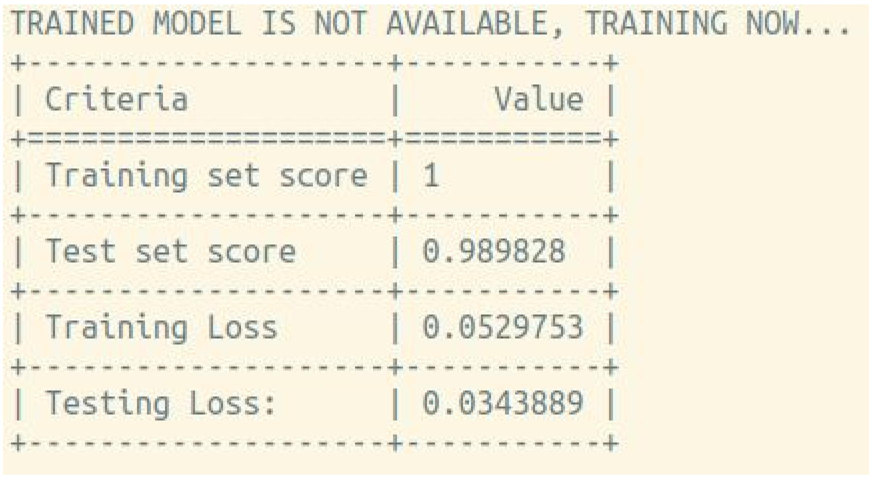
OneHotEncoding: Found that one hot encoding can be done on the label vector before supplying it to the neural network to get better results. So performed OneHotEncoding using sklearn.

Performed a grid search with alpha values = [0.0001,0.001,0.1,10]. Hidden layer size = (100, 50,50). Activation Function = 'logistic' i.e., 'sigmoid'. Max_iterations = 1000. Initial learning rate 0.001.

Then trained the model on the test set generated in part a, and saved it.

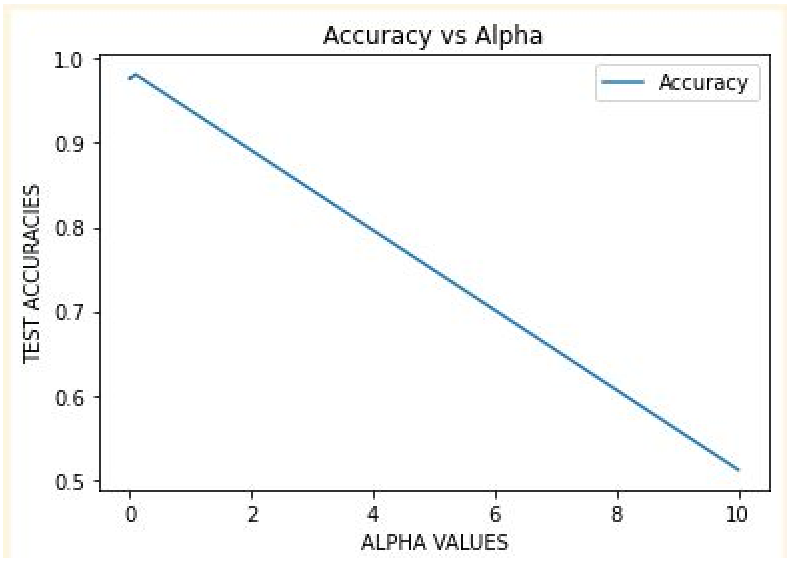
Then loaded the test set and performed the one-hot encoding as well as feature scaling. Then used predict_proba using the best model obtained from grid search. To find the accuracy for both training and testing set, I used the inbuilt method of the MLPClassifier object and for finding the loss, the training loss was obtained from the object of MLPClassifier i.e., the model itself. For testing loss, used the log_loss function of sklearn. Following results were obtained. Used log loss because it's also known as cross-entropy loss which is the default loss function used by the MLPClassifier.

RESULTS:



```
TRAINED MODEL IS NOT AVAILABLE, TRAINING NOW...
+-------------------+-----------+
| Criteria          |     Value |
+===================+===========+
| Training set score | 1         |
+-------------------+-----------+
| Test set score     | 0.989828  |
+-------------------+-----------+
| Training Loss      | 0.0529753 |
+-------------------+-----------+
| Testing Loss:      | 0.0343889 |
+-------------------+-----------+
```

Due to normalization and hot-one encoding achieved the training accuracy of 100% and testing accuracy of 98%. Also, the loss values obtained were very small as they reduced over a large number of iterations (took 1000 as the maximum number of iterations).



Also plotted a graph for different values of alpha and concluded that for small values of alpha such as 0.001 and 0.1 the accuracies were quite high but for high values of alpha the such as 10. This is because the alpha value is the amount of regularization. And for high regularization values the model underfits.

(3) Use the same architecture as in (2) and plot the decision boundary for 3 different values of 'alpha' (2 extreme and 1 middle value) b/w the training samples. [15 Points]

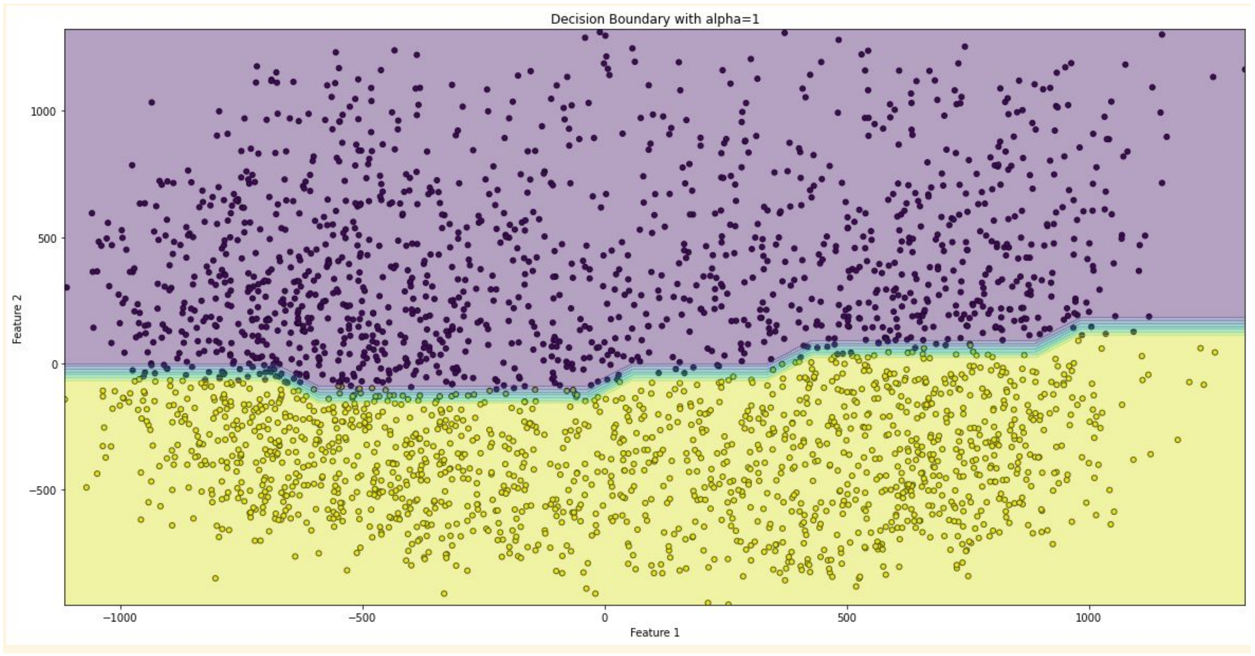
APPROACH:

Took values of alphas as (1, 0.0001, 0.000000001).

1 being extreme large alpha value while 0.000000001 being the extreme small alpha value.

Reduced the number of features to 2 using PCA in the training dataset which was obtained in question 2 part a. Trained the model with the same architecture. Then loaded the test dataset reduced its number of features too using PCA. Used the counterf function to generate the decision boundaries along with respective regions.

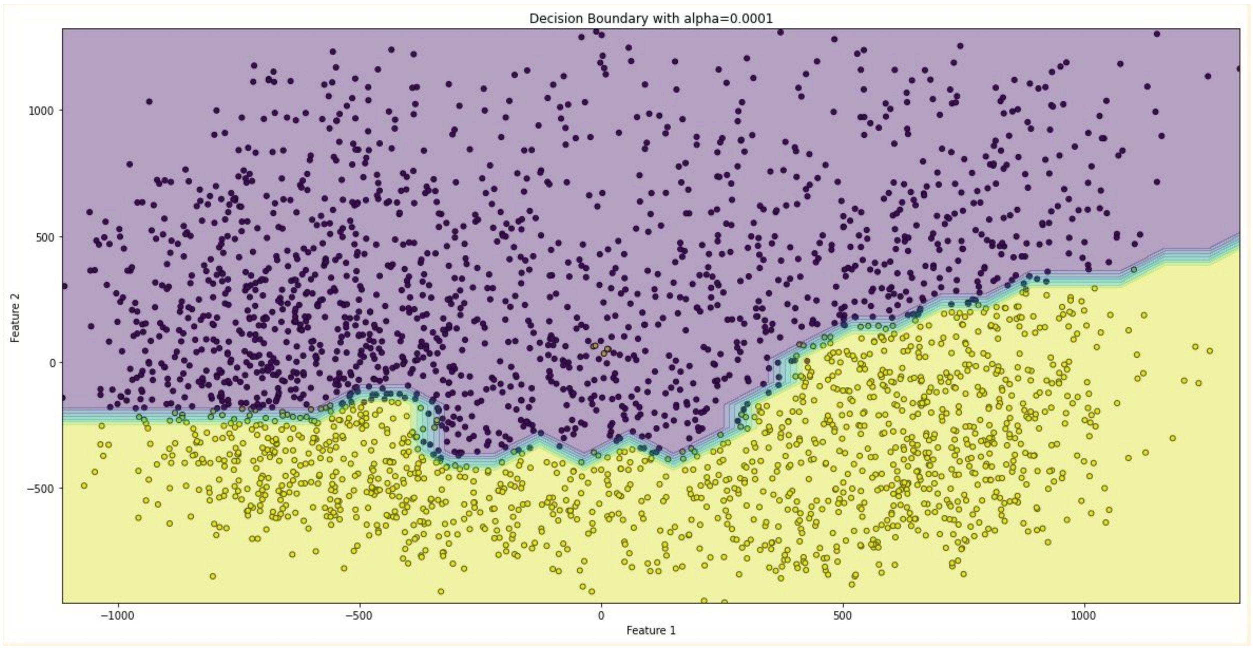
Decision Boundary Using Alpha = 1



It can be observed that the decision plane can easily distinguish the two decision regions linearly in 2D space. There are almost no outliers in the plot. The decision plane seems very horizontal. So one thing can be observed that for samples having feature2 as less than 0 will almost certainly be classified as one class and for value, more than 0 will be classified as another class.

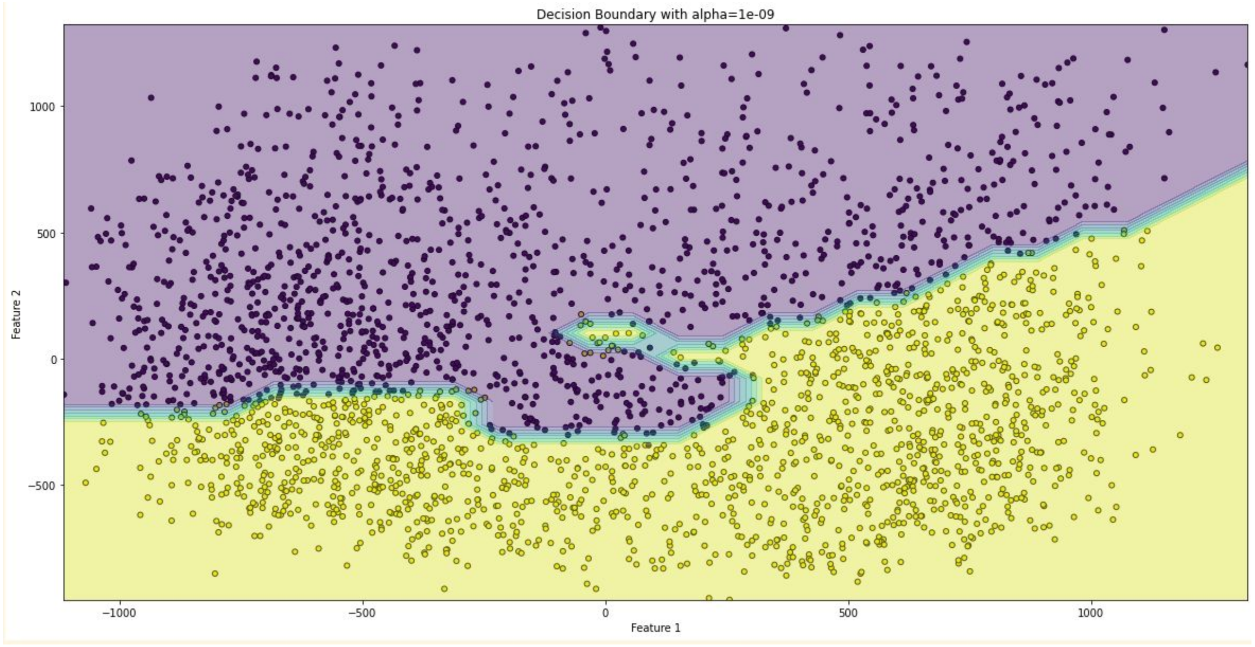
This simple decision plane may also represent the **underfitting scenario**.

Decision Boundary Using Alpha = 0.0001



On this smaller value of alpha, it is observed that the decision plane is more complex and thus does not completely horizontally separate the data point. This means that the model has better learned the dataset and thus must be more accurate than when alpha=1. It can be observed if looked very carefully that some data points of yellow class lie in the region of purple class. So the smaller value of alpha must be used to make the model capture those data points.

Decision Boundary Using Alpha = 1e-09



Using this very small value of alpha also gives good results. It can be seen that the model is not very simple thus there is no underfitting. The model learns the dataset very nicely. It can be observed that there is an overlap of the data points and a few of the data points that earlier lied in the wrong decision region are now lying in their real decision region of yellow. So it can be concluded that this is even better than the previous value of alpha, but decreasing it too much may lead to making the model too complex and lead to overfitting.

REFERENCES:

[1]. Sklearn Neural Networks Documentation, https://scikit-learn.org/stable/modules/neural_networks_supervised.html

[2]. Wikipedia page for MNIST Dataset, https://en.wikipedia.org/wiki/MNIST_database

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/

https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

https://www.geeksforgeeks.org/ml-t-distributed-stochastic-neighbor-embedding-t-sne-algorithm/