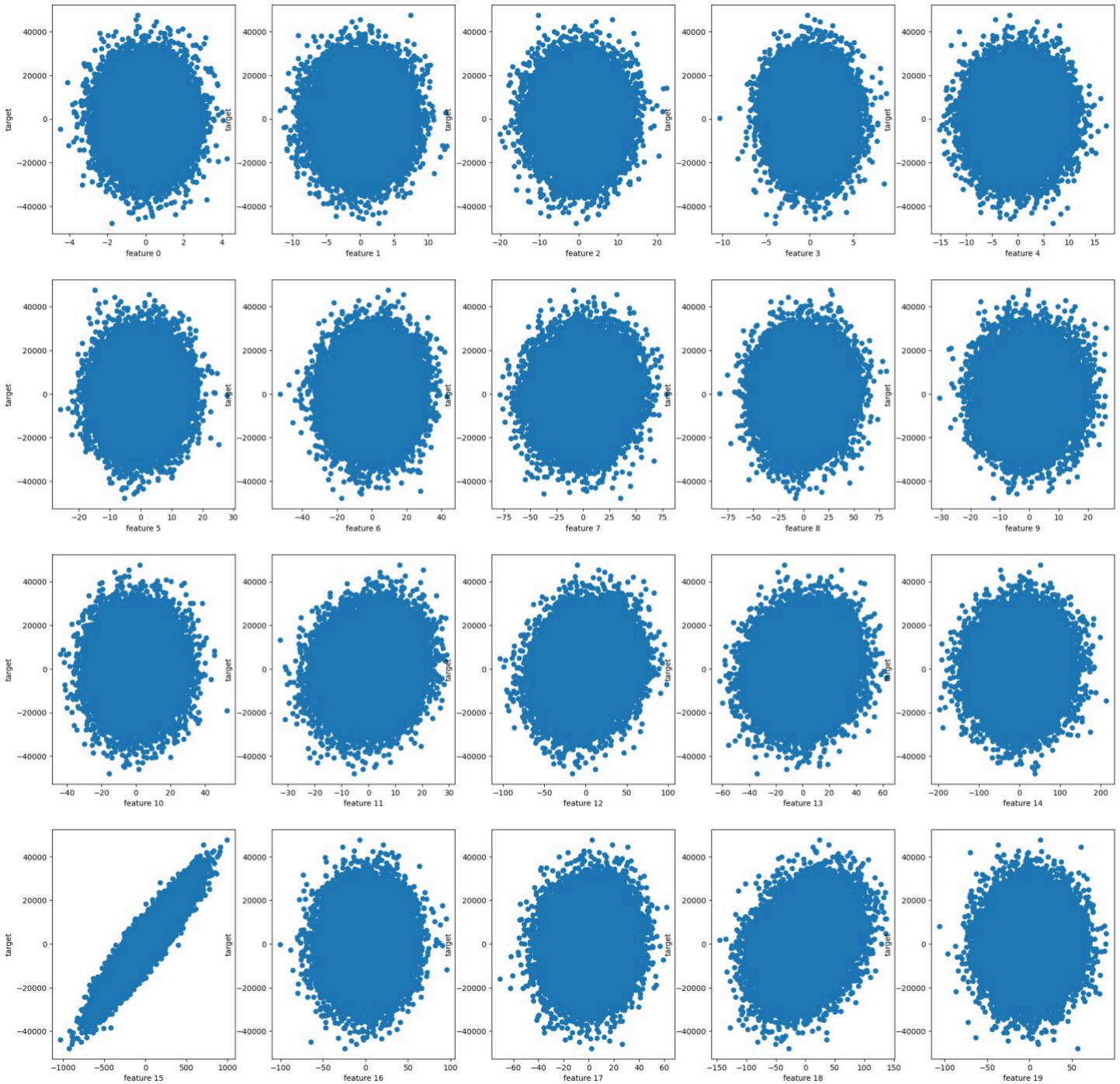
Machine Learning Bootcamp

Before the coding phase of WOC 6, I had started watching Andrew NG's machine learning specialization course on Coursera. Having finished course 1 which consisted of linear regression, polynomial regression and logistic regression, I started with coding on Google colab. To gain experience on both platforms, I made Linear regression, Polynomial regression and Logistic regression on Google colab and Neural Network, KNN and K-means Clustering on Jupyter Notebook. I also watched youtube tutorials and read documentation on numpy, pandas and matplotlib libraries to get basic ideas about their usage. Equipped with this knowledge I started my project.

Linear Regression:

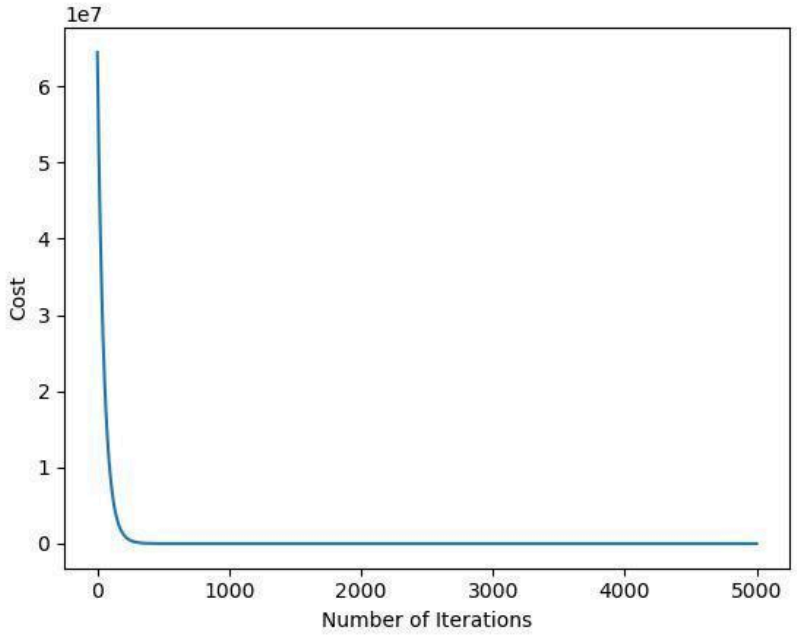
For linear regression, I split the dataset into train set and cross validation set in ratio 80:20 (40000:10000) and plotted the target across each feature using subplots feature of matplotlib



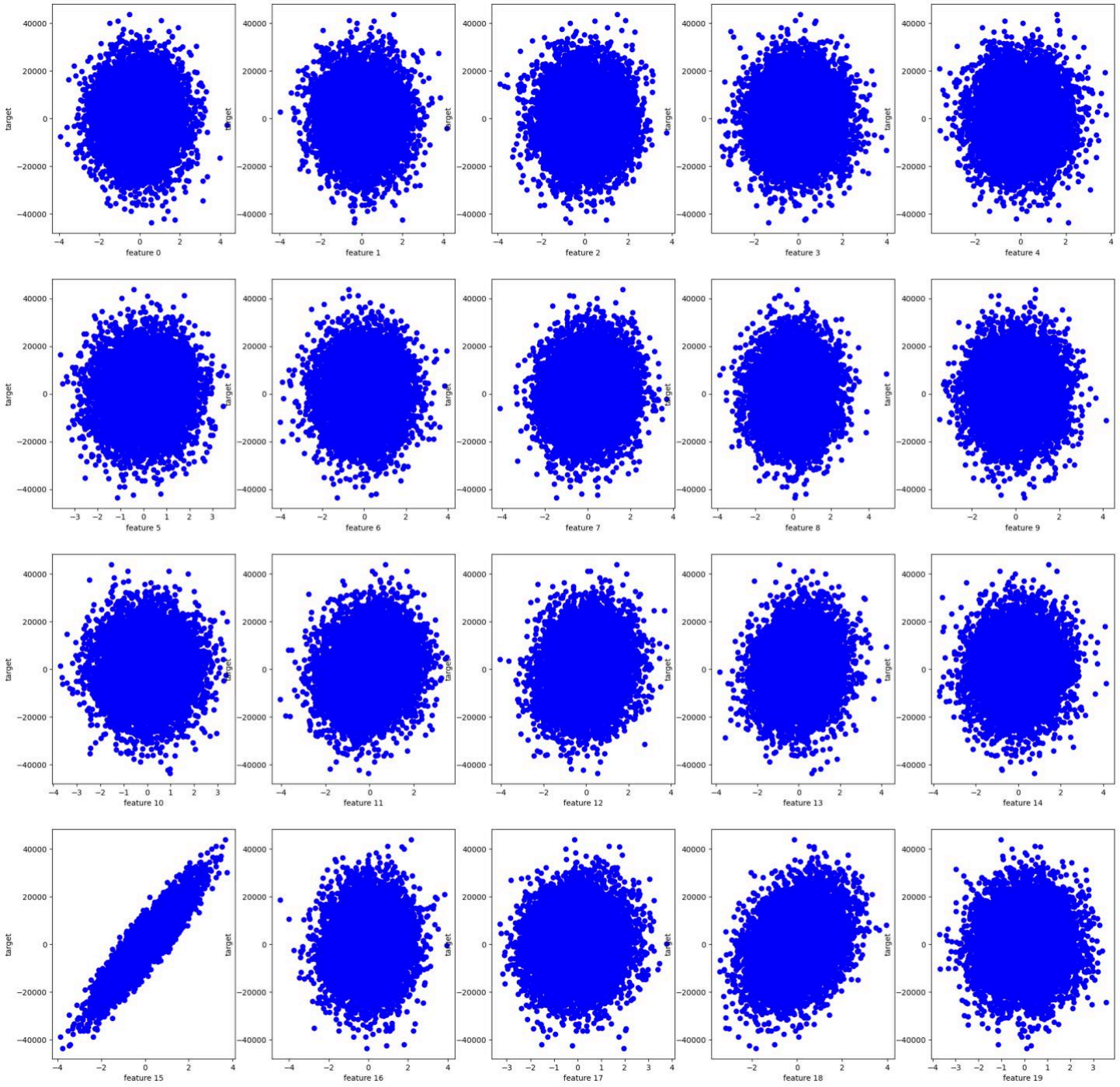
I implemented Z-score normalisation on the training dataset and calculated cost between the predicted (y_pred=wx+b) and expected values using mean square error. I implemented gradient descent to minimise the MSE cost and train the model (Weights and biases). I trained the model for **5000 iterations** and **alpha to be 0.001** and the cost was minimised to **0.005048418072069561**

```
cost at  0  =  64441403.301977016
cost at  1000  =  0.1264348700933828
cost at  2000  =  0.005048418335895111
cost at  3000  =  0.0050484180720696
cost at  4000  =  0.005048418072069561
```

The graph for decrease in cost with respect to number of iterations is as follows-



With this trained model I made final predictions on the train and the CV set. Here's how the predicted values compared against actual values in CV set.

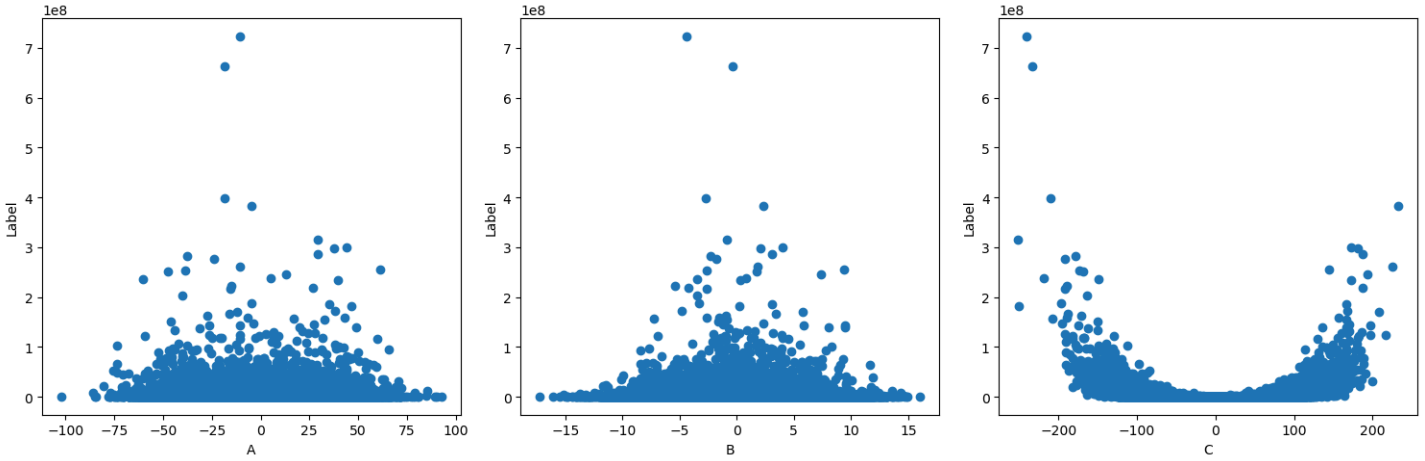

target
feature 0
feature 1
feature 2
feature 3
feature 4
feature 5
feature 6
feature 7
feature 8
feature 9
feature 10
feature 11
feature 12
feature 13
feature 14
feature 15
feature 16
feature 17
feature 18
feature 19

where blue dots represent the predicted values and red dots represent the actual values in the CV set.

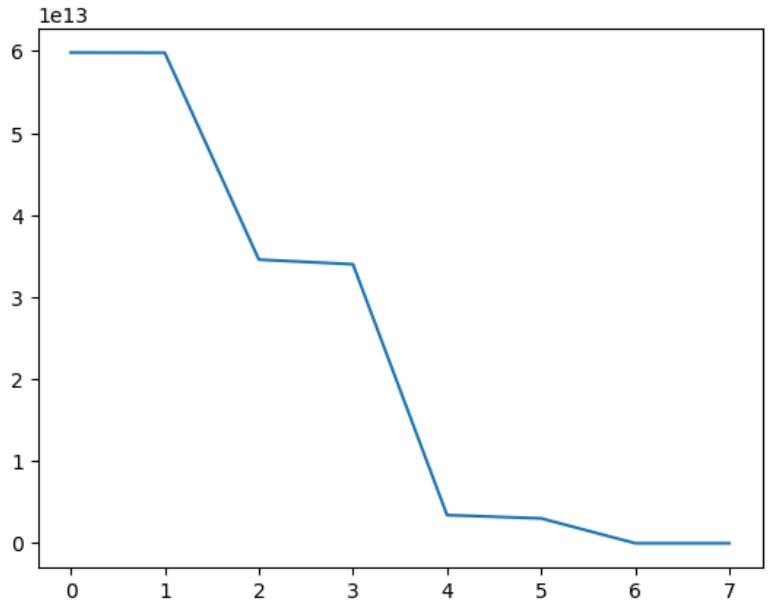
The R2 score of the model on the Training set is **0.9999999999232193** while that on the cross validation set is **0.9999999999216167**.

Polynomial Regression

For Polynomial Regression I split the data in 80:20 (40000:10000) for training and cross validation set and plotted the targets against the three features.



and implemented Z-score normalisation to normalise the data set and stored its mean and deviation on the training set. I modified the features such that it fit a n degree polynomial by generating all the terms of a general n degree polynomial except the degree 0 term. I implemented mean square error and gradient descent to train the model. I plotted the minimised cost against n and deduced that minimum cost was achieved for **n=6**,



I ran the model for **40000 iterations** with **alpha 0.2** and the cost minimised to **3.112704628610564e-16**

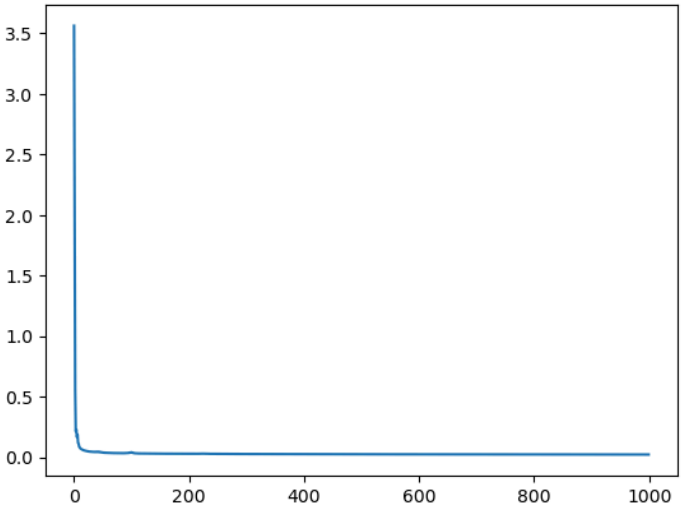
The cost vs degree of polynomial graph is plotted on the left.

The minimised cost came out to be **3.1127046286105 64e-16**. Using the corresponding values of weights and biases, I predicted the values(wx+b) for training and cv set. The R2 score for the training set as well as cross validation set is 1.0.

Logistic Regression

For logistic regression, I made an 90:10 split on the data(27000:3000) and scaled the data between 0 to 1 by dividing the features by 255. I converted the labels for the digit recognition dataset into a one-hot encoded set such that each column had the digit 1 for the corresponding column number. The shape of the one hot encoded matrix is (24000,10). MSE could not work for logistic regression since it gives multiple local minima hence gradient descent could not work properly. I implemented binary cross entropy cost and then gradient descent to minimise cost for each column (labeled as 0,1,2..9) as vectorised code. That is, I converted a multi-class classification into 10 binary classification for each column. After trying multiple learning rates and number of iterations, I chose **learning rate 7** for **1000 iterations**.

```
cost at  0 = 3.5622358920864086
cost at  100 = 0.04007115068800955
cost at  200 = 0.028366228555027146
cost at  300 = 0.026552597072153766
cost at  400 = 0.025417248514540774
cost at  500 = 0.024586127587088728
cost at  600 = 0.023939746749224285
cost at  700 = 0.02341269092080576
cost at  800 = 0.0229698107180506
cost at  900 = 0.022589315375280396
```

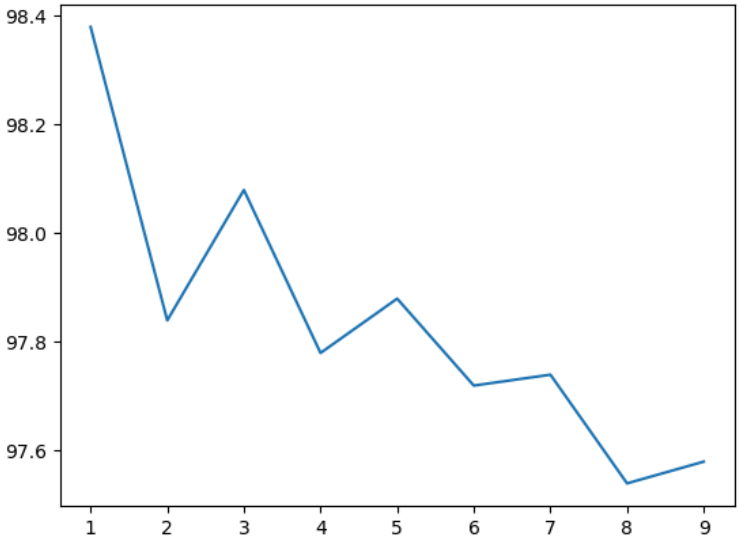


Using the trained weights and biases, I predicted the values for the cross validation set, and passed it through the sigmoid activation function which converted all the real values between (0,1) denoting the probability of the predicted value to be a specific number. Then I chose the column with maximum probability using np.argmax() for each row. The one hot encoded matrix allowed me to directly get the predicted number since the column at i'th position was allotted i'th digit from 0 to 9. I calculated the accuracy of the model

on the cross validation set which came out to be **97.03333333333333**, and that on train set is **97.57407407407406**

K nearest neighbours

For K nearest neighbours I made a 80:20 split on the dataset to make a training and cross validation set and then scaled down the dataset by dividing it by 255 such that all the training features are in between 0 to 1. I stored the distances of each point on the test(cross validation) dataset with all other points as a row in a 2D matrix of shape (no. of points on test set, no.of points on train set). I calculated accuracy for a general value K by taking K indices from the sorted array of distances of one point on test set with each point on train set using np.argsort(), and created two arrays one to store the unique labels and one to store frequency of the labels during using np.unique(). The prediction value will be the value in unique labels at the index corresponding to the maximum frequency in the other set array. Using these predictions, I printed the accuracies of the model against the cross validation set. I implemented this algorithm for different values of K in 1 to 9 and plotted the accuracies for each K against them. The accuracy vs K graph is as follows:




```
98.38
97.84
98.08
97.78
97.88
97.72
97.74000000000001
97.54
97.58
```

From this plot, it is evident that accuracy is highest for k=1 but that does not sit right with me that a data point is the same as the other data point closest to it. Hence I take the next best k=3 for which the accuracy is 98.08%.

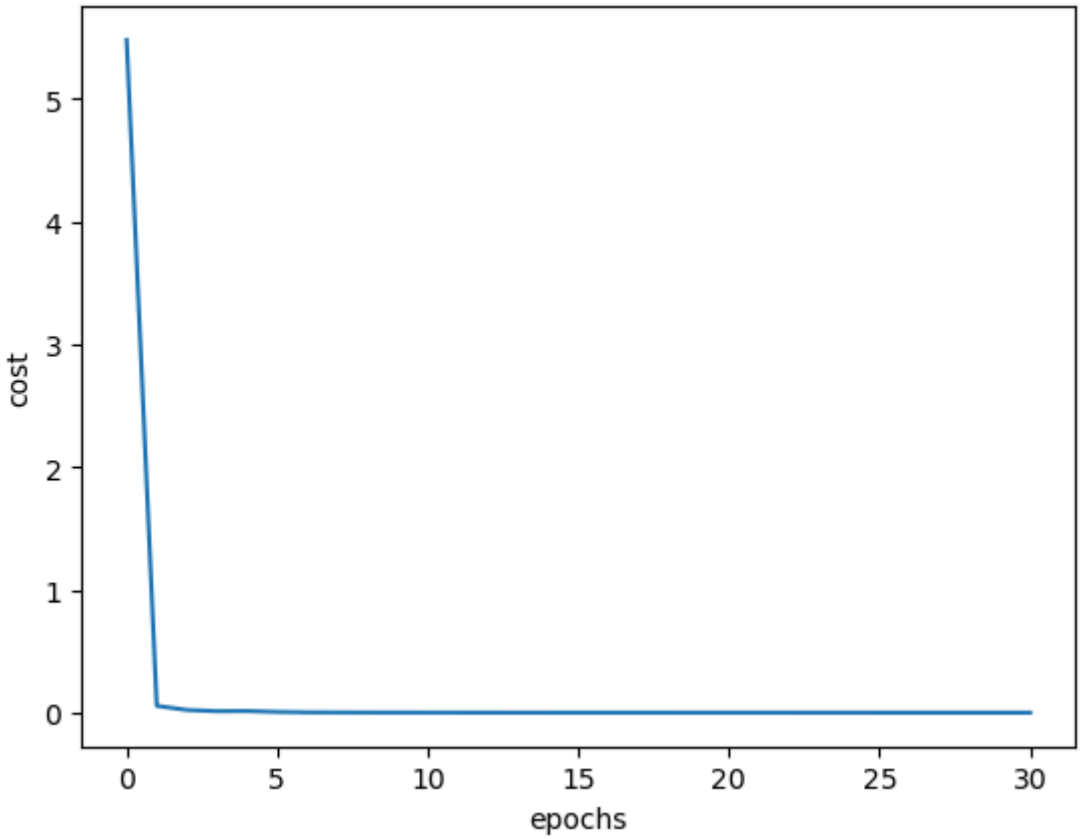
Neural Network

For Neural Network I shuffled the data frame and then made a 80:20 split on the dataset. For this specific algorithm I stored the training data in the form of a matrix of shape (no. of features, no. of inputs). Then I rescaled the features by dividing them by 255 so that all their values are between 0 and 1.

I one - hot encoded the labels for the training set such that they are of shape (10,no. Of features). Then I created functions for activation functions and derivatives for the activation functions including **softmax**, **relu**, and tanh. I took inputs of a vector storing neurons for each layer of the neural network, corresponding to their indices.I initialised weights and biases for each layer and to avoid the problem of vanishing and exploding gradients I divided the random weights by np.sqrt(2/neurons in that layer). I used dictionaries to store all variables in forward propagation and backward propagation(activations and gradients) for easy access. I implemented **sparse categorical cross entropy** error as the cost function and trained the model using back propagation. I used softmax layer as the output layer to get probabilities across the 10 digits and used argmax along axis 0 to get the corresponding predicted values. To train the model, initially I passed the entire dataset in forward propagation to train the model but after a couple of executions, I realised that it took too long to run. After some searching on the net I came across **mini-batch** method which was much faster. I passed the data in batches of **64** data. I created a **4 layer** neural network with neurons in layers being **[784, 512,256,128,10]** with **relu** activation function for each layer other than output. With learning rate to be **0.1** and **30** epochs and the minimised cost came out to be **0.0002607125555786 82**.

For **tanh** activation function, the cost minimised to **0.0007751461148977428**. Hence I decided to go with the relu activation function.

The cost vs epochs graph is as follows:

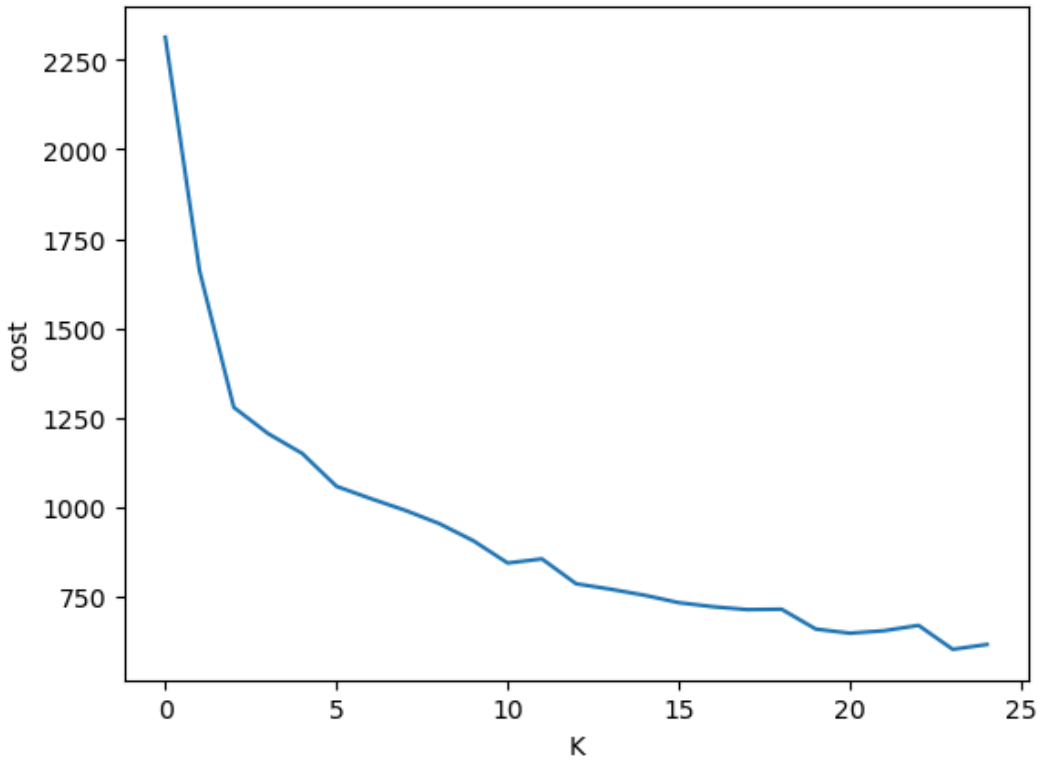


The accuracy of the model on the training set came out to be **1.0** and that on the cross validation set is **0.985**.

K means clustering

For K means clustering, I took input of the training dataset. I implemented Z-score normalisation on the data set and then randomly initialised K data points from the data set as centroids, Then deduced the closest centroid to each point and then updated each centroid to the mean of the points associated with each centroid.

For each value of K the cost is plotted as follows.



From the above graph I deduced that the number of clusters could either be 2 or 5. From the data given, I thought it did not make much sense for it to have only 2 clusters. Thus it would be 5.

REPORT PREPARED

NAME: Divyansh Rastogi

ADM NO: 23JE0320