Karaka Prasanth Naidu

Btech. Electrical Engineering

Project Report-ML Bootcamp

Feb-March 2023

Abstract

Implementing various Machine learning algorithms from scratch using libraries Numpy, pandas, and matplotlib and training them on given datasets.Implemented Linear(and polynomial) regression, logistic regression, L-layer Neural network, K-nearest neighbours

Tech Stack Used:-

Numpy is used for performing all calculations required.

Pandas is used for datahandling and processing.

Matplotlib is used for data visualization,performance analysis.

Contents

Chapter 1

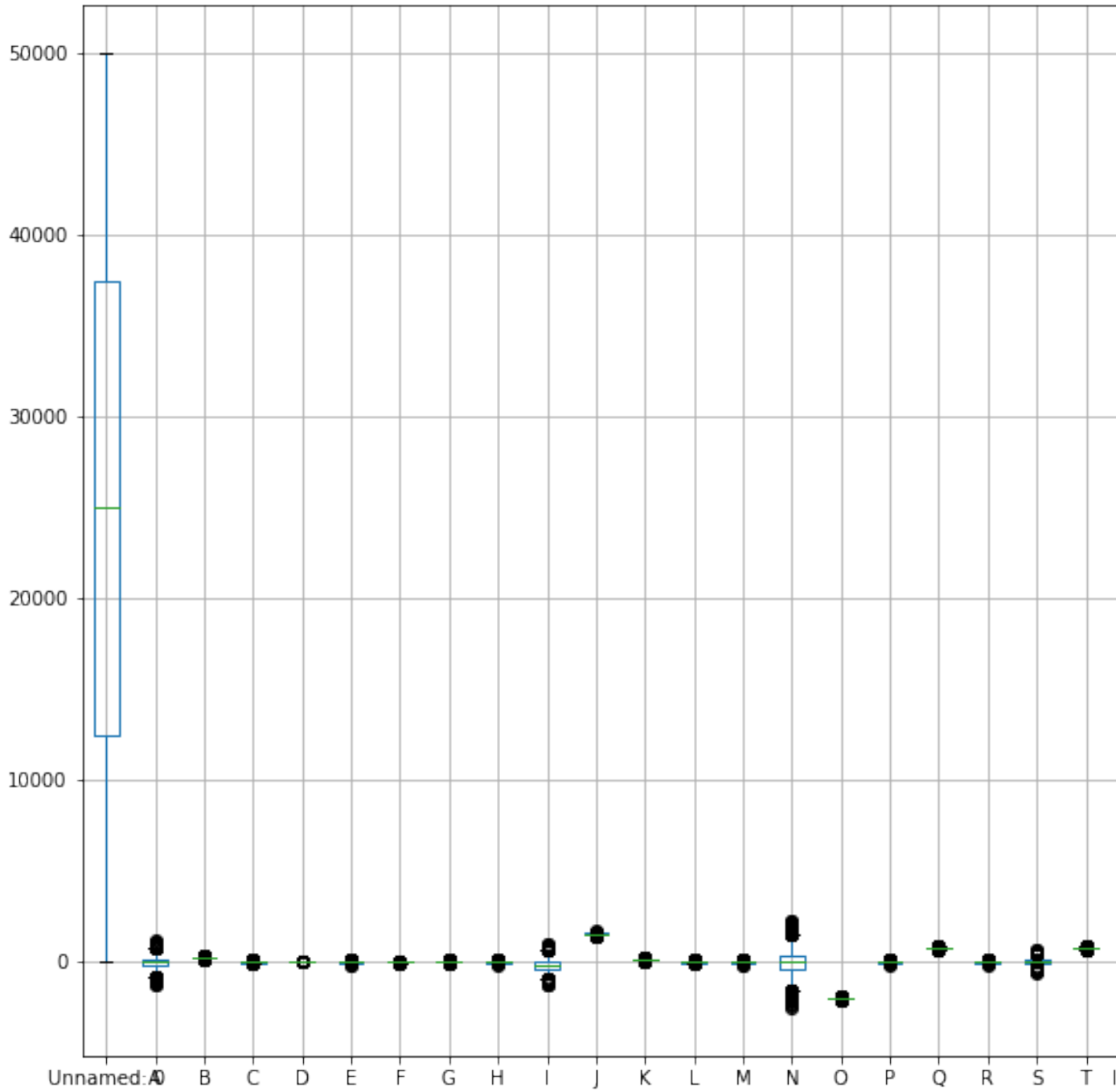
Linear Regression Model

Linear Regression is a simple Machine Learning algorithm that is used to predict a continuous target variable based on one or more predictor variables. In this implementation gradient descent method to minimize the sum of squared errors.

We will start by defining a class called Linear Regression, which will have two methods: fit and predict. The fit method will take in the training data, Xtrain, and ytrain, and use gradient to calculate the coefficients of the linear regression equation. The predict method will take in a test dataset, Xtest,and use the calculated coefficients to predict the target variable, ytest.

1.1 Data Analysis

I have used pandas,matplotlib to analyse the given data.



1.1.1 Standardization

As we can see,each feature has very different range,and this data is not suitable to train on linear regression model as having different range of features will make the gradient descent take different steps for each feature,to ensure gradient descent take similar steps,we should apply

standardization

$$x_i = \frac{x_i - \overline{x}}{\mu}$$

This will make the mean,variance of each feature 1,0

1.1.2 Data Split

I have used pandas to split the given dataset into training and test dataset with a ratio 80:20.

1.2 Implementation

1.2.1 Implementation details

I have created a class Linear regression and implemented fit function which trains the model on given dataset and predict function which makes prediction on the given dataset and implemented different machine learning metrics to evaluate the performance Cost Function:-

$J(\vec{W}, B) = \frac{1}{2m}\sum(\hat{Y} - Y)^2$

$$J(\vec{W}, B) = \frac{1}{2m}\sum(\hat{Y} - Y)^2 \implies J(\vec{W}, B) = \frac{1}{2m}\sum(\vec{W}.\vec{X} + B - \vec{Y})^2$$

Gradient Descent:-

$$\frac{\partial J_{\vec{w},b}}{\partial \vec{W}} = \frac{1}{m}(\vec{X^T}).(\vec{W}.\vec{X} + B - \vec{Y}) \tag{1.1}$$

$$\frac{\partial J_{\vec{w},b}}{\partial B} = \frac{1}{m}\sum(\vec{W}.\vec{X} + B - Y) \tag{1.2}$$

$$W := W - \alpha * \frac{\partial J_{\vec{w},b}}{\partial W} \tag{1.3}$$

$$B := B - \alpha * \frac{\partial J_{\vec{w},b}}{\partial B} \tag{1.4}$$

Prediction:-

$$Y_{pred} = \vec{W}.\vec{X} + B \tag{1.5}$$

Scoring Metrics:-

$$R^2 = 1 - \frac{RSS}{TSS} \tag{1.6}$$

$$MSE = \frac{1}{m}\sum_{i=0}^{m-1}(Y_i - \hat{Y}_i)^2 \tag{1.7}$$

$$MAE = \frac{1}{m}\sum_{i=0}^{m-1}|Y_i - \hat{Y}_i| \tag{1.8}$$

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=0}^{m-1}(Y_i - \hat{Y}_i)^2} \tag{1.9}$$

1.3 Training and testing

After training the model on given dataset,with a learning rate 0.01,for 10,000 iterations.

The following cost,weights and bias are obtained.

weight:-

73.31415097 66.47935187 97.21330636 2.0588644 17.56569253

24.84766668 71.33947836 30.68910035 20.26142815 93.40902237 39.45416784

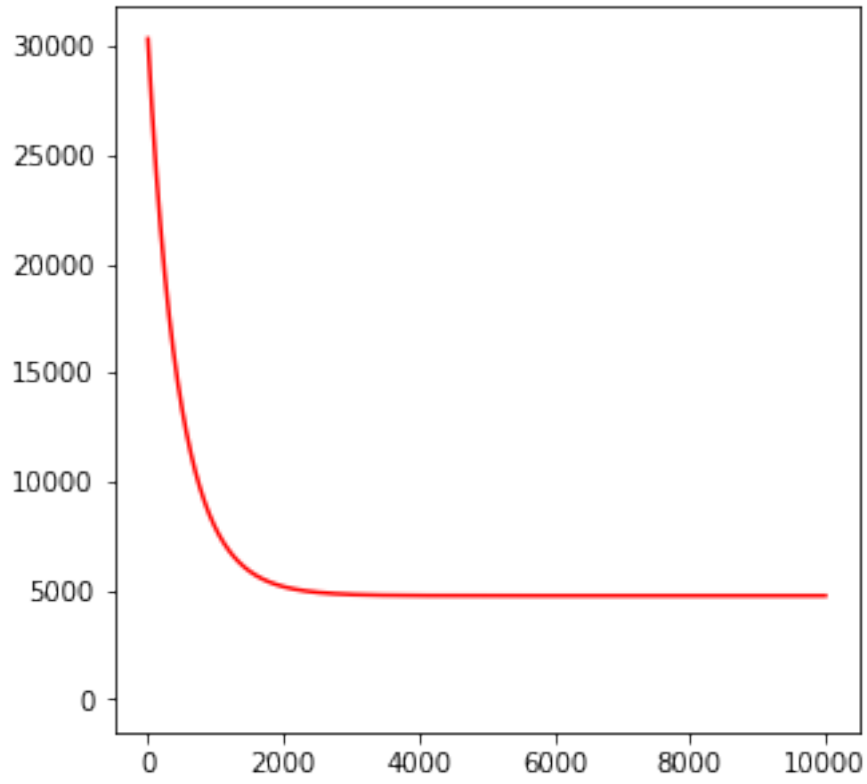
37.17172332 2.99690611 41.67478945 39.64609756 46.26240123 21.36248645

34.10134406 33.76017338 39.26643003

bias:-0.3457803888706665

Costs obtained

Iteration	Cost
0	30281.30786872626
1000	7856.193434199748
2000	5170.276428455614
3000	4811.903837699618
4000	4763.254189240338
5000	4756.624311480748
6000	4755.718982856415
7000	4755.595143680211
8000	4755.578175790414
9000	4755.575847195215



These are the scoring metrics and learning curve obtained.

Scoring metric	score
R2 Score	0.8424090736405616
MSE	9663.444050815566
MAE	78.26321844442849
RMSE	98.30281812245042

Chapter 2

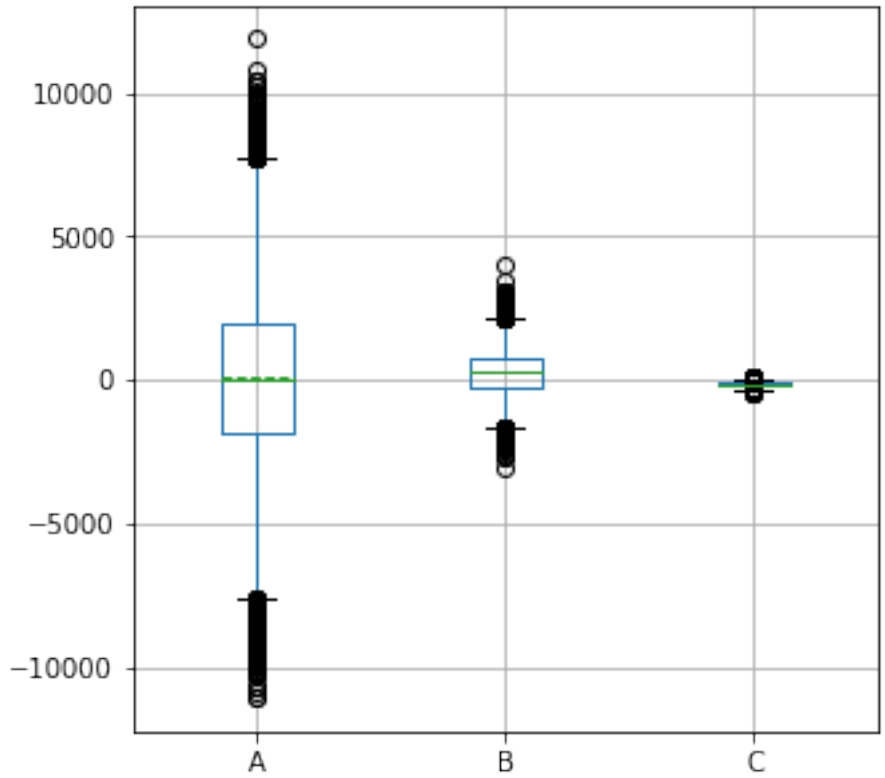
Polynomial Regression

Polynomial Regression is a simple Machine Learning algorithm that is used to predict a continuous target variable based on one or more predictor variables,by fitting a polynomial in the given dataset.

In this implementation gradient descent method is used to minimize sum of squared errors.

2.1 Data Analysis

I have used pandas,matplotlib to analyse the given data.



	A	B	C	label
count	50000.000000	50000.000000	50000.000000	5.000000e+04
mean	66.122302	253.565492	-149.676103	4.760874e+03
std	2851.432362	711.838429	71.174917	2.583795e+06
min	-11086.177958	-3065.807495	-454.476231	-7.994990e+07
25%	-1848.172530	-224.843278	-197.796603	-1.203886e+04
50%	60.054480	250.991294	-149.899035	1.565375e-02
75%	1982.912601	736.291355	-101.725114	1.520390e+04
max	11876.854852	3973.984169	136.969097	1.085096e+08

As we can see ranges of features vary,so feature scaling is done on it and a data split is performed on it using random split with ratio 0.7:0.15:15. into training cross validation and testing set.

2.2 Implementation details

I have created a class PolynomialRegression which takes hyperparameters as input and have built methods which trains the model on given dataset ,make predictions,calculate machine learning metrics,tune the hyperparameters. I have used the below formulas for this model.

Cost Function:-

$$J(\vec{\Theta}) = \frac{1}{2m}\sum(\hat{Y} - Y)^2 + \alpha_{reg} * \frac{\lambda_1}{2m} * \sum|\Theta_i| + (1 - \alpha_{reg}) * \frac{\lambda_2}{2m} * \sum|\Theta_i^2|$$

$$\implies J(\vec{\Theta}) = \frac{1}{2m}\sum(\vec{\Theta}.\vec{X} - \vec{Y})^2 + \alpha_{reg} * \frac{\lambda_1}{2m} * \sum|\Theta_i| + (1 - \alpha_{reg}) * \frac{\lambda_2}{2m} * \sum|\Theta_i^2| \quad (2.1)$$

Gradient Descent:-

$$\frac{\partial J}{\partial \vec{\Theta}} = \frac{1}{m}(\vec{X^T}).(\hat{Y} - Y) + \frac{\partial L_1}{\partial \vec{\Theta}} + \frac{\partial L_2}{\partial \vec{\Theta}} \quad (2.2)$$

$$\frac{\partial L_1}{\partial \vec{\Theta}} = \alpha_{reg} * \lambda_1 * sign(\vec{\Theta})$$

$$\frac{\partial L_2}{\partial \vec{\Theta}} = \alpha_{reg} * \lambda_2 * \vec{\Theta}$$

Updating parameters:-

$$\Theta := \Theta - \alpha * \frac{\partial J}{\partial \Theta} \quad (2.3)$$

Prediction:-

$$Y = \sum_{0<=a+b+c<=degree} \Theta_{a,b,c} X_1^a X_2^b X_3^c \quad (2.4)$$

2.3 Training and testing

2.3.1 regularization

L1 regularization, also known as Lasso regularization, adds a penalty term to the cost function that is proportional to the absolute value of the model weights. This tends to push the weights towards zero, effectively selecting only the most important features and making the model more interpretable.

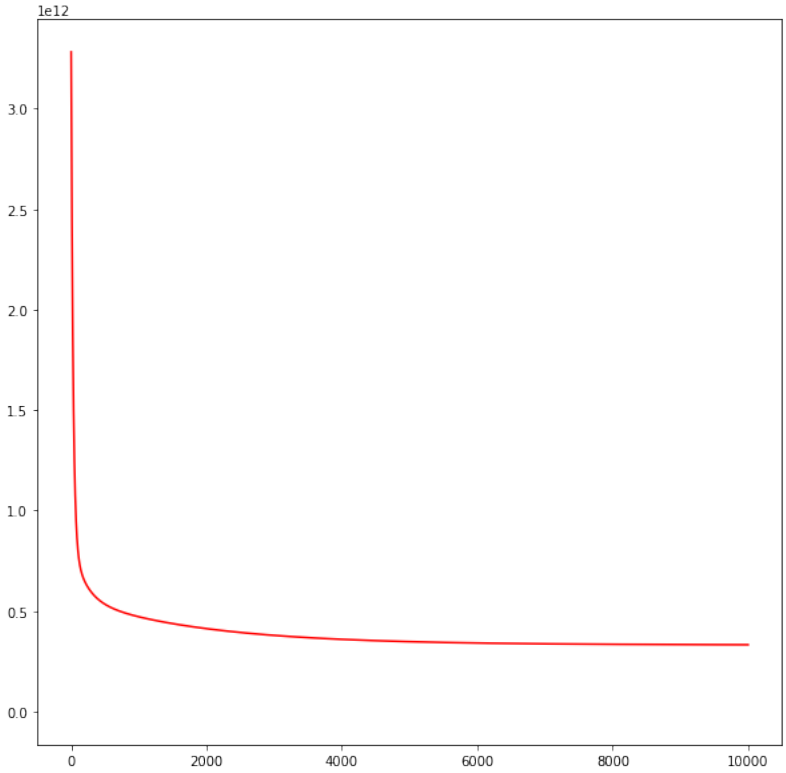
L2 regularization, also known as Ridge regularization, adds a penalty term to the cost function that is proportional to the square of the model weights. This tends to shrink the weights towards zero without forcing them to be exactly zero, and can help to reduce the variance of the model.

since the provided Dataset contains only three features,using Lasso regularization wont be effective,so I have only used Ridge regularization,by setting $\alpha_{reg} = 0$.

2.3.2 Training on random hyperparameters to check the working of model

Initially I have trained the model with degree=3 , $\lambda_2 = 0.02$ with 10,000 epochs

Epoch	Squared mean error
0	3281233432109.108
1000	472029956759.1296
2000	413071192578.57336
3000	379500429109.0748
4000	359832366954.13947
5000	348301548074.60785
6000	341539746073.17053
7000	337573819423.0871
8000	335247371617.16644
9000	333882480374.1356



2.3.3 Hyperparameter Tuning

In our model,the degree of the polynomial(*deg*),learning rate(α) , Ridge regression coefficient (λ_2) are the tunable hyperparameters,I have used random grid search to tune the model and used RMSE on the cross validation dataset as a metric to compare models.

After performing the tuning,I found the best hyperparameters of that execution:-

HyperParameter	value
degree	5
lambda2	0.05
learning rate	0.001

For the above hyperparameters RMSE score obtained was 81404.27842941943

whereas for other degrees,I obtained

2

best score:-2216658.805680621 for 'degree': 2, 'lambda2': 0.01, 'learning rate': 0.01

3

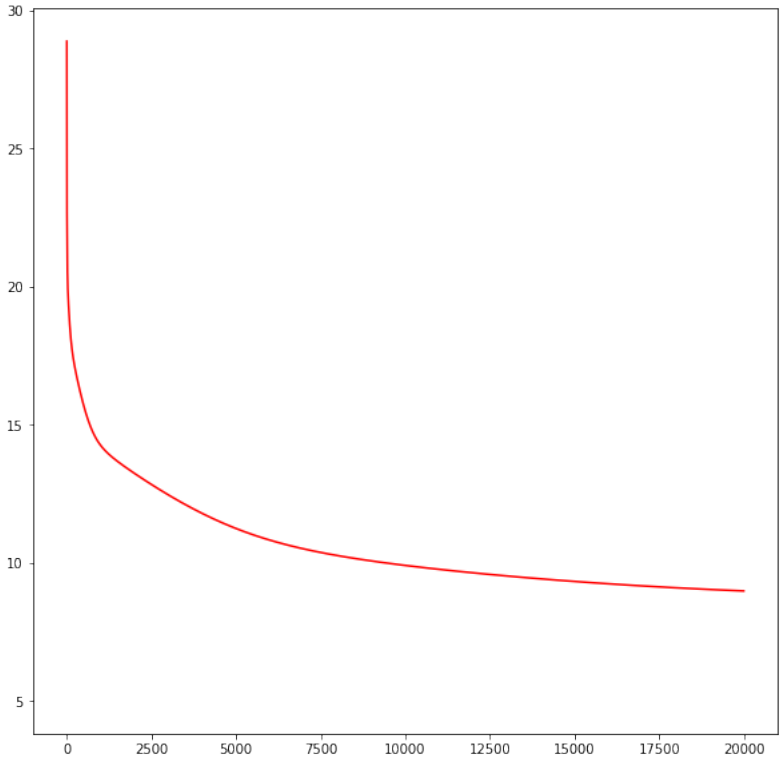
best score:-815733.4675912837 for 'degree': 3, 'lambda2': 0.05, 'learning rate': 0.01

4

best score:-817955.484230131 for 'degree': 4, 'lambda2': 0.05, 'learning rate': 0.01

Final Training and testing of the model with obtained hyperparameters

Learning Curve:-



Scoring Metrics:- These are the scoring metrics obtained on testing it against a hidden dataset

Metric	Score
R2	0.9992052501058277
MSE	4320624495.152336
MAE	14671.26591440149
RMSE	65731.45742452647

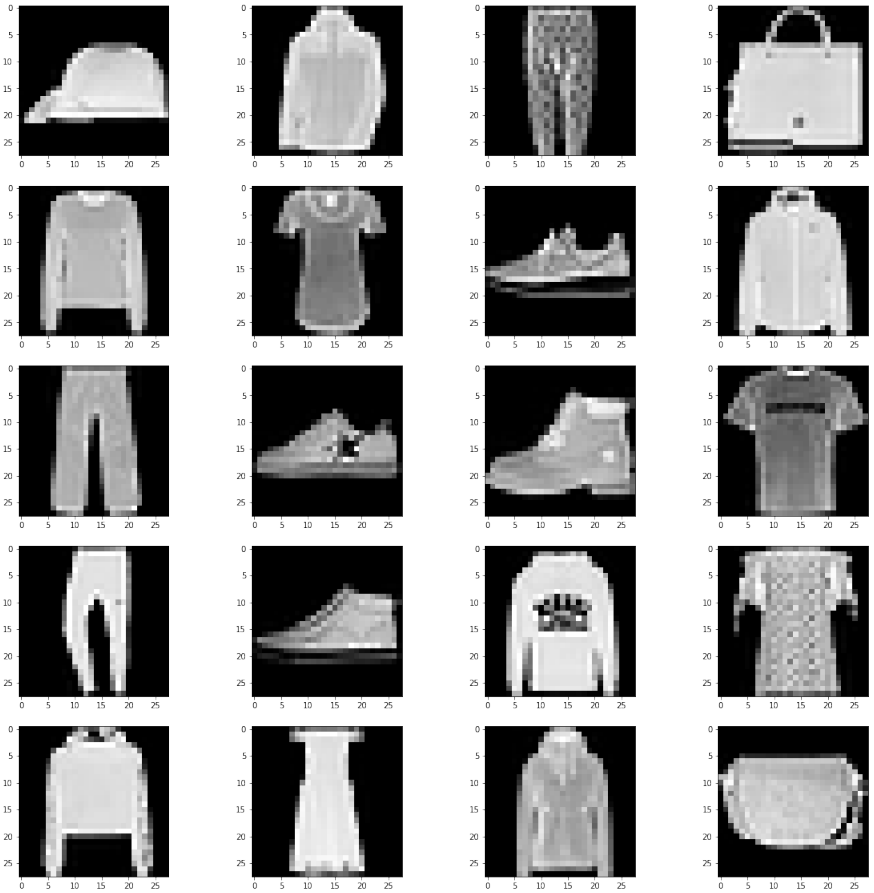
Note:- Higher degrees are not tested as it obvious that they lead to overflow,and there is high chance of overfitting,since degree 5 performed equally well on a hidden set,only upto degree 5 is considered.

Chapter 3

Fashion MNIST DATASET

Note:- The next three models were trained on this dataset,i.e Logistic Regression,N-layered Neural Network,K-nearest Neighbours

This dataset contained pixel values of multiple 28X28 greyscale images,and a labelled to 10 different classes.
i.e ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
few images are:-



I have performed feature scaling by dividing each pixel value with 256.

And Used Stratified split to divide the dataset into training,cross-validation and testing.In the ratio 60:20:20.

Chapter 4

Logistic Regression

Logistic regression is a Machine Learning algorithm,that is used to make predictions for classification problems based on one or more predictor variables.In this implementation,gradient descent method is used to minimize sparse categorical cross entropy function.
Note:- In the code,I have added regularization,but have not used as the model isnt overfitting in the first place,and it is effecting the performance.

4.1 Implementation Details

We will start by defining a class Logistic regression,which will have two methods fit and predict which perform the training and predictions respectively,and few other helper methods. The fit method will take in the training data,Xtrain and Ytrain,and use gradient descent to calculate the gradients using sparse categorical cross entropy function Sigmoid Function:-

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{4.1}$$

Softmax function:-

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K \tag{4.2}$$

Loss function:-

$$J = \sum_{c=1}^{M} y_{oc} \log(p_{o,c}) \tag{4.3}$$

where y_{oc} is a binary indicator whether this label is the correct classification or not, and $\log(p_{o,c})$ is the predicted probability that the example is this label.

Gradient Descent:-

$$\frac{\partial J}{\partial \Theta} = \frac{1}{m}(\vec{X^T}.(\hat{Y} - Y)) \tag{4.4}$$

Updating parameters:-

$$\Theta := \Theta - \frac{\partial J}{\partial \Theta} \tag{4.5}$$

Prediction:-

$$P\left(y^{(i)} = k \mid x^{(i)}; \theta\right) = \frac{\exp\left(\theta^{(k)\top} x^{(i)}\right)}{\sum_{j=1}^{K} \exp\left(\theta^{(j)\top} x^{(i)}\right)} \tag{4.6}$$

label with maximum probabilty will be the predicted label.

4.2 Training and testing

The model is trained against the train dataset,and following is observed.

Epoch	training loss	accuracy
0	2.6316582475330796	0.11788888888888889
100	1.424379857642282	0.544
200	1.124893755049132	0.6416666666666667
300	0.9900207397334976	0.6791666666666667
400	0.9087012783374866	0.6993333333333334
500	0.8524649117411917	0.7152777777777778
600	0.8104961922296295	0.7272222222222222
700	0.7776157129653198	0.7384444444444445
800	0.7509598110693144	0.7462777777777778
900	0.7287874508164198	0.753

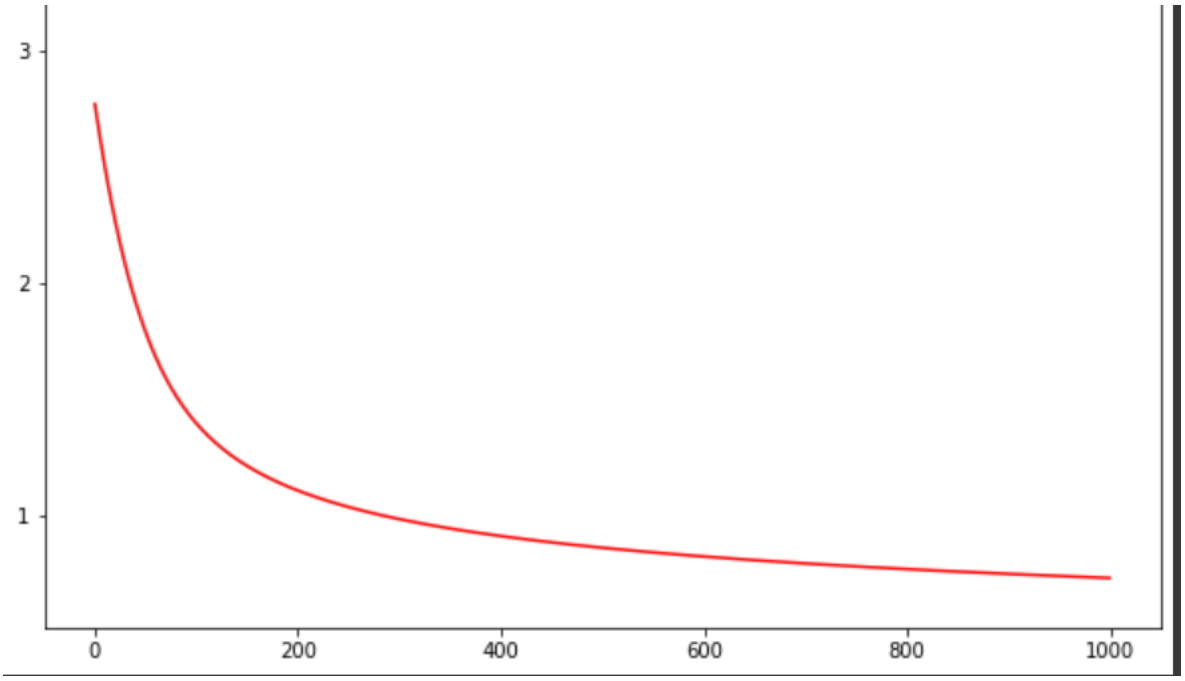


Figure 4.1: Learning curve obtained

Figure 4.2: few of the misclassified images

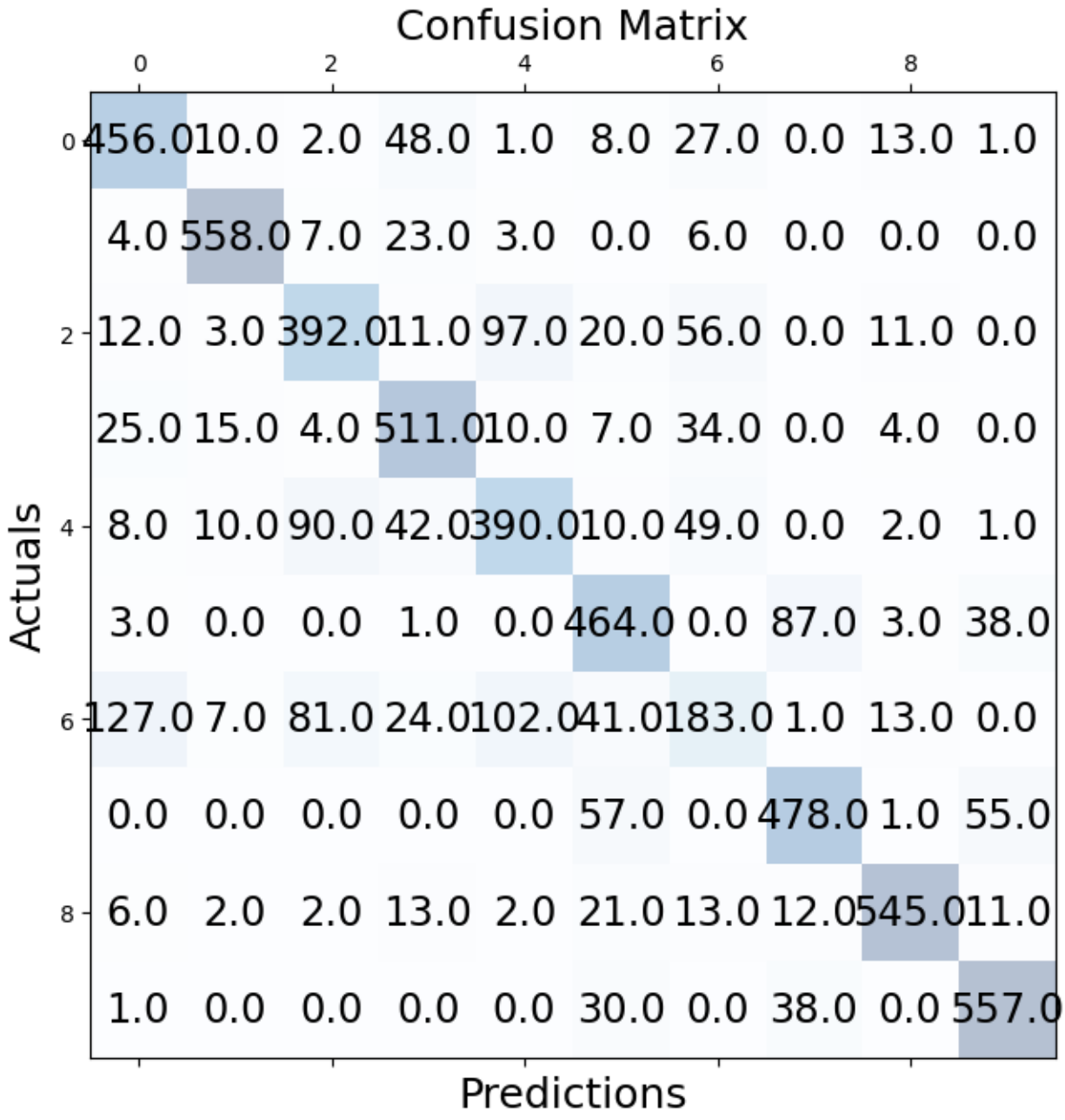


Figure 4.3: Confusion matrix on the hidden Test data from train dataset

Chapter 5

L-layered Neural Network

Neural Network is a Machine learning model,which give a way of defining non-linear,complex form of hypothesis, $h_{W,b}(\vec{X})$ with parameters W, b that we can fit to our data.It contains multiple layers with neurons.
Each neuron is a simple computational unit which takes input from previous layers and a intercept term(bias) and a activation function is applied on it.We can see that it corresponds exactly to a input-output mapping defined by logistic regression.

5.1 Implementation details

I have created a class Neural network which takes layer sizes(Architecture of the model) and activation function.
This model uses mini batch gradient descent with ADAM optimization,sparse categorical cross entropy as the loss function.
I have created train method which performs the training of the model which uses two other methods forward propagation and backward propagation,also a predict method which makes the predictions.

During training,first the forward propagation method is called,which computes activations of each neuron for the given weights and biases,and using the output of the final layer,backward propagation method is called,which computes the gradients,which are used to update the parameters and this gets repeated for all batches.

The model will contain first layer of 784 neurons which is the input layer,and with hidden layers and a softmax output layer of 10 neurons which show the probabilty of the given example belonging to each class.

FORWARD PROPAGATION

$$Z^{(l)} = W^{(l)}.A^{(l-1)} + B^{(l)} \tag{5.1}$$

$$A^{(l)} = f(Z^{(l)}) \tag{5.2}$$

where $W^{(l)}, B^{(l)}$ are the weights and bias of the current layer, $A^{(l-1)}$ is the activations of the previous layer, and f is the activation function.

BACKWARD PROPAGATION

Gradient of the output layer

$$\delta\hat{Y} = \hat{Y} - Y \tag{5.3}$$

As we go back from current layer to previous layer,with the activations of the current layer,and gradient of the activations of the current layer,we can find out the gradients of the weights,bias and the activations of the previous layer.

$$\frac{dA^{(l+1)}}{dW^{(l)}} = \delta A^{(l+1)} * f'(Z^{(l+1)}).A^{(l)} \tag{5.4}$$

$$\frac{dA^{(l+1)}}{dB^{(l)}} = \delta A^{(l+1)} * f'(Z^{(l+1)}) \tag{5.5}$$

$$\delta A^{(l)} = \frac{dA^{(l+1)}}{dA^{(l)}} = W^{(l)T}.(\delta A^{(l+1)} * f'(A^{(l+1)})) \tag{5.6}$$

where f' is the derivative of the activation function.

After finding out the gradients,I have used ADAM optimization on the mini batch gradient descent,with amsgrad.

$$v_t = \beta_1 * v_{t-1} + (1 - \beta_1) * (\frac{\delta L}{\delta w_t}) \tag{5.7}$$

$$s_t = \beta_2 * s_{t-1} + (1 - \beta_2) * (\frac{\delta L}{\delta w_t})^2 \tag{5.8}$$

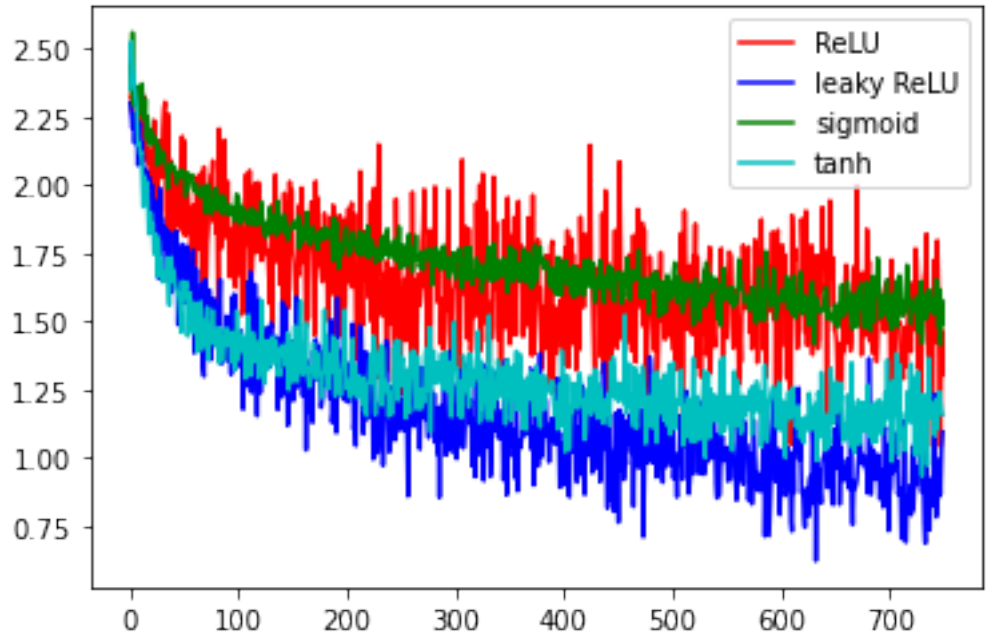
$$\hat{s}_t = max(\hat{s}_t, s_t) \tag{5.9}$$

where v_t is the first moments vector,s_t is the second moment vector,$\hat{s}_t$ is the maximum of previous and current second moment vector.t is the iteration.

5.2 Training and testing

5.2.1 Choosing activation function

I have taken a simple architecture with a input layer,a hidden layer with 128 neurons,and an output layer,check the performance of different activation functions.



as we can see leaky ReLU performed better than other activation functions.

5.2.2 Single Hidden Layer

I have trained a neural network architecture with a input layer,a hidden layer with 128 neurons,and an output layer.

After 100 epochs,with a learning rate of 0.0001 I have obtained an accuracy of 79.1% on hidden test dataset.

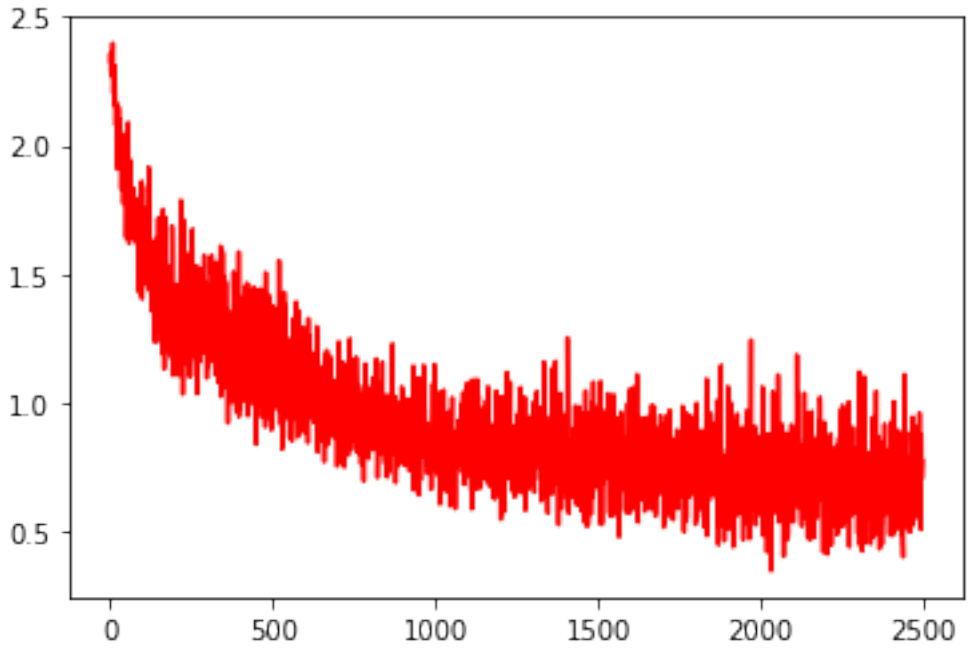


Figure 5.1: Learning curve obtained

The spikes are due to the usage of mini batch gradient descent.

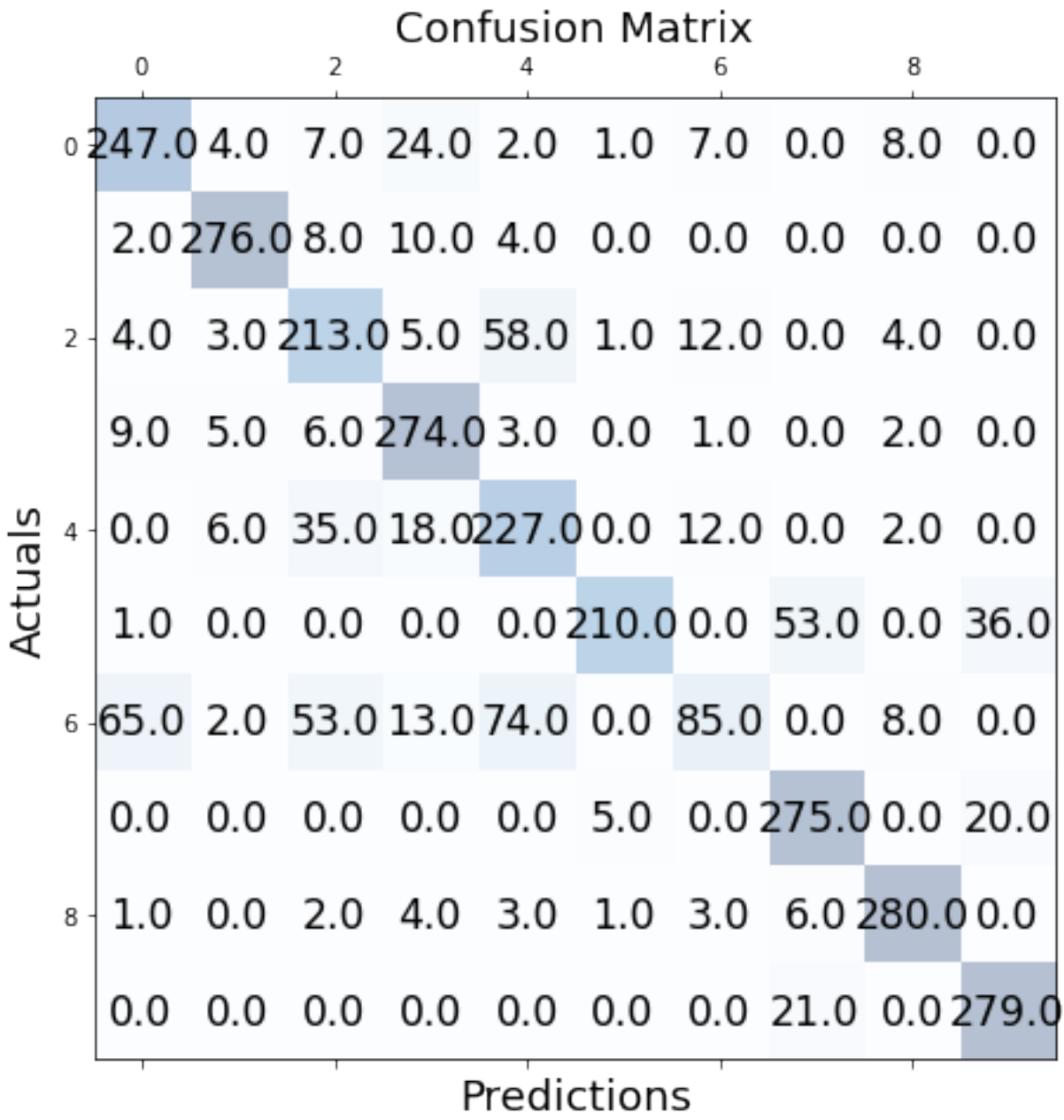


Figure 5.2: Confusion matrix obtained

5.2.3 Double Hidden Layer

I have trained a neural network architecture with a input layer,a hidden layer with 128 neurons,and an output layer.

After 100 epochs,with a learning rate of 0.0001 I have obtained an accuracy of 80.7% on hidden test dataset.

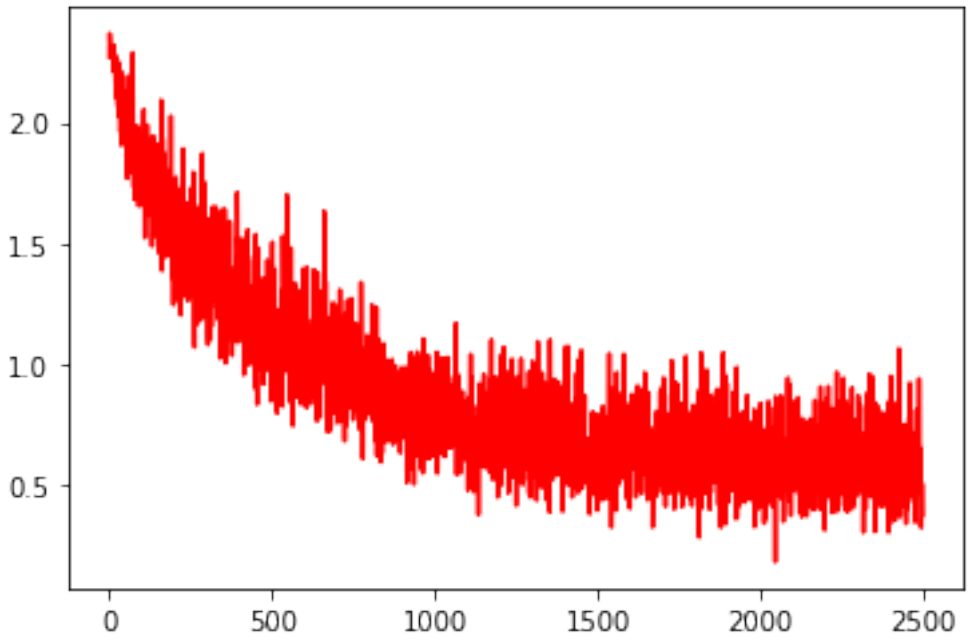


Figure 5.3: Learning Curve Obtained

5.2.4 Three Hidden Layer Neural Network

For three hidden layers I obtained an accuracy of 81.53% .

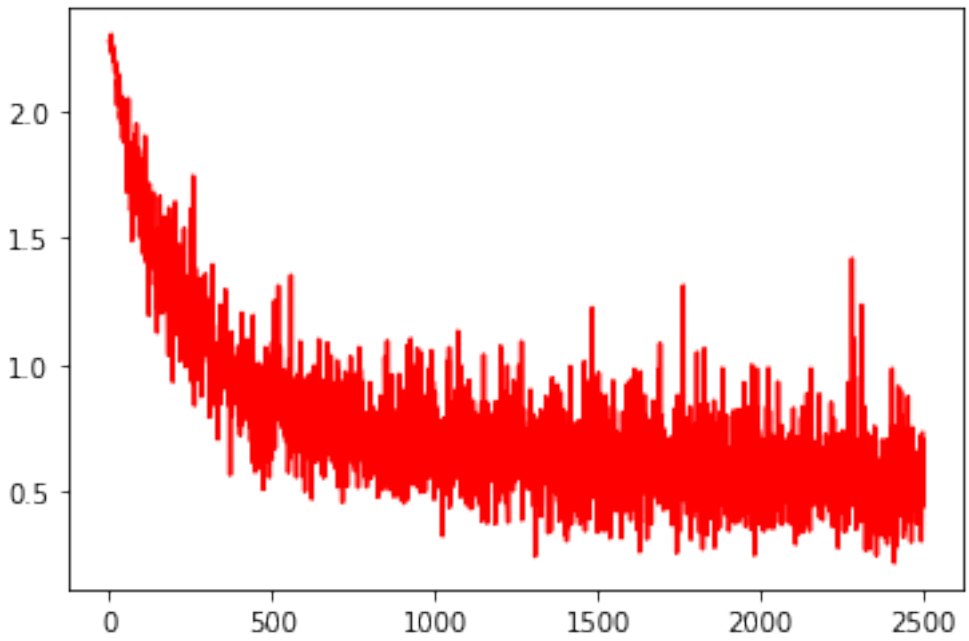


Figure 5.4

5.2.5 Four hidden layer Neural Network

For four hidden layers with a learning rate of 0.001,I have obtained an accuracy of 85.6%

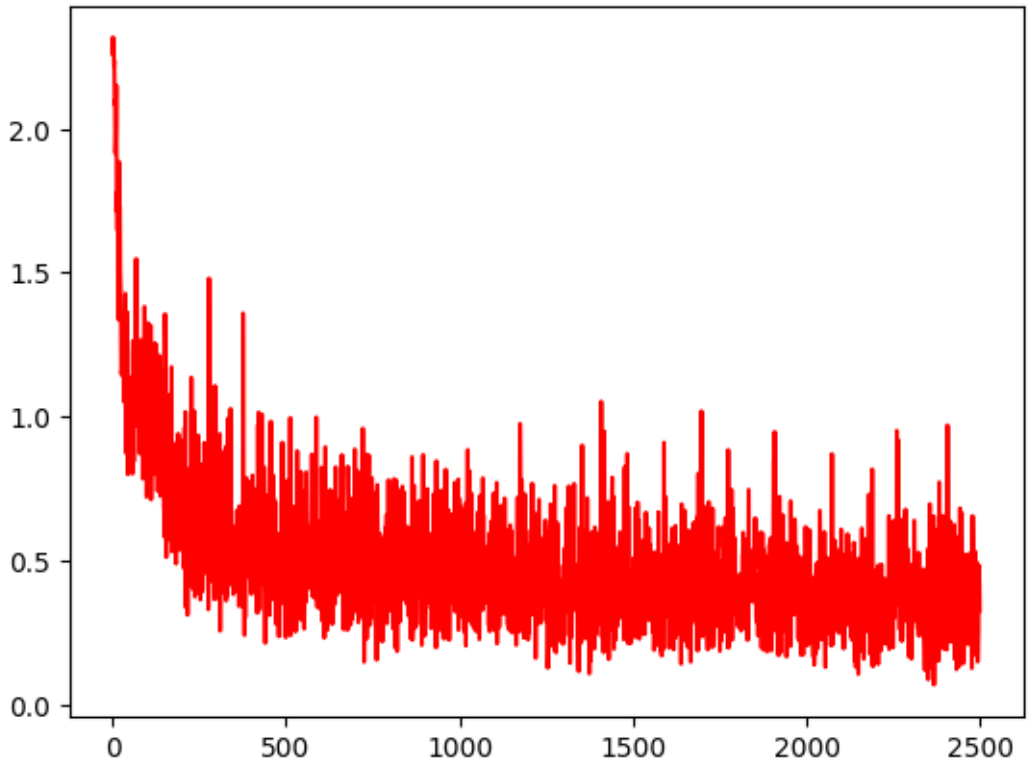


Figure 5.5

Observation:- with higher learning rate or deeper networks,the performance gets reduced or slightly increases but it never crossed 86% so 4 hidden layers seems to be the best fit for the dataset.

5.3 Final Performance Evaluation

On the test dataset the model obtained an accuracy of 85.53% .

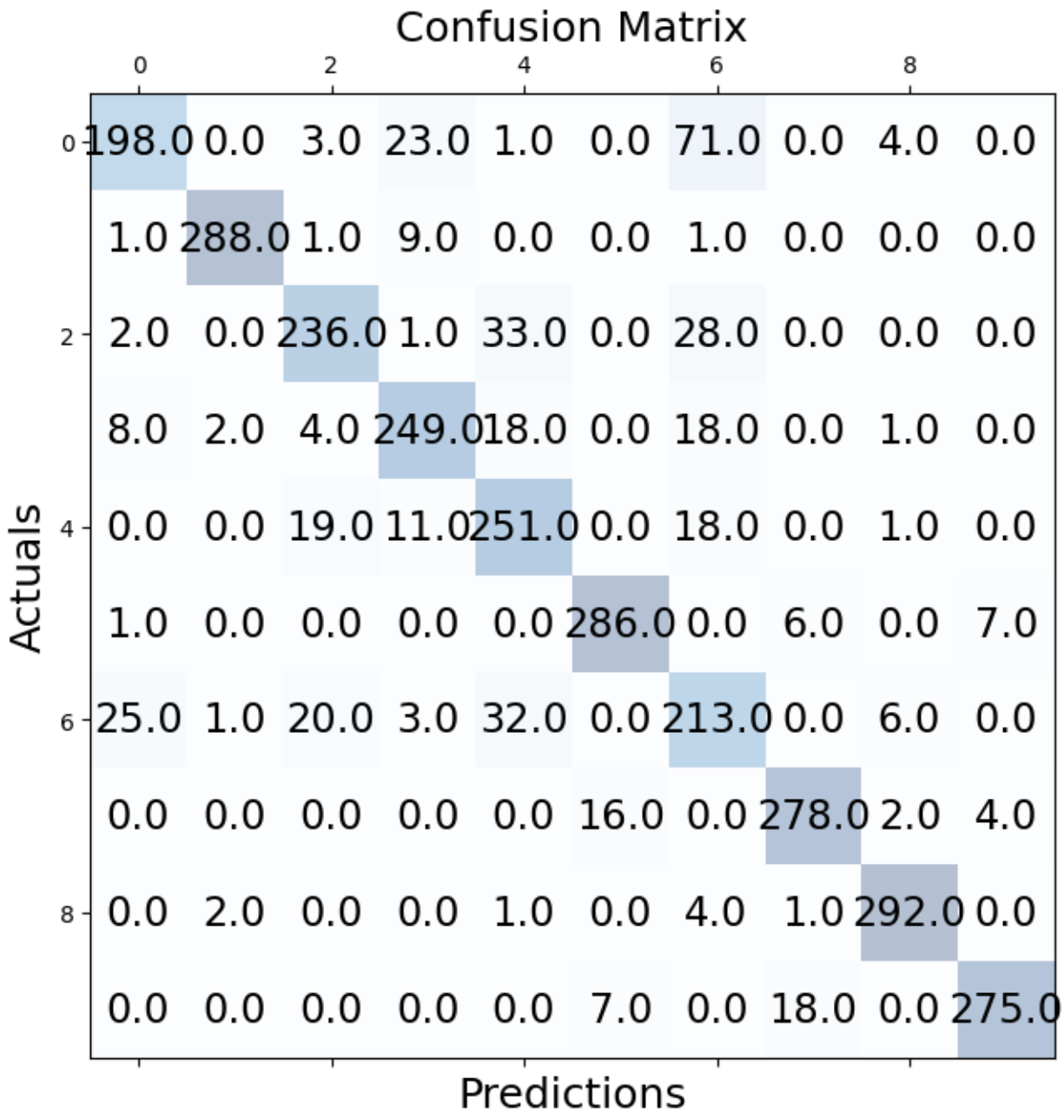


Figure 5.6: Confusion matrix obtained

Chapter 6

K-nearest Neighbours

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.
For classification problems, a class label is assigned on the basis of a majority vote—i.e. the label that is most frequently represented around a given data point is used.

For each datapoint,the k-nearest neighbours are observed and the most frequent class will be assigned to that datapoint.

6.1 Implementation Details

I have created a class KNearestNeighbours Which takes the parameters as distance formula i.e Euclidean or Manhattan.

for Euclidean I have used vectorization to make the model classify fast.

6.2 Testing

I have tested the model on the cross validation dataset with accuracy as the scoring metric to find the best K value,with manhattan distance.

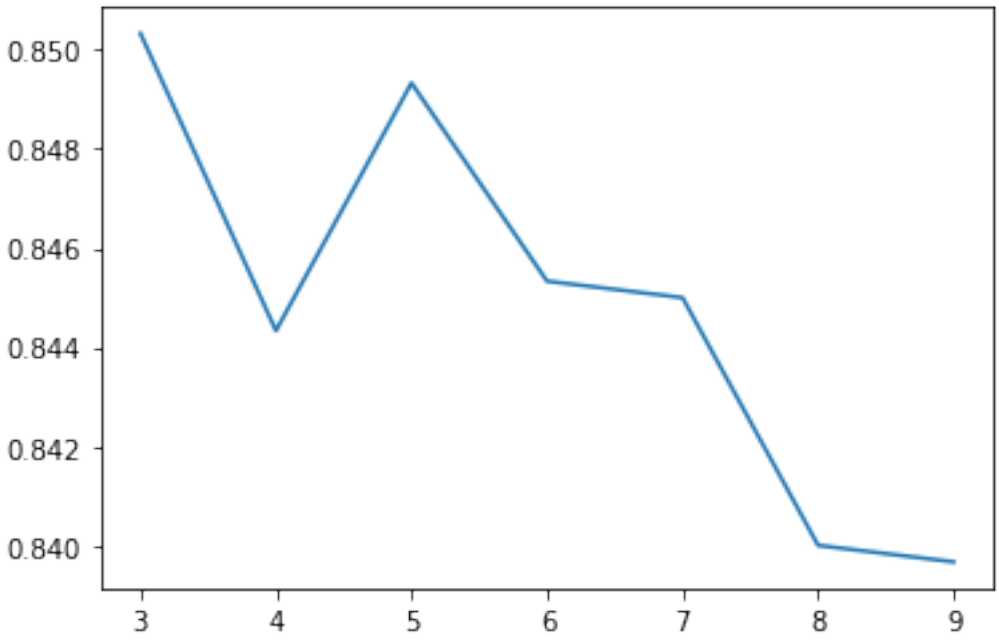


Figure 6.1: As we can see 5 is the optimum K since 3 leads to high bias model

Using Euclidian Distance:-

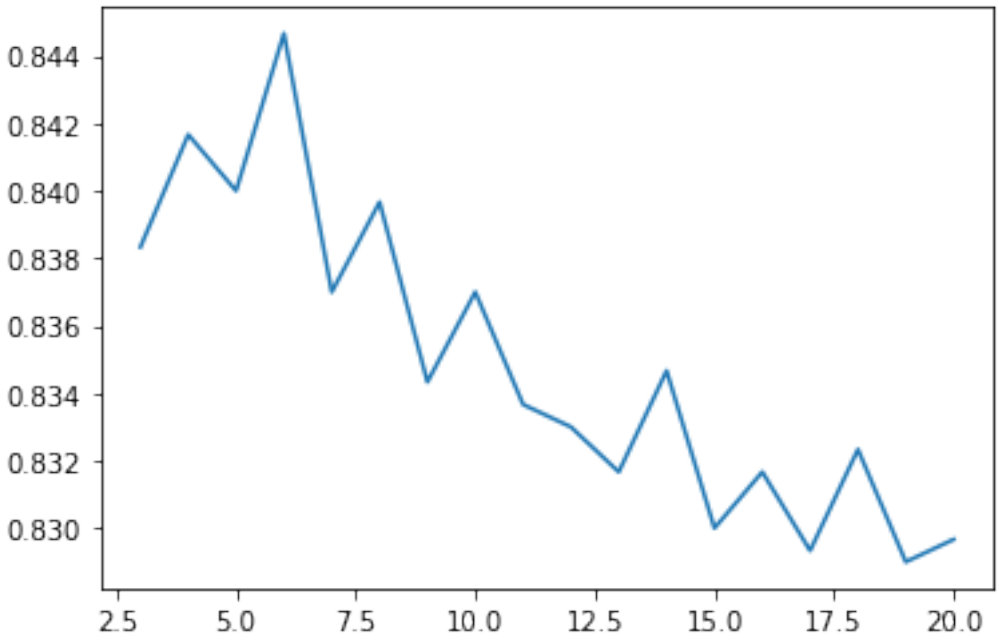


Figure 6.2: As we can see 6 is the optimum K

Since Euclidian Distance seems to be a better Distance function than Manhattan,the final model will be with Euclidian distance and K=6,

This model gave an accuracy of 83.73% on hidden dataset.