ML BOOTCAMP

IIT ISM DHANBAD

By :- Samyak Jha

REPORT

Introduction

The aim of the project was to implement all the Linear Regression, Polynomial Regression , Logistic Regression for multiclass Classification , KNN for Classification and a

n- layer neural network from scratch .

Linear Regression:-

The main idea was to fit a straight line as a estimation of the datasets

An overview of what I did was to define a linear function and try to find its gradients at any particular dataset and then apply gradient descent.I think one of the most important steps was to **Normalize the X dataset**(I applied the **Z - score normalization** to be specific)because if the mean of dataset is zero and std deviation is 1 all the features are in similar ranges of values it will cause **gradient descent to run faster** , why is that the case , because if there is very big variation in sizes of ranges of features then attempts to take steps towards the minimum will cause **oscillation of gradients** since we are taking a same learning rate for both of the features and hence gradient descent would need a lot more iterations to reach the minimum , now if you try and reduce the learning rate to reduce the oscillations again that might reduce the oscillations but the model would take a lot more iterations to be trained.

So as i implemented it using for loops i.e i didn't leverage the existence of broadcasting and **ran a nested loop with n*m iterations** where n is number of features and m is number of training examples.

```
for j in range (0,n):

  for i in range (0,m):

     dj_dw[0][j] +=(np.dot(self.w,X[i])+self.b-Y[i][0])*X[i][j]

      dj_dw[0][j] /= m
```

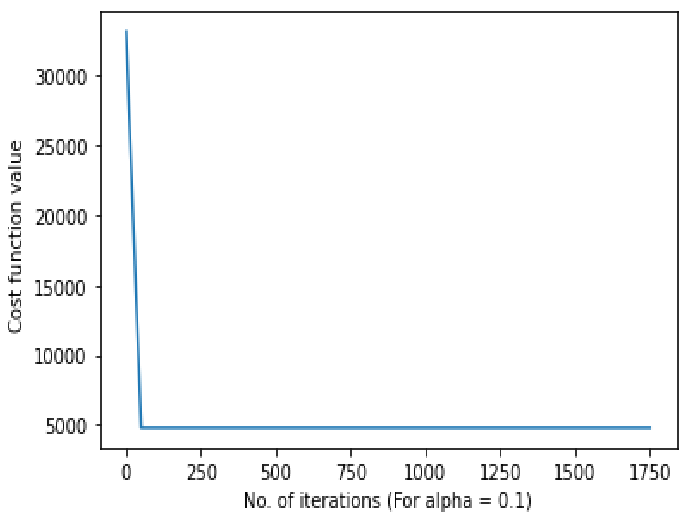
(Note why the above loop consumed more time than others because it is doing the step of addition n*m times which is the most in the entire code and hence this for loop is highlighted)

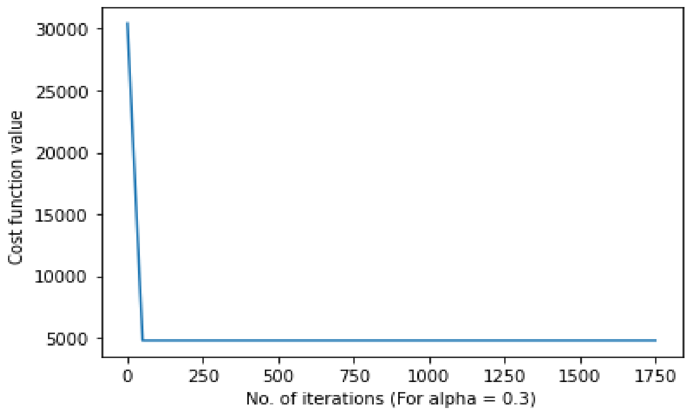
```
for i in range (0,m):

   Z = (np.dot(self.w,X[i])+self.b-Y[i][0])

   dj_db +=Z

dj_db /= m
```

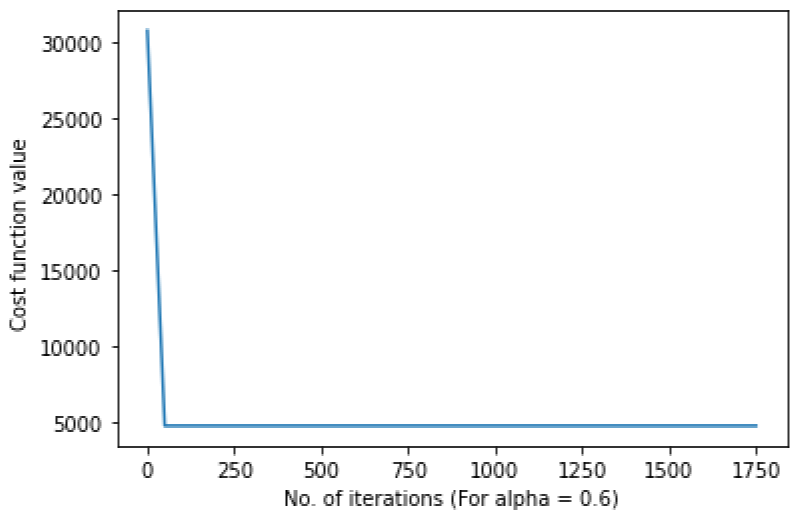
Then I learnt more about broadcasting and how it can be helpful in vectorizing your code.

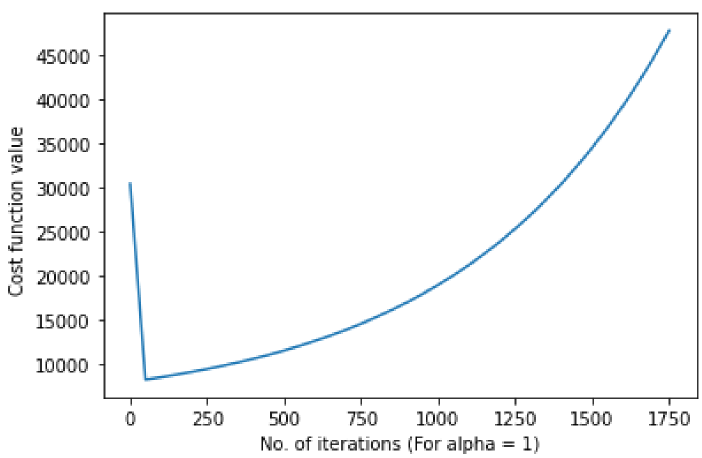
```
dj_dw = np.dot((np.dot(self.w,X.transpose())+self.b-Y.reshape(1,-1)),X)
dj_dw /= m
dj_db =
np.sum((np.dot(self.w,X.transpose()) + self.b-Y.reshape(1,-1)),axis =1)
dj_db /= m
```

Then i arrived at picking up **the right choice for alpha**









Hence the **optimal choice of alpha was 0.6** for me

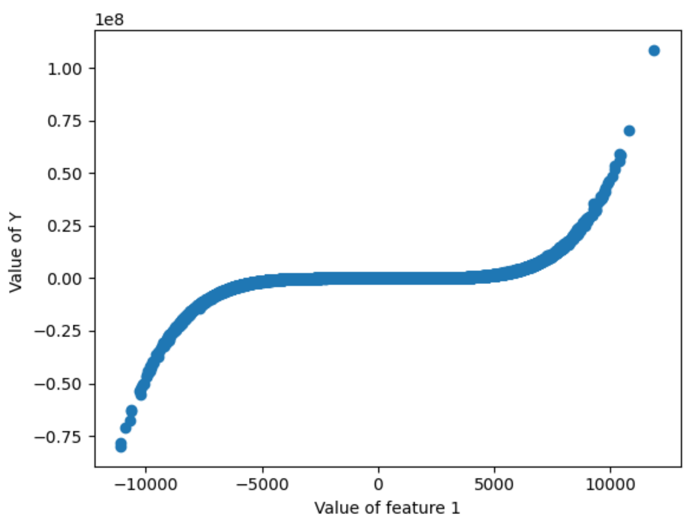
And i divided the dataset into 45000 examples as training set and 5000 examples as cross validation set and the model had an

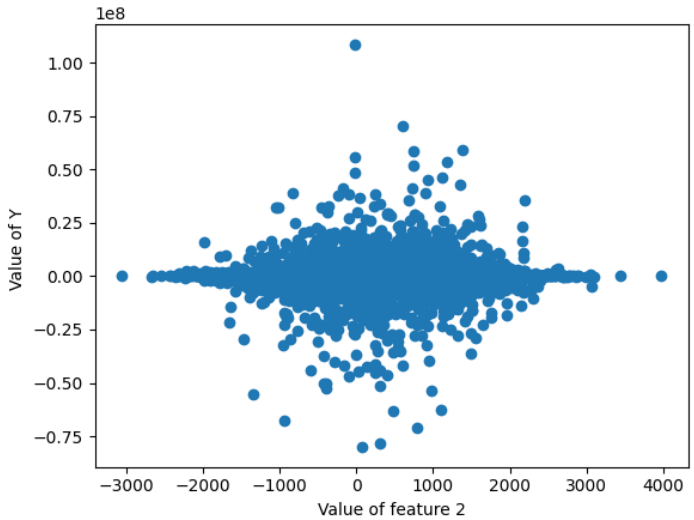
R2_score of 0.8443047602

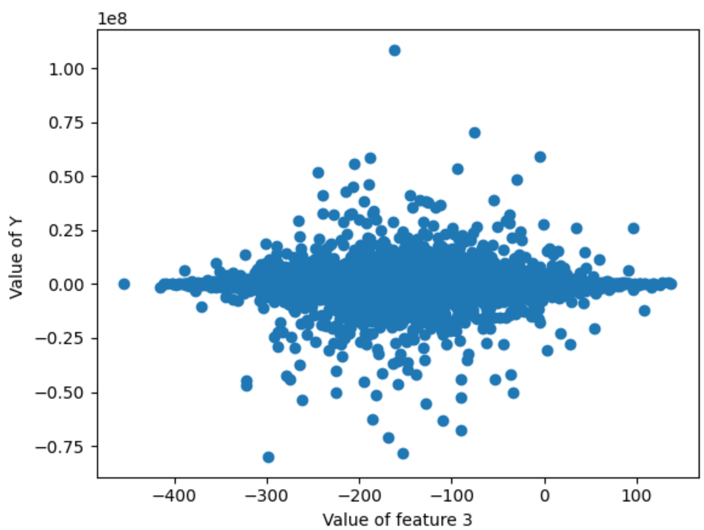
Saturation value of Cost was 4779.97042004

Polynomial Regression:-

For a n degree polynomial the first question was of **selecting the degree** so i tried to plot the datasets,







As i saw these three graphs i observed that plot of value of first feature vs the value of y i saw that it **is a graph of an odd function** so this gave some ideas of on taking only odd degree terms in the polynomial model , the terms were mixed terms i.e if i say that the maximum degree of my polynomial model is n then all the terms less than or equal to n will be included even mixed terms , now why this would work is because if any term itself is insignificant for the given dataset , it's weight i.e a measure of importance of that particular term would be and when trained it will eventually itself will be approaching to zero if it is that insignificant, sure it will be computational slower but will give better results.

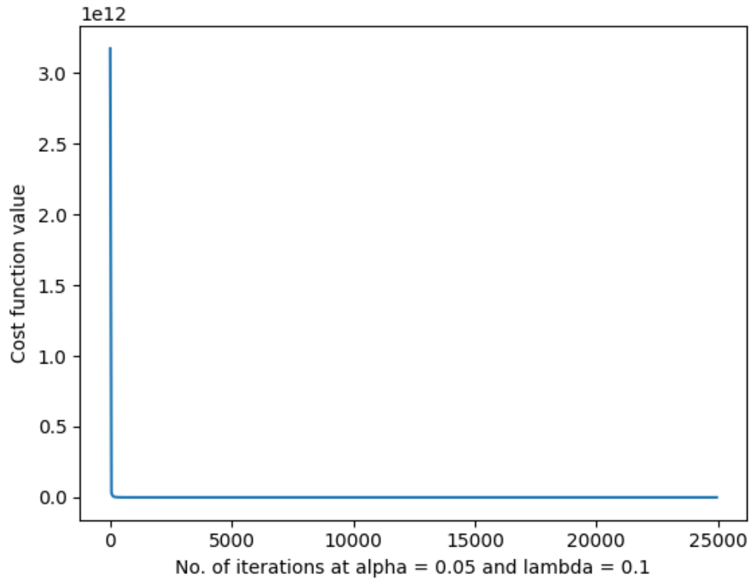
Since there is always a possibility of overfitting I did introduce regularization in my model but it wasn't that needed in this dataset but I did keep it in my model for sake of generalization and **used a common value for lambda i.e 0.1.**

About the learning rate i took the learning rate as 0.1 and what i saw was there was just too much oscillations and in the end cost increased so much that there was overflow even in taking z = z**2 .

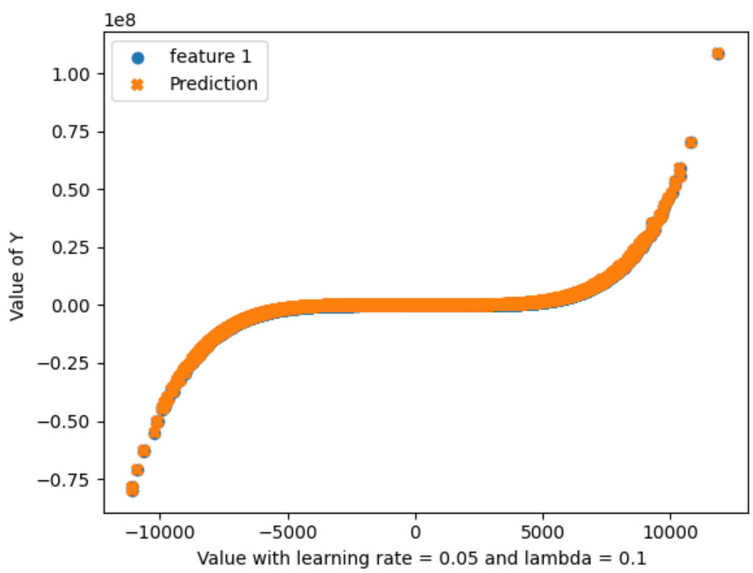
And then reduced my **learning rate to 0.05** and my curve was cost was decreasing smoothly another my data was able to get **an R2_Score as 1 i,e cost function was tending to be zero** on a cross validation set which i spilt by dividing the dataset with 45000 training examples as training set and 5000 training examples as cross validation set

But I saw that if I decrease **lambda** there is no overflow error and oscillations of gradients , and since the model didn't show signs of high variance and overfitting I reduced **lambda = 0** which caused the gradient descent to work properly even at **a learning rate of 0.1.**

The graphs that follow are what I got using the two models.

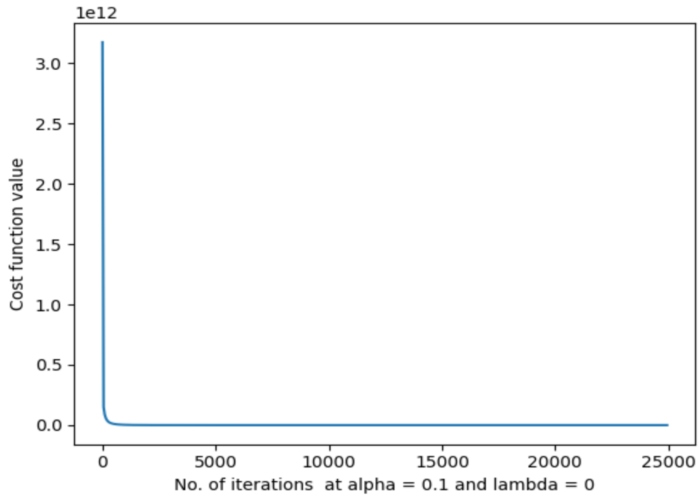


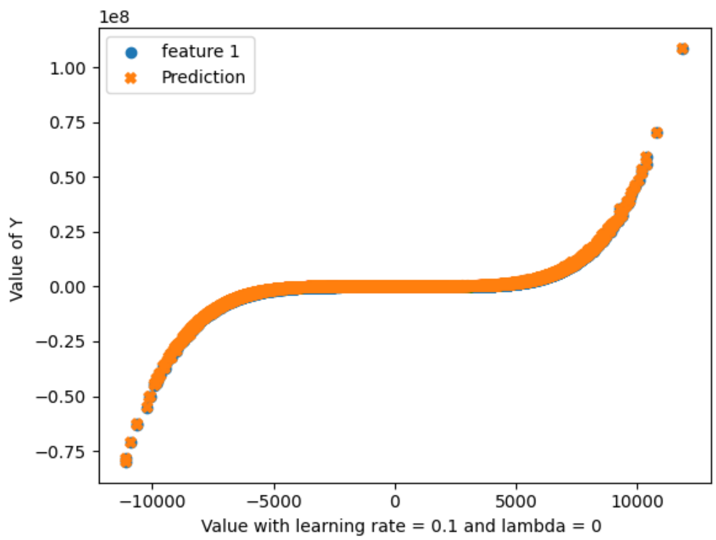
1e12
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Cost function value
0
5000
10000
15000
20000
25000
No. of iterations at alpha = 0.05 and lambda = 0.1



1e8
feature 1
Prediction
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
Value of Y
−10000
−5000
0
5000
10000
Value with learning rate = 0.05 and lambda = 0.1

•





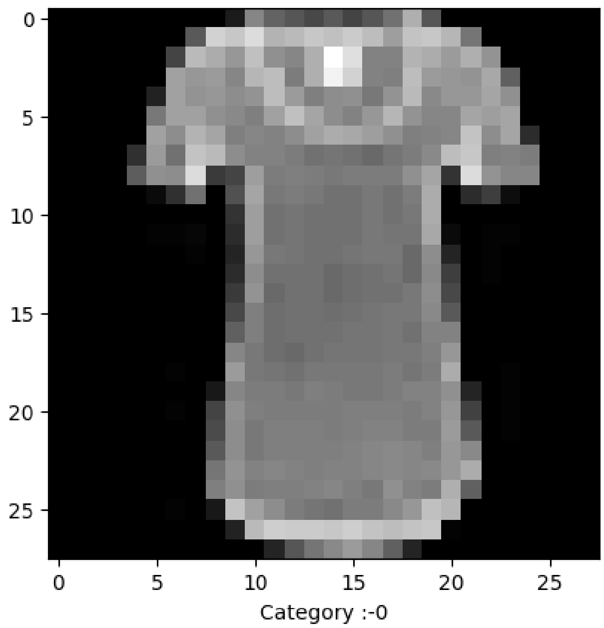
Though the R2_score of learning rate = 0.1 and lambda = 0 was close to 1 it was exactly equal to **0.9999999895** . So I ultimately decided to go with a **Learning rate = 0.01 and lambda = 0.1** for my model.

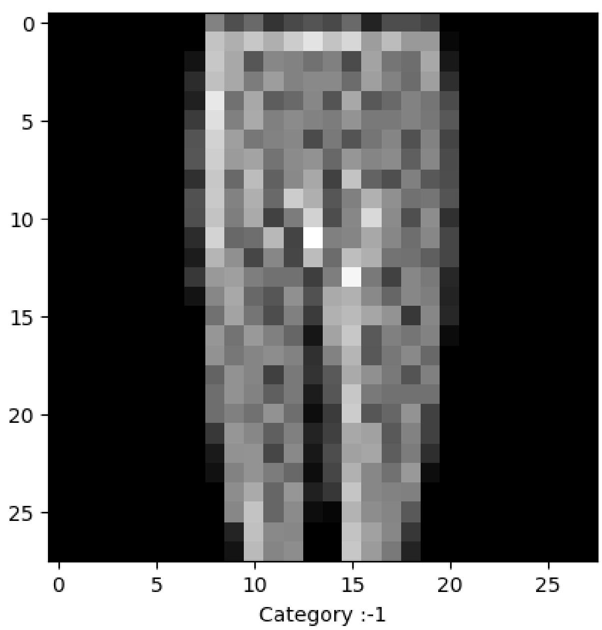
Now another thought that comes to mind while looking at the other two plots and even the first plot of value of Y vs the features is that **The function Y is mostly dependent on the first feature and almost independent of the second and third feature.**

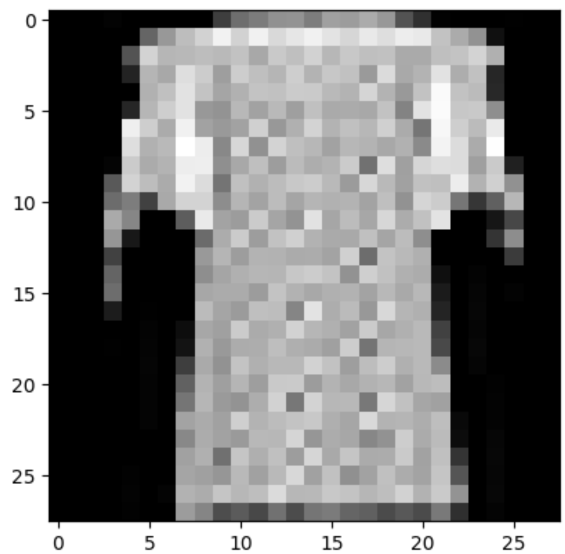
So I tried to test this hypothesis and removed the values of other two features from the input feature array, and trained my model on that and i found out that the model was getting **an R2_score of 0.999140892** , now why do i think this is so important to highlight because removing the other features made my model to be able to train itself like 100 times faster and even saturate quickly.Although i do agree this happened because this was a very special case with the data but i think it is notable mention to be made.

CLASSIFICATION :-

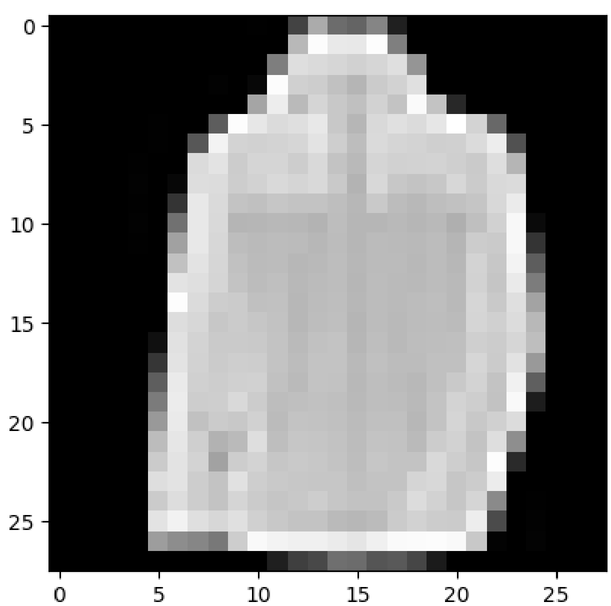
We can plot the Dataset given and the categories(About how did i form categories was via using np.unique function) we have which are as follows :-


0
5
10
15
20
25
0
5
10
15
20
25
Category :-0

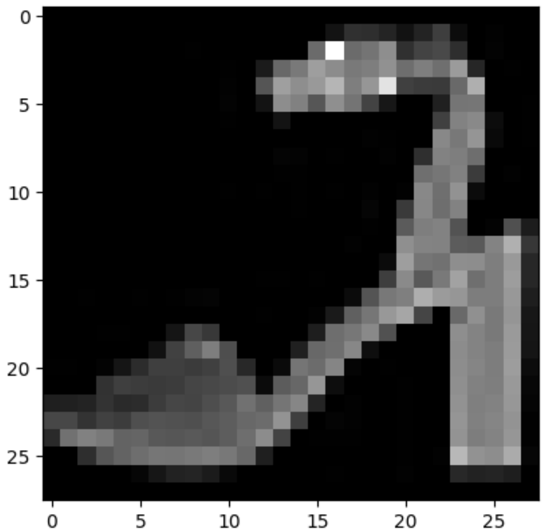

0
5
10
15
20
25
0
5
10
15
20
25
Category :-1



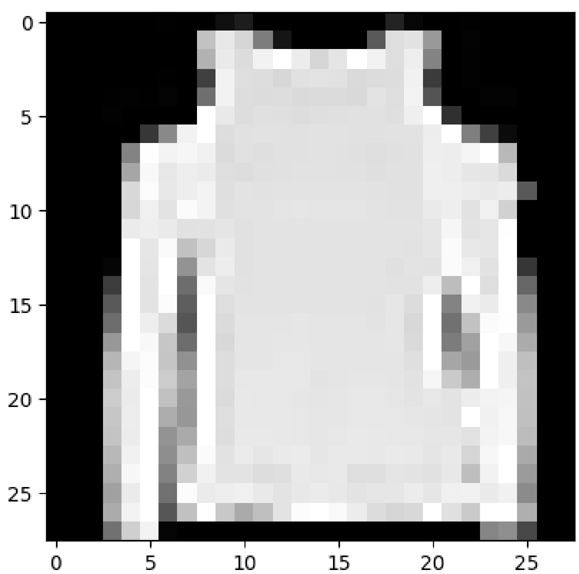
Category :-3



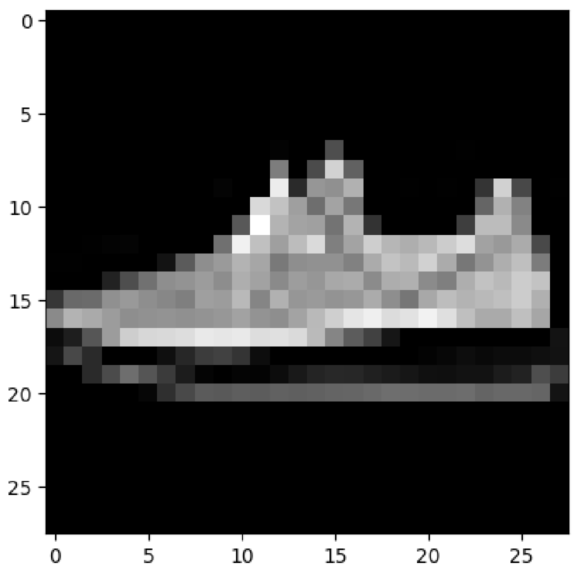
Category :-4



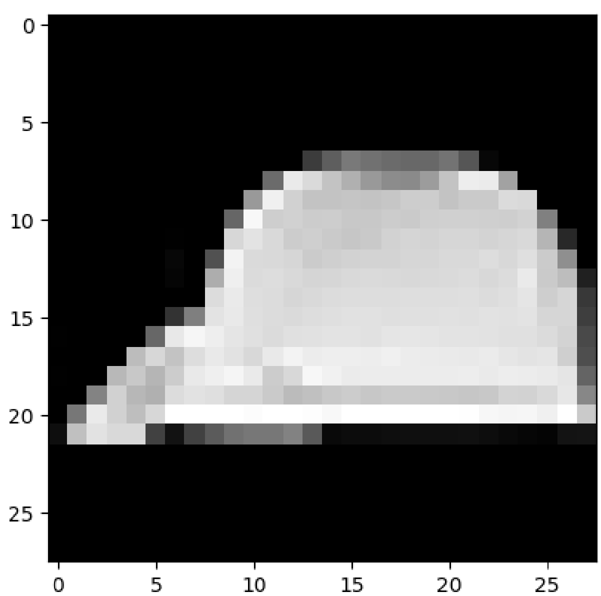
Category :-5



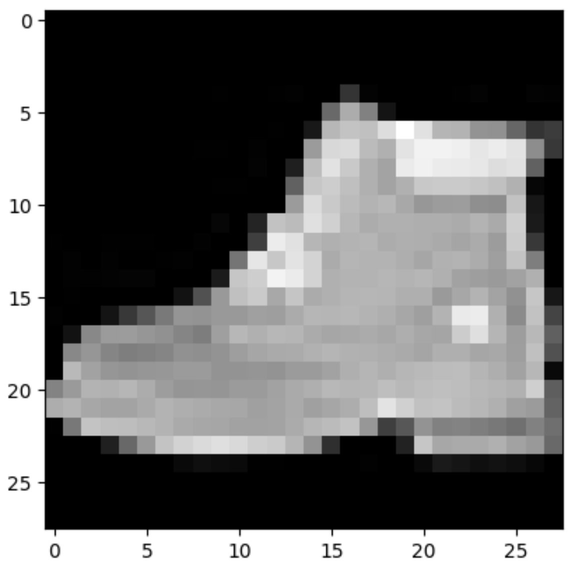
Category :-6



Category :-7



Category :-8



Category :-9

So we had to create a Classifier using logistic regression, KNN and Neural Network.

LOGISTIC REGRESSION :-

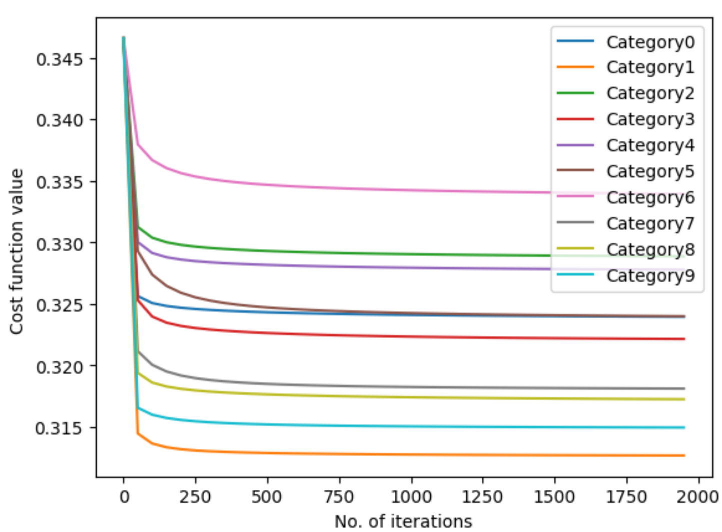
Since I had learned about Logistic regression for Binary classification but the dataset was of a multiclass classification so my initial thought was to use softmax activation function but logistic function means use of sigmoid and so I found we had to use one vs all algorithm , and since i know how to make a binary classification model of logistic regression i tried to convert Y which contained values at category value into a matrix with 10 copies of initial matrix but the difference was that this matrix had all the elements as 0 other than the row value of the category and column value of the training value we are talking about which are assigned as 1 this was done using np.where and np.repeat

```
Y_new = np.repeat(Y,10,axis = 0)

A = np.array([[0],[1],[2],[3],[4],[5],[6],[7],[8],[9]])

Y_new = np.where(Y==A,1,0)
```

Now the rest of the code is the same as linear regression other than the fact i had to use sigmoid to find output and this represented the probability of a category being the actual category and for each training example found out the category for which this probability is maximum.

Honestly I didn't feel the need to introduce a regularization parameter for this dataset so that isn't included so my learning rate was the only hyperparameter **. Initial learning rate was set as 0.05.**

But that caused error of overflow in finding sigmoid value and loss value as that cause too much oscillating gradients and thereby making Z very small and causing such overflow and so then I decided to take **learning rate as 0.03** and encountered the same issue so i took the **learning rate as 0.01** and my cost was decreasing properly . The following is the plot of loss function of the model wrt each of the 10 categories :-



I divided the training data and cross validation data into a ratio of 1:5 and got an **accuracy of 82.02 on the cross validation set** .

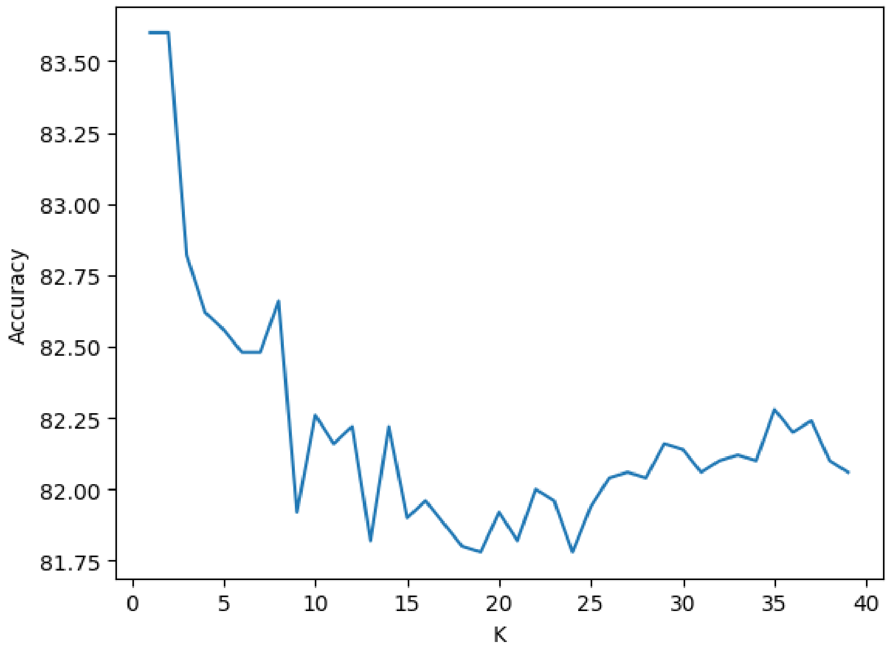
KNN :-

So main idea of KNN is that any new dataset will have same category as the most common among it's "K" nearest neighbors where K is a hyperparameter

So the first hurdle was to find the distances of points in training examples and the cross validation examples (which I divided in ratio of 1:5) although using loop was the initial attempt but I learned my lesson of not vectorising from linear regression so I tried to vectorize the finding distances completely.

And leveraged the fact that **(a-b)**2 is a**2 + b**2 - 2*a*b** to find distances between points.

Since we don't have to train this model **the only choice of hyperparameter here was the value of K** . How i found that out was that i plotted a graph of accuracy on the cross validation set vs value of K for values of K = 1 to 40



The above graph gave the idea that highest value of **accuracy on Cross validation set is 83.6** and it happened at both K =1 and K = 2 , but i **choose K = 1.**

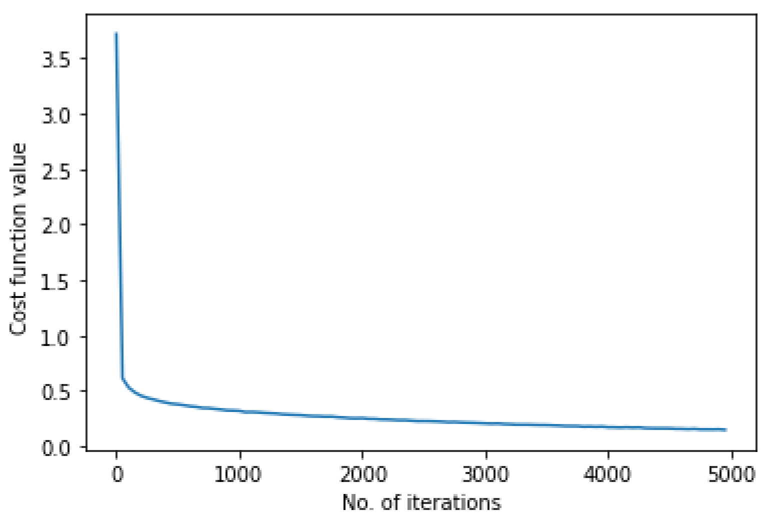
Neural Network

So I began by implementing forward propagation and backward propagation and implementing Batch gradient descent to reduce the lost function and reaching the minima of the loss function.

The first problem i faced was the fact that When i divided Final Y by A i got getting a invalid divide because the value of A was too small which i removed by **initializing the value of w as np.random.randn()*0.0**1 this 0.01 removed that error and this reduced the value of wait and hence their decrease(value of gradient).Then I tried training my Neural Network **on learning rate = 0.1** as any more it was showing an issue in dividing Y by A.

But it turned out my model was very slow even for being able to fit properly i.e over 99 percent accuracy on training set , it took more than 8000 iterations which took like 4 hours to train not only that i was getting an Accuracy of 87.86 percent accuracy on the Cross validation set.

My choice of number of neurons and number of layers were number of hidden layers as 2 and number of neurons in those layers as 200 ,100 respectively.



But then i learned about the concept of vanishing and exploding gradients and the t=fact that **initializing weights as He initialization (np.random.randn(*shape)*(np.sqrt(2/number of neurons in the**

previous layer))is the most optimum and fastest setup for the RELU layers

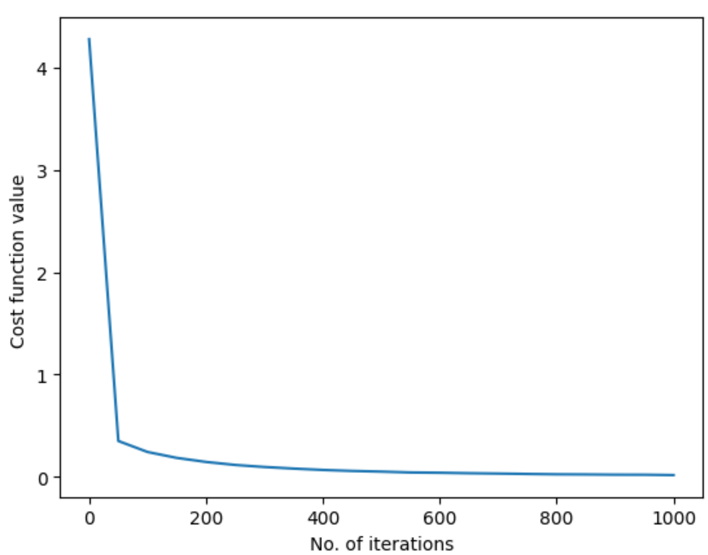
And then I applied **Adam 's optimizer** and **mini- Batch Gradient descent**.

Adam 's optimizer consists of using momentum and RMS prop simultaneously and their **hyperparameters were taken as beta1 and beta2 = 0.9 and 0.999** respectively.

Another **hyperparameter was the size of the mini batch** and I took that **as 4096** because there was an error of the denominator term approaching 0 in activation value of the final layer.

The accuracy I was getting was **88.62%** by taking the number of neurons in different layers as [X.shape[0],400,100,10] and **learning rate of 0.001.**

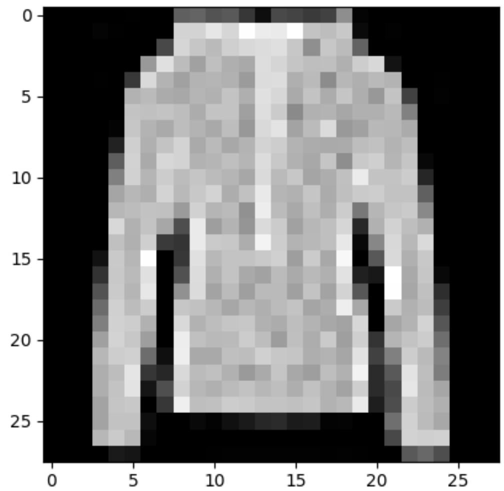
And now the function is reaching its minimum in 20 minutes .



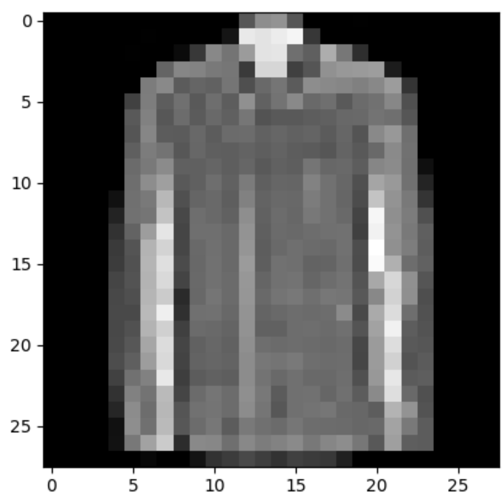
Since i also tried to calculate **Accuracy on the training which turned out to be 99.9924% accuracy** , Now my initial instinct was to assume that my model is having **high variance** and that i need to apply regularization on the model and so i decided to apply **Dropout Regularization** and turned the hidden layer 's probability of any neurons to get trained in any said iteration as **0.7.** But even after that my model didn't have any significant differences in terms of Accuracy on the Cross Validation set. Since it was a Neural Network I presumed that my model could reach accuracy of over 99% on training set and i must be doing something wrong and tried changing number of neurons and number of hidden layers but that didn't help So next i decided to plot the examples in the Cross Validation set on which my model was giving the wrong predictions

WRONG OUTPUTS IN CROSS VALIDATION SET:-

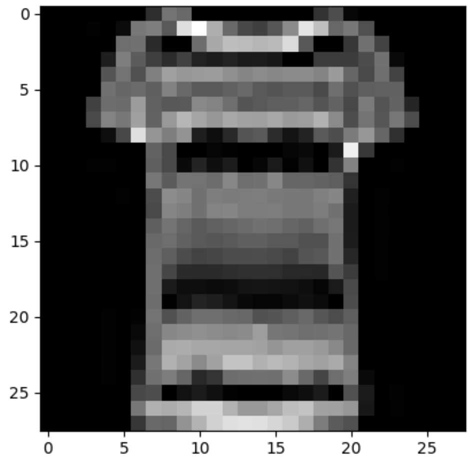
Since there were many errors so instead I will put like a common type of error via my model .These errors are common errors that existed in all the models that were used for classification.



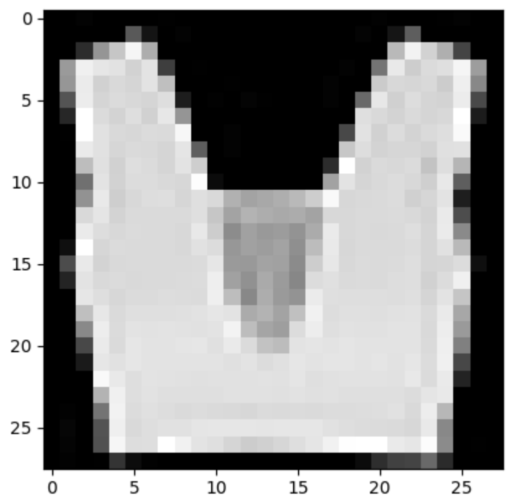
Predicted_Category :- 2 Actual_Category :- 4



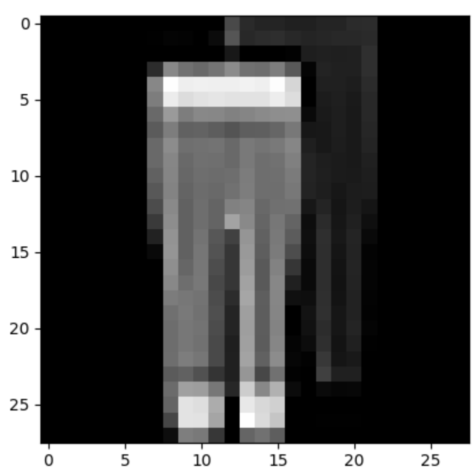
Predicted_Category :- 6 Actual_Category :- 4



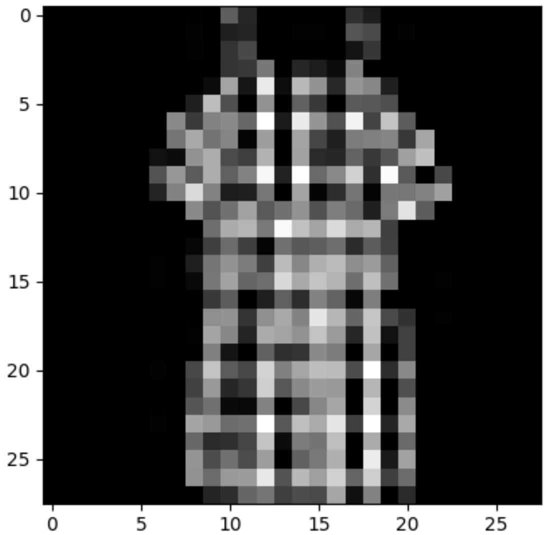
Predicted_Category :- 0 Actual_Category :- 6



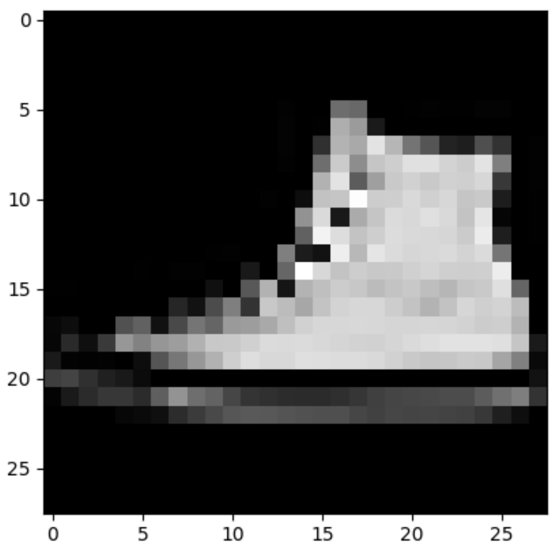
Predicted_Category :- 8 Actual_Category :- 0



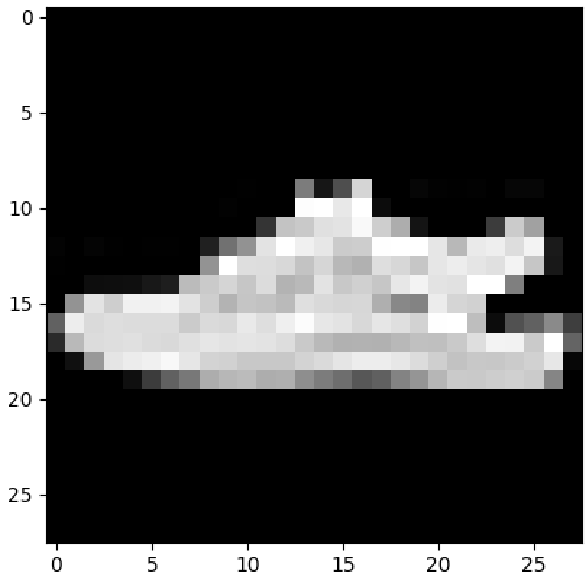
Predicted_Category :- 3 Actual_Category :- 1



Predicted_Category :- 0 Actual_Category :- 3



Predicted_Category :- 9 Actual_Category :- 7



From this what i saw was that all the images that my model was struggling to had a pattern like claiming 2 instead of 4 and so on but the thing that i saw was the fact that the images were really hard to classify accurately even by a human like me because plenty of times pixel values at the critical features aren’t that accurate for example in case of category 2 and category 4 i.e a full sleeve T-shirt and a jacket the whether or not they have the zip part near their collar would determine the classification between them which as i could see is hard for the above images even for a human like me   , so hence i realized even the **Human level of performance** i.e the **Baseline Level of performance is not that very good either .**

Even though I wasn't able to find a proper method to quantify my human level of error, looking at the images since I was also making a lot of mistakes gave me a sense of satisfaction that my neural network model is doing fine .

So even if the model is having an error of about 11 % on the cross validation set i would say it is pretty good considering the **Baseline Level of performance**