ML ASSIGNMENT-1 REPORT

UDIT BHATI
2019281

Solution-1)

The dataset has 9 columns and 4177 rows, out of these 9 columns 8 are model parameters and the last column i.e. the ring is the output vector.

Preprocessing:

On running data.describe(), I notice that the column “Height” has some 0 values, I replace the 0 values with the median of that column.

Then I set the random seed to 0 so that random outputs can be consistent in future.

Then I perform the random shuffling on the data so that we can achieve IID in our data.

I also checked for any NA values in the dataset, luckily there were no NA values.

Since the column “Sex” does not have numeric values we can not perform the regression so I mapped M-0, F-1 and I-2. After that, I performed the 8:2 train: spilt.

Since the maximum, median and standard deviation of the columns is less than 1, We do not need Normalization in this case.

Part-1:

Define hypothesis, H(x)=X*theta where X=feature vector and theta=model parameters.

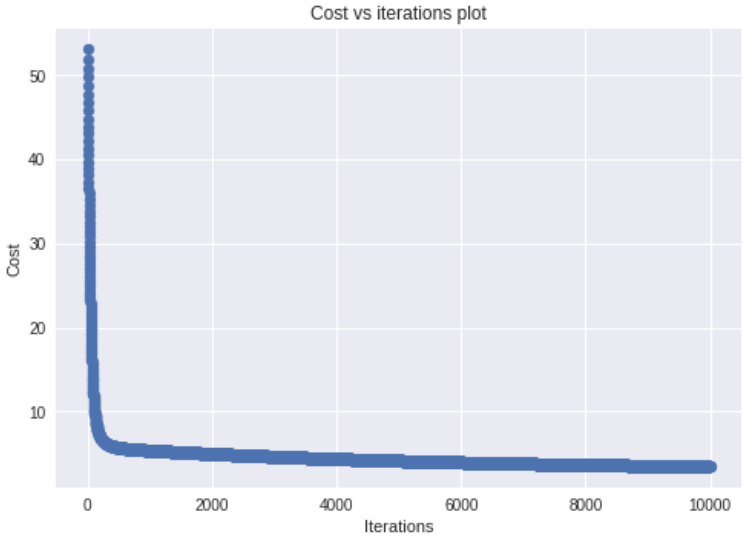
Define cost function(L2 loss), J(theta)=(average of (y_predicted- y_true)^2)

Gradient descent runs for iterations times with alpha. For each iteration, we calculate the predicted y, update the model parameters and cost.

Gradient descent returns the cost list and tuned parameters theta which now can be used to predict the results.

I take alpha=0.005 and iterations=10000.

RMSE is calculated using the formula RMSE=sqrt(average of (y_predicted- y_true))



Results:

RMSE in Training: 1.3723412202071286

RMSE in Testing: 1.4272339304229533

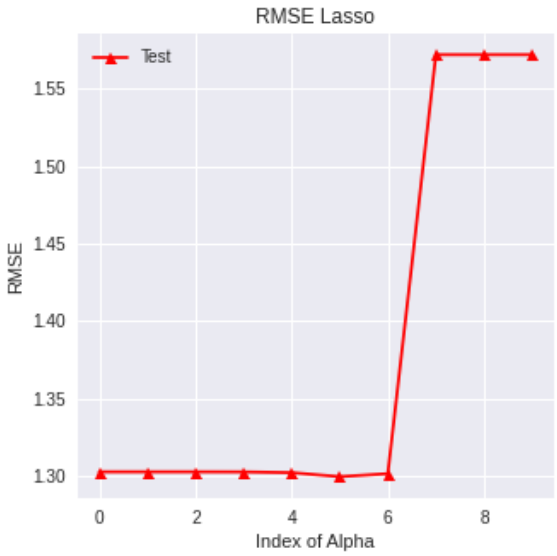
Model parameters, theta=[0.4751367276565551 8.95799279128411 7.108142469583458
2.8087202072919637 1.776984532601285 -2.6831658525579942 0.08162936021305318
2.890054504377442]

Part-2a:

Alpha values=[1e-10, 1e-8,1e-6,1e-5,1e-4,1e-3,1e-2, 1, 5, 10]

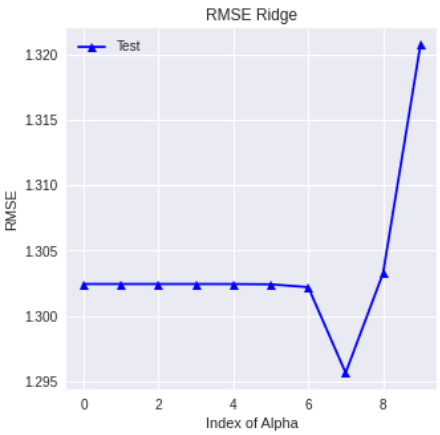
iteration=10e5

Results:



Best alpha for Lasso=0.001

Model Parameter for Lasso=[-0.34392101 0. 8.23848036 21.29784301 8.3469814
-18.85506825 -9.56786127 8.2550227]



Best alpha for Ridge=1

Model Parameter for Ridge=[-0.36505381 2.14715945 6.7883872 12.08093437 6.8659893
-17.03308958 -6.91630525 10.17225279]

Part-2b:

Using GrdSearchCV we can get the following result:

Results:

For RIDGE

alpha=0.01

Model parameters=[-0.33711002 -1.55401871 10.12409545 24.19306169 9.07358908
 -19.50986754 -11.42934178 7.40285644]
For LASSO
alpha=0.0001
Model parameters=[-0.33782805 -1.09011706 9.58031986 24.12853045 9.03805392
 -19.49165774 -11.34093658 7.43472472]
Mean Absolute difference between coeff. of Lasso=1.148721534138283
Mean Absolute difference between coeff. of Rigde=3.8929710197566703
We can see Lasso is performing twice as well as that of ridge, which is justified because lasso trys to make coefficient zero which are not giving accurate results. On the other hand ridge never sets the coefficient value to zero.

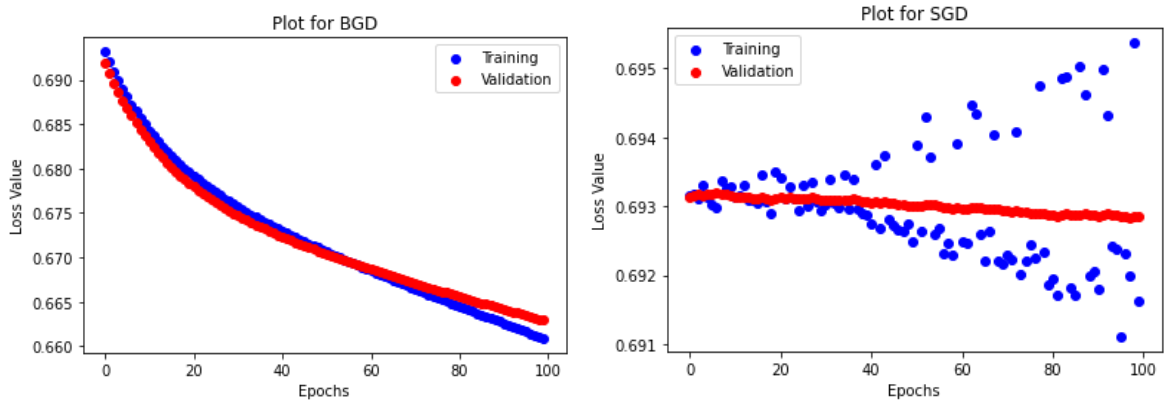
Solution-2)
The dataset has 9 columns and 768 rows, out of these 9 columns 8 are model parameters and the last column i.e. the Outcome is the output vector.
Since outcome has only 0,1 value so it is a binary classification.

Preprocessing:

On running data.describe(), I notice that some of the columns has some 0 values, I replace the 0 values with the median of each individual column.
I observe that the data was not Normalised so I used the min-max normalizarion to normalize the data.
Then I set the random seed to 0 so that random outputs can be consistent in future.
Then I perform the random shuffling on the data so that we can achieve IID.
I also checked for any NA values in the dataset, there were no NA values present.
Splitting of the data in performed in 7:2:1 train:val:test.

Part-2.1.a:

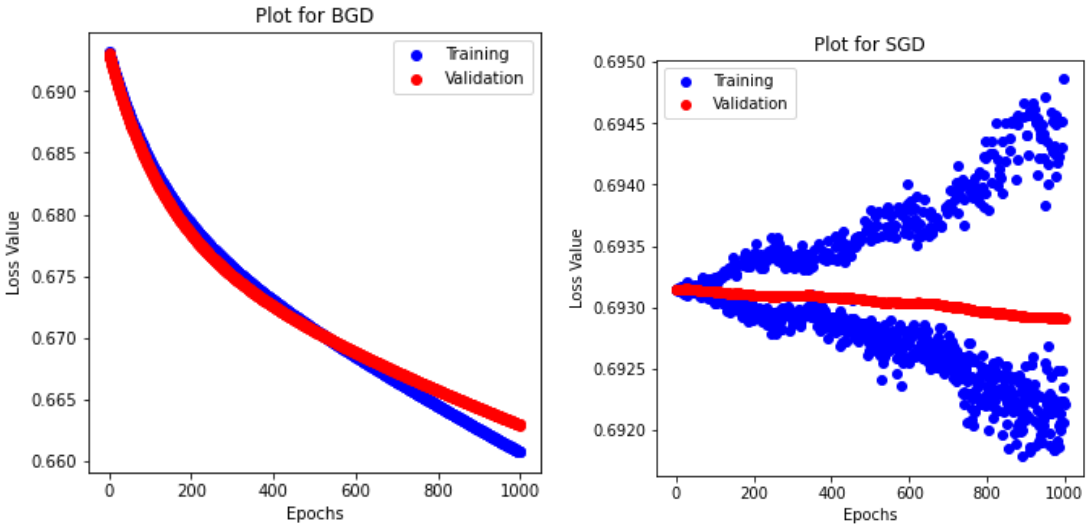
I created a LogisticRegressionModel class which has epoches, learning rate, number of features, data size as class variables and sigmoid, lossfunction, predictresult, trainBGD, trainSGD as class functions.
Training loss and validation losses are calculated for SGD, BGD are calculated is respective functions.
The plot we get is:



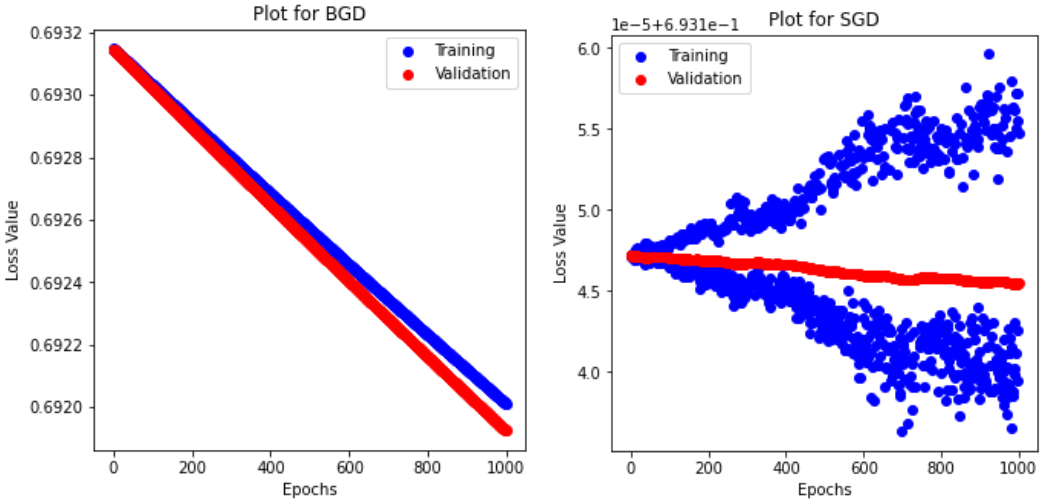
Part-2.1.b:

For alpha=0.01,0.0001,10 graphs are:

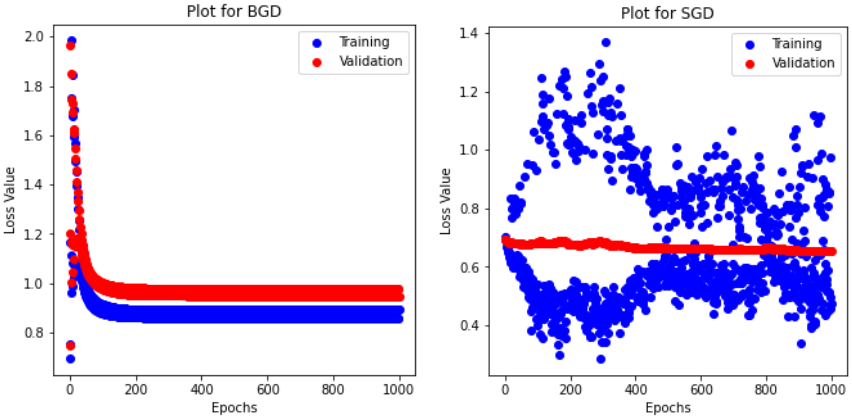
alpha=0.01, converges at around 800 epochs, minimum loss value.



alpha=0.0001, did not converge, SGD has high loss.



alpha=10, did not converges, minimum loss is relatively high as compared to other alpha



For the graph we can observe that the best value of alpha is 0.01, as the loss in both BGD and SGD is minimum from other two values of alpha.

Part-2.1.c:

Confusion matrix= [TP FP]
[FN TN]

accuracy=(TP+TN)/(TP+TN+FP+FN)
precision=TP/(TP+FP)
recall=TP/(TP+FN)
flScore=2 * ((precision * recall)/(precision + recall))

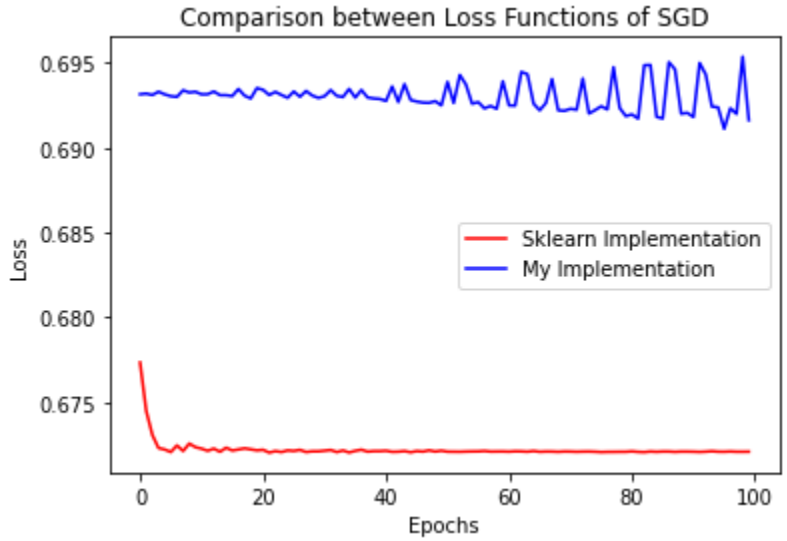
Results:

[[54 0]
[23 0]]
Accuracy is: 0.7012987012987013
Precision is: 1.0
Recall is: 0.7012987012987013
F1 Score is: 0.8244274809160306

Part-2.2.a:

To get the values of losses from sklearn I first print the result on console using verbose=1 on SGDClassifier then I collected the outstream results and seperated out the losses, to get 100 data points I changed the n_iter_no_change to 100, without this I were only getting 15 data points.

Result:



Part-2.2.b:

My implementation converges at around 60-80 epochs while Sklearn implementation converges at around 15-20 epochs.

Part-2.2.c:

Confusion matrix= [TP FP]
[FN TN]
accuracy=(TP+TN)/(TP+TN+FP+FN)
precision=TP/(TP+FP)
recall=TP/(TP+FN)
flScore=2 * ((precision * recall)/(precision + recall))

Result:
**********Sklearn Performance**********
Confusion Matrix
[[54 0]
[23 0]]
Accuracy is: 0.7012987012987013
Precision is: 1.0
Recall is: 0.7012987012987013
F1 Score is: 0.8244274809160306

**********My Performance**********

Accuracy is: 0.7012987012987013
Precision is: 1.0
Recall is: 0.7012987012987013
F1 Score is: 0.8244274809160306

Solution-3:
Part-3.1:
Preprocessing:
Seed set to 0.
Rows are filtered to 1 and 2 because only we only have two classes Trouser
and Pullover.
Random shuffling is done on training and testing data.
Then I Binarize all the pixel columns to 0 and 1 where 0=0 to 127, and 1 otherwise. This means now our image is grey scaled.
Then I implemented the NaiveBayes class which uses gaussian distribution to classify the two classes.
Result:
Accuracy of the model is: 0.5
Part-3.2:
Similar steps in preprocessing are done as above.
I combined the sample from train and test into one so that we can use k-fold cross validation.
crossValidation function divides the data into folds.
scoreAverage function reuters the list of accuracy score for each fold.
Result: Using k=4
Accuracy of the Fold 1 is: 0.49838095238095237

Accuracy of the Fold 2 is: 0.5022857142857143
Accuracy of the Fold 3 is: 0.4978095238095238
Accuracy of the Fold 4 is: 0.5015238095238095
Average Accuracy is: 0.5

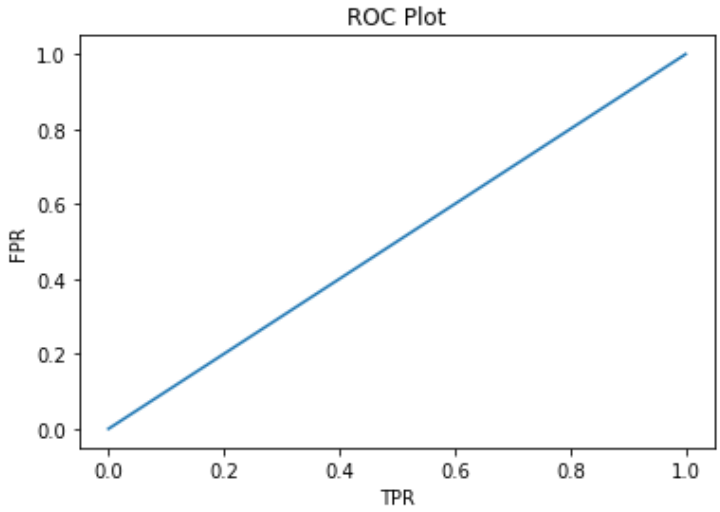
Part-3.3:

a)

Confusion matrix= [TP FP]
[FN TN]

accuracy=(TP+TN)/(TP+TN+FP+FN)
precision=TP/(TP+FP)
recall=TP/(TP+FN)
flScore=2 * ((precision * recall)/(precision + recall))

b)



c)

Result:

Confusion Matrix
[[1000 0]
 [1000 0]]
Accuracy is: 0.5
Precision is: 1.0
Recall is: 0.5
F1 Score is: 0.6666666666666666

Solution-4)

Part-2:

Regularisation is used to prevent overfitting of the data. This is done by introducing a penault term in loss function. loss=LS_obj + lambda*(sum of square of slopes)
Since the loss function depends on the square of slopes the features with higher coefficients will have more loss. In this way to minimize the loss coefficients values are reduced.

Part-3:

Let the linear regression be $Yn = MXn + c$, where c being the intercept or noise.
Let c be the Normal/Gaussian Noise whose variance is var, and mean=0.
Then the likelihood function will be=product over n(F(Yn|MXn,var)).
Taking Gaussian prior F(M|0,(lamda)^-1). Where lamba>0
Multiplying likelihood and prior we get,

Product over n (F(Yn|MXn,var))*F(M|0,(lamda)^-1).

taking logrithm we get, (sum over n)1/(-var)*(Yn-MXn)^2 -lamda*(M^2)
To get the Maximum A Posterior Inference. We maximise the above term with respect to M.