MACHINE LEARNING
ASSIGNMENT 1 REPORT

AASTHA -2019224

1. Linear Regression

Data Preprocessing

1. Importing the modules
2. Importing the dataset with the header set to None
3. Encoding column 0 with values F, I, M to 0,1, 2 using LabelEncoder.
4. Splitting the dataset into train and test sets with test size set to 0.2 and random state set to 0.

Implementing train_test_split From Scratch

1. Calculate the test size by multiplying the test_size parameter with dataset length.
2. Set random state to the random seed.
3. Find a random index, add it to the test set and remove it from the dataset. Continue this process till the test set size is less than the test_size needed.
4. Return the train and test sets.

Implementing Linear Regression From Scratch

Linear regression is implemented from scratch using Gradient descent algorithm. The equation for gradient descent is $\theta j = \theta j - \alpha\,(\partial J(\theta)/\partial \theta j)$ for all j. $J(\theta)$is given by $J(\theta) = \sum_{i=1}^{m} (\theta T\, \phi(xi) - yi)^2$. A bias term has also been taken into account in the code.

For Training

1. We iterate over the training dataset and for each iteration we find the y_predicted value using the dot product of x and theta and adding the constant term.
2. We then find the gradients of theta and the bias terms and update the values of theta and bias for each iteration.

For Prediction

We do a dot product of testing dataset X and theta and add the bias term to it to predict the y values.

To Find RMSE

RMSE is given by $\sqrt{1/n * \sum_{i=1}^{n}(y - f(x)^2}$. Here y is taken as y_true and f(x) is taken as y_pred to calculate the RMSE.

The results obtained for 40,000 iterations and learning rate 0.1 are:

```
Root Mean Squared Error
Train  2.1917792438562023
Test  2.3550307203907757
```

Applying Regularization Techniques-Ridge and Lasso Regression

Used 500 alpha values from 1e-5 to 10 using np.linspace(). The graphs obtained to show the effect of alpha value on training and testing data's RMSE are shown below.

The best model coefficient obtained for Ridge and Lasso Regression(values for which RMSE is minimum) are as follows:

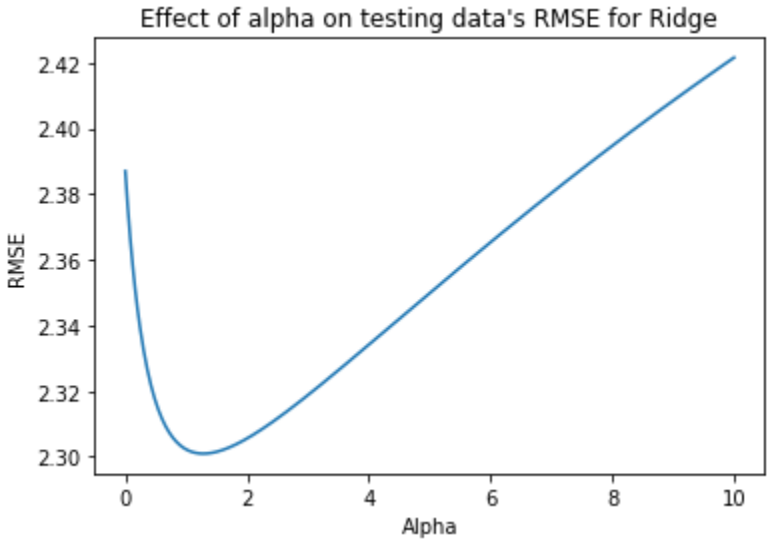
For Ridge Regression

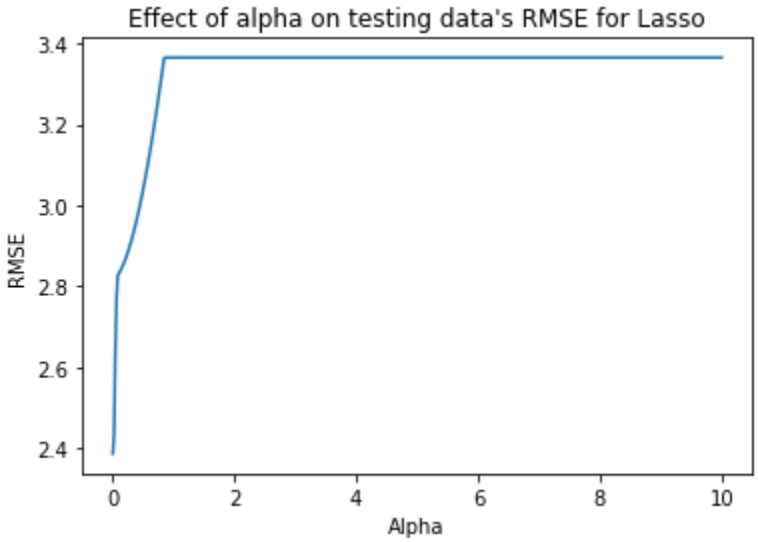
```
{'alpha': 1.282573847695391,
 'coeff': array([[  0.05188745,   2.7531959 ,   6.80260005,
10.78568595,
          6.44183854, -16.43319683,  -5.3326316 ,  10.16730724]]),
```

```
 'rmse': 2.3009253750557725}
```

For Lasso Regression

```
{'alpha': 1e-05,
 'coeff': array([  0.05912666,  -1.42729057,  10.68826337,  23.7601288 ,
          8.7365166 , -19.24686393, -10.02253708,   7.49497595]),
 'rmse': 2.3866406862721976}
```





Results obtained using Sklearn's GridSearchCV to find the best alpha value and best model coefficients

For Ridge Regression

```
Best Negative RMSE:  -2.236280173646031
Best Alpha:  {'alpha': 0.12025036072144289}
Best Model Coefficients:  [[  0.05876589  -0.42352831   9.82225519
21.11722349   8.37414862
  -18.89306024  -9.27050044   8.09695838]]
```

For Lasso Regression

```
Best Negative RMSE:  -2.237444565893175
Best Alpha:  {'alpha': 1e-05}
Best Model Coefficients:  [[  0.05876589  -0.42352831   9.82225519
21.11722349   8.37414862
  -18.89306024  -9.27050044   8.09695838]]
```

Comparing the results

We can see that the best model coefficient and alpha values obtained from parts A and B and the results from the graph are very close. The slight deviation can be due to the scoring parameter which is negative RMSE in GridSearchCV and RMSE in part A.

2. Logistic Regression

Data Preprocessing

1. Importing the modules
2. Importing the dataset
3. Splitting the dataset into train, validation and test sets.
4. Feature scaling using MinMaxScalar

Implementing train_val_test_split From Scratch

1. Calculate the validation and test sizes by multiplying the val_size and test_size parameters with dataset length.
2. Set random state to the random seed.
3. Find a random index, add it to the validation set and remove it from the dataset. Continue this process till the validation set size is less than the val_size needed.
4. Repeat the same process for the train set.
5. Return the train, validation and test sets.

Classification Metrics

Created a class for classification metrics that takes the y_pred and y_test as the parameters and calculates the true positive, true negative, false positive and false negative values. It has functions for confusion matrix, accuracy, precision, recall and F1 Score.

Confusion Matrix = [[TP FP] , [FN TN]]

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Precision = TP/(TP + FP)

Recall = TP / (TP + FN)

F1 Score = 2*TP / (2*TP + FP + FN)

Implementing Logistic Regression From Scratch

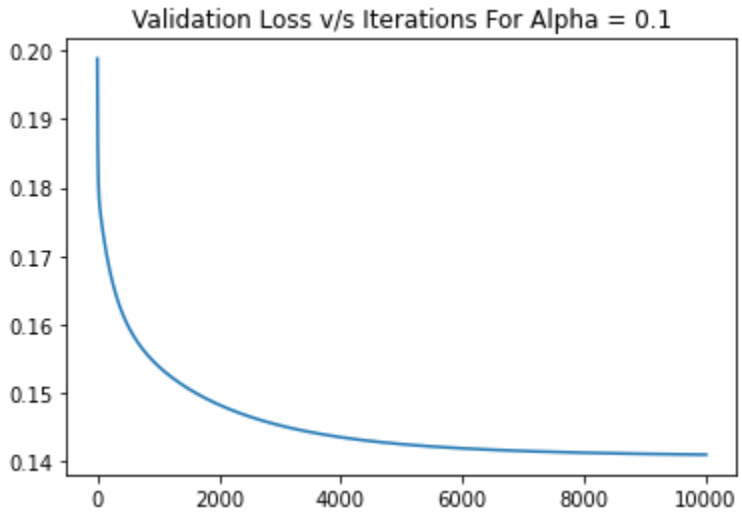
Using Batch Gradient Descent

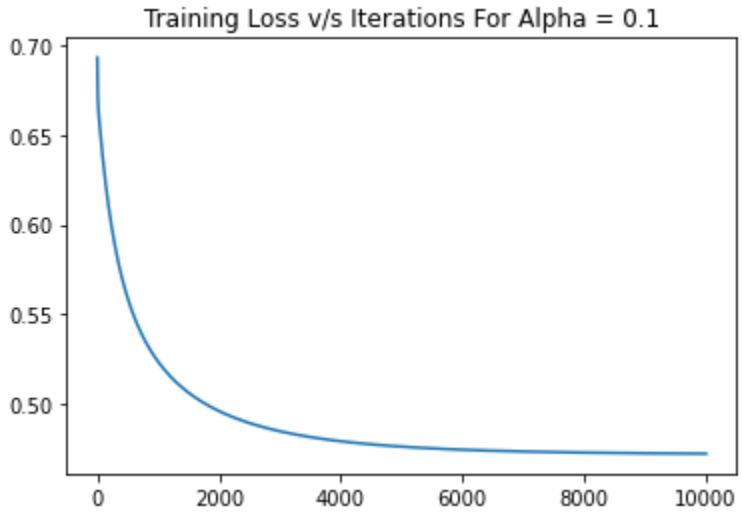
1. For each iteration, find y_predicted using the dot product of X and theta and adding the bias term and then applying the sigmoid function.
2. Store the loss for training and validation for each iteration.
3. Calculate the gradients for alpha and bias terms and update the terms.

Using Stochastic Gradient Descent

In Stochastic Gradient Descent, alpha and bias values are updated for each value in y, hence it converges faster than BGD. Training and validation loss is stored after each iteration to plot the loss v/s iteration curves.

Plots and metrics for different alpha values using BGD





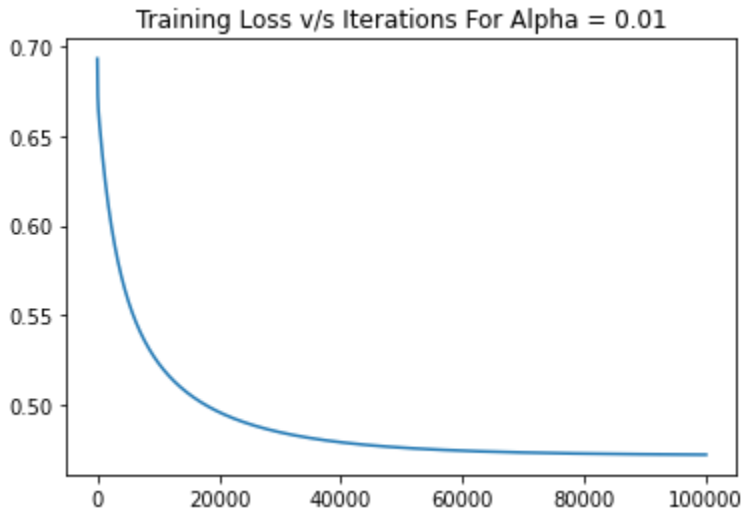
Convergence of the model

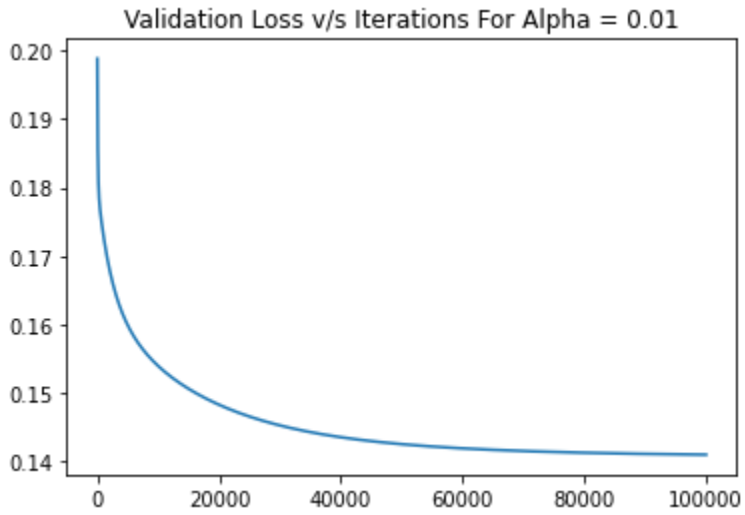
The line starts becoming straight between 4000 and 6000 iterations hence the model converges around 4000 iterations.

```
Alpha: 0.1, Iterations: 10000

For Validation Data
Accuracy:  0.7727272727272727
Precision:  0.5869565217391305
Recall:  0.627906976744186
F1 Score:  0.6067415730337079
Confusion Matrix
 [[27 19]
 [16 92]]


For Training Data
Accuracy:  0.7922077922077922
Precision:  0.7777777777777778
Recall:  0.5384615384615384
F1 Score:  0.6363636363636364
Confusion Matrix
 [[14  4]
 [12 47]]
```





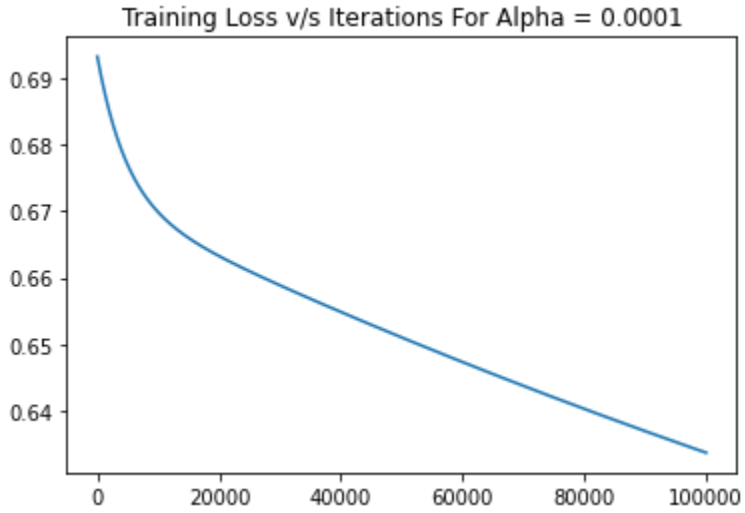
Convergence of the model

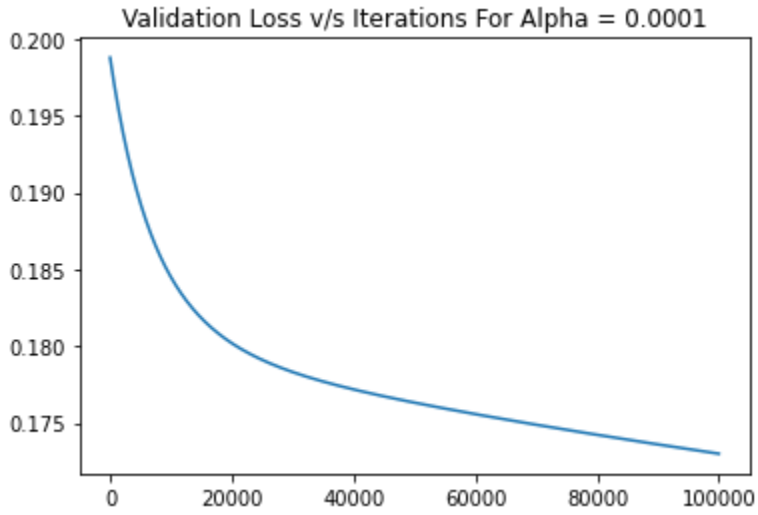
The line starts becoming straight between 40000 and 60000 iterations hence the model converges around 40000 iterations.

```
Alpha: 0.01, Iterations: 100000

For Validation Data
Accuracy:  0.7727272727272727
Precision:  0.5869565217391305
Recall:  0.627906976744186
F1 Score:  0.6067415730337079
Confusion Matrix
 [[27 19]
 [16 92]]


For Testing Data
Accuracy:  0.7922077922077922
Precision:  0.7777777777777778
Recall:  0.5384615384615384
F1 Score:  0.6363636363636364
Confusion Matrix
 [[14  4]
 [12 47]]
```





Convergence of the model

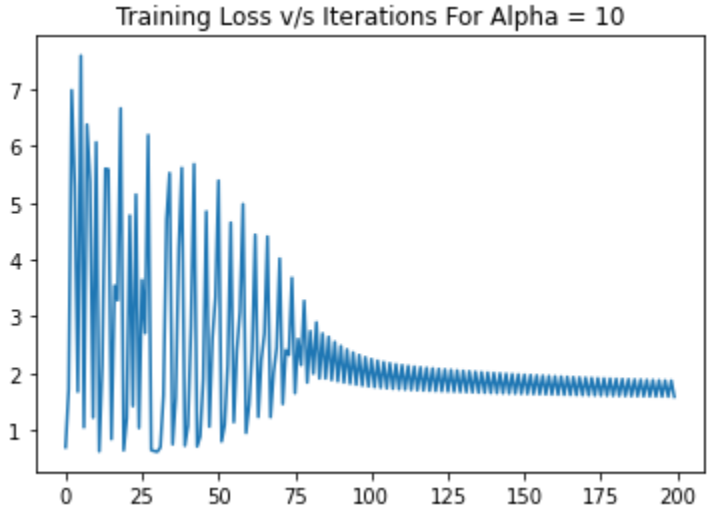
The line starts becoming straight around 1,00,000 iterations and hence the model starts to converge.

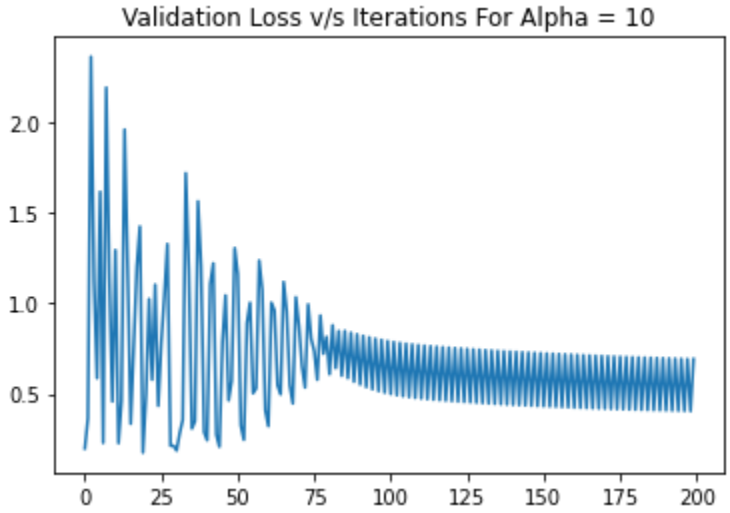
```
Alpha: 0.0001, Iterations: 100000

For Validation Data
Accuracy:  0.7207792207792207
Precision:  0.0
Recall:  0.0
```

```
F1 Score:  0.0
Confusion Matrix
 [[  0   0]
 [ 43 111]]


For Testing Data
Accuracy:  0.6623376623376623
Precision:  0.0
Recall:  0.0
F1 Score:  0.0
Confusion Matrix
 [[ 0  0]
 [26 51]]
```





```
Alpha: 10, Iterations: 200

For Validation Data
Accuracy:  0.7532467532467533
Precision:  0.7777777777777778
Recall:  0.16279069767441862
F1 Score:  0.2692307692307692
Confusion Matrix
 [[  7   2]
 [ 36 109]]


For Testing Data
Accuracy:  0.6623376623376623
Precision:  0.5
Recall:  0.07692307692307693
F1 Score:  0.13333333333333333
Confusion Matrix
 [[ 2  2]
 [24 49]]
```

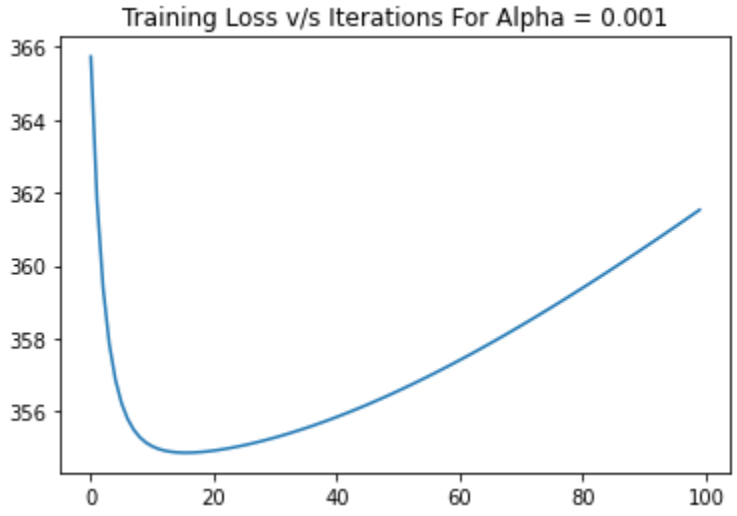
Convergence of the model

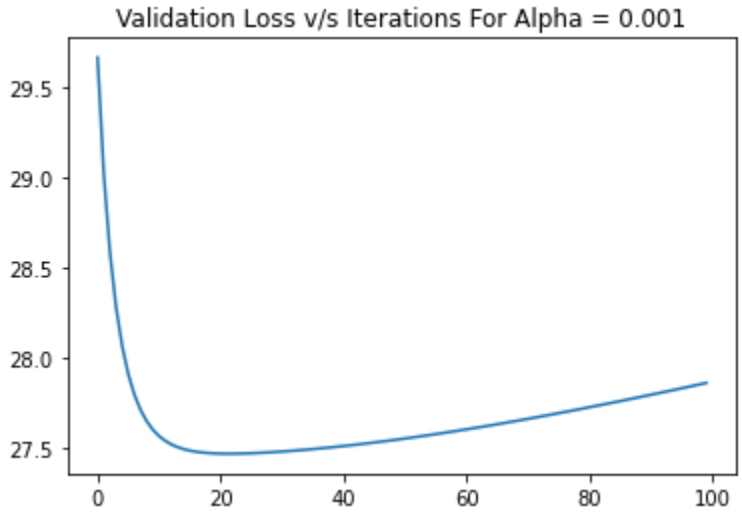
Since the value of alpha is very large the model oscillates and converges quickly to a suboptimal solution. From the graph, the model converges around 200 iterations

Analysis

1. In all the plots, the training loss is greater than the validation loss for the same number of iterations.
2. When the learning rate is small(0.0001) the model converges slowly and when the learning rate is large the model converges very fast. The optimal learning rate from the 3 values would be 0.01
3. The metrics accuracy, precision, recall and F1 score are better for 0.01 as compared to the other 2 learning rates.

Plots and metrics for different alpha values using SGD





Convergence of the model

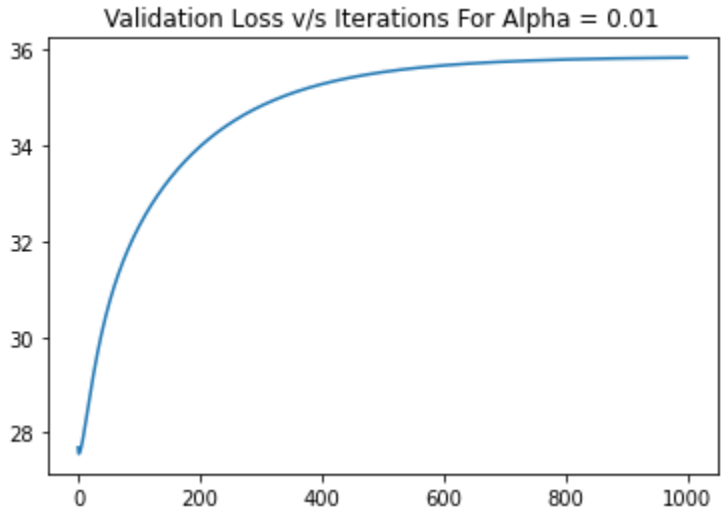
The model converges around 16 iterations where we have the minima in the plot.

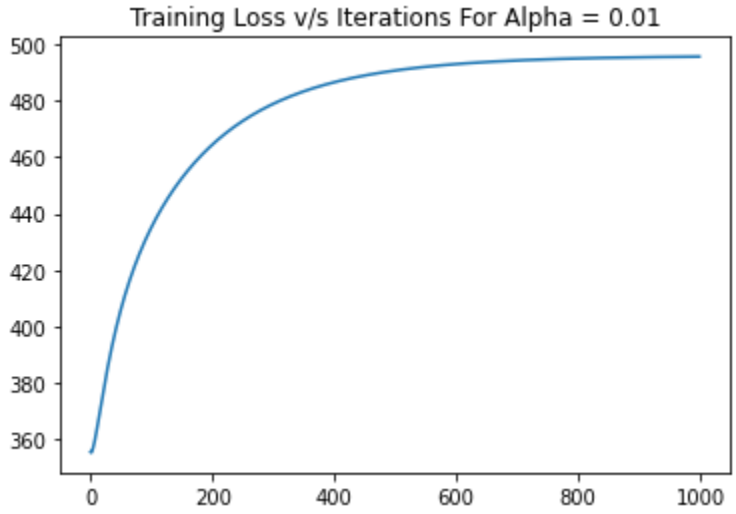
```
Alpha: 0.001 Iterations: 100

For Validation Data
Accuracy:  0.7337662337662337
Precision:  0.6
Recall:  0.13953488372093023
F1 Score:  0.22641509433962265
Confusion Matrix
```

```
 [[  6   4]
 [ 37 107]]


For Testing Data
Accuracy:  0.6883116883116883
Precision:  0.75
Recall:  0.11538461538461539
F1 Score:  0.2
Confusion Matrix
 [[ 3  1]
 [23 50]]
```





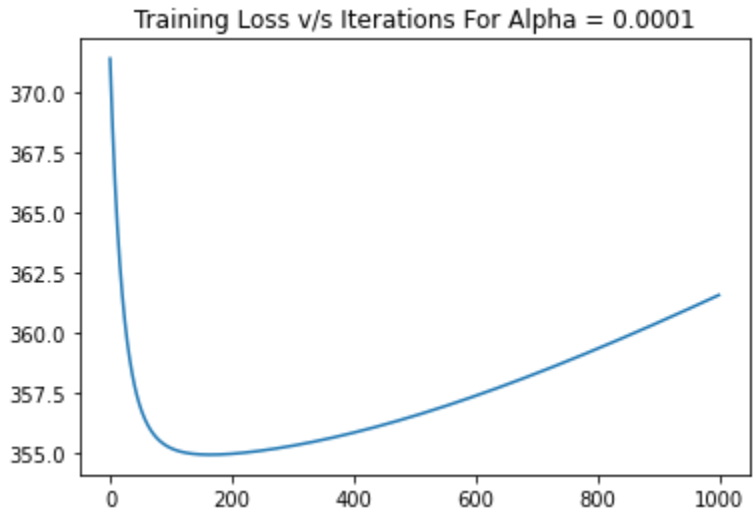
Convergence of the model

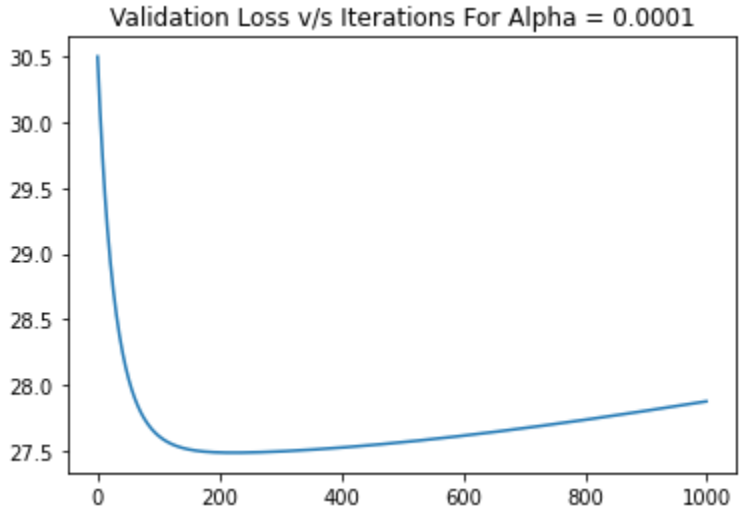
The model converges around 2 iterations where we have the minima.

```
Alpha: 0.01, Iterations: 1000

For Validation Data
Accuracy:  0.7727272727272727
Precision:  0.5869565217391305
Recall:  0.627906976744186
F1 Score:  0.6067415730337079
Confusion Matrix
 [[27 19]
 [16 92]]


For Testing Data
Accuracy:  0.7922077922077922
Precision:  0.7777777777777778
Recall:  0.5384615384615384
F1 Score:  0.6363636363636364
Confusion Matrix
 [[14  4]
 [12 47]]
```





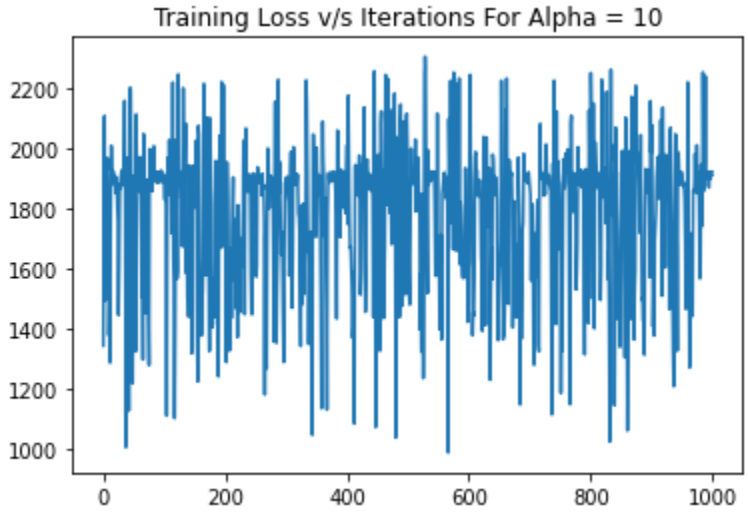
Convergence of the model

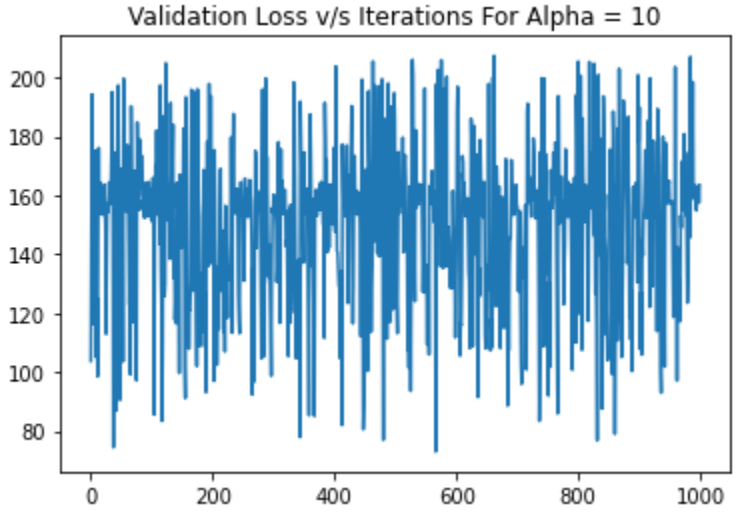
The model converges around 150 iterations where we have the minima.

```
Alpha: 0.0001, Iterations: 1000

For Validation Data
Accuracy:  0.7337662337662337
Precision:  0.6
Recall:  0.13953488372093023
F1 Score:  0.22641509433962265
Confusion Matrix
 [[  6   4]
 [ 37 107]]


For Testing Data
Accuracy:  0.6883116883116883
Precision:  0.75
Recall:  0.11538461538461539
F1 Score:  0.2
Confusion Matrix
 [[ 3  1]
 [23 50]]
```





Convergence of the model

Since the value of alpha is very large the model oscillates and converges quickly to a suboptimal solution. From the graph, the model does not converge till 1000 iterations.

```
Alpha: 10, Iterations: 1000

For Validation Data
Accuracy:  0.6298701298701299
Precision:  0.4222222222222222
Recall:  0.8837209302325582
```

```
F1 Score:  0.5714285714285714
Confusion Matrix
 [[38 52]
 [ 5 59]]


For Testing Data
Accuracy:  0.7662337662337663
Precision:  0.5952380952380952
Recall:  0.9615384615384616
F1 Score:  0.7352941176470589
Confusion Matrix
 [[25 17]
 [ 1 34]]
```

Analysis

1. In all the plots, the training loss is greater than the validation loss for the same number of iterations.
2. When the learning rate is small(0.0001) the model converges slowly. For the learning rate 0.01, the model converges faster. For alpha=10 the error oscillates.
3. The metrics accuracy and precision are better for alpha=0.01 and recall and f1 score are better for alpha =10.

Using Sklearn's Implementation

The appropriate learning rate from A part is 0.01 for BGD and the number of iterations is around 40,000

A)

1. The loss v/s iteration plot first decreases and after a minima, it starts increasing.
2. For greater values of alpha the loss oscillates.
3. For smaller values of alpha the curve converges very slowly.
4. The validation loss is less than the training loss.

B)

The number of iterations before reaching the stopping criteria for sklearn is given by n_iter_ and is 16. Our model implemented using gradient descent converges slower due to the difference in the internal implementation of sklearn. Also, there are several other factors like regularization which have not been taken into account which makes the number of epochs different.

C) The performance of sklearn's implementation is shown below.

```
For Validation Data
Accuracy:  0.7792207792207793
Precision:  0.6097560975609756
Recall:  0.5813953488372093
F1 Score:  0.5952380952380952
Confusion Matrix
 [[25 16]
 [18 95]]


For Testing Data
Accuracy:  0.7402597402597403
Precision:  0.6875
Recall:  0.4230769230769231
F1 Score:  0.5238095238095238
Confusion Matrix
 [[11  5]
 [15 46]]
```

For 16 iterations using our BGD Classifier the performance is as follows.

```
Alpha: 0.01, Iterations: 16

For Validation Data
Accuracy:  0.7207792207792207
Precision:  0.0
Recall:  0.0
F1 Score:  0.0
Confusion Matrix
 [[  0   0]
 [ 43 111]]


For Training Data
Accuracy:  0.6623376623376623
Precision:  0.0
```

```
Recall:  0.0
F1 Score:  0.0
Confusion Matrix
 [[ 0  0]
 [26 51]]
```

For 40,000 iterations using our BGD Classifier the performance is as follows

```
Alpha: 0.01, Iterations: 40000

For Validation Data
Accuracy:  0.7662337662337663
Precision:  0.574468085106383
Recall:  0.627906976744186
F1 Score:  0.6
Confusion Matrix
 [[27 20]
 [16 91]]


For Training Data
Accuracy:  0.7792207792207793
Precision:  0.7368421052631579
Recall:  0.5384615384615384
F1 Score:  0.6222222222222222
Confusion Matrix
 [[14  5]
 [12 46]]
```

Comparison

While our model does not perform too well for the same number of iterations on which sklearn's model converges that is 16, it performs better on larger number of iterations(40,000) than sklearn's model and has better accuracy, precision, recall and F1 Score

3. Naive Bayes

Data Preprocessing

1. Importing the modules
2. Importing the FMNIST dataset using train and test CSV files from Kaggle.
3. Filtering Data For Binary Classification-Trouser and Pullover were represented by 1 and 2 in the dataset. Filtered 1 and 2 values and updated the encoding of pullover to 0.
4. Binarizing the data-Values less than 126 set to 0 and greater than or equal to 126 set to 255.
5. Separating the columns X and y.

Implementing Naive Bayes Binary Classifier From Scratch

Binary Classifier has been implemented using Gaussian Naive Bayes algorithm.

The formula for Gaussian Naive Bayes is given by

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

and the equation of Naive Bayes Classifier is

$$Y^{new} = argmax P(Y = Y_j) \prod_i P(X^{new} | Y = Y_j)$$

For Training

Calculated and stored the mean and variance along the columns for each class and the ratio of items of that class in the dataset.

For Prediction

For each row in the testing dataset, we store the probabilities using the Gaussian elimination formula and then use argmax to return the class with maximum probability.

Classification Metrics

Created a class for classification metrics that takes the y_pred and y_test as the parameters and calculates the true positive, true negative, false positive and false negative values. It has functions for confusion matrix, accuracy, precision and recall.

Confusion Matrix = [[TP FP] , [FN TN]]

Accuracy = (TP + TN) / (TP + TN + FP + FN)

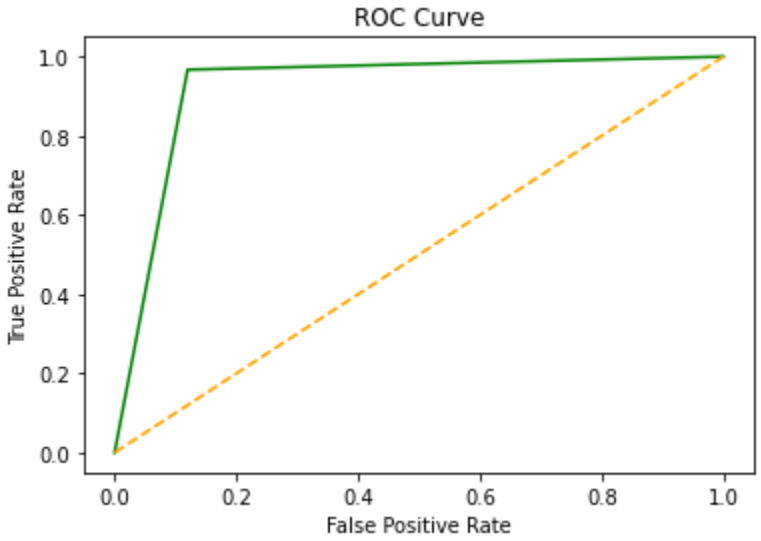
Precision = TP/(TP + FP)

Recall = TP / (TP + FN)

The results obtained are as follows:

```
Confusion Matrix
 [[967 120]
 [ 33 880]]
Accuracy  0.9235
Precision  0.889604415823367
Recall  0.967
```

Receiver Operating Characteristic Curve



K-Fold Cross-Validation

K-Fold Cross Validation is performed by splitting the dataset into k-folds and performing k-iterations. In each iteration, one of the k folds is taken as the testing data and the remaining k-1 folds as the training and accuracy is calculated. The average of the accuracy from the k folds is returned.

The value chosen for k is 5 since it is a medium-sized dataset. The choice of folds depends on the size of the dataset and since it is a medium-sized dataset(after filtering out the values of trouser and pullover) an appropriate value of k would be 5. Increasing the folds can lead to smaller bias and larger variance and an increase in computation time. Though a common choice for k is 10, taking k = 10 will drastically increase the computation time.

The accuracy reported from k fold cross-validation is

```
Accuracy Using K-Fold Cross Validation:  0.9217857142857142
```