CSE 343: Machine Learning

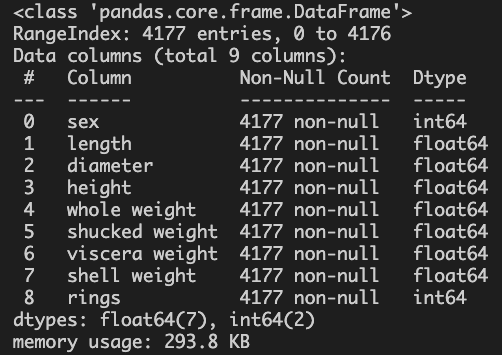
Assignment 1 | Report

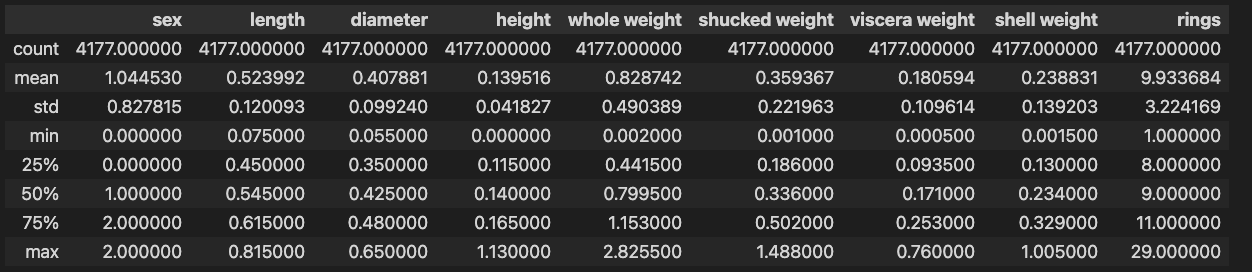
# Question 1. Linear Regression

1. Loading the Dataset
   * The dataset was loaded and converted to DataFrame using the pandas library.
   * The Columns header was added and the following names were given to all the columns: abaloneDF.columns = ['sex', 'length', 'diameter', 'height', 'whole weight', 'shucked weight', 'viscera weight', 'shell weight', 'rings']
2. Preprocessing the Data
   * Firstly the unique values in the ‘sex’ column were found. Those were:

array(['F', 'I', 'M'], dtype=object)

* + The above categorical data was converted to numeric data as per the following mapping:
    - I = 0
    - F = 1
    - M = 2
  + Basic info about the data was obtained:





* + The data was randomly divided into train and test sets in the ratio of 8:2
  + The train and test sets were divided into X\_train, y\_train and X\_test, y\_test respectively such that first 8 columns were part of X and last column was part of y
  + The data was then normalized such that each column had mean 0 and standard deviation 1
    - It was made sure that the test set was normalized using the mean and standard deviation of training data itself.

X\_mean = np.mean(X\_train, axis=0)

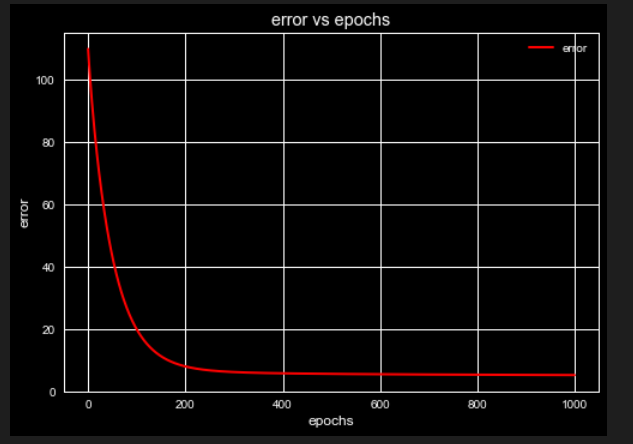
X\_std = np.std(X\_train, axis=0)

X\_train = (X\_train - X\_mean) / X\_std

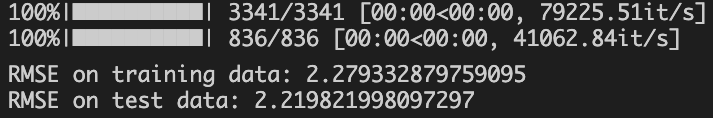
X\_test = (X\_test - X\_mean) / X\_std

* + A column of 1’s was added in both the training and the test sets so as to denote X0

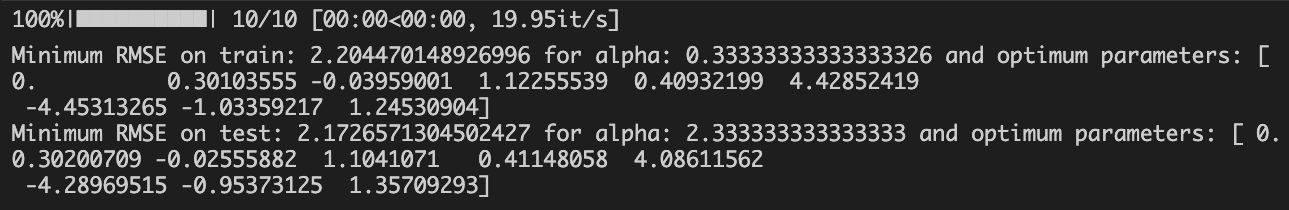
1. Performing Linear Regression
   * A set of functions were written to calculate:
     + hypothesis
     + rmse
     + gradient
     + gradient descent
   * Linear regression was performed using Batch Gradient Descent and theta vector and errorlist corresponding to each epoch was obtained
   * The error list was plotted against the number of epochs:



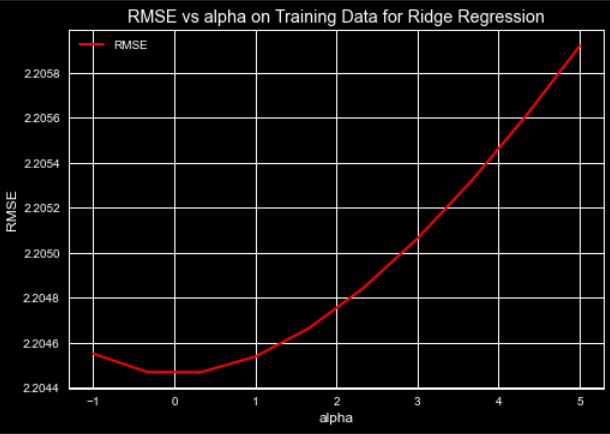
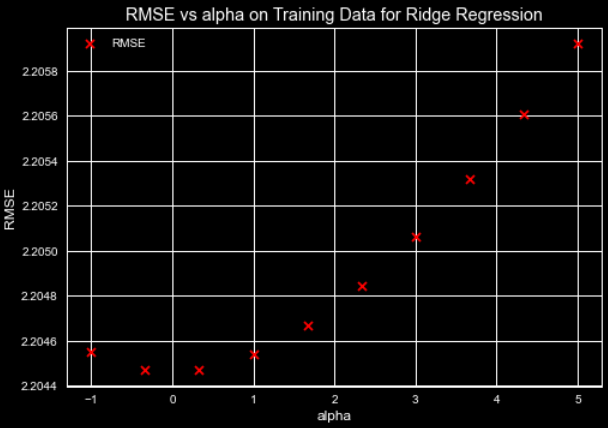
* + To get the predictions following functions were written:
    - getPredictionList
    - rmse
  + Using the above functions, RMSE on training and testing data was calculated



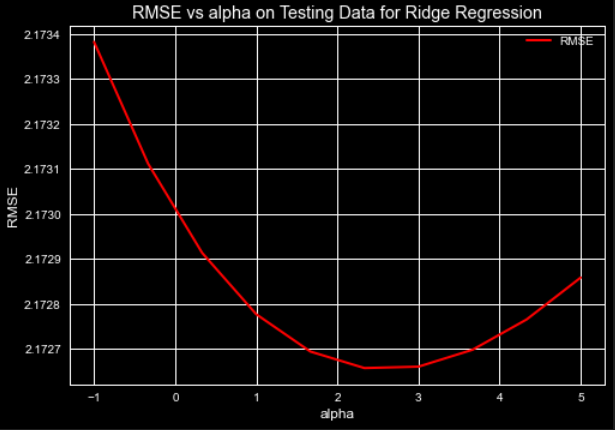
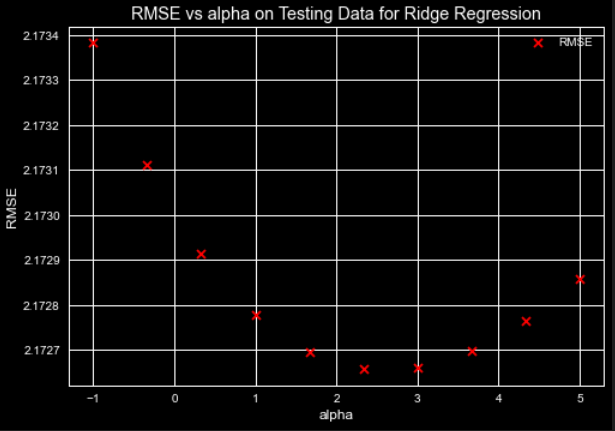
1. Ridge Regression
   * To use Ridge Regression, sklearn library was used
   * Function to calculate RMSE error for ridge regression was written
   * Minimum RMSE and its corresponding alpha and theta values were stored. They came out to be:



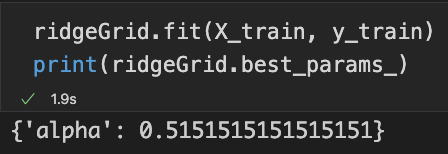
* + Corresponding scatter and line plots were also plotted.
    - Training Data:



* + - Test Data:



* + For the above data, 10 different alpha values were takes which were equally distributed in the rage of -1 to 5.
  + The most optimum alpha was also calculated using sklearn’s GridSearchCV function.
  + For the same, the alphas used were equally distributed over 100 different values between -1 and 4, and the training data was used.
  + The optimum received was:

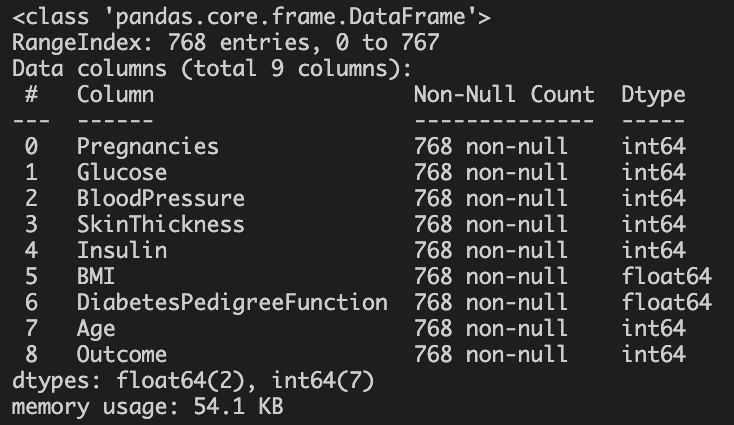


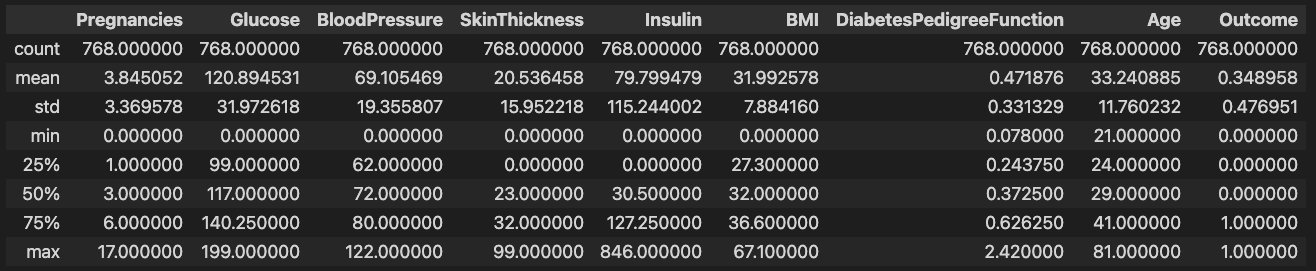
* + The best model coefficients in the above are close but not the same as the values of alpha used in Ridge regression were more than those done manually.

1. Lasso Regression
   * To use Lasso Regression, sklearn library was used
   * Function to calculate RMSE error for ridge regression was written
   * Minimum RMSE and its corresponding alpha and theta values were stored. They came out to be:

# Question 2. Logistic Regression

1. Loading the dataset
   * The dataset was loaded and converted to DataFrame using the pandas library.
2. Preprocessing the data
   * Basic info about the data was obtained.



* + The data was randomly divided into training, validation and testing datasets in the ratio of 7:2:1
  + All the parameters were converted to np.float128 datatype
  + The train and test sets were divided into X\_train, y\_train, X\_validation, Y\_validation and X\_test, y\_test respectively such that first 8 columns were part of X and last column was part of y
  + The data was then normalized such that each column had mean 0 and standard deviation 1
    - It was made sure that the test and validation sets were normalized using the mean and standard deviation of training data itself.

X\_mean = np.mean(X\_train, axis=0)

X\_std = np.std(X\_train, axis=0)

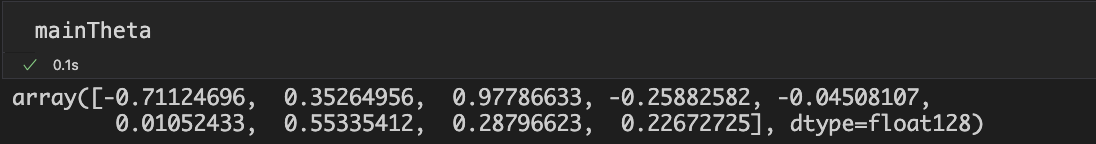
X\_train = (X\_train – X\_mean)/X\_std

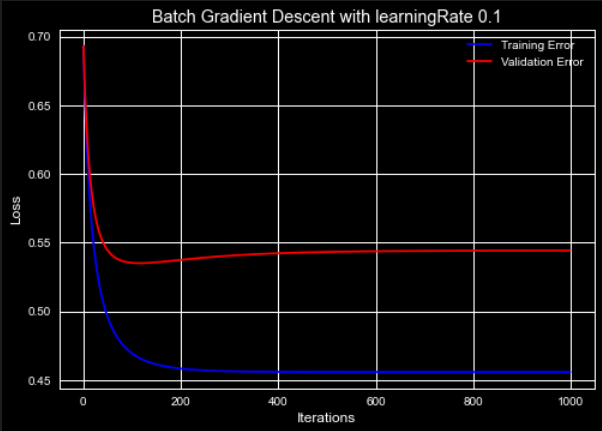
X\_validation = (X\_validation – X\_mean)/X\_st

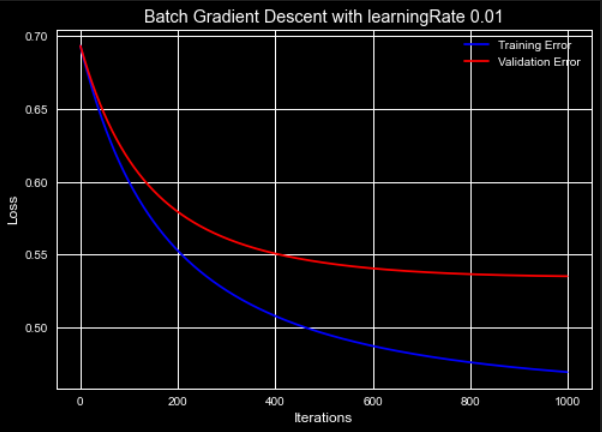
X\_test = (X\_test – X\_mean)/X\_std

* + A column of 1’s was added in both the training and the test sets so as to denote X0

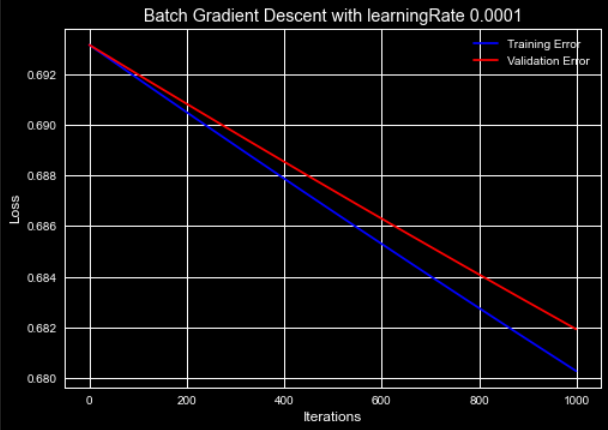
1. Performing Logistic Regression
   * Certain functions were made to calculate:
     + sigmoid
     + hypothesis
     + error
     + gradient
2. Batch Gradient Descent(BGD)
   * Batch Gradient Descent was performed
   * The final theta parameters as reported by the model are:



* + A function to draw plots for both training and validation data for the BGD performed was written
  + Plots for various learning rates were as follows:
    - Learning Rate = 0.1
    - Learning Rate = 0.01



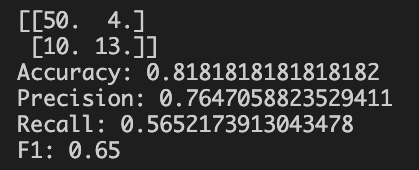
* + - Learning Rate = 0.001



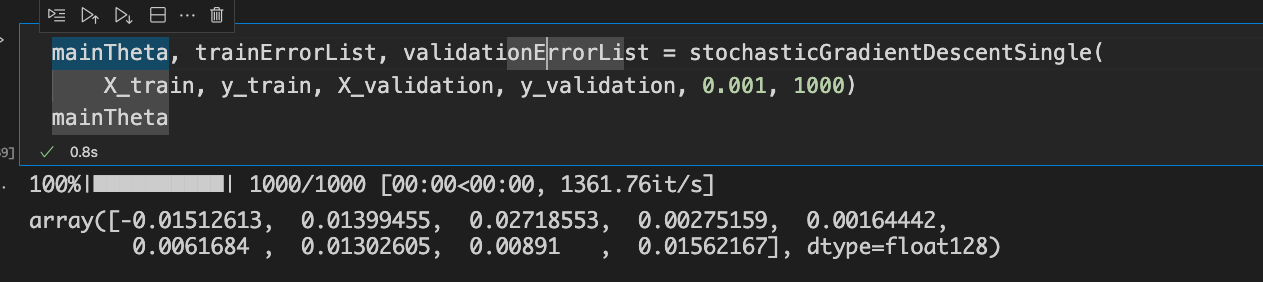
* + - Learning Rate = 10

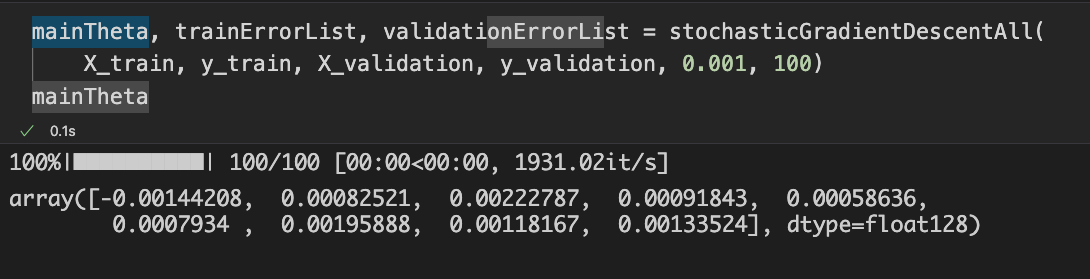


* + Functions to get the following were written:
    - prediction list
    - Confusion matrix, accuracy, precision, recall and F1 scroe
  + Confusion matrix, accuracy, precision, recall and F1 score for the BGD were:



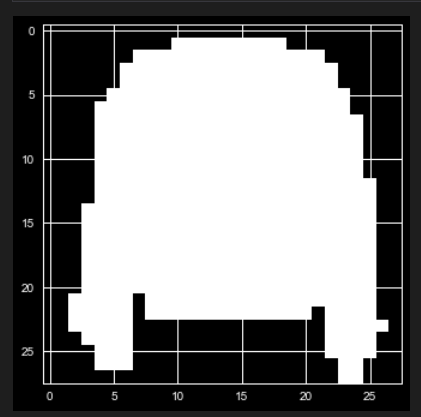
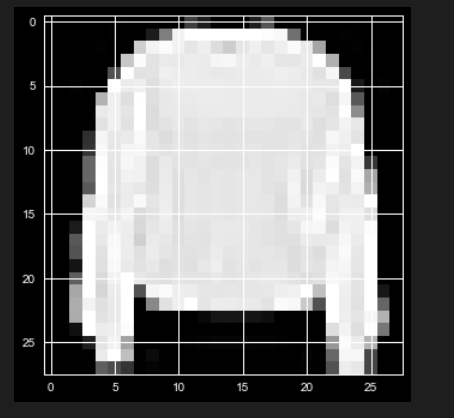
1. Stochastic Gradient Descent
   * For performing stochastic gradient descent, 2 functions were created.
     + StochasticGradientDescentSingle(): It randomly picked 1 value from the entire training dataset for each epoch and updated the theta correspondingly
     + StochasticGradientDescentAll(): For each epoch, it iterated over all datapoints once and updated theta value corresponding to each datapoint
   * The final theta values for both the implementations are as follows:
     + For StochasticGradientDescentSingle():



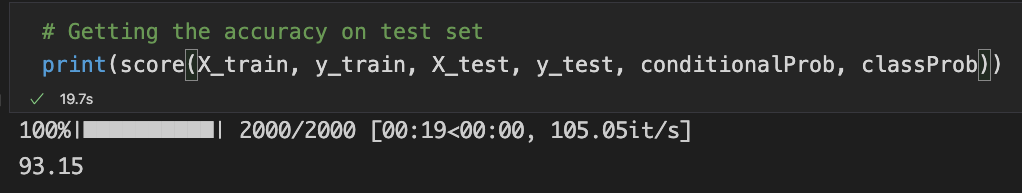
* + - For StochasticGradientDescentAll():
    - 

# Question 3. Naïve Bayes

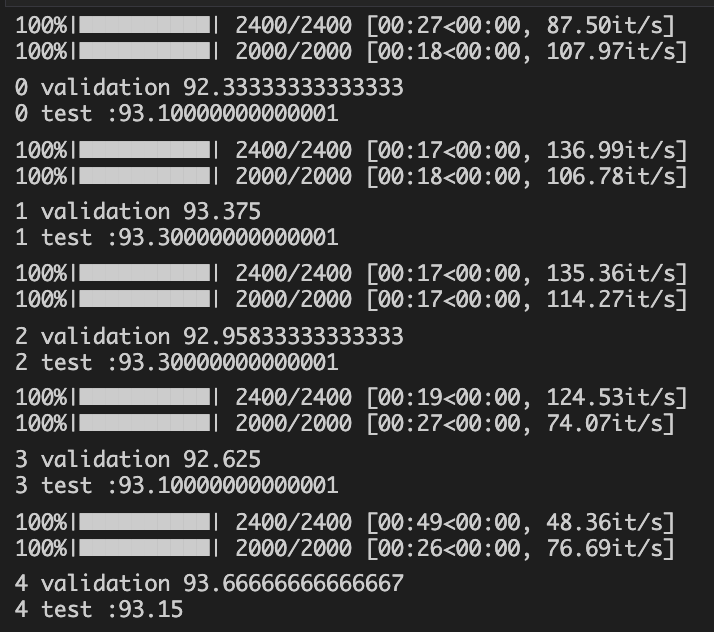
1. Loading the dataset
   * To load the dataset, the function available in the github repo provided to us was used.
   * The data was directly loaded in X and y, training and testing divisions
   * Copy of the above loaded dataset was created with the same variable names so as to make the data editable
2. Preprocessing the Dataset
   * From the github repo, it was found that the labels for trousers and pullovers were 1 and 2
   * Using this information at hand, the complete dataset was pruned to remove the data having any other label other than 1 and 2
   * For ease in the future tasks, the labels 1, and 2 were converted to 0 and 1 respectively.
   * The data had to be binarized as given in the problem statement
   * To do this, the threshhold was fixed at 128.
   * All the values above 128 were made 255 and all those under it were made 0
   * Sample image before and after binarizing:



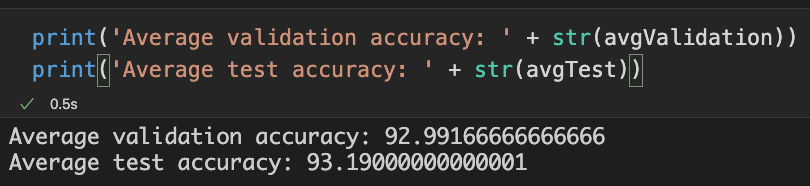
1. Implementing Naïve Bayes
   * Certain functions were made to compute the following:
     + Prior Probability
     + Conditional Probability
     + List of all prior probabilities and conditional probabilities
     + Prediction Labels
     + Accuracy of the model
   * Naïve Bayes was performed on the training data and the class probabilities and conditional probabilities were obtained
   * Using the above, the model was run on testing set and the following accuracy was obtained



1. K-Fold Cross-Validation
   * the value of K was taken to be 5
   * the data was shuffled and then broken down into 5 sets of equal length
   * the above made model was trained again every time taking 4 out of 5 sets and taking the remaining as validation data
   * every time the model was retrained and accuracy for both validation and testing data was calculated



* + The average test and validation accuracies over all K validation sets were also calculated



1. Confusion Matrix:
   * Confusion matrix was calculated over the entire training data and it was used to calculate precision, recall, accuracy and F1 score

