[CSE343/ ECE343] Machine Learning

# REPORT

Assignment: Linear / Logistic Regression and Naive Bayes

Submitted By
Dolly Sidar
2019304
CSD

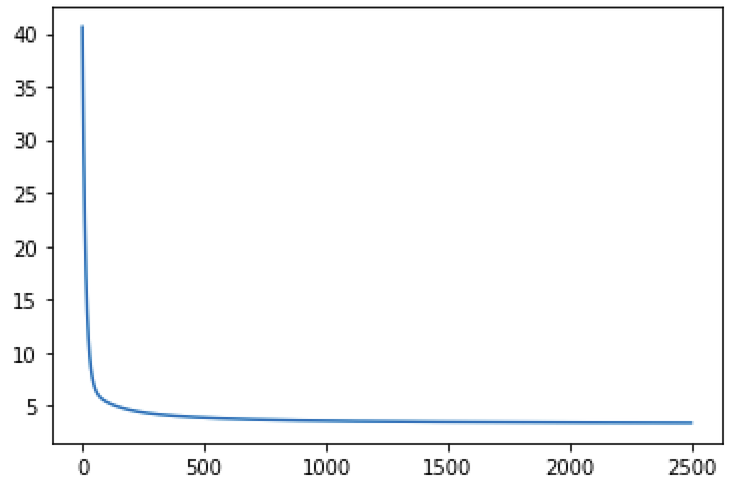
## Question 1: Linear Regression

### 1.1.

Step 1 -  Imported the  important libraries and loaded the dataset

Step 2 - preprocessing and splitting the dataset into 8:2 of train and test
Assumptions-
1.  As regression analysis requires a numerical variable, so to fit the categorical variable of column Sex(Character) into the model we have to create dummy variables i.e M=1 F=2 I=0.
2. Shuffling of data to ensure iid
3. Separate features and the target variable
4. Dividing the data into training and testing sets

Step 3 - Implementing the cost and gradient descent function from scratch
Note - Implemented the cost function just to see the performance of the model against each iteration



we can see how the cost is decreasing and approaching zero.

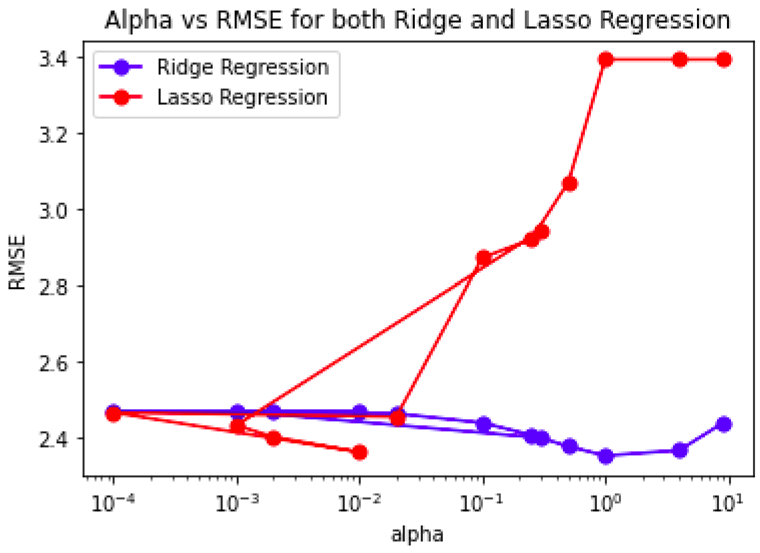
Step 4- Evaluating the model
Calculated RMSE on training and testing data

```
RMSE for training data  : 2.6020211149087733, RMSE for testing data : 2.78200861361564
```

### **1.2.a** Using Sklearn's Ridge and Lasso implementation

As we know lower values of RMSE indicate better fit, so I have used the minimum calculated RMSE in an array and then find its value of coefficient when in turn the best coefficient among all others.
Assumption -
alpha = [0.3, 0.001, 0.002, 0.01, 0.0001, 0.02, 0.1, 0.25, 0.5, 1, 4, 9]

```
Best model coefficients in Lasso:
[-6.92788529e-03  0.00000000e+00  7.02251102e+00  0.00000000e+00
  4.62919697e+00 -1.35740546e+01 -0.00000000e+00  1.29932760e+01]
Best model coefficients in Ridge:
[ 2.17702354e-04  2.00232104e+00  7.30998612e+00  1.27771640e+01
  6.96501139e+00 -1.70601236e+01 -6.65028574e+00  1.02662711e+01]
```



## 1.2.b Using Sklearn's Grid Search Function

Calculated the best model coefficients and alpha values using the best_estimator function. The GridSearchCV instance fitted with the best hyperparameters is stored in best_estimator_. The coef_ and intercept_ are the fitted parameters of that best model.

```
Lasso best alpha : 0.0001
Lasso best coef : [ 1.05626470e-02 -1.81023244e+00  1.06972874e+01  2.50760776e+01
  8.98929451e+00 -1.94063956e+01 -1.09133080e+01  7.68309185e+00]
Ridge best alpha : 0.1
Ridge best alpha : [ 9.64905191e-03 -1.35158940e+00  1.04107361e+01  2.29255383e+01
  8.73339103e+00 -1.91518049e+01 -1.03794786e+01  8.16059198e+00]
```

**Comparison of the best model coefficients reported in Q1.2.a.**
The best model coefficients founded by the Grid search function are slightly different from the first estimation i.e Q1.2.a where we used min RMSE to find the best coefficients. Lasso and ridge are regularizing linear models that try to search out the best combination of coefficients, but we can't say that min RMSE gives the best coefficient model. On the other hand grid function by default uses the cross-validation scheme to make the model more generalize, it builds the model for every given hyperparameter and takes the best model. It gives the best coefficient model having the best hyperparameters value.

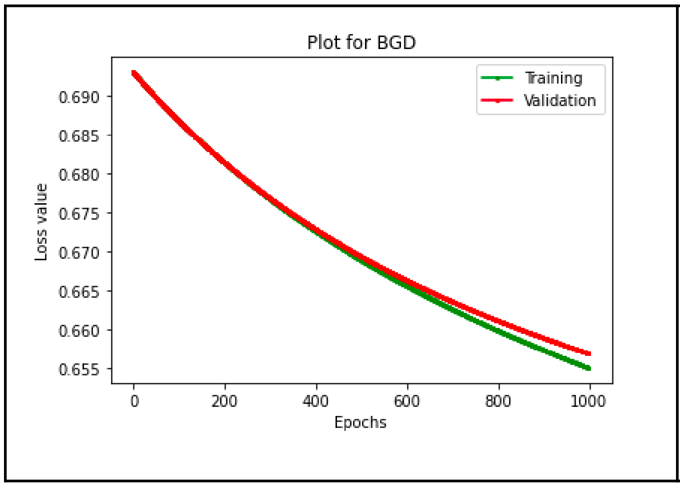
## Question 2: Logistic Regression

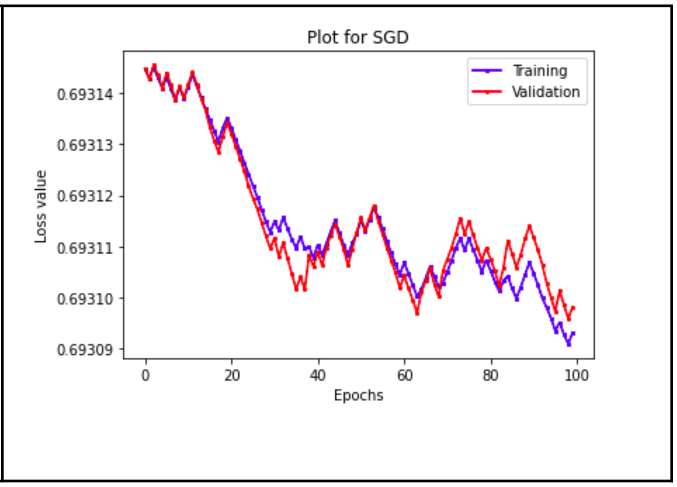
### 2.1.a

1. Importing the required libraries
2. Loading the dataset
3. preprocessing and splitting the dataset into training and testing sets

As data points have 0 values i.e we have some 0 values in the pregnancies column and other columns, also there is a disturbance in input value, so to handle this.
Assumptions :
1. Shuffling of data to ensure iid
2. Replacing the 0 values of all the columns to a median of its column
3. performing mean normalization to change the range of input variable as there is noise in the data
4. Dividing the data into training, validation, and testing sets 7:2:1
5. Separate features and target

4. Implementing the Loss and Gradient descent function from scratch



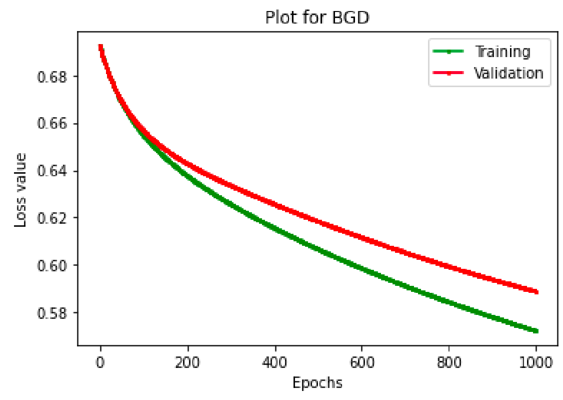


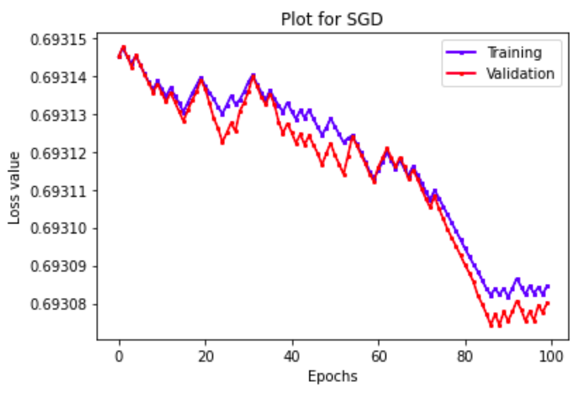
**Comment on the convergence of the model.**

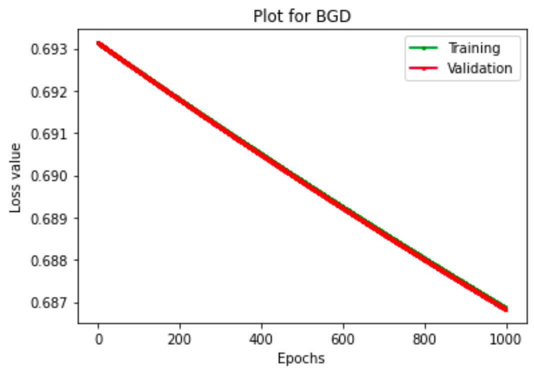
In both the graph we can see that both the graph starts approaching the local minimum as the number of epochs increases. In the BGD graph, the curve gradually decreases, at first the loss is very high but as iteration increases, it started stabilizing and after a certain value, there is not much reduction in the loss. On the other hand, the graph of SDG is scattered , because in stochastic we used one randomly picked training sample to calculate the loss.

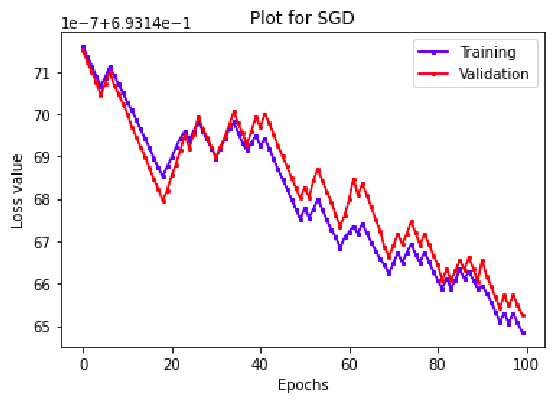
## 2.1.b

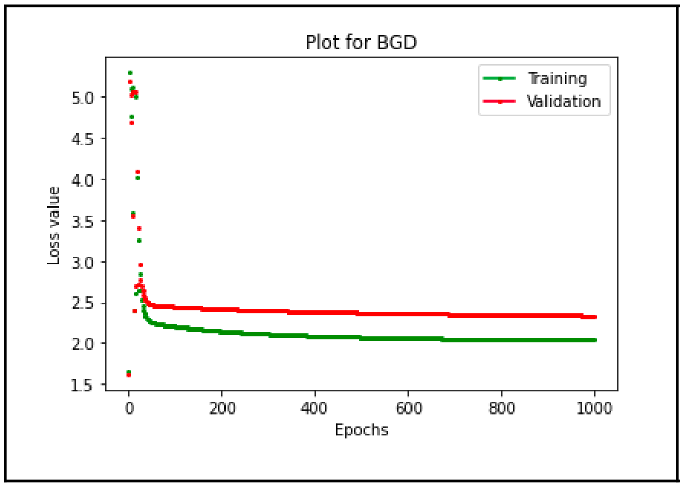
1. For a learning rate of 0.01
2. For a learning rate of 0.0001
3. For a learning rate 10

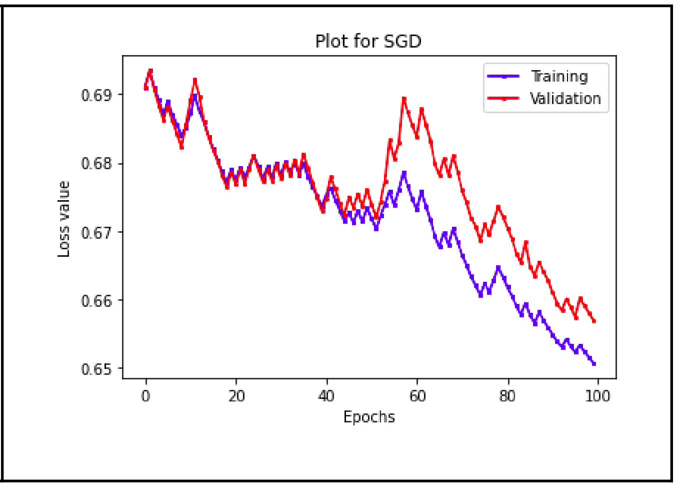








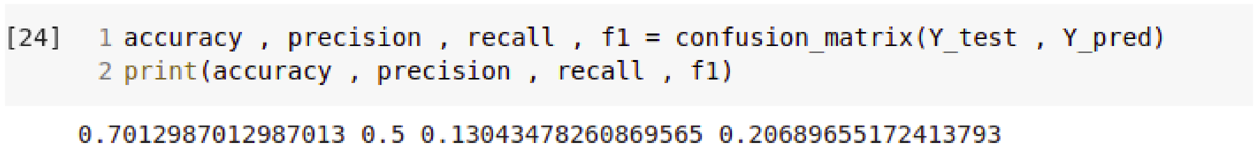




**Compare and give your analysis.**
As we can see above that at a learning rate of 0.01 the BGD and SGD performed best which is not higher or not lower learning rate, for a smaller value of learning rate i.e 0.0001 the model taking a lot of time to converge or to reach the local minimum whereas, for the large learning rate, it will overshoot as we can see in the last graph which is performed on learning rate 10, the learning rate is directly proportional to an increase in the rate of minimization of the cost function but sometimes it overshoots there we should pick the best learning rate by trying the different values as there is no specific rule for the perfect learning rate.

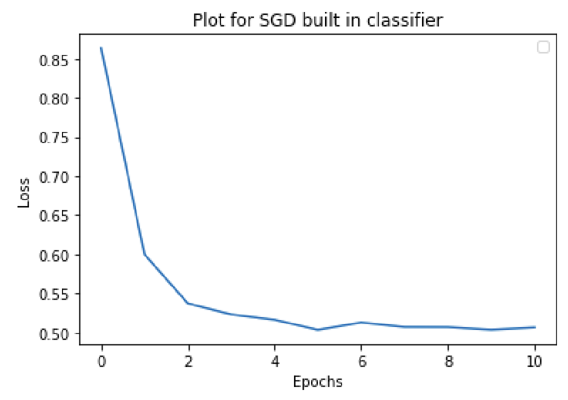
## 2.1.c

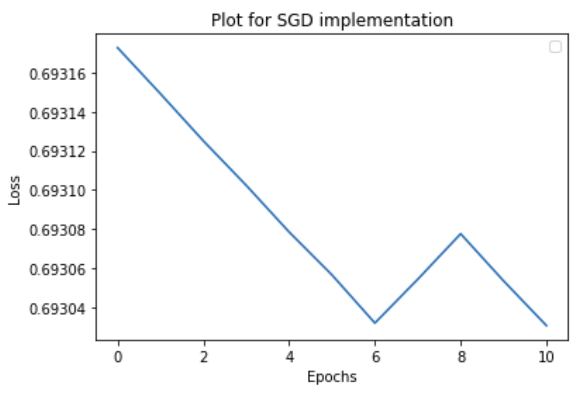
Calculated accuracy, precision, recall, and f1score



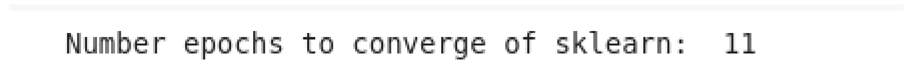
```
[24]  accuracy , precision , recall , f1 = confusion_matrix(Y_test , Y_pred)
      print(accuracy , precision , recall , f1)

    0.7012987012987013 0.5 0.13043478260869565 0.20689655172413793
```

## 2.2.a Loss plots of sklearn and your implementation.



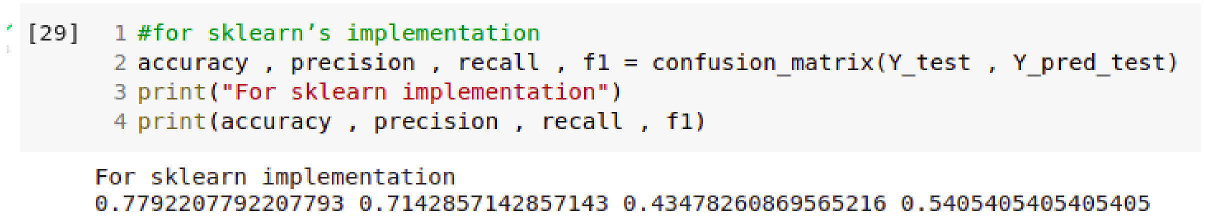


Number epochs to converge on sklearn and own implementation.

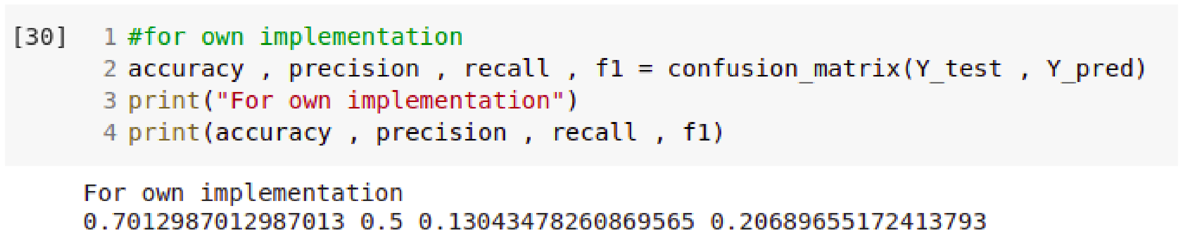


```
Number epochs to converge of sklearn:  11
```

Performance sklearn’s implementation and own implementation. Report accuracy, precision, recall, and f1 score.



```
[29]  #for sklearn’s implementation
      accuracy , precision , recall , f1 = confusion_matrix(Y_test , Y_pred_test)
      print("For sklearn implementation")
      print(accuracy , precision , recall , f1)

   For sklearn implementation
   0.7792207792207793 0.7142857142857143 0.43478260869565216 0.5405405405405405
```



```
[30]  #for own implementation
      accuracy , precision , recall , f1 = confusion_matrix(Y_test , Y_pred)
      print("For own implementation")
      print(accuracy , precision , recall , f1)

   For own implementation
   0.7012987012987013 0.5 0.13043478260869565 0.20689655172413793
```

## Question 3: Naive Bayes

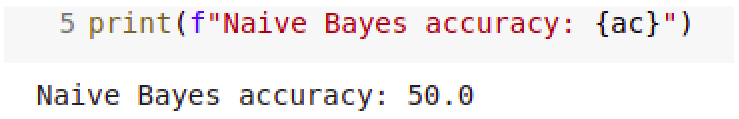
### **3.1.a** Naive Bayes using Train and Test Split

1. Importing the required libraries
2. Loading the dataset
3. preprocessing and splitting the dataset into training and testing sets

Assumptions :

1. Shuffling of data to ensure iid
2. Extracting the required data i.e trouser and pullover label
3. Dividing the data into target and feature sample
4. Binarizing the images to 0, 255

4. Implementing the Naive Bayes from scratch
The accuracy comes out to be

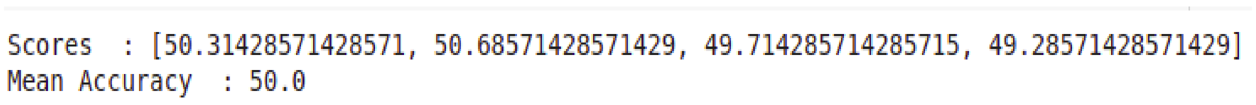


```
print(f"Naive Bayes accuracy: {ac}")
```

```
Naive Bayes accuracy: 50.0
```

### **3.1.b** Naive Bayes using k-fold Cross Validation Split

The accuracy comes out to be 50.0, taken k = 4



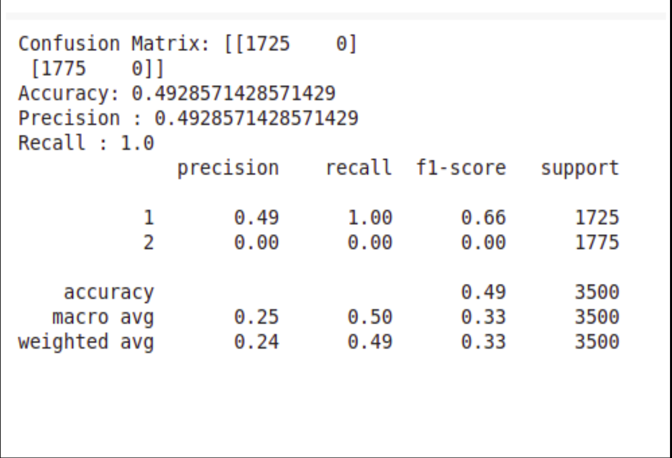
```
Scores  : [50.31428571428571, 50.68571428571429, 49.714285714285715, 49.28571428571429]
Mean Accuracy  : 50.0
```

Preprocessing
1. Spilt the dataset into k fold
Fold size = total rows / total folds
Best value of K = 4

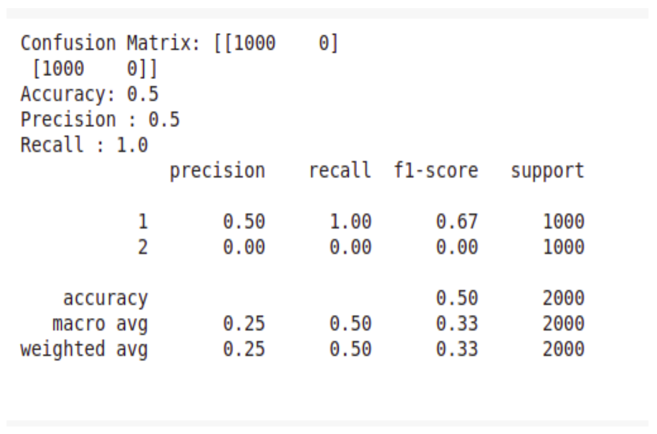
### **3.1.c** Making the Confusion Matrix and calculating accuracy, precision, recall

(a) Make the Confusion Matrix
(b) Plot the ROC curve
(c) Find the Accuracy, Precision, Recall

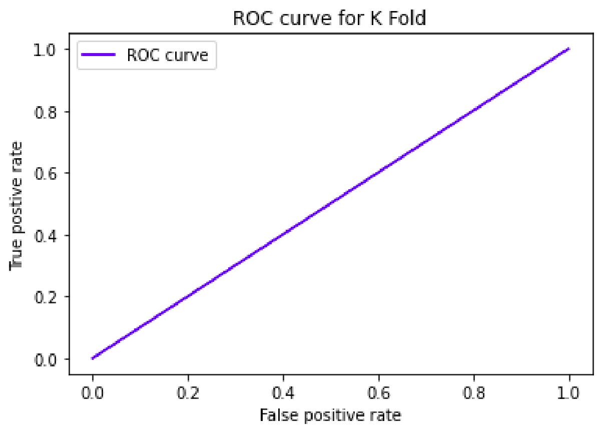
| For k-fold Naive Bayes | Simple Naive Bayes |
|---|---|

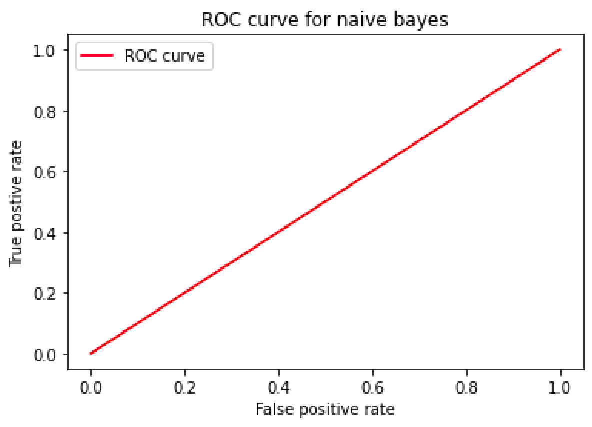


```
Confusion Matrix: [[1725    0]
 [1775    0]]
Accuracy: 0.4928571428571429
Precision : 0.4928571428571429
Recall : 1.0
              precision    recall  f1-score   support

           1       0.49      1.00      0.66      1725
           2       0.00      0.00      0.00      1775

    accuracy                           0.49      3500
   macro avg       0.25      0.50      0.33      3500
weighted avg       0.24      0.49      0.33      3500
```



```
Confusion Matrix: [[1000    0]
 [1000    0]]
Accuracy: 0.5
Precision : 0.5
Recall : 1.0
              precision    recall  f1-score   support

           1       0.50      1.00      0.67      1000
           2       0.00      0.00      0.00      1000

    accuracy                           0.50      2000
   macro avg       0.25      0.50      0.33      2000
weighted avg       0.25      0.50      0.33      2000
```





### 4.1.a

We can test the suspicion by adding the datasets into one and then create a new dummy variable that identifies the condition and then runs the model for both the dataset and if the final output for both the model gives different values then it is suspicious.

### 4.1.b

We can test the suspicion by adding the datasets into one and then create a new parameter that identifies the condition and then runs the model for both the dataset and if the final output for both the model gives different values then it is suspicious.

### 4.1.c

We can test the suspicion by adding the datasets into one and then create a new parameter that identifies the condition and then runs the model for both the dataset and if the final output for both the model gives different values then it is suspicious.

## 4.2

L2 regression promotes smaller coefficients, shrinking the coefficients close to zero with a minor contribution to the outcome. The shrinkage of the coefficients is attained by penalizing the model with a penalty  L2-norm, which is the sum of the squared coefficients.

## 4.3

4) ~~B~~ Bayes theorem says

$$\text{Posterior} = \frac{\text{Likelihood} \cdot \text{prior}}{\text{evidence}}$$

$$P(\theta|y) = \frac{P(y|\theta)\,P(\theta)}{P(y)}$$

$$\hat{\theta}_{map} = \arg\max_{\theta} P(\theta|y)$$
$$= \arg\max_{\theta} \frac{P(y|\theta)\,P(\theta)}{P(y)}$$
$$= \arg\max_{\theta} P(y|\theta)P(\theta)$$
$$= \arg\max_{\theta} \log\left(P(y|\theta)P(\theta)\right)$$
$$= \arg\max_{\theta} \log\left(P(y|\theta)\right) + \log P\theta \quad \text{—} \textcircled{1}$$

Thus, $\text{prior} = \prod_{j=0}^{p} \frac{1}{\tau\sqrt{2\pi}} e^{-\frac{\beta_j^2}{2\tau^2}} = P(\theta)$ — ②

$$\text{likelihood} = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y_i-(\beta_0+\beta_1 x_1 \cdots \beta_p x_{ip}))^2}{2\sigma^2}} = P\left(\frac{y}{\theta}\right) \quad \textcircled{3}$$

substituting ② and ③ in ①

$$\hat{\theta}_{map} \Rightarrow \arg\max_{\beta}\left[\log \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-(y_i-(\beta_0+ \cdots - \beta_p x_{ip}))^2} + \log \prod_{j=0}^{p} \frac{1}{\tau\sqrt{2\pi}} e^{-\frac{\beta_j^2}{2\tau^2}}\right]$$

$$\Rightarrow \arg\min \frac{1}{2\sigma^2}\left[\sum_{i=1}^{n} (y_i-(\beta_0+\beta_1 x_1 + \ldots + \beta_p x_{i,p}))^2 + \frac{\sigma^2}{\tau^2}\sum_{j=0}^{p} \beta_j^2\right]$$

$$\Rightarrow \arg\min_{\beta} \left[\sum_{i=1}^{n} (y_i - (\beta_0+\beta_1 x_1 + \ldots \beta_p x_{ip}))^2 + \lambda \sum_{j=0}^{p} \beta_j^2\right]$$