Assignment 1

Source Files included: Hw_10602864.py, question1.py, question2.py, question3.py, preprocessing.py, regression.py, evaluation.py, plot.py.
Output files included: StudentID_1, StudentID_2

Question 1: File "question1.py" implements all the steps of question 1.

a) Step by step process is given below.
   - After loading the data, the required columns data is separated. The shape of data which we obtain is (800 *18).
   - The categorical columns in data were then then converted to hot encoded data. As all the categorical columns are binary categorical, a single column can represent each original column. So, there is no change in shape of data. The function "one_hot_encoding" in file "processing.py" does this task.
   - The data is then split into train/test. We take 800 samples randomly from the data and call it our Train_data. Those samples which are not included in Train_data we call it Test data. Finally, we separate the G3 column from both Train data and test data and call it Train_y and Test_y respectively. The function "one_hot_encoding" in file "processing.py" does this task. The function "split_data" in file "processing.py" does this task.
   - The train and test data were then normalized by subtracting mean and from the data and dividing it by standard deviation. The function "normalize" in file "processing.py" does this task.

   At the end of this task we have train_X, train_Y, test_X, test_Y. This data will be further used in all parts of question 1.

b) The RMSE value of Linear regression without bias term and regularization is 11.937880503702567. The function "LReg" implemented in "regression.py" file does this task when 0 is passed as lambda.

c) The RMSE value of Linear regression when regularization term (with λ =1.0) is used is 11.983046478107665. The RMSE increased because we have added a penalty term. The penalty term was added to MSE while Lambda*identity term was also added in weight calculation step. The function "LReg" implemented in "regression.py" file does this task when 1 is passed as lambda.
   The regression is modeled as gaussian random variable $P(\hat{y}x,W)=N(\hat{y}|WTX,\sigma 2)$.
   Optimal weights can be calculated as $WMLE=arg\max WN(\hat{y}|WTX,\sigma 2)$. We can simplify this equation by using PDF of gaussian and removing the normalizing constant $arg\max W\exp(-(\hat{y}-WTX)\text{^}2\sigma 2$ Taking the natural log we get the cost function. We can use minimization to calculate the optimal weights. $arg\min W12\sigma 2(\hat{y}-WTX)2$. We will update our weights iteratively through the above equations until we get optimal weights.

d) The RMSE value of Linear regression with bias term and with regularization term (with λ=1.0) is 3.323889582851475. The function "LReg_with_bias_reg" implemented in "regression.py" calculates RMSE for this method.

e) The RMSE of Bayesian Linear regression recorded is **3.3248682419662394**. The function "Bayesian_LR" implemented in "regression.py" calculates RMSE for this method.

f) When I plotted all together shown in Figure 1, (c) hides (b) and (e) hides (d). So, I also plotted it separately in Figure 2,3,4,5. The function "all_reg" in "plot.py" is implemented to plot these graphs.

Figure 1. linear regression (b, c, d, e)

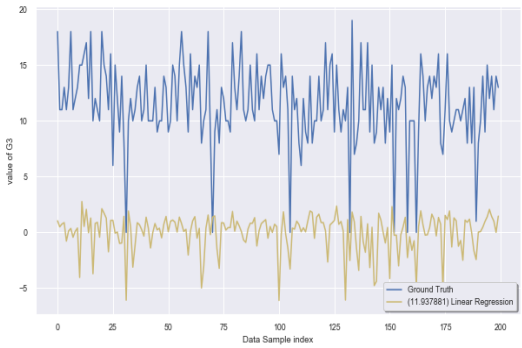

Figure 2. linear regression (b)

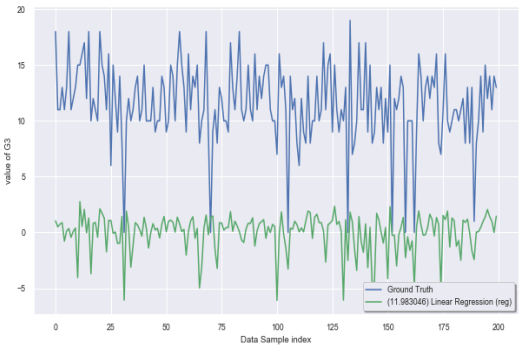

Figure 3. linear regression (c)

Figure 4. linear regression (d)

Figure 5. linear regression (e)

Question 2: File "question2.py" implements all the steps of question 2.

In classification task, the target row "G3" is first converted to binary values based on threshold value 10. The same preprocessing steps explained in question 1(a) is used to get Train/test data and labels.

a) A linear regression method with bias and regulation term 1.0 is used to predict the test class values.
First "LReg" function of "regression.py was used to get values for target class. The values were then converted to binary using threshold values = 0.1, 0.5, 0.9. For each of these thresholds TP, TN, FP, FN values were found by "get_confusion_matrix" implemented in "evaluate.py". Accuracy and Precision values were calculated by "accuracy_precision" function of "evaluate.py". Figure 6,7,8 shows the results.

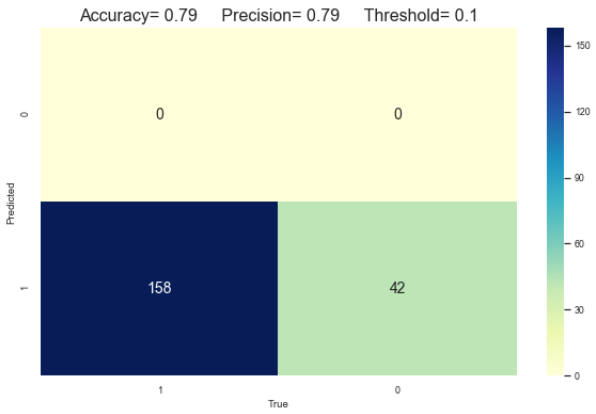

Figure 6. Linear regression (Threshold=0.1)

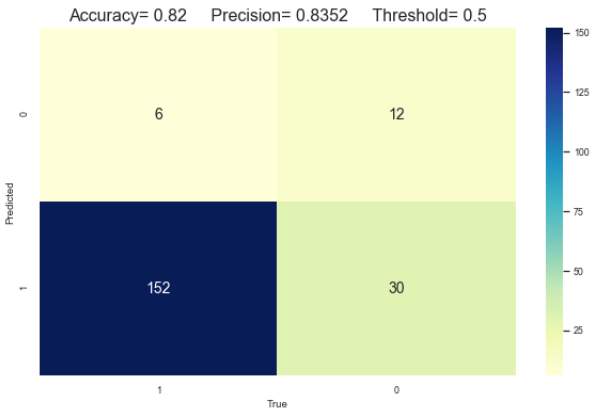

Figure 7. Linear regression (Threshold=0.5)

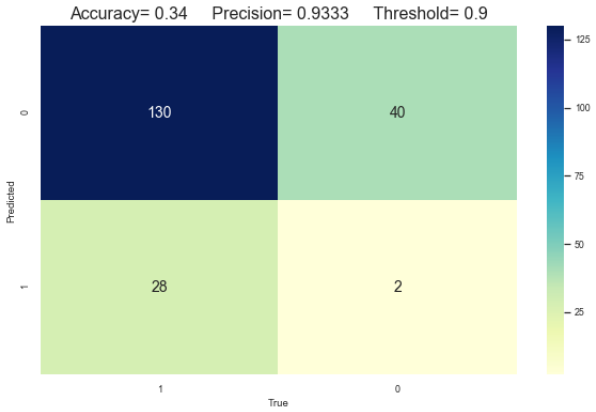

Figure 8. Linear regression (Threshold=0.9)

b) The method "Logistic_regression" in "regression.py" implements Logistic regression.
The loss directly depends upon weight values. To minimize loss, we have to fit the weights. We get optimize weights by iteratively updating weights. We get updated weights by subtracting gradient times learning rate.
weights = weights – (learning rate * gradient)
The gradient is basically the derivative of loss function with respect to each weight shown below.

gradient= $\frac{1}{m} X^T\left(g(X\theta) - y\right)$ Where $g(z) = \frac{1}{1 + e^{-z}}$ and $z = \text{weightsT} * X$

For each of these thresholds TP, TN, FP, FN values were found by "get_confusion_matrix" implemented in "evaluate.py". Accuracy and Precision values were calculated by "accuracy_precision" function of "evaluate.py". Figure 9,10,11 shows the results.

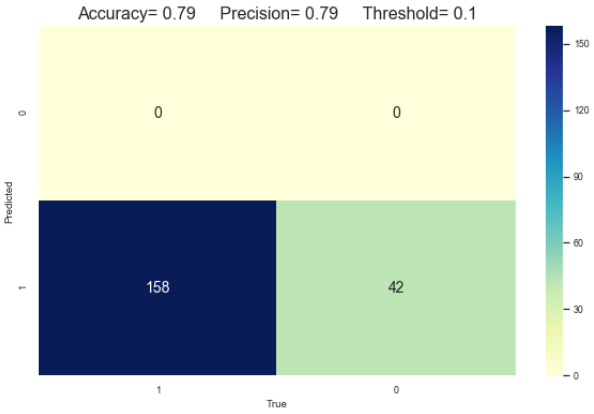

Figure 9. Logistic Regression (Threshold=0.1)

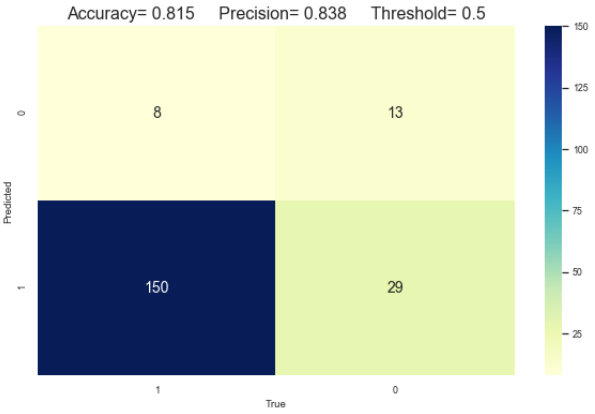

Figure 10. Logistic Regression (Threshold=0.5)

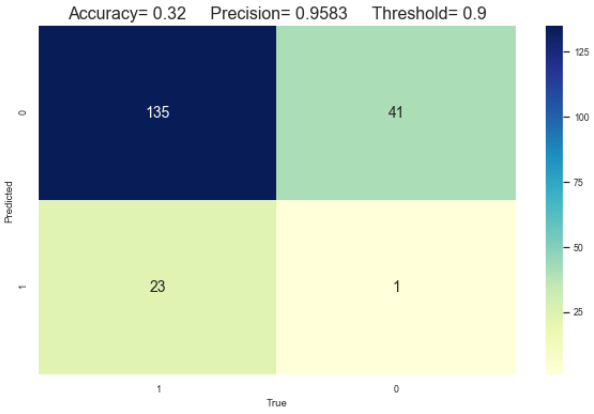

Figure 11. Logistic Regression (Threshold=0.9)

c) The confusion matrices for threshold= 0.5. Figure 13 and Figure 14 shows the results.

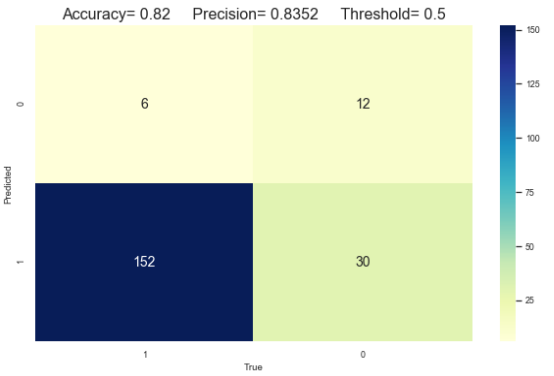

Figure 12. Linear regression (Threshold=0.5).

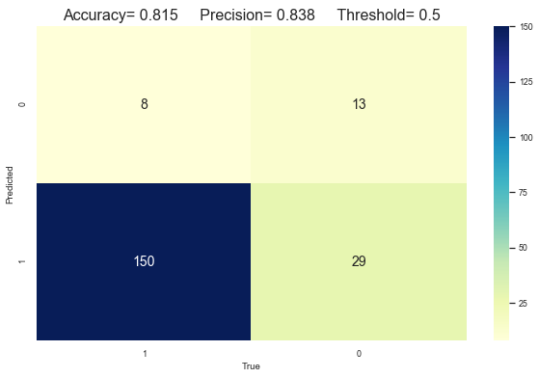

Figure 13. Logistic Regression (Threshold=0.5)

d) The confusion matrices for threshold= 0.9.

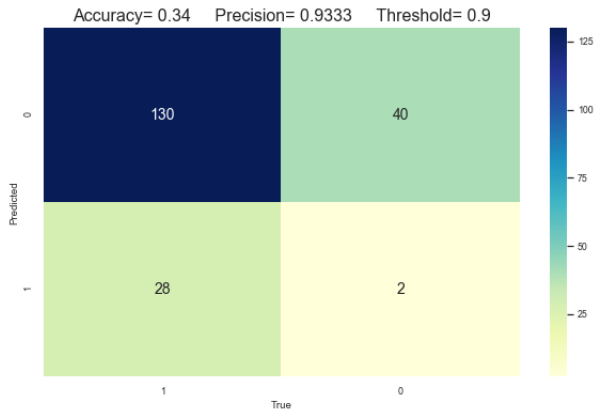

Figure 14. Linear regression (Threshold=0.9).

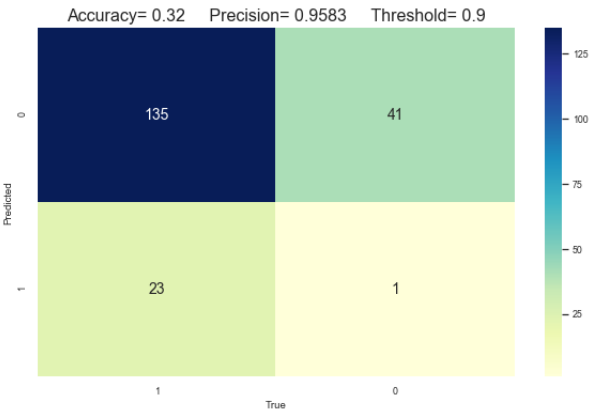

Figure 15. Logistic regression (Threshold=0.9).

e) Increasing the threshold value from the low to high i.e. 0.5 to 0.9 decreases the accuracy. Values which are predicted represent confident. If its higher it represents its high confident on class one and if its low it means class zero. So, increasing the threshold means because increasing the threshold means we are classifying majority of the values into one class.
The summary of accuracy and precision is shown on every confusion matrix plot. A summary is also given in Table 1.

		Threshold=0.1	Threshold=0.5	Threshold=0.9
Linear Regression	Accuracy	0.79	0.82	0.34
	Precision	0.79	0.83	0.93
Logistic Regression	Accuracy	0.79	0.81	0.83
	Precision	0.79	0.32	0.95

Table 1.Accuracy and Precision

Question 3: File "question3.py" implements all the steps of question 3.

a) Instances of "test_no_G3" are predicted using model made in 1(d) that is linear regression with bias and regularization. File "student_id_1" is created and the result is saved in it against the index number.
[12.08, 11.7, 11.21, 11.78, 11.99, 10.97, 11.6, 13.24, 11.59, 10.14, 12.55, 9.82, 12.47, 12.43, 10.99, 12.43, 4.6, 13.73, 6.83, 12.84, 11.67, 10.5, 10.69, 11.83, 11.75, 11.42, 13.11, 12.43, 11.71, 9.41, 10.87, 6.98, 12.01, 11.66, 12.17, 12.95, 12.59, 12.22, 11.51, 12.95, 6.2, 12.22, 11.8, 11.16]

b) Instances of "test_no_G3" are predicted using model made in 2(b) that is logistic regression. File "student_id_1" is created and the result is saved in it against the index number.
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1]

Flow chart of Source code:

Run the hw1_studentID.py file to see all the results at once.