ML Assignment 1

Name: Meetakshi Setiya
Roll No: 2019253

Note: everywhere "iteration" is used in the output of Kfold, it means the "fold".

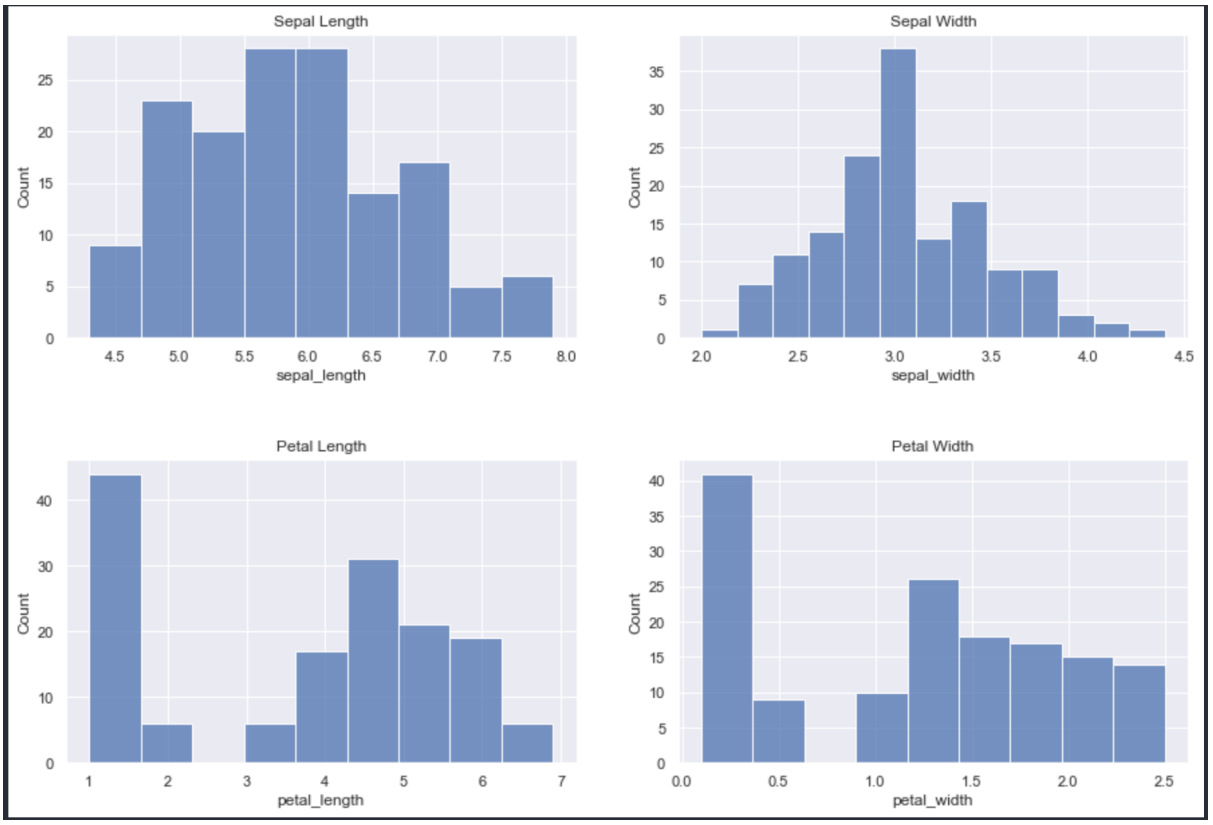
1. *The code for this question is given in the file* `question1.ipynb`

1.
(a). I have used `urlretrieve()` to load the dataset from its url without downloading it. I have also mentioned a commented command if the IRIS dataset has to be downloaded.

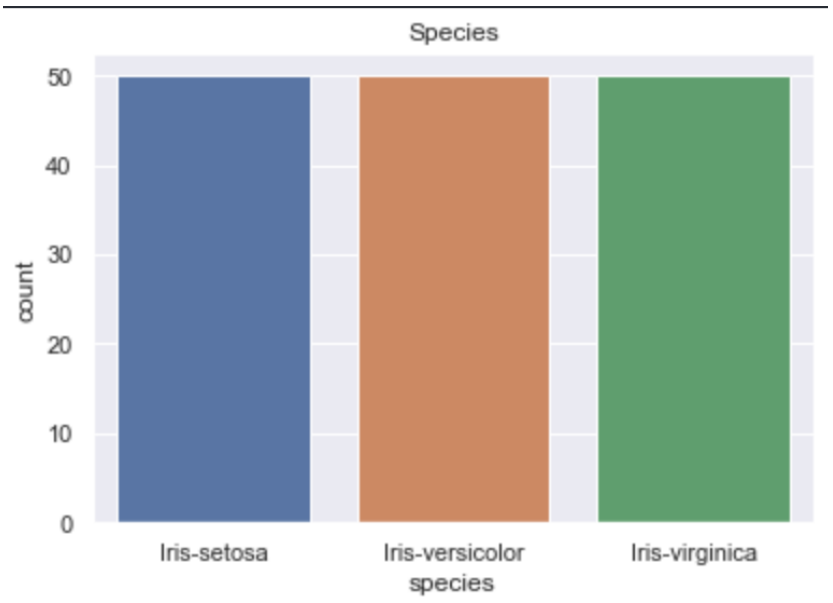
(b). This is the column information that I am printing for IRIS dataset:

```
Number of columns in the IRIS dataset: 5

--------------------------------

Column name: sepal_length

Data type: float64

Value range: 3.60

Number of non-null values: 150

--------------------------------

Column name: sepal_width

Data type: float64

Value range: 2.40

Number of non-null values: 150

--------------------------------

Column name: petal_length

Data type: float64

Value range: 5.90

Number of non-null values: 150

--------------------------------

Column name: petal_width

Data type: float64

Value range: 2.40

Number of non-null values: 150

--------------------------------

Column name: species

Data type: object

Unique values: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']

Number of instances for each label:
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: species, dtype: int64
```

(c) **Continuous valued attributes in the IRIS dataset:** `sepal_length, sepal_width, petal_length, petal_width.`
The histograms generated for these features are shown below:



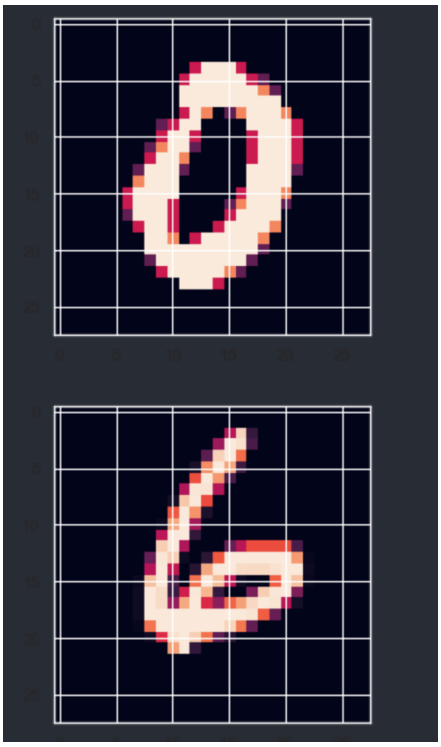
Discrete valued attribute(s) in the IRIS dataset: the target class `species.`
The generated barplot/countplot is given below:



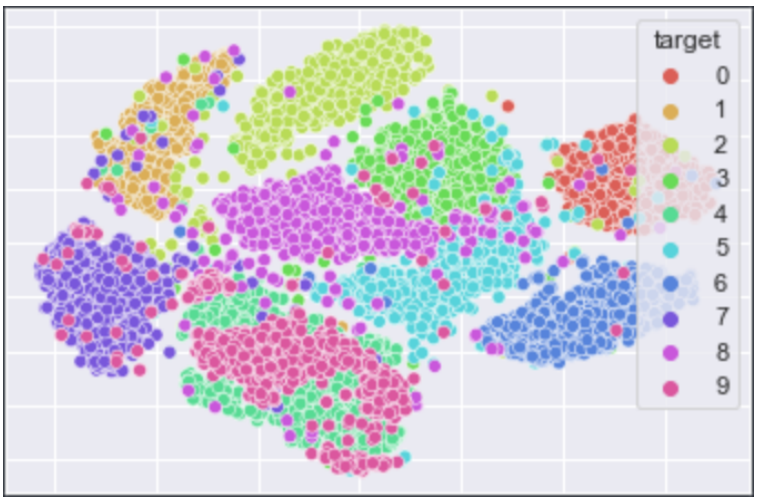
2.
(a). The dataset as gzip files is downloaded using `curl` and the corresponding commands have been provided in the jupyter notebook, commented. I used gzip() to extract index files from the zip files and then used `idx2numpy.convert_from_file()` to convert IDX files to numpy arrays.

(b). Two images are picked at random and visualised using `plt.imshow()`. `imshow()` might throw a warning when run on google colab, so for that, I have provided alternative commands that use `cv2_imshow()` from `google.colab.patches`, commented.

Example output:



(c). Since t-SNE would have taken a lot of time to run on the entire 60k images training set, I have randomly sampled 1000 images of each class (0 to 9) and used those to create a shortened dataset consisting of 10k images. t-SNE has been used on this shortened dataset.
Given below is the scatterplot after the data was reduced to 2 dimensions:



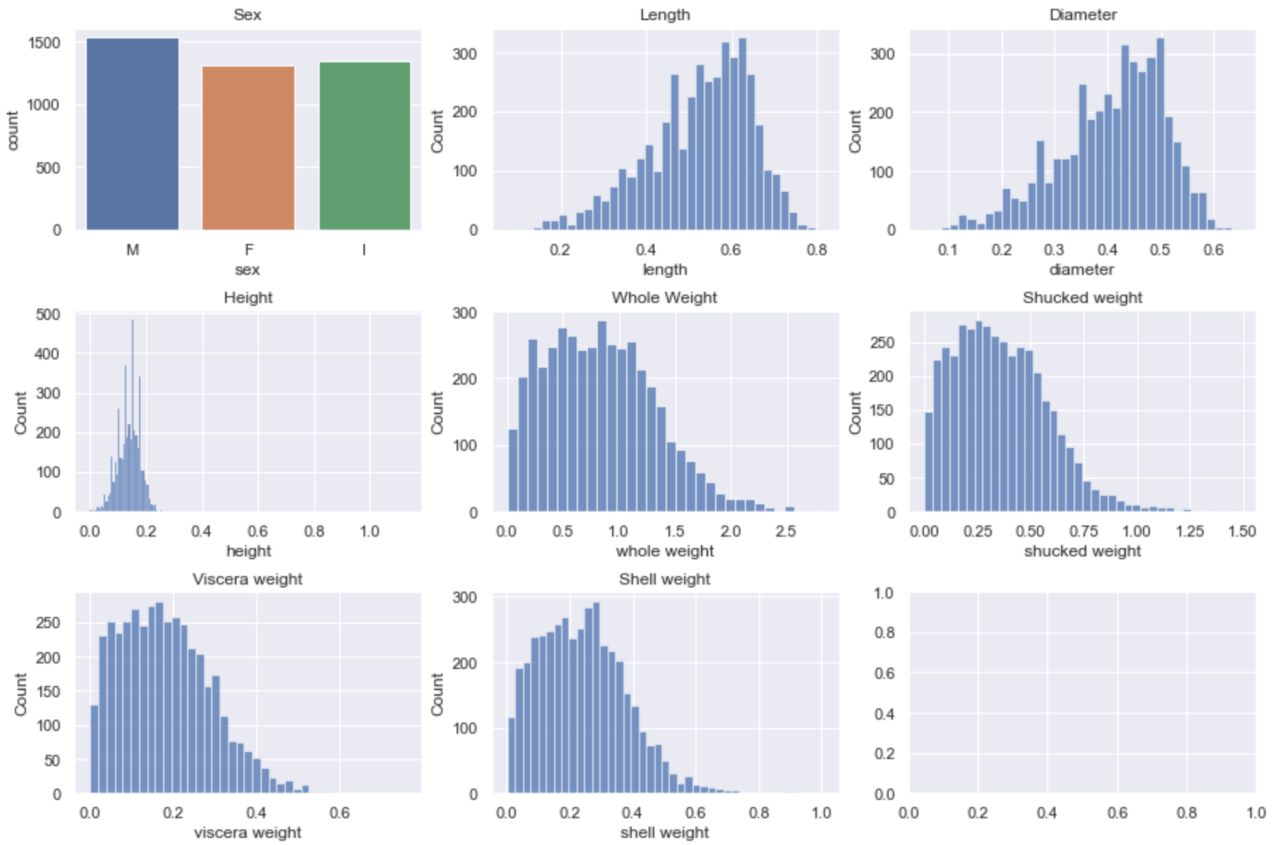
The clusters look pretty much separable although here are a few close observations:

- '5' has been separated into two clusters - one that seems to be closer to '3' and the other to '8'. In fact, '5', '3' and '8' are very close to each other.
- There are some '9's in '4's and '7's clusters too. These numbers have an almost similar letter stroke while writing them.
- '5's data points seem to be comparatively more spread out.
- There is linear separability in all classes except: '4' and '9', '5', '8' and '3'. Most of '1' is also linearly separable although there is a decent amount of noise because of confusion with other numbers.

---

2. *The code for this question is given in the file* `question2.ipynb`

1.
I have used `urlretrieve()` to load the Abalone dataset from its url without downloading it. I have visualised the different features of the dataset in the ipython notebook. Here is how it looks like:



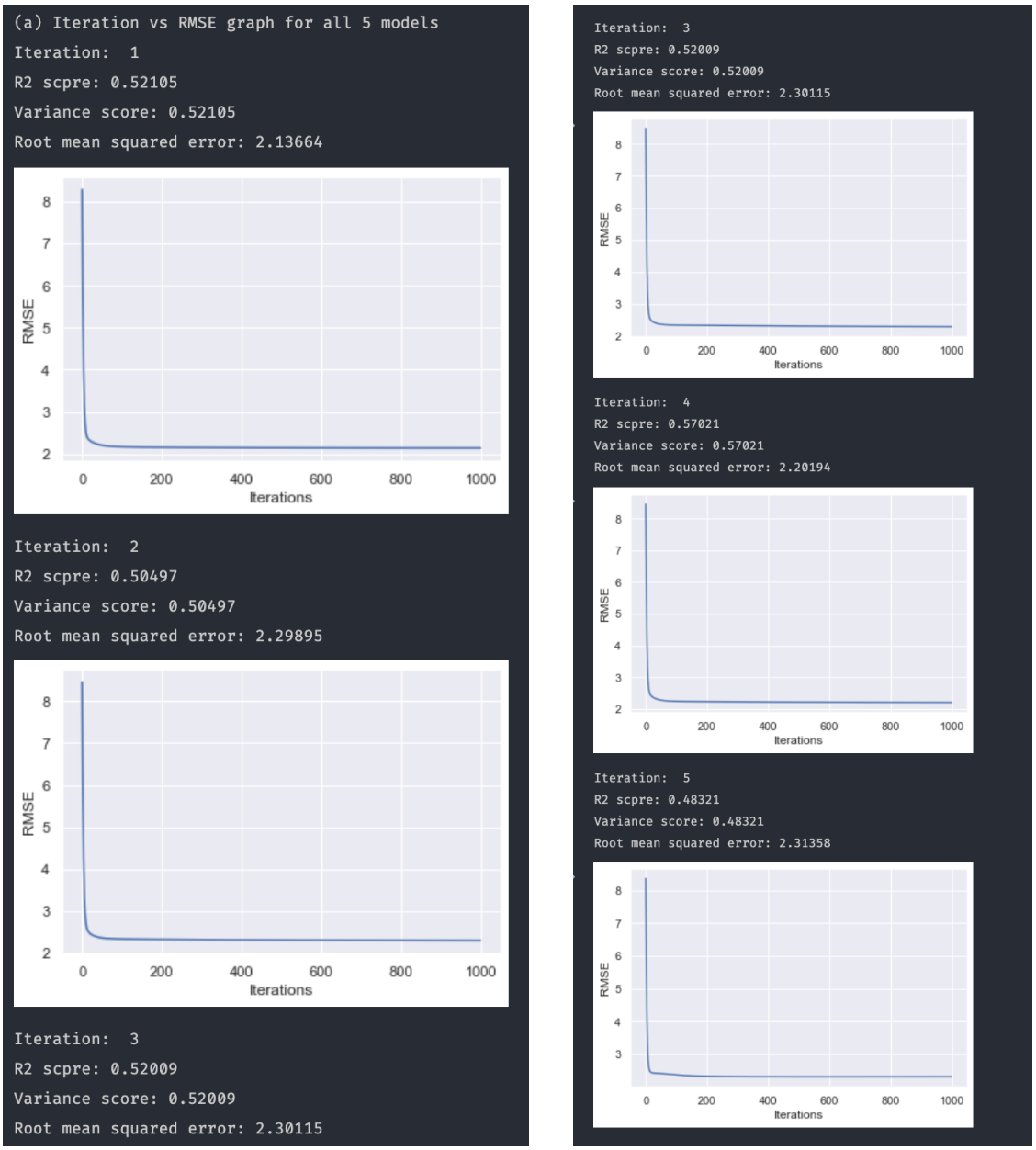
I tried one giving labels to sex (M-0, F-1, I-2) and one hot encoding too but did not get much better RMSE. So, I ended up removing this column altogether.
I performed feature standardisation using `StandardScalar` from sklearn on rest of the dataset obtained after removing the 'sex' column.

I kept the number of iterations as 1000 in each linear regression implementation I did from scratch (simple, l1 regularisation and l2 regularisation). The learning rate was 0.1.

I fixed a random seed in train_test_split to maintain consistency of results on each run. It might be possible that the RMSE values and other results vary if a different split of the dataset is done because of the way the model selection was asked to be done (best performance on validation set from KFold cross validation).

I used RMSE to compare model performance on the validation sets. This is because it is easy to calculate, universal and R Squared value always increases with the addition of the independent variables which might lead to the addition of the redundant variables in our model.

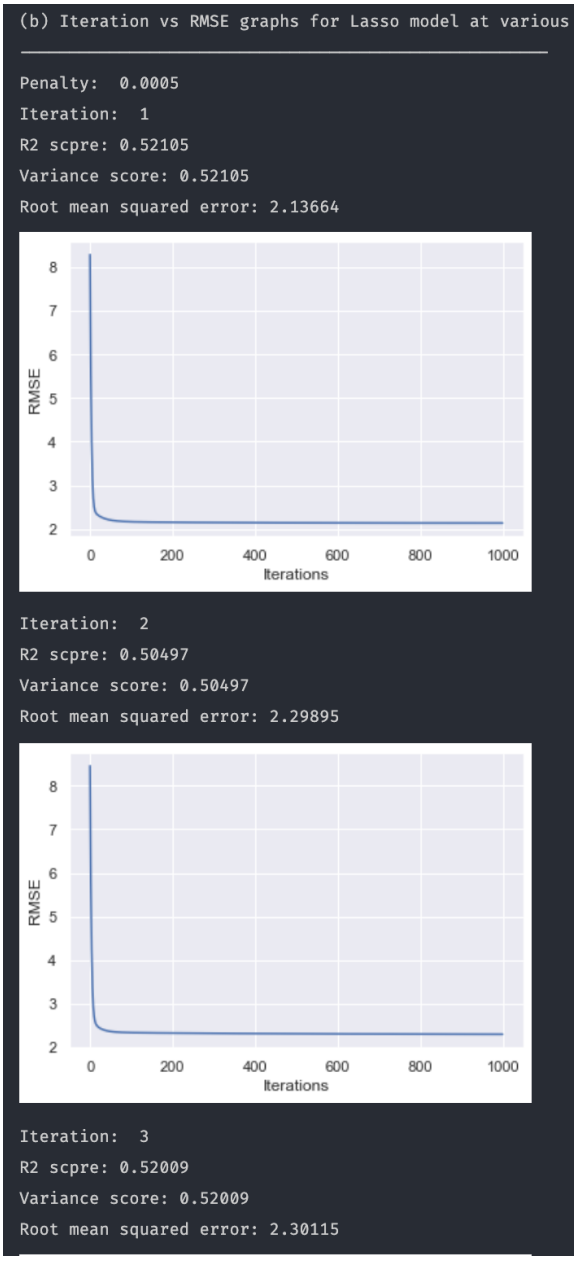
(a). Iteration vs RMSE graphs for all 5 models from 5-fold cross validation on the validation set:

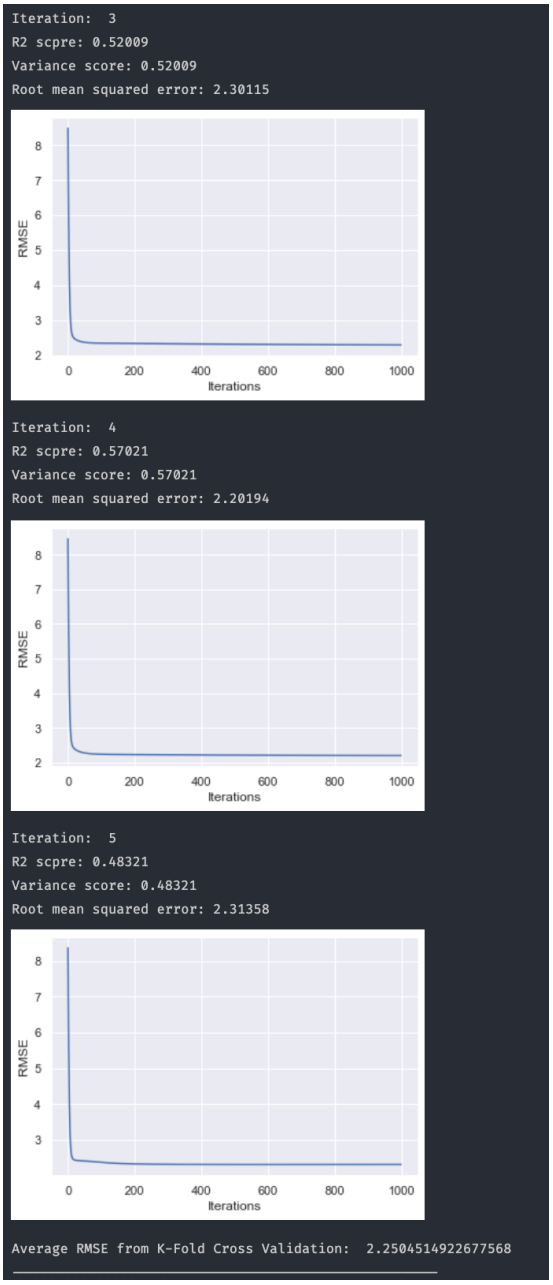


(b) Differential of the sum of absolute weights (-1 if some weight<0 and 1 if some weight>0) multiplied by the penalty lambda was added to dw (see code) to implement LASSO.

Different values of the regularisation parameter (penalty) that I tried in L1 Regularisation were [0.0005, 0.001, 0.01, 0.05, 0.1, 5]. The best model was chosen based on the minimum RMSE value obtained on the validation set during 5-fold cross validation. The penalty that returned the weights and intercept with the least

RMSE on the validation set was 0.0005 and the corresponding weights for that instance were used to make predictions on the test set in the next part.
I have displayed the Iterations vs RMSE for all penalties in the ipython notebook. However, here I will just show the plots for the penalty= 0.0005.

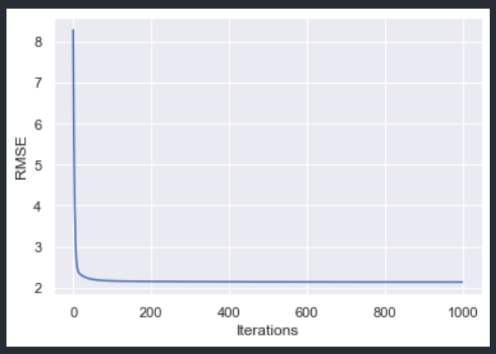




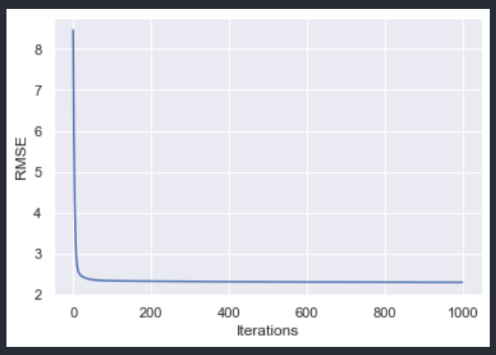
Differential of the sum of squared weights (twice the weight's sum) multiplied by the penalty lambda was added to dw (see code) to implement RIDGE.
Different values of the regularisation parameter (penalty) that I tried in L2 Regularisation were [0.001, 0.01, 0.05, 0.1, 1, 5]. The best model was chosen based

on the minimum RMSE value obtained on the validation set during 5-fold cross validation. The penalty that returned the weights and intercept with the least RMSE on the validation set was 5 and the corresponding weights for that instance were used to make predictions on the test set in the next part.
I have displayed the Iterations vs RMSE for all penalties in the ipython notebook. However, here I will just show the plots for penalty = 5.

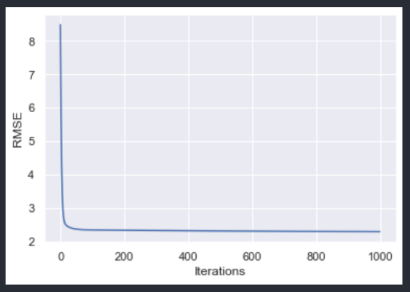
```
Penalty:  5
Iteration:  1
R2 scpre: 0.52114
Variance score: 0.52114
Root mean squared error: 2.13644
```



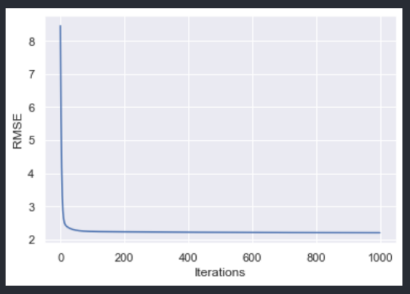

```
Iteration:  2
R2 scpre: 0.50495
Variance score: 0.50495
Root mean squared error: 2.29899
```



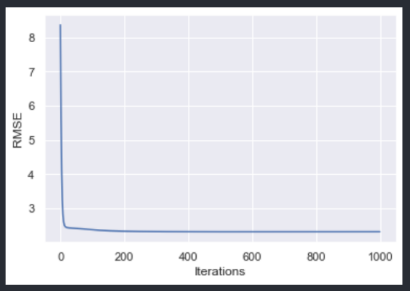

```
Iteration:  3
R2 scpre: 0.52018
Variance score: 0.52018
Root mean squared error: 2.30093
```




```
Iteration:  4
R2 scpre: 0.57013
Variance score: 0.57013
Root mean squared error: 2.20214
```




```
Iteration:  5
R2 scpre: 0.48348
Variance score: 0.48348
Root mean squared error: 2.31296
```




```
Average RMSE from K-Fold Cross Validation:  2.2504514977523464
```

(c) I have mentioned before how the best models (weights and intercept) were chosen for each regression method. Here is their comparative accuracy on the test dataset:

```
(c) Test set accuracy of the best models:
_____________________________________________________

Only Regression:
R2 scpre: 0.45313
Variance score: 0.45313
Root mean squared error: 2.17520
_____________________________________________________

Regression + L1:
R2 scpre: 0.45313
Variance score: 0.45313
Root mean squared error: 2.17520
_____________________________________________________

Regression + L2:
R2 scpre: 0.45316
Variance score: 0.45316
Root mean squared error: 2.17513
```

(d) I used Sklearn's models, used K-fold cross validation on each (simple, l1 and l2 linear regressors) and used the respective models that gave the minimum RMSE on the validation dataset. The performance of these three models are given below:

```
(d) Test set accuracy of the sklearn implementation:
_____________________________________________________

Only Regression:
R2 scpre: 0.45709
Variance score: 0.45709
Root mean squared error: 2.16730
_____________________________________________________

Regression + L1:
R2 scpre: 0.45735
Variance score: 0.45735
Root mean squared error: 2.16679
_____________________________________________________

Regression + L2:
R2 scpre: 0.46256
Variance score: 0.46256
Root mean squared error: 2.15636
```

Compare and analyse any differences:

- The RMSE and other metrics from both implementations are comparable.
- The most notable difference in results is that the RMSE values are lower in the results obtained from Sklearn.
- L2 regularisation gave the best RMSE in both implementations, although the variance was lower in my implementation and R2 score was higher in sklearn's.
- This difference can be attributed to the following reasons:
    - I used 1000 iterations in my implementation. The optimal number of iterations (I used 1000) or optimal learning rate (I used 0.1) might not have been used. I tried my best to come to an optimal value but it might be possible that if the model is left running for a few more iterations, an even minimum RMSE can be achieved.
    - The lambda (penalty) used in L1 and L2 might not have been optimal.
    - Sklearn's implementation might differ.

(e). The respective metrics on the validation sets after performing 5-fold cross validation on the closed form of linear regression is shown below:

```
Iteration:  1
R2 scpre: 0.51930
Variance score: 0.51930
Root mean squared error: 2.14055
Iteration:  2
R2 scpre: 0.50843
Variance score: 0.50843
Root mean squared error: 2.29091
Iteration:  3
R2 scpre: 0.52568
Variance score: 0.52568
Root mean squared error: 2.28770
Iteration:  4
R2 scpre: 0.57180
Variance score: 0.57180
Root mean squared error: 2.19786
Iteration:  5
R2 scpre: 0.47857
Variance score: 0.47857
Root mean squared error: 2.32393
```

3.

1. *This question is done in the ipython notebook question3-1.ipynb*

I downloaded the UCI Ionosphere dataset uploaded on drive and separated the target class from the rest of the features. Then, I used LabelBinarizer() to convert the target classes from string (b, g) to binary classes (0, 1). LabelBinarizer() mapped b to 0 and g to 1. 1 was assumed to be the positive class.

I found the 5 features with highest variance in the training set without preprocessing the dataset in any way. Standardising etc was performed later.
I used StandardScalar() from sklearn to standardise all training and testing features.

I used F1 score as the metric to compare different models in K-fold cross validation since it aggregates both precision and recall. Accuracy by itself is not enough as it can lead to wrongly assuming that the model is good when the data is imbalanced, even when the minor class is not being identified at all. Thus, F1 score, I felt, was a more reliable metric.

I fixed a random seed in train_test_split to maintain consistency of results on each run. It might be possible that the F1 scores and other results vary if a different split of the dataset is done because of the way the model selection was asked to be done (best performance on validation set from KFold cross validation).

The max_iterations used were 10000 in logistic regression models and the solver used was “saga”. A new model instance was used by K Fold cross validation performing function (using clone()) and the best one with the highest F1-score on the validation set was returned.

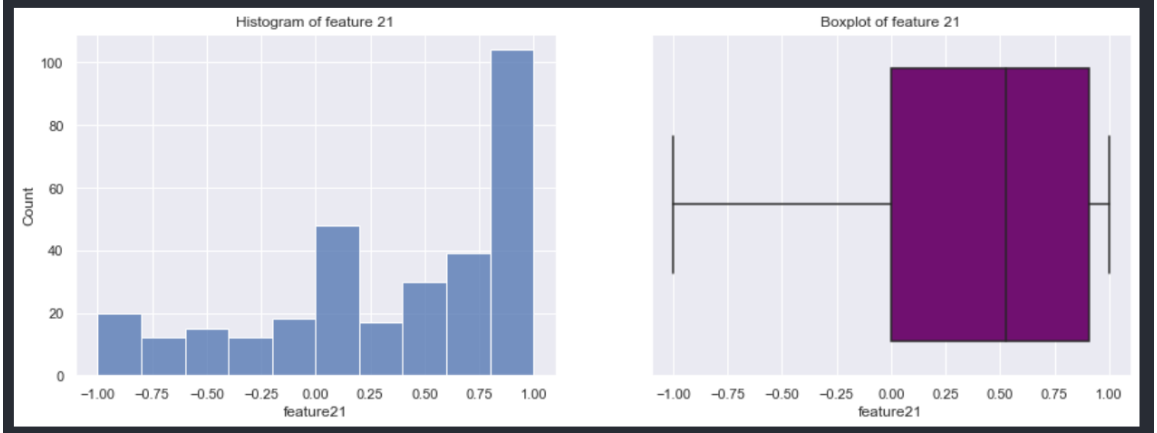
Mean and variance for all independent variables in the unprocessed dataset:

```
Mean and variance for all independent features in the UCI Ionosphere dataset:

Mean of feature 1 :  0.8920634920634921
Variance of feature 1 :  0.09659286219795775

Mean of feature 2 :  0.0
Variance of feature 2 :  0.0

Mean of feature 3 :  0.6308179365079365
Variance of feature 3 :  0.2602032881259832

Mean of feature 4 :  0.03464301587301588
Variance of feature 4 :  0.1911938292835507

Mean of feature 5 :  0.6055236825396825
Variance of feature 5 :  0.258096742394039

Mean of feature 6 :  0.11338358730158729
Variance of feature 6 :  0.2080500373377272

Mean of feature 7 :  0.5580497142857143
Variance of feature 7 :  0.23625978871424932

Mean of feature 8 :  0.11812219047619046
Variance of feature 8 :  0.27579562964837123

Mean of feature 9 :  0.5082888888888889
Variance of feature 9 :  0.2588541226283793

Mean of feature 10 :  0.18534961904761904
Variance of feature 10 :  0.23705533713807098

Mean of feature 11 :  0.4723110476190477
Variance of feature 11 :  0.31130301838966334

Mean of feature 12 :  0.14626996825396826
Variance of feature 12 :  0.2356386818130563

Mean of feature 13 :  0.4002824444444445
Variance of feature 13 :  0.3873872994764897

Mean of feature 14 :  0.1038431746031746
Variance of feature 14 :  0.2449093145675968

Mean of feature 15 :  0.34472593650793654
Variance of feature 15 :  0.4247939809541361

Mean of feature 16 :  0.06529834920634922
Variance of feature 16 :  0.21503897070363562

Mean of feature 17 :  0.3816451111111111
Variance of feature 17 :  0.38745071012379334

Mean of feature 18 :  0.0005986349206349161
Variance of feature 18 :  0.2361420017347548
```

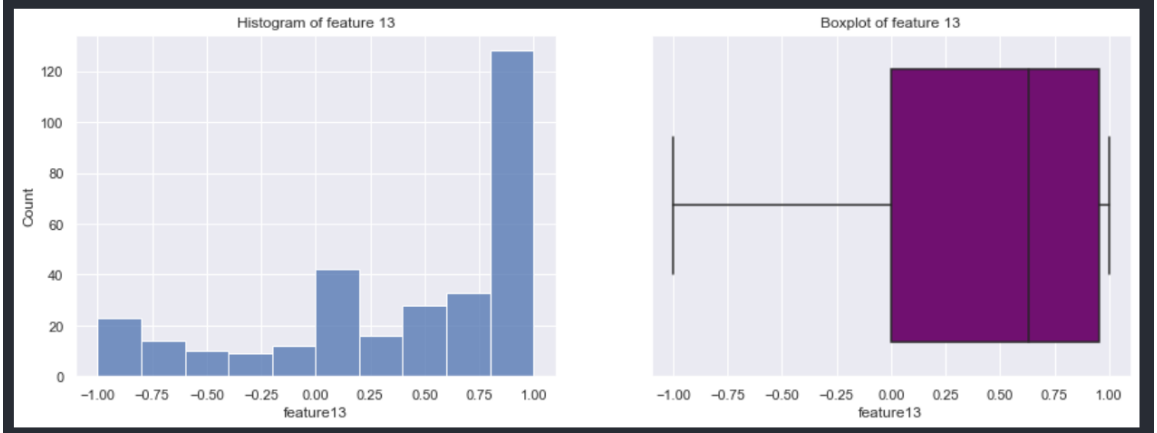
```
Mean of feature 19 :  0.3600667619047619
Variance of feature 19 :  0.3947847416181438

Mean of feature 20 :  -0.019576253968253972
Variance of feature 20 :  0.2647008508642665

Mean of feature 21 :  0.34753257142857147
Variance of feature 21 :  0.37153949731725205

Mean of feature 22 :  0.010754031746031744
Variance of feature 22 :  0.27367667020248304

Mean of feature 23 :  0.3682361904761905
Variance of feature 23 :  0.3633948278510463

Mean of feature 24 :  -0.06128361904761906
Variance of feature 24 :  0.2763703709900455

Mean of feature 25 :  0.40988285714285716
Variance of feature 25 :  0.3311888105363967

Mean of feature 26 :  -0.07118555555555556
Variance of feature 26 :  0.25441046204961076

Mean of feature 27 :  0.5422044444444444
Variance of feature 27 :  0.27069918842859164

Mean of feature 28 :  -0.062267555555555555
Variance of feature 28 :  0.2911586102936872

Mean of feature 29 :  0.39711955555555556
Variance of feature 29 :  0.31858965886922863

Mean of feature 30 :  -0.036432063492063495
Variance of feature 30 :  0.24261598598904052

Mean of feature 31 :  0.3572594603174603
Variance of feature 31 :  0.3224317165541664

Mean of feature 32 :  -0.008864476190476187
Variance of feature 32 :  0.25616022585792536

Mean of feature 33 :  0.36738088888888887
Variance of feature 33 :  0.2722886385762774

Mean of feature 34 :  0.008300603174603172
Variance of feature 34 :  0.22323017107447576
```

The five features with highest variance are: feature 21, feature 13, feature 17, feature 19 and feature 15. Their histograms and boxplots:

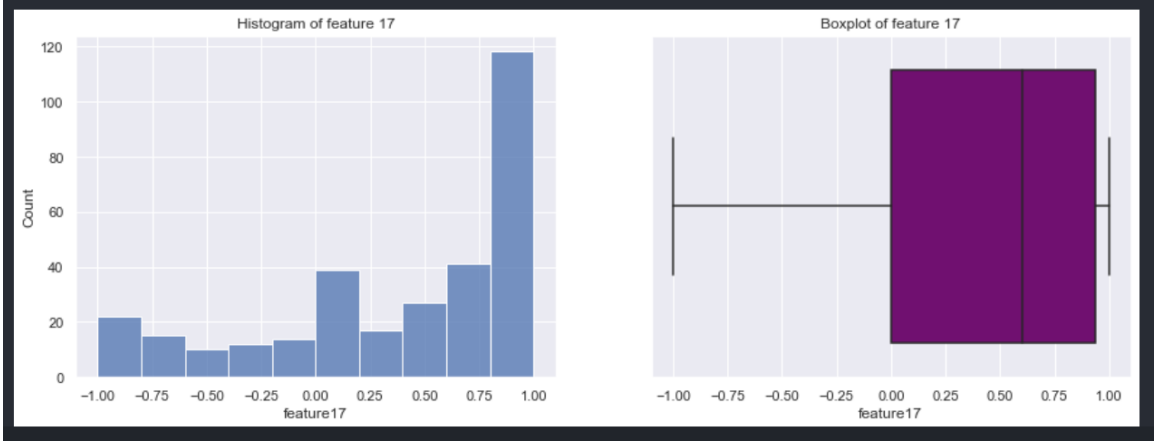
```
Features with higest variance:

Variance of feature 21 :  0.37153949731725205
```



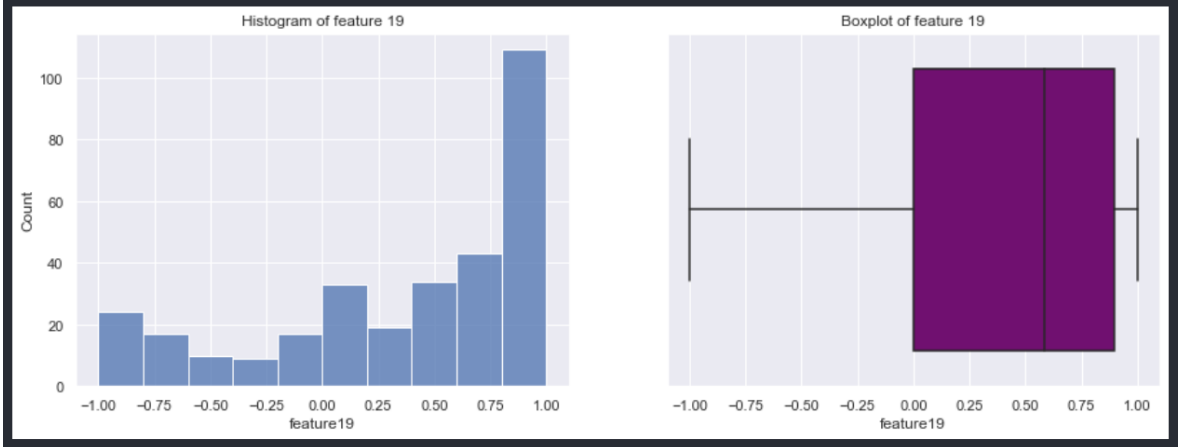

```
Variance of feature 13 :  0.3873872994764897
```



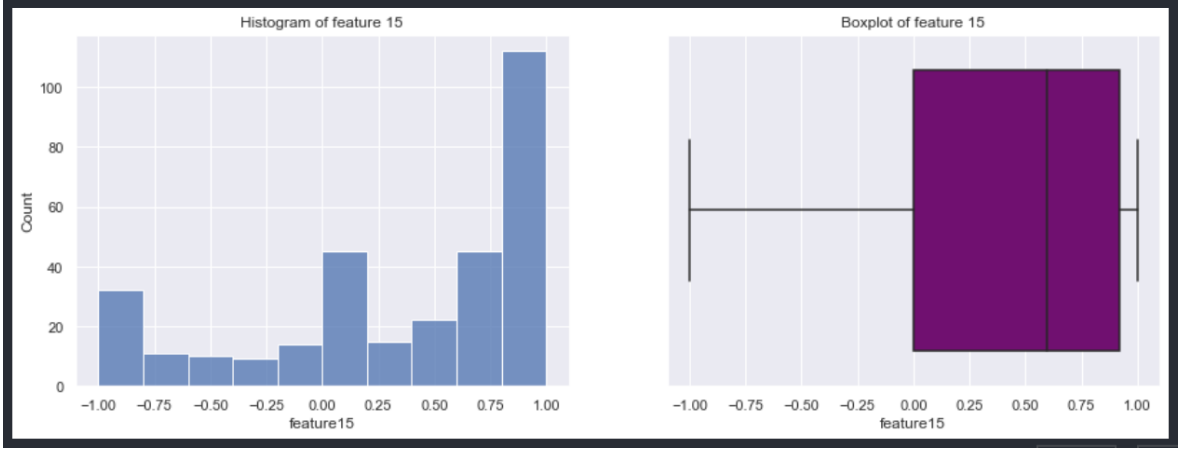

```
Variance of feature 17 :  0.38745071012379334
```



```
Variance of feature 19 :  0.3947847416181438
```




```
Variance of feature 15 :  0.4247939809541361
```



(a). **BONUS**

I passed the 11 variances as parameters to sklearn's PCA() one by one and recorded the one that gave the max f1 score on its validation data across all variances.
The optimal variance was 0.91- this gave the highest F1 score on its validation data.
The comparison of metrics on the test dataset of the best PCA and best non-PCA models:

```
Optimal PCA model:  PCA(n_components=0.91)      F1 score:  0.9473684210526315

Performance comparison of the best LR model with and without PCA with optimal variance on the test dataset:
   Method  Accuracy  Precision    Recall  F1 score
0  No PCA  0.944444   0.954545  0.954545  0.954545
1     PCA  0.916667   0.913043  0.954545  0.933333
```

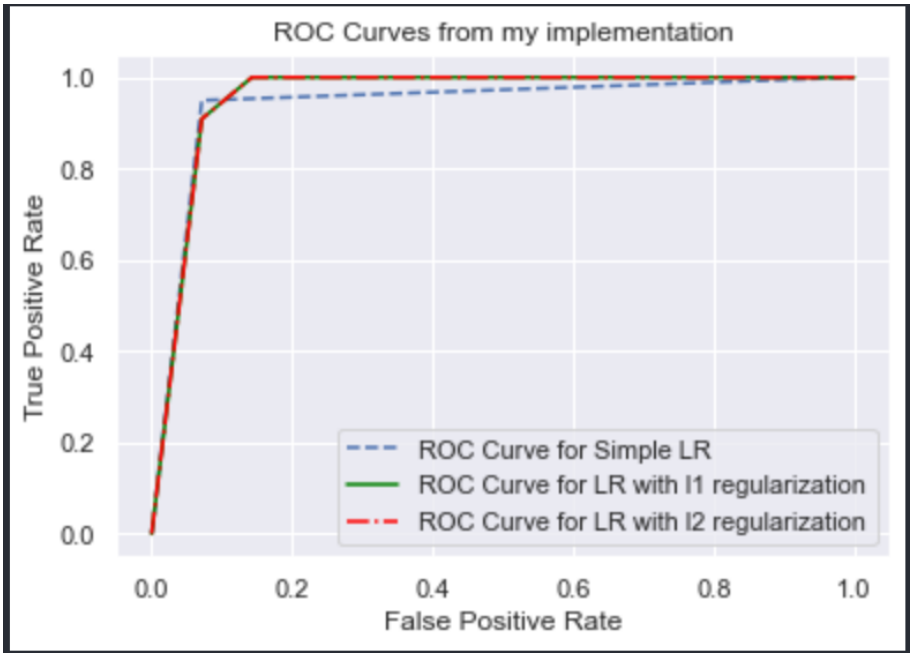
The result can vary depending on how the data is split because of our model selection technique (already described before).

(b). The output metrics on comparing simple logistic regression and L1 and L2 regularised logistic regression:

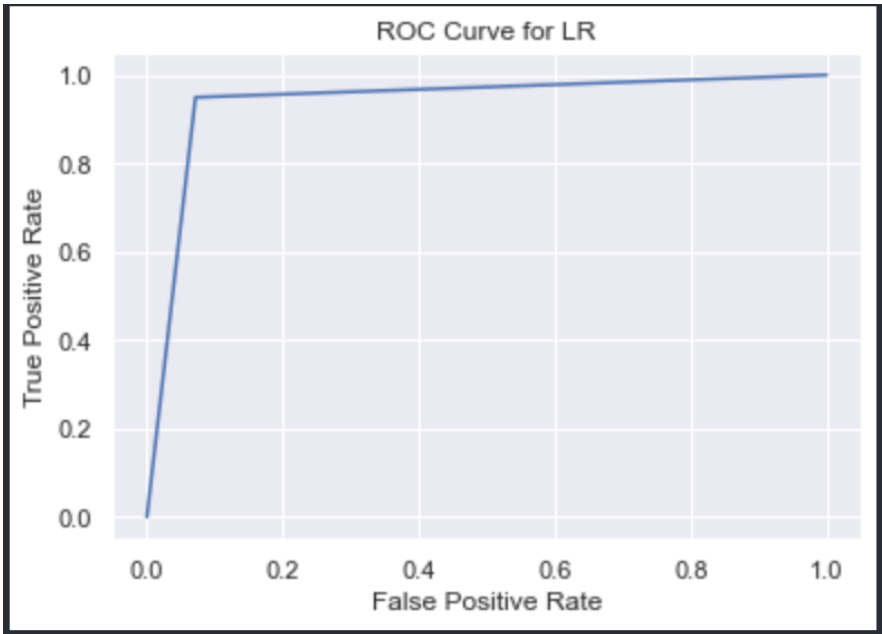
	Penalty	Accuracy	Precision	Recall	F1 score
0	none	0.944444	0.954545	0.954545	0.954545
1	l1	0.972222	0.956522	1.000000	0.977778
2	l2	0.944444	0.916667	1.000000	0.956522

L1 regularised Logistic Regression works the best in my case.

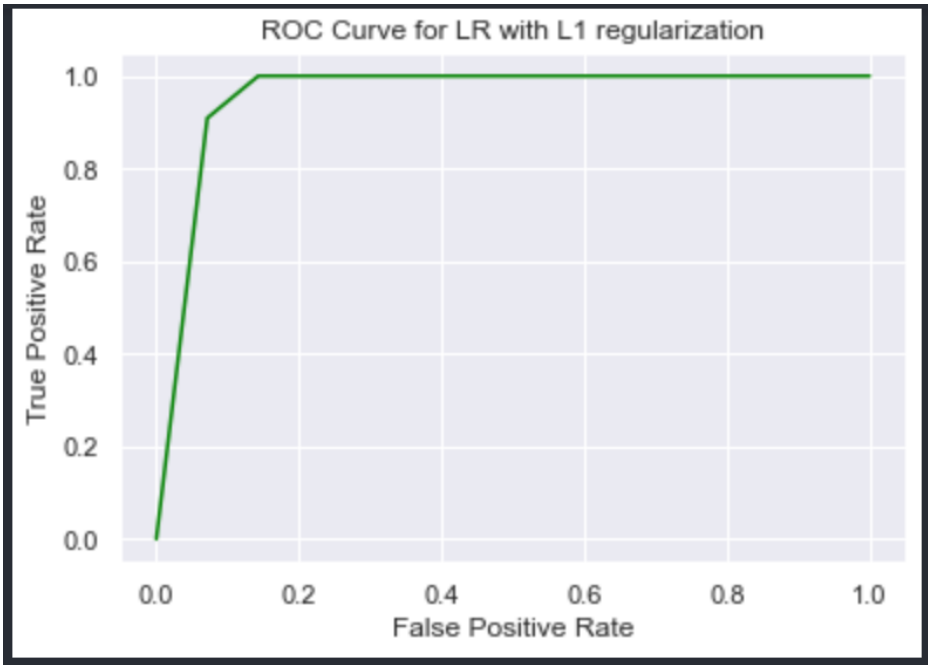
(c). ROC curves from my implementation together:



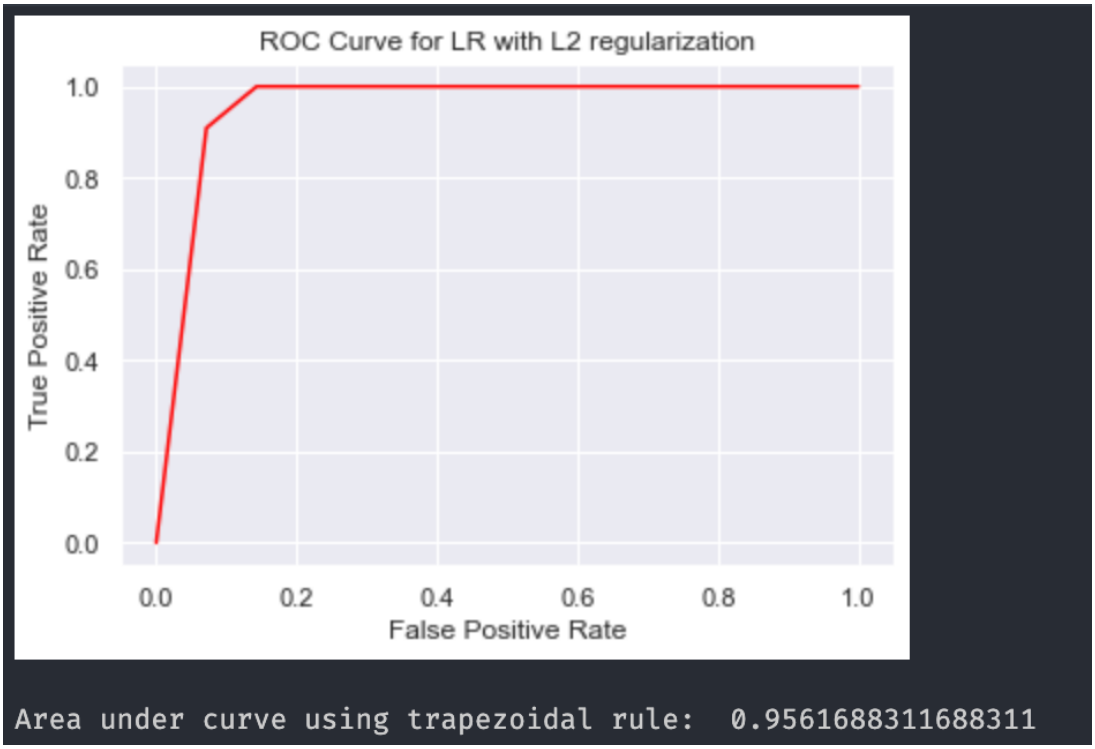
Individual ROC curves




```
Area under curve using trapezoidal rule:  0.9594155844155845
```



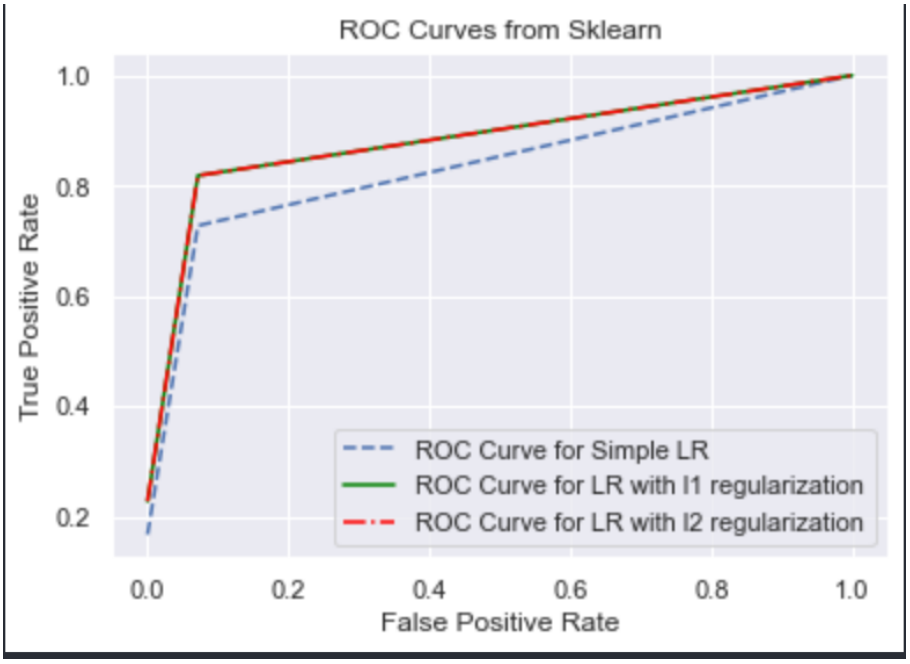

```
Area under curve using trapezoidal rule:  0.9561688311688311
```



The values used to plot these curves:

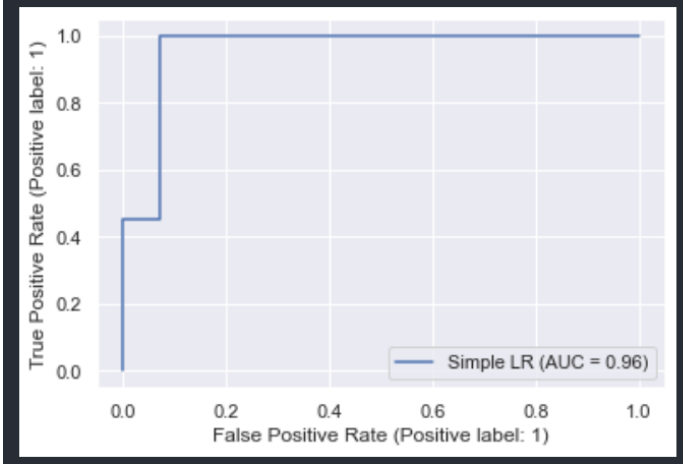
```
True positive rates for simple LR at given threshold values:  [1.0, 1.0, 0.9545454545454546, 0.9545454545454546, 0.9545454545454546, 0.9545454545454546,
0.9545454545454546, 0.9545454545454546, 0.9545454545454546, 0.8636363636363636, 0.0]
False positive rates for simple LR at given threshold values:  [1.0, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142,
0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.0]
_____

True positive rates for LR with l1 regularization at given threshold values:  [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.9545454545454546, 0.9545454545454546,
0.8636363636363636, 0.7727272727272727, 0.0]
False positive rates for LR with l1 regularization at given threshold values:  [1.0, 0.35714285714285715, 0.21428571428571427, 0.21428571428571427,
0.14285714285714285, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.0]
_____

True positive rates for LR with l2 regularization at given threshold values:  [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.9545454545454546, 0.9090909090909091,
0.7727272727272727, 0.0]
False positive rates for LR with l2 regularization at given threshold values:  [1.0, 0.42857142857142855, 0.21428571428571427, 0.14285714285714285,
0.14285714285714285, 0.14285714285714285, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.07142857142857142, 0.0]
```

(d) ROC curves using sklearn:

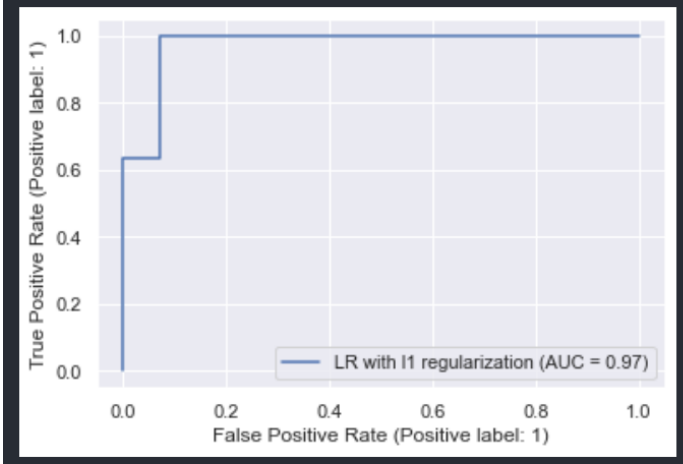


Individual curves with their areas are also given below:

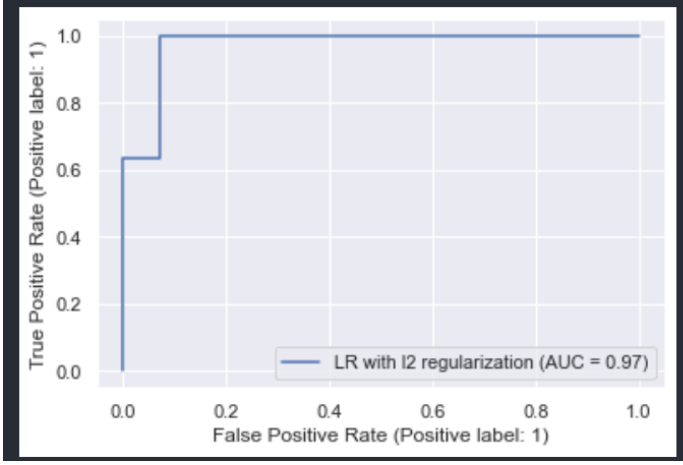
```
Area under curve using sklearn's roc_auc_score function:  0.961038961038961
```




```
Area under curve using sklearn's roc_auc_score function:  0.974025974025974
```




```
Area under curve using sklearn's roc_auc_score function:  0.974025974025974
```



Comment on any observed differences:

- There is certainly a difference in the area under the curve in both implementations but not much.

- Moreover, the curves for L1 and L2 kind of coincide in both the implementations.
- One other difference is that my curves start at 0 while sklearn’s start at some non-zero values too.
- As I increase the resolution of the thresholds from 11 to 1000, I observed that my curves' shape and area start approaching those reported by Sklearn.
- This is probably because we are now perhaps more likely to use the threshold values that are used by sklearn.

2. *This question is done in the ipython notebook question3-2.ipynb*

Extraction of the dataset was done in the same way as mentioned in Q1 part2.
I flattened each 28x28 image in the dataset to 1x784. The training set has dimensions 60000x784 and the testing set has dimensions 10000x784. The datasets were then standardised using sklearn's StandardScalar()

I used the "saga" solver for logistic regression since it was reported to run faster on large datasets. I kept the number of iterations to the default 1000. Python will throw warnings that the model does not converge because of a shortage of iterations, but we were told to not pay attention to that. Each classifier (OVO, OVR) takes about 10 mins to run as is, I tried running the models by increasing the max iterations to 10000 but that was taking over an hour to run.

(a). For One-vs-One classifier with simple logistic regression, the accuracy metrics found are given below:

```
Metrics for One-vs-One Logistic Regression without regularization on test dataset
Accuracy:  0.9347
Precision:  0.9347
Recall:  0.9347
F1 Score:  0.9347
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.97       980
           1       0.96      0.98      0.97      1135
           2       0.94      0.91      0.93      1032
           3       0.92      0.93      0.92      1010
           4       0.94      0.96      0.95       982
           5       0.91      0.89      0.90       892
           6       0.95      0.96      0.95       958
           7       0.94      0.92      0.93      1028
           8       0.91      0.89      0.90       974
           9       0.93      0.92      0.92      1009

    accuracy                           0.93     10000
   macro avg       0.93      0.93      0.93     10000
weighted avg       0.93      0.93      0.93     10000
```

For One-vs-One classifier with l2 regularised logistic regression, the accuracy metrics found are given below:

```
Metrics for One-vs-One Logistic Regression with L2 regularization on test dataset
Accuracy:  0.9347
Precision:  0.9347
Recall:  0.9347
F1 Score:  0.9347
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.97       980
           1       0.96      0.98      0.97      1135
           2       0.94      0.91      0.93      1032
           3       0.92      0.93      0.92      1010
           4       0.94      0.96      0.95       982
           5       0.91      0.89      0.90       892
           6       0.95      0.96      0.95       958
           7       0.94      0.92      0.93      1028
           8       0.91      0.89      0.90       974
           9       0.93      0.92      0.92      1009

    accuracy                           0.93     10000
   macro avg       0.93      0.93      0.93     10000
weighted avg       0.93      0.93      0.93     10000
```

(b) For One-vs-Rest classifier with simple logistic regression, the accuracy metrics found are given below:

```
Metrics for One-vs-Rest Logistic Regression without regularization on test dataset
Accuracy:  0.9028
Precision:  0.9028
Recall:  0.9028
F1 Score:  0.9028
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.98      0.95       980
           1       0.91      0.97      0.94      1135
           2       0.93      0.87      0.90      1032
           3       0.91      0.90      0.90      1010
           4       0.88      0.93      0.90       982
           5       0.88      0.84      0.86       892
           6       0.92      0.94      0.93       958
           7       0.89      0.90      0.90      1028
           8       0.87      0.84      0.85       974
           9       0.90      0.85      0.87      1009

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000
```

For One-vs-Rest classifier with l2 regularised logistic regression, the accuracy metrics found are given below:

```
Metrics for One-vs-Rest Logistic Regression with l2 regularization on test dataset
Accuracy:  0.9029
Precision:  0.9029
Recall:  0.9029
F1 Score:  0.9028999999999999
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.98      0.95       980
           1       0.91      0.97      0.94      1135
           2       0.93      0.87      0.90      1032
           3       0.91      0.90      0.90      1010
           4       0.88      0.93      0.90       982
           5       0.88      0.84      0.86       892
           6       0.92      0.94      0.93       958
           7       0.89      0.90      0.90      1028
           8       0.87      0.84      0.85       974
           9       0.90      0.85      0.87      1009

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000
```

DOMS | Page No.
Date / /

Question 4.

1- Given,

dataset $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$

$x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$, $i = 1, 2, \dots, N$.

Regression model $y = X\theta + \epsilon$

(θ: weights, ϵ: error)

$\vec{y} = [y_1, y_2, \dots, y_N]^T$ $y_i \Rightarrow$ target variable

$X_{N \times d} = [x_1, x_2, \dots, x_N]^T$ with input data vectors.

We have to derive the closed form solution to linear regression to estimate θ.

Let our hypothesis h_θ take in x, use the estimated weights $\theta_0, \theta_1, \dots \theta_d$ on the features X and spits out y as a linear function of x.

i.e. $h_\theta(x_i) = \theta_0 + \theta_1 x_i^1 + \theta_2 x_i^2 + \dots + \theta_d x_i^d$

or, $h_\theta(x_i) = (x_i)^T \theta$ (predictions)

Error in predictions would be given as →

$$X\theta - \vec{y} = \begin{bmatrix} x_1^T\theta \\ x_2^T\theta \\ \vdots \\ x_N^T\theta \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

$$= \begin{bmatrix} x_1^T\theta - y_1 \\ x_2^T\theta - y_2 \\ \vdots \\ x_N^T\theta - y_N \end{bmatrix}$$

We have to minimise the square of difference.

The square of $X\theta - \vec{y}$ can be written as -

$$(X\theta - \vec{y})^T \cdot (X\theta - \vec{y}) = \sum_{i=1}^{N} (x_i^T\theta - y_i)^2$$

(dot product)

$$\text{or} \quad \sum_{i=1}^{N} (h_\theta(x_i) - y_i)^2$$

Now, we have to minimize the sum of square by calculating its derivative w.r.t. θ and finding the minimum θ.

$$\Rightarrow \nabla \sum_{i=1}^{N} (h_\theta(x_i) - y_i)^2 = \frac{\partial}{\partial\theta}\left[(X\theta - \vec{y})^T (X\theta - \vec{y})\right]$$

$$= \frac{\partial}{\partial\theta}\left[(X\theta)^T \cdot X\theta - \vec{y}^T \cdot X\theta - (X\theta)^T \cdot \vec{y} + \vec{y}^T \cdot y\right]$$

since $\vec{y}^T X\theta$ and $X\theta^T \vec{y}$ are scalars, we can write :-

$$= \frac{\partial}{\partial\theta}\left[(X\theta)^T \cdot X\theta - (X\theta)^T\vec{y} - (X\theta)^T\vec{y} + \vec{y}^T y\right]$$

$y^T X\theta$ and $X\theta^T y$ are scalars, so, $y^T X\theta = X\theta^T y$

$$= \frac{\partial}{\partial\theta}\left[\theta^T (X^T X)\theta - 2(X\theta)^T\vec{y} + \vec{y}^T y\right]$$

Now, differentiate,

$$\frac{\partial}{\partial\theta}\theta^T X^T X\theta - 2\frac{\partial}{\partial\theta}\theta^T X^T \vec{y} + \frac{\partial}{\partial\theta} y^T y$$

$$= 2X^T X\theta - 2X^T y + 0$$ (derivative with respect to a vector)

(I consulted The Matrix Cookbook for finding derivatives).

$$\frac{\partial}{\partial\theta}\sum_{i=1}^{N}(h_\theta(x_i) - y_i^2) = 2X^T X\theta - 2X^T\vec{y}$$

Set the derivative = 0 for critical points

$$2X^T X\theta - 2X^T\vec{y} = 0$$
$$X^T X\theta - X^T\vec{y} = 0$$
$$X^T(X\theta - y) = 0 \quad X^T X\theta = X^T\vec{y}$$

Assuming that $X^T X$ is invertible,

$$\boxed{\theta = (X^T X)^{-1} X^T\vec{y}}$$

This is the closed form solution to the linear regression problem on the mentioned dataset.

2- Closed form solution to linear Regression can only exist when X^TX is invertible i.e. it is positive has to be full column rank i.e. when all the columns of X are linearly independent. which makes X^TX positive definite and hence invertible.

In case X is not full column rank, we can make it so by removing columns or equivalently, features.

If rank of X is less than d, there exists $v \in \mathbb{R}^N$ such that $Xv = 0$

which causes $X^TXv = 0$ and X^TX to be non invertible.

3-

If the closed form solution is not computationally expensive, it is good to use. But as soon as the matrix X becomes very large - eg 100k rows and 100k columns, it requires large storage space and finding X^TX and then its inverse can become very computationally expensive ($O(n^3)$) In this case, using gradient descent ($O(kn^2)$) is a better option than closed form calculations

4- Given,

$$\bar{Y} = \frac{1}{N} \sum_{i=1}^{N} y_i$$

or,

$$\bar{Y} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i + \epsilon_i) \quad \text{--- (1)}$$

where $\hat{y}_i$ is estimated by x_i using the regression coefficients θ and ϵ_i are the residuals.

Now, using linear regression,

$$\hat{y_i} = \theta_0 + \theta_1 x_i^1 + \theta_2 x_i^2 + \theta_3 x_i^3 + \cdots + \theta_d x_i^d$$

- Plugging this $\hat{y_i}$ in (1),

$$\bar{Y} = \frac{1}{N} \sum_{i=1}^{N} (\theta_0 + \theta_1 x_i^1 + \cdots + \theta_d x_i^d + \epsilon_i)$$

Now, sum of residuals has to be 0.

DOMS | Page No.
Date / /

So, we have,

$$\bar{Y} = \frac{1}{N}\sum_{i=1}^{N}(\theta_0 + \theta_1 x_i^1 + \cdots + \theta_d x_i^d)$$

$$\bar{Y} = \underbrace{\frac{1}{N}\sum_{i=1}^{N} x_i}_{\bar{X}}\theta$$

$$\bar{Y} = \bar{X}\theta$$

Hence, the least square fit line passes through $(\bar{X}, \bar{Y})$.

5- Technically, linear regression can be used for classification by setting thresholds (eg. predicted values above 0.5 belong to class 0 and others to class 1)
But the basic concept of linear regression is to find the best fit for the target values, which are continuous - not categorical as in classification problems. We need a solution that gives optimum decision boundary for the target classes. If we use linear regression here, everytime we add new data, our line will change, even if...

DOMS | Page No.
Date / /

the data does not carry much information. Thus, using linear regression for classification problems will cause inconsistencies