



MIDDLE EAST TECHNICAL UNIVERSITY  
NORTHERN CYPRUS CAMPUS

**CNG 562**  
**MACHINE LEARNING**

ASSIGNMENT-1

**Report**

*Nisa Nur Odabaş*  
*Kaan Taha Köken*

June 2, 2020

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Dataset . . . . .	3
1.2	Preprocess . . . . .	3
1.2.1	L1 Normalization . . . . .	3
1.2.2	Mean Removal . . . . .	4
1.3	Validation . . . . .	4
1.3.1	Random 1-Hold Out . . . . .	4
1.3.2	Stratified 1-Hold Out . . . . .	4
1.3.3	Random k-Fold . . . . .	4
1.4	Predictor Model . . . . .	5
1.4.1	Linear Regression . . . . .	5
1.4.2	Logistic Regression . . . . .	5
<b>2</b>	<b>Experiment</b>	<b>6</b>
<b>3</b>	<b>Results</b>	<b>9</b>
<b>4</b>	<b>Final Model</b>	<b>10</b>
<b>5</b>	<b>ROC</b>	<b>15</b>
5.1	Linear Regression Model . . . . .	16
5.2	Logistic Regression Model . . . . .	17
<b>6</b>	<b>Appendix</b>	<b>19</b>
6.1	Project Link . . . . .	19
6.2	Code . . . . .	19

## List of Figures

1	L1 Normalization formula . . . . .	3
2	Mean removal formula . . . . .	4
3	5-Fold . . . . .	4
4	Linear Regression Model Visualization . . . . .	5
5	Logistic Regression Model Visualization . . . . .	5
6	Train and Test Sets for the Experiments . . . . .	6
7	Main Method . . . . .	6
8	K-Fold Method . . . . .	7
9	Random 1-Hold Out Method . . . . .	7
10	Stratified 1-Hold Out Method . . . . .	8
11	Display Method . . . . .	8
12	Logistic Regression Accuracy Results . . . . .	9
13	Linear Regression Accuracy Results . . . . .	9
14	Splitting dataset into 4 subsets . . . . .	10
15	First data split . . . . .	10
16	Second data split . . . . .	10
17	Training code . . . . .	11
18	Train - TrainDev testing . . . . .	11
19	Train - Dev testing . . . . .	11
20	Train - Test testing . . . . .	12
21	Train - Dev+Test testing . . . . .	12
22	Machine Learning Chart . . . . .	13
23	All errors . . . . .	13
24	Prediction of [6 3 5 1.5] . . . . .	13
25	Threshold Method for Linear Regression . . . . .	14
26	Creating new binary classes . . . . .	15
27	Finding True Positive and False Positive Rates for All Classes . . . . .	16
28	ROC for Class 0 - Linear Regression Model . . . . .	16
29	ROC for Class 1 - Linear Regression Model . . . . .	17
30	ROC for Class 2 - Linear Regression Model . . . . .	17
31	ROC for Class 0 - Logistic Regression Model . . . . .	17
32	ROC for Class 1 - Logistic Regression Model . . . . .	18
33	ROC for Class 2 - Logistic Regression Model . . . . .	18

# 1 Introduction

In this assignment, our aim is to compare random 1-hold out, random 5-folds, random 10-folds and stratified 1-hold out validation methods for both logistic and linear regression predictors. In addition, we will look at the differences between using raw data, L1 normalization and mean removal preprocessors.

## 1.1 Dataset

We are using Iris dataset for the assignment. It contains 3 classes of 50 instances each, where each class refers a type of iris plant. It also has four attributes.

Attributes:

- sepal length(cm)
- sepal width(cm)
- petal length(cm).
- petal width(cm)
- class
  - Iris Setosa
  - Iris Versicolour
  - Iris Virginica

## 1.2 Preprocess

### 1.2.1 L1 Normalization

L1 normalization adds a penalty to the original loss function, and shrinks some parameters to zero. Hence, some variables do not play any role in the model. Therefore, it reduces the over-fitting and the generalization abilities of the model.

$$Error_{L1} = Error + \sum_{i=0}^N |\beta_i|$$

*where the  $\beta_i$  are the parameters*

Figure 1: L1 Normalization formula

### 1.2.2 Mean Removal

Mean removal or standardization is simply centers data by removing the average value of each characteristic, and then scales it by dividing standard deviation. Thus, it helps to remove bias from features.

$$x_{scaled} = \frac{x - mean}{sd}$$

Figure 2: Mean removal formula

## 1.3 Validation

### 1.3.1 Random 1-Hold Out

Random 1-hold out is basically splitting up the dataset into a ‘train’ and ‘test’ set randomly. The training set is what the model is trained on, and the test is used to see performance of the model on unseen data.

### 1.3.2 Stratified 1-Hold Out

Stratified 1-hold out is splitting up the dataset into a ‘train’ and ‘test’ set so that each split has same percentage of samples of each targets as the complete set.

### 1.3.3 Random k-Fold

Random k-fold is a cross-validation technique which splits up the dataset into ‘k’ groups. One of the groups is used as the test set and the rest are used as the training set. The model is trained on the training set and scored on the test set. The process is repeated until each unique group as been used as the test set.

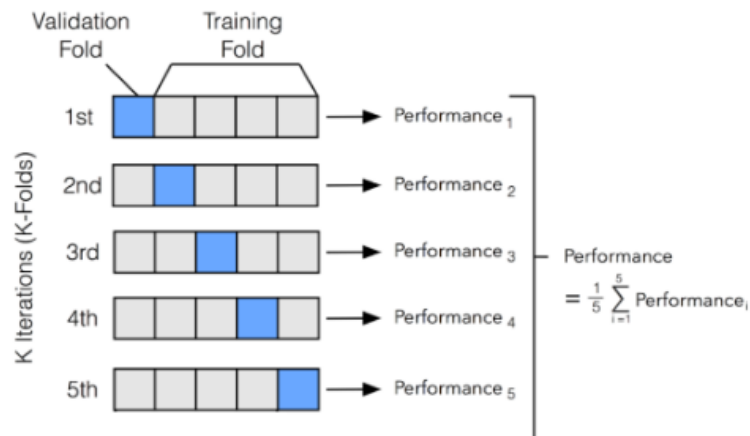


Figure 3: 5-Fold

## 1.4 Predictor Model

### 1.4.1 Linear Regression

Linear regression technique involves the continuous dependent variable and the independent variables can be continuous or discrete. By using best fit straight line or hyperplane in multidimensional cases, linear regression sets up a relationship between dependent variable and independent variables.

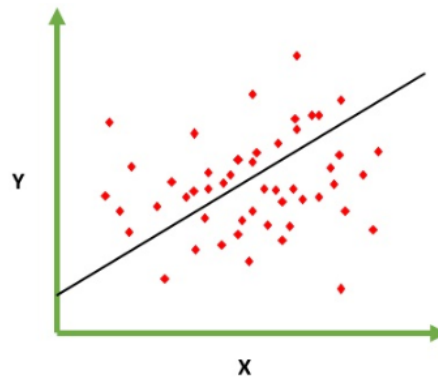


Figure 4: Linear Regression Model Visualization

### 1.4.2 Logistic Regression

Logistic regression technique involves dependent variable which can be represented in the binar (0 or 1, true or false) values. Therefore, the outcome can only be in either one form of two.

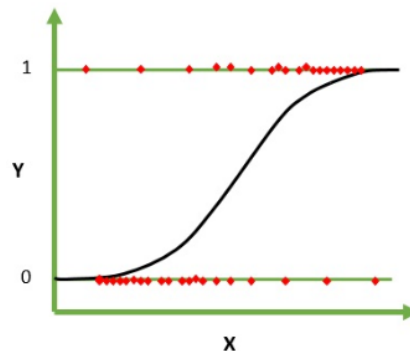


Figure 5: Logistic Regression Model Visualization

## 2 Experiment

Aim of this part is to decide our final strategy. As you can see in below, first, our approach is splitting dataset into a ‘train’ and ‘test’ set randomly. Then, we created 24 different models with training set.

<b>TRAIN</b> <b>%80</b>	<b>TEST</b> <b>%20</b>
----------------------------	---------------------------

Figure 6: Train and Test Sets for the Experiments

First, we trained our models with raw data. Then, we preprocessed data with L1 normalization and mean removal techniques, and created new models using pre-processed data.

### Main

```
[8] if __name__ == '__main__':  
  
    iris = datasets.load_iris()  
  
    X = iris.data  
    Y = iris.target  
  
    # L1 normalization  
    l1_norm = preprocessing.normalize(X, norm="l1")  
    # Mean removal  
    mean_removal = preprocessing.scale(X)  
  
    #Displaying result according to each type of methods and regression model  
    print("\nRaw: ")  
    displayAccuracy(X,Y)  
    print("\nL1 Normalization: ")  
    displayAccuracy(l1_norm,Y)  
    print("\nMean Removal: ")  
    displayAccuracy(mean_removal,Y)
```

Figure 7: Main Method

For writing more readable and reusable code, we used 3 different methods for validation techniques. Each of them takes data and target, then trains models with both linear regression and logistic regression predictors. Finally, they display accuracies for both models.

#### K-Fold method

```
[2] def kFold(foldNumber, X_train, Y_train):  
  
    kf = KFold(n_splits=foldNumber, shuffle=False)  
  
    logReg = LogisticRegression(solver='liblinear', multi_class='ovr')  
    linReg = LinearRegression()  
  
    cv_result_log = cross_val_score(logReg, X_train, Y_train, cv=kf, scoring='accuracy')  
    cv_result_lin = cross_val_score(linReg, X_train, Y_train, cv=kf, scoring='neg_mean_squared_error')  
  
    print(str(foldNumber) + "Fold")  
    print("Logistic Regression Accuracy: ", cv_result_log.mean())  
    print("Linear Regression Accuracy: ", 1 + cv_result_lin.mean())
```

Figure 8: K-Fold Method

#### Random 1-Hold Out method

```
[3] def randomOneHoldout(X_train, Y_train):  
  
    x_train, x_test, y_train, y_test = train_test_split(X_train, Y_train, test_size=0.2, random_state=0)  
  
    logReg = LogisticRegression(solver='liblinear', multi_class='ovr')  
    linReg = LinearRegression()  
  
    logReg.fit(x_train, y_train)  
    linReg.fit(x_train, y_train)  
  
    y_pred_log = logReg.predict(x_test)  
    y_pred_lin = linReg.predict(x_test)  
  
    print("Random One Hold Out")  
    print("Logistic Regression Accuracy: ", 1 - metrics.mean_squared_error(y_test, y_pred_log))  
    print("Linear Regression Accuracy: ", 1 - metrics.mean_squared_error(y_test, y_pred_lin))
```

Figure 9: Random 1-Hold Out Method

In random 1-hold out method, we used 20% of data for validation and 80% for training. Since we had a small set, we did not split data 30%-70%.



**Stratified 1-Hold Out method**

```
[4] def stratifiedOneHoldout(X_train, Y_train):  
  
    x_train, x_test, y_train, y_test = train_test_split(X_train, Y_train, test_size=0.2, random_state=0, stratify=Y_train)  
  
    logReg = LogisticRegression(solver='liblinear', multi_class='ovr')  
    linReg = LinearRegression()  
  
    logReg.fit(x_train, y_train)  
    linReg.fit(x_train, y_train)  
  
    y_pred_log = logReg.predict(x_test)  
    y_pred_lin = linReg.predict(x_test)  
  
    print("Stratified")  
    print("Logistic Regression Accuracy: ", 1 - metrics.mean_squared_error(y_test, y_pred_log))  
    print("Linear Regression Accuracy: ", 1 - metrics.mean_squared_error(y_test, y_pred_lin))
```

Figure 10: Stratified 1-Hold Out Method

Stratified 1-hold out method is almost same as random 1-hold out method. However, it splits data regarding to target set.

Finally, we displayed all accuracies for 24 experiments in total.

**Displaying accuracies for all validation methods**

```
[5] def displayAccuracy(X, Y):  
  
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)  
  
    kFold(5, X_train, Y_train)  
    kFold(10, X_train, Y_train)  
    randomOneHoldout(X_train, Y_train)  
    stratifiedOneHoldout(X_train, Y_train)
```

Figure 11: Display Method

### 3 Results

#### Logistic Regression Accuracy Results

	Random 1- Hold out	5-Fold	10-Fold	Stratified 1- Hold out
<b>Raw Data</b>	0.83	0.93	0.93	0.95
<b>L1 Normalization</b>	0.70	0.69	0.69	0.70
<b>Mean Removal</b>	0.79	0.87	0.88	0.91

Figure 12: Logistic Regression Accuracy Results

#### Linear Regression Accuracy Results

	Random 1- Hold out	5-Fold	10-Fold	Stratified 1- Hold out
<b>Raw Data</b>	0.91	0.94	0.95	0.95
<b>L1 Normalization</b>	0.88	0.91	0.91	0.92
<b>Mean removal</b>	0.91	0.94	0.95	0.95

Figure 13: Linear Regression Accuracy Results

When we look at the dataset and its attributes, it can be easily realized that this dataset is actually not suitable for regression. In Iris dataset, the aim is predicting one of the 3 classes. Since targets are not continuous values, linear regression is not a good choice. In addition, since this is not a binary or true/false case, logistic regression is also not suitable. This dataset is perfect for classification. However, in this assignment our aim is to compare logistic and linear regression.

When we started to compare the results of the experiments before deciding which dataset we were going to use, we looked at the cross validation methods. For both models and for every dataset, Stratified 1-Hold Out validation is provided best results. Then, we moved on to compare datasets. So far, we got the best results using raw dataset, and we got the worst results with the L1 Normalized dataset. Especially for logistic regression, L1 normalization gave low accuracy since when we normalized data we added some errors. Therefore, we decided not to use this dataset for further use.

After we eliminated the L1 normalization, we started to compare mean removal and raw datasets. If we look at the results closely, we can see that some of the results of raw and mean removal data are same. So we decided to continue with raw data.

As we mentioned before, Iris dataset is not suitable for regression. We can see that especially our logistic regression models are unsuccessful since we have three classes. Therefore, we decided to choose linear regression technique as a predictor.

## 4 Final Model

For creating a final model, we decided to move on with raw data. We used stratified 1-fold out validation and linear regression.

In order to validate and test our model, we split our data into four as mentioned by Andrew Ng, and also in the class.

<b>TRAIN</b> <b>%56</b>	<b>TRAIN-DEV</b> <b>%14</b>	<b>DEV</b> <b>%15</b>	<b>TEST</b> <b>%15</b>
----------------------------	--------------------------------	--------------------------	---------------------------

Figure 14: Splitting dataset into 4 subsets

Since our dataset is small, we split it with big percentages. Our approach is %56 for train, %14 train-dev, %15 for dev and %15 for test. In order to fulfill our goal, first we split into %70 and %30. Then, we split the train (%70) dataset, and test (%30), and created 'Train', 'Train-Dev', 'Dev', 'Test'.

```
[ ] X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=0, stratify=Y)
```

Figure 15: First data split

```
Train_x, TrainDev_x, Train_y, TrainDev_y = train_test_split(X_train, Y_train, test_size=0.2, random_state=0, stratify=Y_train)
Dev_x, Test_x, Dev_y, Test_y = train_test_split(X_test, Y_test, test_size=0.5, random_state=0, stratify=Y_test)
```

Figure 16: Second data split

After data separation, we started to experiment and tuning our model. Then, we trained with Train data.

### Training

```
[ ] linReg = LinearRegression()  
    linReg.fit(Train_x, Train_y)
```

Figure 17: Training code

In order to test our model, using Train-Dev, Dev, Test, Dev + Test, we recorded four error and four accuracy score.

```
trainDev_pred = linReg.predict(TrainDev_x)  
round_trainDev_pred = roundPredict(trainDev_pred)  
  
print("Train-Train Dev, e1:", metrics.mean_squared_error(TrainDev_y, trainDev_pred), "\n")  
print("Rounded Stratify One Hold Out - TrainDev")  
print("Linear Regression Accuracy: ", 1 - metrics.mean_squared_error(TrainDev_y, trainDev_pred))  
print("Linear Regression R^2 score: ", metrics.r2_score(TrainDev_y, trainDev_pred))  
  
Train-Train Dev, e1: 0.05876313845669336  
  
Rounded Stratify One Hold Out - TrainDev  
Linear Regression Accuracy: 0.9412368615433067  
Linear Regression R^2 score: 0.91185529231496
```

Figure 18: Train - TrainDev testing

```
dev_pred = linReg.predict(Dev_x)  
round_dev_pred = roundPredict(dev_pred)  
  
print("Train-Dev, e2", metrics.mean_squared_error(Dev_y, dev_pred), "\n")  
print("Rounded Stratify One Hold Out - Dev")  
print("Linear Regression Accuracy: ", 1 - metrics.mean_squared_error(Dev_y, dev_pred))  
print("Linear Regression R^2 score: ", metrics.r2_score(Dev_y, dev_pred))  
  
Train-Dev, e2 0.050911109070897693  
  
Rounded Stratify One Hold Out - Dev  
Linear Regression Accuracy: 0.9490888909291023  
Linear Regression R^2 score: 0.9251034140112022
```

Figure 19: Train - Dev testing

```
test_pred = linReg.predict(Test_x)
round_test_pred= roundPredict(test_pred)

print("Train-Test,   e3: ", metrics.mean_squared_error(Test_y, test_pred), "\n")
print("Rounded Stratify One Hold Out - Test set")
print("Linear Regression Accuracy: ", 1 - metrics.mean_squared_error(Test_y, test_pred))
print("Linear Regression R^2 score: ", metrics.r2_score(Test_y, test_pred))
```

```
Train-Test,   e3:  0.05383934801884544

Rounded Stratify One Hold Out - Test set
Linear Regression Accuracy:  0.9461606519811545
Linear Regression R^2 score:  0.9172063514477639
```

Figure 20: Train - Test testing

```
devTest_pred = linReg.predict(X_test)
rounded_lin = roundPredict(devTest_pred)

print("Train-(Dev+Test),   e4: ", metrics.mean_squared_error(Y_test, devTest_pred), "\n")
print("Rounded Stratify One Hold Out - Test set")
print("Linear Regression Accuracy: ", 1 - metrics.mean_squared_error(Y_test, devTest_pred))
print("Linear Regression R^2 score: ", metrics.r2_score(Y_test, devTest_pred))
```

```
Train-(Dev+Test),   e4:  0.05240776453318211

Rounded Stratify One Hold Out - Test set
Linear Regression Accuracy:  0.9475922354668179
Linear Regression R^2 score:  0.9213883532002268
```

Figure 21: Train - Dev+Test testing

After we got the four different values, we were asking ourselves this question: Did we do good job? To understand this, we needed to use one particular chart[22].

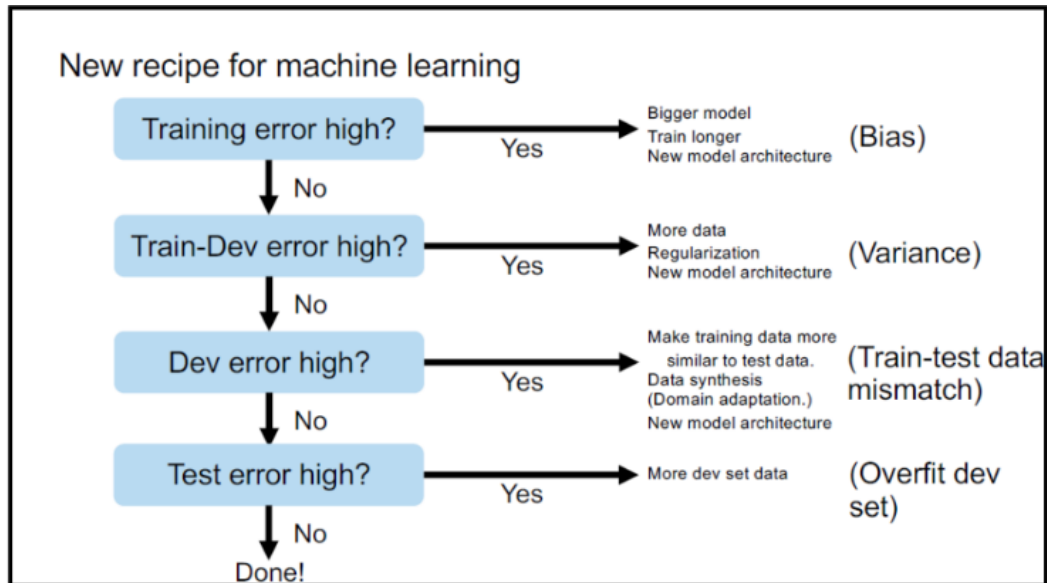


Figure 22: Machine Learning Chart

Error 1	Error 2	Error 3	Error 4
0.0587	0.0509	0.0538	0.0524

Figure 23: All errors

When we looked at the result and the chart on top, we did not do a bad job. Also, we wanted to check with given value in PDF which is [6 3 5 1.5].

```

Y_pred = linReg.predict([[6, 3, 5, 1.5]])
rounded = roundPredict(Y_pred.copy())

print("Prediction: \t\t", Y_pred)
print("Predicted class: \t", rounded)
print("Mean squared error: \t", metrics.mean_squared_error(rounded, Y_pred))
print("Mean absolute error: \t", metrics.mean_absolute_error(rounded, Y_pred))

Prediction:          [1.39983899]
Predicted class:     [1.]
Mean squared error:  0.15987121534579546
Mean absolute error: 0.3998389867756713
  
```

Figure 24: Prediction of [6 3 5 1.5]

Our model predicted '[6 3 5 1.5]' as 1.3998. Since we used linear regression, it did not give exact class number. However, according to our threshold method[25], we predicted the class as 1. To get error, we could use the offset of the predicted class value and prediction value. However, we went with mean squared error and mean absolute error which gave same result with offset. With these error techniques, in order, we got 0.1598 and 0.3998.

#### Round method for linear regression prediction

```
[6] def roundPredict(p):  
    r = p.copy()  
    for i in range(len(r)):  
        if r[i] <= 0.5: r[i] = 0  
        elif r[i] >= 1.5: r[i] = 2  
        else: r[i] = 1  
    return r
```

Figure 25: Threshold Method for Linear Regression

Although it looks like we did a good job, as we mention above, since our data size is small, and the dataset is not suitable for regression problem, these results are creating an illusion. We should not get low error percentages.

## 5 ROC

When we are trying to draw ROC curve, we faced with issues. Our dataset is not compatible since it has multiclass. In order to draw ROC, we developed a strategy. We treated our data has binary class. First, we used class 0, as one ,1, and rest stayed minus one, -1. We did it for all three classes(0,1,2).

```
#Linear Regression ROC calculation

#calculating the value according 0, 0 is 1 rest -1
roc_0 = Y_test.copy()
for i in range(len(Y_test)):
    if Y_test[i] != 0: roc_0[i] = -1
    else: roc_0[i] = 1

#calculating the value according 1, 1 is 1 rest -1
roc_1 = Y_test.copy()
for i in range(len(Y_test)):
    if Y_test[i] != 1: roc_1[i] = -1
    else: roc_1[i] = 1

#calculating the value according 2, 2 is 1 rest -1
roc_2 = Y_test.copy()
for i in range(len(Y_test)):
    if Y_test[i] != 2: roc_2[i] = -1
    else: roc_2[i] = 1
```

Figure 26: Creating new binary classes

Then, we calculated True Positive Rate and False Positive Rate. Using these values, we found ROC and ROC area under curved values. We keep doing these procedure for other labels. We used this procedure for both linear and logistic regression models.



```
#Individual ROC curve calculation
#For label 0
fpr_0, tpr_0, thresholds = roc_curve(roc_0, scores)
roc_auc_0 = auc(fpr_0, tpr_0)

#For label 1
fpr_1, tpr_1, thresholds = roc_curve(roc_1, scores)
roc_auc_1 = auc(fpr_1, tpr_1)

#For label 2
fpr_2, tpr_2, thresholds = roc_curve(roc_2, scores)
roc_auc_2 = auc(fpr_2, tpr_2)
```

Figure 27: Finding True Positive and False Positive Rates for All Classes

At the end, we got individual results and graphs for each label, and finally, we calculated average area under all these three graphs. As an average score for linear regression, we got 0.5. We also got the same average for logistic regression.

## 5.1 Linear Regression Model

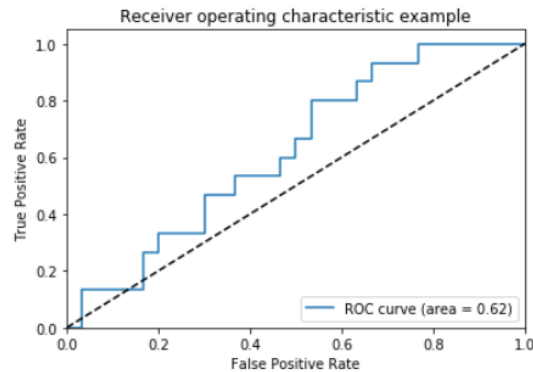


Figure 28: ROC for Class 0 - Linear Regression Model

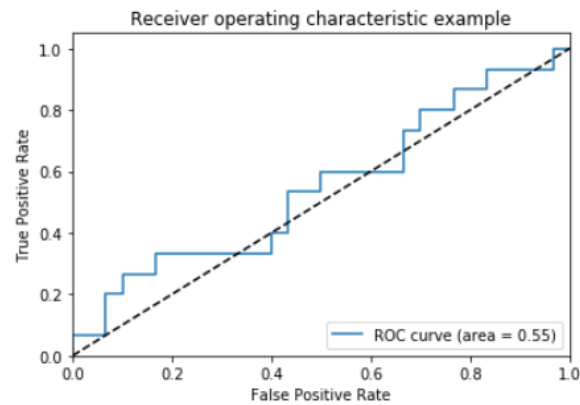


Figure 29: ROC for Class 1 - Linear Regression Model

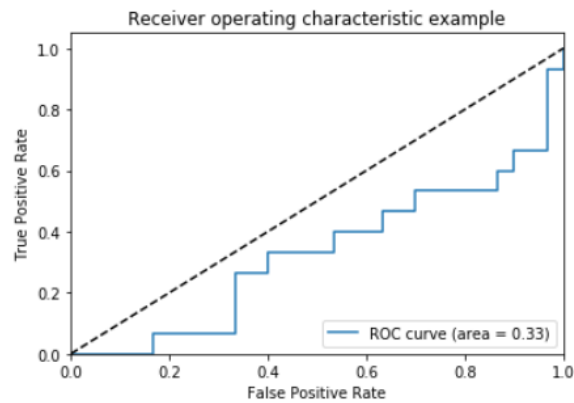


Figure 30: ROC for Class 2 - Linear Regression Model

## 5.2 Logistic Regression Model

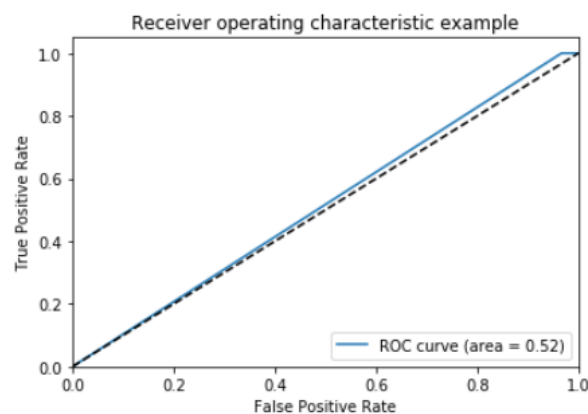


Figure 31: ROC for Class 0 - Logistic Regression Model

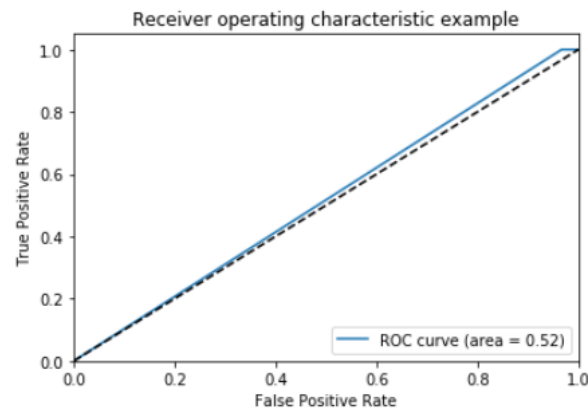


Figure 32: ROC for Class 1 - Logistic Regression Model

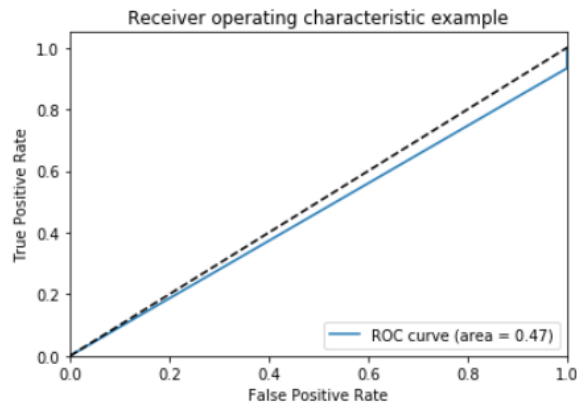


Figure 33: ROC for Class 2 - Logistic Regression Model

## 6 Appendix

### 6.1 Project Link

<https://github.com/nisanuro/CNG562-Assignment-1>

### 6.2 Code

```
0      """CNG562-Assignment1.ipynb
2
3      Automatically generated by Colaboratory.
4
5      Original file is located at
6          https://colab.research.google.com/github/nisanuro/CNG562-
7              Assignment-1/blob/master/
8              CNG562_Assignment1.ipynb
9
10     # **CNG 562 - Assignment #1**
11
12     Linear Regression vs Logistic Regression using Iris Dataset\
13     Comparing:
14     *   Random 1-Hold Out
15     *   5-Fold
16     *   10-Fold
17     *   Stratified 1-Hold Out
18
19     \
20     Nisa Nur Odabas\
21     Kaan Taha Koken
22
23     ---
24     """
25
26     # Commented out IPython magic to ensure Python compatibility.
27     import matplotlib.pyplot as plt
28     from sklearn.model_selection import train_test_split, KFold,
29         StratifiedKFold, cross_val_score
30     from sklearn.linear_model import LinearRegression,
31         LogisticRegression
32     from sklearn import metrics, datasets, preprocessing
33     from sklearn.metrics import roc_curve, auc
34     # %matplotlib inline
35
36     """**K-Fold method**"""
37
38     def kFold(foldNumber, X_train, Y_train):
39
40         kf = KFold(n_splits=foldNumber, shuffle=False)
```

```
38     logReg = LogisticRegression(solver='liblinear', multi_class='
                                     ovr')
    linReg = LinearRegression()
40
    cv_result_log = cross_val_score(logReg, X_train, Y_train, cv=
                                     kf, scoring='accuracy')
42     cv_result_lin = cross_val_score(linReg, X_train, Y_train, cv=
                                     kf, scoring='
                                     neg_mean_squared_error')

44     print(str(foldNumber) + "Fold")
    print("Logistic Regression Accuracy: ", cv_result_log.mean())
46     print("Linear Regression Accuracy: ", 1 + cv_result_lin.mean
          ())

48     """**Random 1-Hold Out method**"""

50     def randomOneHoldout(X_train, Y_train):

52         x_train, x_test, y_train, y_test = train_test_split(X_train,
                                                                Y_train, test_size=0.2,
                                                                random_state=0)

54         logReg = LogisticRegression(solver='liblinear', multi_class='
                                     ovr')
        linReg = LinearRegression()

56         logReg.fit(x_train, y_train)
58         linReg.fit(x_train, y_train)

60         y_pred_log = logReg.predict(x_test)
        y_pred_lin = linReg.predict(x_test)

62         print("Random One Hold Out")
        print("Logistic Regression Accuracy: ", 1 - metrics.
              mean_squared_error(y_test,
                                y_pred_log))
        print("Linear Regression Accuracy: ", 1 - metrics.
              mean_squared_error(y_test,
                                y_pred_lin))

66     """**Stratified 1-Hold Out method**"""

68     def stratifiedOneHoldout(X_train, Y_train):

70         x_train, x_test, y_train, y_test = train_test_split(X_train,
                                                                Y_train, test_size=0.2,
                                                                random_state=0, stratify=Y_train)

72
```

```
logReg = LogisticRegression(solver='liblinear', multi_class='
                                ovr')
74 linReg = LinearRegression()

76 logReg.fit(x_train, y_train)
linReg.fit(x_train, y_train)
78
y_pred_log = logReg.predict(x_test)
80 y_pred_lin = linReg.predict(x_test)

82 print("Stratified")
print("Logistic Regression Accuracy: ", 1 - metrics.
      mean_squared_error(y_test,
                          y_pred_log))
84 print("Linear Regression Accuracy: ", 1 - metrics.
      mean_squared_error(y_test,
                          y_pred_lin))

86 """**Displaying accuracies for all validation methods**"""

88 def displayAccuracy(X, Y):

90     X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                                                           test_size=0.2, random_state=0)

92     kFold(5, X_train, Y_train)
kFold(10, X_train, Y_train)
94 randomOneHoldout(X_train, Y_train)
stratifiedOneHoldout(X_train, Y_train)
96

98 """**Round method for linear regression prediction**"""

def roundPredict(p):
100     r = p.copy()
    for i in range(len(r)):
102         if r[i] <= 0.5: r[i] = 0
        elif r[i] >= 1.5: r[i] = 2
104         else: r[i] = 1
    return r

106

108 """**Display method for ROC curve**"""

def displayROC(fpr, tpr, roc_auc):
110     plt.figure()
    plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc
             )
112     plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
114     plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
```

```
116     plt.ylabel('True Positive Rate')
117     plt.title('Receiver operating characteristic example')
118     plt.legend(loc="lower right")
119     plt.show()
120
121     """**Main**"""
122
123     if __name__ == '__main__':
124
125         iris = datasets.load_iris()
126
127         X = iris.data
128         Y = iris.target
129
130         # L1 normalization
131         l1_norm = preprocessing.normalize(X, norm="l1")
132         # Mean removal
133         mean_removal = preprocessing.scale(X)
134
135         '''#mean & standart deviation before mean removal
136         print(X.mean(axis=0))
137         print(X.std(axis=0))
138
139         #mean & standart deviation after mean removal
140         print(mean_removal.mean(axis=0))
141         print(mean_removal.std(axis=0))'''
142
143         #Displaying result according to each type of methods and
144                                     regression model
145         print("\nRaw: ")
146         displayAccuracy(X,Y)
147         print("\nL1 Normalization: ")
148         displayAccuracy(l1_norm,Y)
149         print("\nMean Removal: ")
150         displayAccuracy(mean_removal,Y)
151
152         """# **Final**
153         **Training and Testing using:**
154         * **Raw data**
155         * **Stratified 1-Hold Out**
156         * **Linear Regression**
157
158         **Dividing Train and Test sets**
159         """
160
161         X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
162                                                         test_size=0.3, random_state=0,
163                                                         stratify=Y)
164
165         """Splitting dataset into 4.
```

```

164 *   Train - 56%
165 *   Train Dev - 14%
166 *   Dev - 15%
167 *   Test - 15%
168 """
169
170 Train_x, TrainDev_x, Train_y, TrainDev_y = train_test_split(
171     X_train, Y_train, test_size=0.2,
172     random_state=0, stratify=Y_train)
173
174 Dev_x, Test_x, Dev_y, Test_y = train_test_split(X_test, Y_test
175     , test_size=0.5, random_state=0,
176     stratify=Y_test)
177
178 """**Training**"""
179
180 linReg = LinearRegression()
181 linReg.fit(Train_x, Train_y)
182
183 """**Testing**"""
184
185 trainDev_pred = linReg.predict(TrainDev_x)
186 round_trainDev_pred = roundPredict(trainDev_pred)
187
188 print("Train-Train Dev,   e1:", metrics.mean_squared_error(
189     TrainDev_y, trainDev_pred), "\n")
190 print("Rounded Stratify One Hold Out - TrainDev")
191 print("Linear Regression Accuracy: ", 1 - metrics.
192     mean_squared_error(TrainDev_y,
193     trainDev_pred))
194 print("Linear Regression R^2 score: ", metrics.r2_score(
195     TrainDev_y, trainDev_pred))
196
197 print("\ntrainDev_pred   \tTrainDev_y\trounded")
198 for i, (j, k) in sorted(zip(trainDev_pred, zip(TrainDev_y,
199     round_trainDev_pred))):
200     print(i , "\t" , j, "\t\t", k)
201
202 """Add TrainDev to Train and create new model"""
203
204 dev_pred = linReg.predict(Dev_x)
205 round_dev_pred = roundPredict(dev_pred)
206
207 print("Train-Dev,   e2", metrics.mean_squared_error(Dev_y,
208     dev_pred), "\n")
209 print("Rounded Stratify One Hold Out - Dev")
210 print("Linear Regression Accuracy: ", 1 - metrics.
211     mean_squared_error(Dev_y,
212     dev_pred))
213 print("Linear Regression R^2 score: ", metrics.r2_score(Dev_y,
214     dev_pred))

```



```

200     print("\ndev_pred      \t      Dev_y\trounded")
202     for i, (j, k) in sorted(zip(dev_pred, zip(Dev_y, round_dev_pred
203                               ))):
204         print(i , "\t" , j, "\t", k)
206
208     test_pred = linReg.predict(Test_x)
209     round_test_pred= roundPredict(test_pred)
210
211     print("Train-Test,    e3: ", metrics.mean_squared_error(Test_y,
212                                                             test_pred),"\n")
213     print("Rounded Stratify One Hold Out - Test set")
214     print("Linear Regression Accuracy: ", 1 - metrics.
215           mean_squared_error(Test_y,
216                               test_pred))
217     print("Linear Regression R^2 score: ", metrics.r2_score(Test_y,
218                                                             test_pred))
219
220     print("\ntest_pred      \t      Test_y\trounded")
221     for i, (j, k) in sorted(zip(test_pred, zip(Test_y,
222                               round_test_pred))):
223         print(i , "\t" , j, "\t", k)
224
225     devTest_pred = linReg.predict(X_test)
226     rounded_lin = roundPredict(devTest_pred)
227
228     print("Train-(Dev+Test),    e4: ", metrics.mean_squared_error(
229           Y_test, devTest_pred),"\n")
230     print("Rounded Stratify One Hold Out - Test set")
231     print("Linear Regression Accuracy: ", 1 - metrics.
232           mean_squared_error(Y_test,
233                               devTest_pred))
234     print("Linear Regression R^2 score: ", metrics.r2_score(Y_test,
235                                                             devTest_pred))
236
237     print("\ndevTest_pred \t\t Dev+Test \trounded")
238     for i, (j, k) in sorted(zip(devTest_pred, zip(Y_test,
239                               rounded_lin))):
240         print(i , "\t" , j, "\t\t", k)
241
242     """**Predicting [6, 3, 5, 1.5]**"""
243
244     Y_pred = linReg.predict([[6, 3, 5, 1.5]])
245     rounded = roundPredict(Y_pred.copy())
246
247     print("Prediction: \t\t",Y_pred)
248     print("Predicted class: \t", rounded)

```

```
238     print("Mean squared error: \t", metrics.mean_squared_error(
        rounded, Y_pred))
240     print("Mean absolute error: \t", metrics.mean_absolute_error(
        rounded, Y_pred))

242     """**ROC**\
    Using rounded predictions since linear regression has no ROC.
    """

244     #calculating the score for roc curve
246     scores=[]

248     for i, j in sorted(zip(devTest_pred, Y_test)):
        scores.append(float(j)- i)
250     scores

252     #Linear Regression ROC calculation

254     #calculating the value according 0, 0 is 1 rest -1
    roc_0 = Y_test.copy()
256     for i in range(len(Y_test)):
        if Y_test[i] != 0: roc_0[i] = -1
258         else: roc_0[i] = 1

260     #calculating the value according 1, 1 is 1 rest -1
    roc_1 = Y_test.copy()
262     for i in range(len(Y_test)):
        if Y_test[i] != 1: roc_1[i] = -1
264         else: roc_1[i] = 1

266     #calculating the value according 2, 2 is 1 rest -1
    roc_2 = Y_test.copy()
268     for i in range(len(Y_test)):
        if Y_test[i] != 2: roc_2[i] = -1
270         else: roc_2[i] = 1

272     #Individual ROC curve calculation
    #For label 0
274     fpr_0, tpr_0, thresholds = roc_curve(roc_0, scores)
    roc_auc_0 = auc(fpr_0, tpr_0)
276
    #For label 1
278     fpr_1, tpr_1, thresholds = roc_curve(roc_1, scores)
    roc_auc_1 = auc(fpr_1, tpr_1)
280
    #For label 2
282     fpr_2, tpr_2, thresholds = roc_curve(roc_2, scores)
    roc_auc_2 = auc(fpr_2, tpr_2)
284
    #plotting ROC curve individual
```

```
286
288     #For 0
288     displayROC(fpr_0, tpr_0, roc_auc_0)
290
290     #For 1
290     displayROC(fpr_1, tpr_1, roc_auc_1)
292
292     #For 2
292     displayROC(fpr_2, tpr_2, roc_auc_2)
294
294     #Average ROC
294     sum_roc = roc_auc_2 + roc_auc_1 + roc_auc_0
296     sum_roc = sum_roc / 3.0
298     print("Average ROC curve (area) score: ", sum_roc)
300
300     #Logistic Regression ROC calculation
300     logReg = LogisticRegression(solver='liblinear', multi_class='
302                                     ovr')
302     logReg.fit(Train_x, Train_y)
304     devTest_pred_log = logReg.predict(X_test)
306
306     #calculating the score for roc calculation
306     scores=[]
308
308     for i, j in sorted(zip(devTest_pred_log, Y_test)):
310         scores.append(float(j)- i)
312
312     #calculating the value according 0, 0 is 1 rest -1
312     roc_0 = Y_test.copy()
314     for i in range(len(Y_test)):
314         if Y_test[i] != 0: roc_0[i] = -1
316         else: roc_0[i] = 1
318
318     #calculating the value according 1, 1 is 1 rest -1
318     roc_1 = Y_test.copy()
320     for i in range(len(Y_test)):
320         if Y_test[i] != 1: roc_1[i] = -1
322         else: roc_1[i] = 1
324
324     #calculating the value according 2, 2 is 1 rest -1
324     roc_2 = Y_test.copy()
326     for i in range(len(Y_test)):
326         if Y_test[i] != 2: roc_2[i] = -1
328         else: roc_2[i] = 1
330
330     #Individual ROC curve calculation
330     #For label 0
332     fpr_0, tpr_0, thresholds = roc_curve(roc_0, scores)
332     roc_auc_0 = auc(fpr_0, tpr_0)
334
```

```
336     #For label 1
    fpr_1, tpr_1, thresholds = roc_curve(roc_1, scores)
    roc_auc_1 = auc(fpr_1, tpr_1)
338
    #For label 2
340     fpr_2, tpr_2, thresholds = roc_curve(roc_2, scores)
    roc_auc_2 = auc(fpr_2, tpr_2)
342
    #plotting ROC curve individual
344
    #For 0
346     displayROC(fpr_0, tpr_0, roc_auc_0)
348
    #For 1
    displayROC(fpr_1, tpr_1, roc_auc_1)
350
    #For 2
352     displayROC(fpr_2, tpr_2, roc_auc_2)
354
    #Average ROC
    sum_roc = roc_auc_2 + roc_auc_1 + roc_auc_0
356     sum_roc = sum_roc / 3.0
    print("Average ROC curve (area) score: ", sum_roc)
```