MIDDLE EAST TECHNICAL UNIVERSITY

NORTHERN CYPRUS CAMPUS

CNG 562
MACHINE LEARNING

Assignment-1

Report

Nisa Nur Odabaş

Kaan Taha Köken

June 2, 2020

Contents

List of Figures

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.

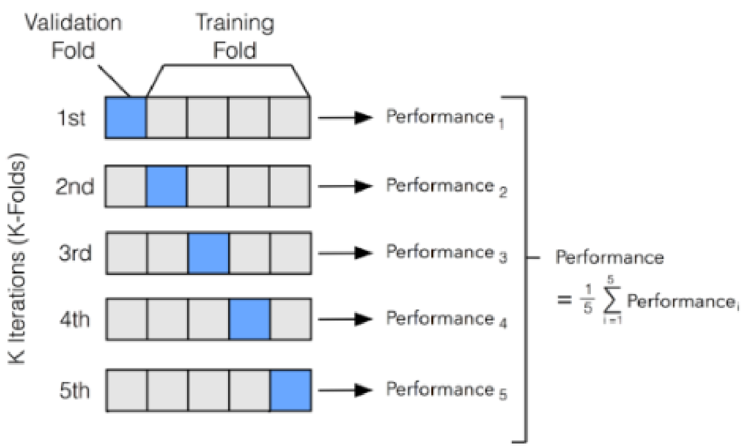

Figure 3: 5-Fold

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.

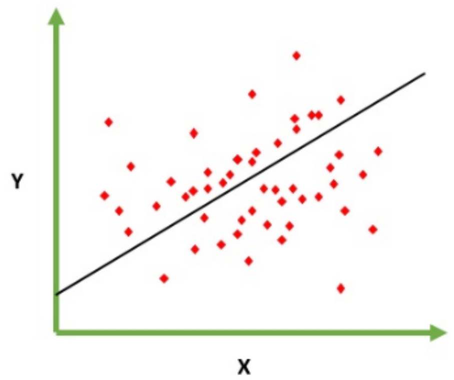

Figure 4: Linear Regression Model Visualization

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.

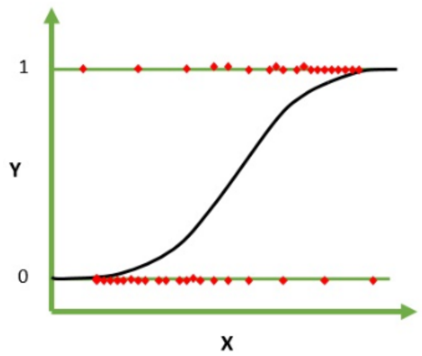

Figure 5: Logistic Regression Model Visualization

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.

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 preprocessed 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
Raw Data	0.83	0.93	0.93	0.95
L1 Normalization	0.70	0.69	0.69	0.70
Mean Removal	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
Raw Data	0.91	0.94	0.95	0.95
L1 Normalization	0.88	0.91	0.91	0.92
Mean removal	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 the 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-hold 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.

TRAIN	TRAIN-DEV	DEV	TEST
%56	%14	%15	%15

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].

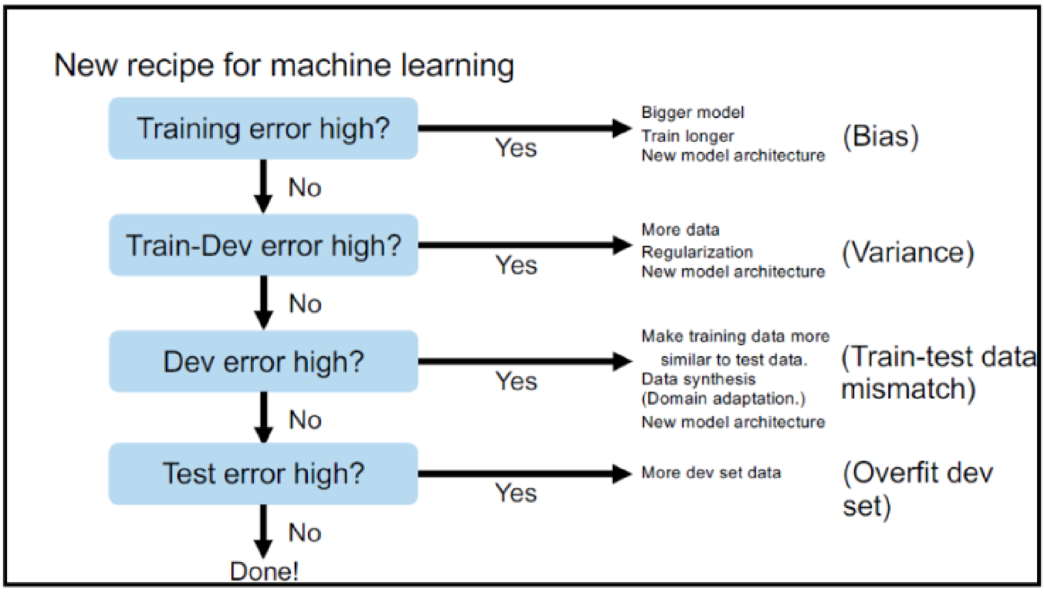

Figure 22: Machine Learning Chart

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 multiclasses. 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

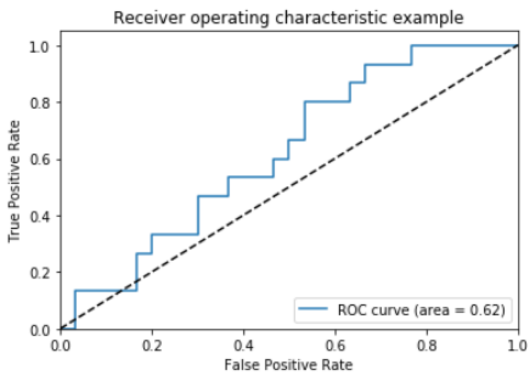

Figure 28: ROC for Class 0 - Linear Regression Model

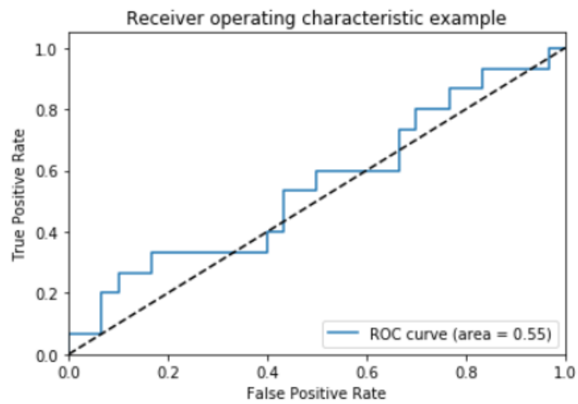

Figure 29: ROC for Class 1 - Linear Regression Model

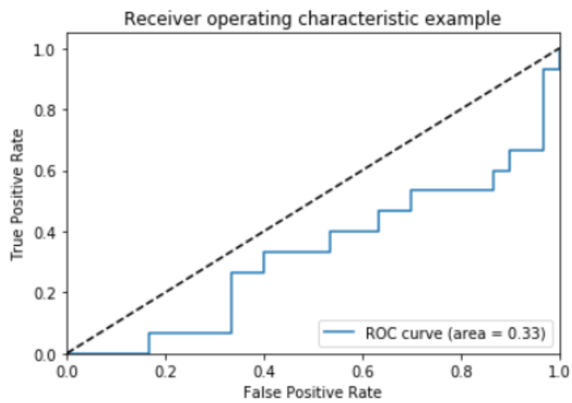

Figure 30: ROC for Class 2 - Linear Regression Model

5.2 Logistic Regression Model

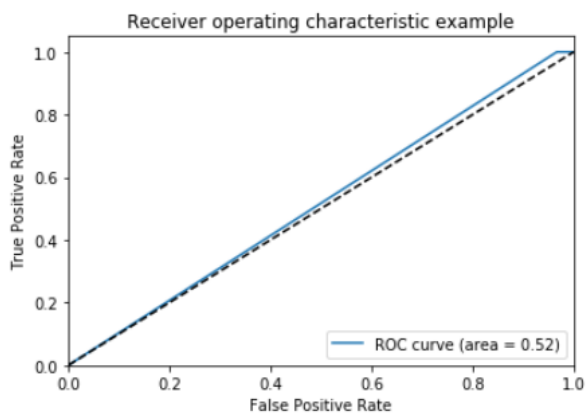

Figure 31: ROC for Class 0 - Logistic Regression Model

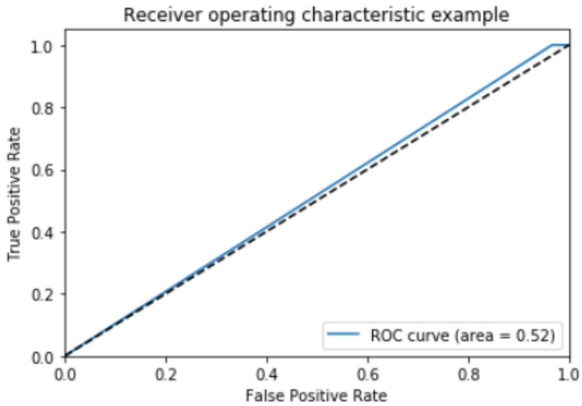

Figure 32: ROC for Class 1 - Logistic Regression Model

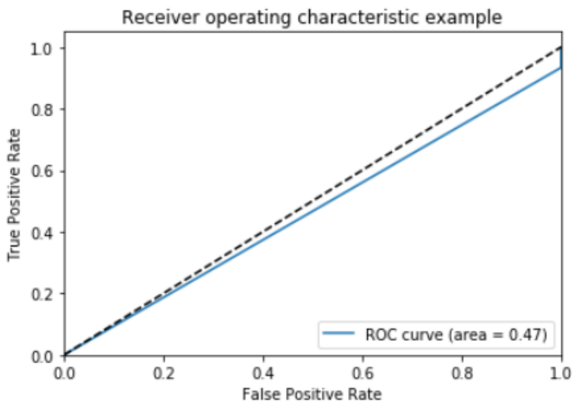

Figure 33: ROC for Class 2 - Logistic Regression Model

6 Appendix

6.1 Project Link

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

6.2 Code

```
"""CNG562-Assignment1.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/github/nisanuro/CNG562-Assignment-1/blob/master/CNG562_Assignment1.ipynb

# **CNG 562 - Assignment #1**

Linear Regression vs Logistic Regression using Iris Dataset\
Comparing:
*   Random 1-Hold Out
*   5-Fold
*   10-Fold
*   Strafied 1-Hold Out

\
Nisa Nur Odabas\
Kaan Taha Koken

---
"""

# Commented out IPython magic to ensure Python compatibility.
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, KFold, StratifiedKFold, cross_val_score
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn import metrics, datasets, preprocessing
from sklearn.metrics import roc_curve, auc
# %matplotlib inline

"""**K-Fold method**"""

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())

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

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))

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

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))

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

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)

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

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

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

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

```
  plt.ylabel('True Positive Rate')
  plt.title('Receiver operating characteristic example')
  plt.legend(loc="lower right")
  plt.show()

"""**Main**"""

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)

  '''#mean & standart deviation before mean removal
  print(X.mean(axis=0))
  print(X.std(axis=0))

  #mean & standart deviation after mean removal
  print(mean_removal.mean(axis=0))
  print(mean_removal.std(axis=0))'''

  #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)

"""# **Final**
**Training and Testing using:**
* **Raw data**
* **Stratified 1-Hold Out**
* **Linear Regression**

**Dividing Train and Test sets**
"""

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

"""Splitting dataset into 4.
```

```
    *   Train - 56%
    *   Train Dev - 14%
    *   Dev - 15%
    *   Test - 15%
    """

    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)

    """**Training**"""

    linReg = LinearRegression()
      linReg.fit(Train_x, Train_y)

    """**Testing**"""

    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))

      print("\ntrainDev_pred   \tTrainDev_y\trounded")
      for i, (j, k) in sorted(zip(trainDev_pred, zip(TrainDev_y,
                                  round_trainDev_pred))):
        print(i , "\t" , j, "\t\t", k)

    """Add TrainDev to Train and create new model"""

    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))
```

```

    print("\ndev_pred      \t       Dev_y\trounded")
    for i, (j, k) in sorted(zip(dev_pred, zip(Dev_y, round_dev_pred
                                  ))):
        print(i , "\t" , j, "\t", k)

    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))

    print("\ntest_pred      \t       Test_y\trounded")
    for i, (j, k) in sorted(zip(test_pred, zip(Test_y,
                                  round_test_pred))):
      print(i , "\t" , j, "\t", k)

    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))

    print("\ndevTest_pred  \t\t Dev+Test \trounded")
    for i, (j, k) in sorted(zip(devTest_pred, zip(Y_test,
                                  rounded_lin))):
      print(i , "\t" , j, "\t\t", k)

    """**Predicting [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))

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

    #calculating the score for roc curve
    scores=[]

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

    #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

    #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)

    #plotting ROC curve individual
```

```
    #For 0
    displayROC(fpr_0, tpr_0, roc_auc_0)

    #For 1
    displayROC(fpr_1, tpr_1, roc_auc_1)

    #For 2
    displayROC(fpr_2, tpr_2, roc_auc_2)

    #Average ROC
    sum_roc = roc_auc_2 + roc_auc_1 + roc_auc_0
    sum_roc = sum_roc / 3.0
    print("Average ROC curve (area) score: ", sum_roc)

    #Logistic Regression ROC calculation
    logReg = LogisticRegression(solver='liblinear', multi_class='
                                ovr')
    logReg.fit(Train_x, Train_y)
    devTest_pred_log = logReg.predict(X_test)

    #calculating the score for roc calculation
    scores=[]

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

    #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

    #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)

#plotting ROC curve individual

#For 0
displayROC(fpr_0, tpr_0, roc_auc_0)

#For 1
displayROC(fpr_1, tpr_1, roc_auc_1)

#For 2
displayROC(fpr_2, tpr_2, roc_auc_2)

#Average ROC
sum_roc = roc_auc_2 + roc_auc_1 + roc_auc_0
sum_roc = sum_roc / 3.0
print("Average ROC curve (area) score: ", sum_roc)
```