Assignment

*Name , Student Number *

`

```
import numpy as np
import pandas as pd
import operator
from collections import Counter
from sklearn.metrics import plot_confusion_matrix
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

from sklearn import neighbors
from math import sqrt
import math
```

- Note

Change the file path of class2.csv or class1.csv

Change the file path regr1.csv or regr2.csv

```
data= pd.read_csv('/content/class1.csv') # Read the CSV file
print('That shape  is',data.shape)
regr_2=pd.read_csv('/content/regr1.csv') # Read the CSV file
print('That shape is',regr_2.shape)
```

```
That shape  is (200, 3)
That shape is (400, 3)
```

Class label and features.

✓ 0s completed at 4:56 PM

```
features_c2=data.drop(['y'],axis=1) # Features of Class 2 File
print(features_c2.iloc[0:3])
label_c2 = data['y']            # Label of Class 2 File
label_c2.values.reshape(-1,1)       # Reshape the label into one coloumn
features_r2=regr_2.drop(['y'],axis=1)  # Features of reg 2 File
print(features_r2.iloc[0:5])        # Print the first three features of data
label_r2 = regr_2['y']
label_r2.values.reshape(-1,1)      # Reshape the label into one coloumn
print(label_r2.iloc[0:5])
```

```
     x1    x2
0  2.97  8.28
1  9.91  4.35
2  3.27  8.47
     x1    x2
0  0.62  0.99
1  0.71  0.71
2  0.99  0.77
3  0.78  0.84
4  0.77  0.05
0    0.97
1    1.74
2    1.63
3    1.91
4    2.36
Name: y, dtype: float64
```

KNN classification algorithm.

- Training 80% and Testing 20%

```
split = 0.8
end_of_training = int(len(data) * split)
train_set = data.iloc[:end_of_training]
test_set = data.iloc[end_of_training:]
# Converting dataset to train & test
train_set = train_set.values
test_set = test_set.values


#find the distance between test and training example

def euclideandistance(row1, row2):
  distance = []
  distance = [pow((row1[feature_number] - row2[feature_number]),2) for feature_numb
  euclideandistance = math.sqrt(sum(distance))
```

```
  euclideandistance = math.sqrt(sum(distance))
  return euclideandistance
  # find the k closest distances of example to the training set

def getk_closest(Traindata, Testdata, k):
  distances = []
  example1 = Testdata
  for train_number in range((Traindata).shape[0]):
    example2 = Traindata[train_number]
    distances.append((example2[2] , euclideandistance(example1, example2)))
  distances.sort(key=operator.itemgetter(1))
  k_closest = []
  k_closest = distances[:k]
  return k_closest
def prediction(K_c):
  lis2 = [x[0] for x in K_c]
  prediction = Counter(lis2).most_common(1)[0][0]
  return(prediction)

def score(test_set, prediction):
  score = []
  for test_number in range(len(test_set)):
    if test_set[test_number] == prediction[test_number]:
      score.append(1)
    else:
      score.append(0)
  accuracy = float(float(sum(score))/float(len(score))*100)

  return ((accuracy))
```

```
k  = 5
predictions = []
def main():
  for x in range(len(test_set)):
    k_closest = getk_closest(train_set, test_set[x], k)
    #print((k_closest))
    prediction(k_closest)
    predictions.append(prediction(k_closest))
    #print ('Test Number', x , '=> Predicted:', prediction(k_closest), ': Actual' ,
    #print ('Accuracy:' , score(test_set, predictions),'%')


main()
```

For knn classification for different value of k.

For knn classification for different value of k:

```
X_train, X_test, y_train, y_label = train_test_split(features_c2,label_c2, test_si
```

```
def NormalizeData(data):
    return (data - np.min(data)) / (np.max(data) - np.min(data))
```

```
X_train_std=NormalizeData(X_train)
X_test_std=NormalizeData(X_test)
```

- Dataset for validation

```
# Use the same function above for the validation set
X_train_val, X_val, y_train_val, y_val = train_test_split(X_train_std, y_train, tes
```

```
X_train_val.to_numpy()
X_val.to_numpy()
y_train_val.to_numpy
y_val.to_numpy()
# Converting dataset to train & test
train_set_val = X_train_val.values
test_set_val = X_val.values
np.size(train_set)
```

```
480
```

Find value of value of K Test vs Prdeict.

```
k_range = range(1, 18)
predictions = []
scores = []
for k in k_range:
  for x in range(len(test_set)):
    k_closest = getk_closest(train_set, test_set[x], k)
    #print((k_closest))
    prediction(k_closest)
    predictions.append(prediction(k_closest))
    #print ('Test Number', x , '=> Predicted:', prediction(k_closest), ': Actual' ,
    #print ('Accuracy:' , score(test_set, predictions),'%')
```

Develop The Model For best value of K

- For File class1 the value of K is 8
- For File class2 the value of K is 7

```
k  = 7
predictions = []
def main():
  for x in range(len(test_set)):
    k_closest = getk_closest(train_set, test_set[x], k)
    #print((k_closest))
    prediction(k_closest)
    predictions.append(prediction(k_closest))
    #print ('Test Number', x , '=> Predicted:', prediction(k_closest), ': Actual' ,
    #print ('Accuracy:' , score(test_set, predictions),'%')


main()
```

KNN regression algorithm.

```
features_r2  =features_r2.to_numpy()
label_r2=label_r2.to_numpy()


train_split_percent = 0.8

size = features_r2.shape[0]
X_train = features_r2[:int(train_split_percent * size),:]
X_test = features_r2[int(train_split_percent * size):,:]
y_train = label_r2[:int(train_split_percent * size)]
y_test = label_r2[int(train_split_percent * size):]

print("size of train data",X_train.size)
print("size of test data",X_test.size)
print("Train label",y_train.size)
print("Test label",y_test.size)
```

```
size of train data 640
size of test data 160
Train label 320
Test label 80
```

```
Test label 80
```

```
mu = np.mean(X_train, 0)
sigma = np.std(X_train, 0)
X_train = (X_train - mu ) / sigma
X_test = (X_test - mu ) / sigma
#Standardizing the y_train data
mu_y = np.mean(y_train, 0)
sigma_y = np.std(y_train, 0, ddof = 0)
y_train = (y_train - mu_y ) / sigma_y


#Changing the shape of the target varibale for easy computation

y_train = y_train.reshape(len(y_train),1)
y_test = y_test.reshape(len(y_test),1)
y_pred = np.zeros(y_test.shape)
y_train.shape, y_test.shape,y_pred.shape
```

```
((320, 1), (80, 1), (80, 1))
```

For KNN Regression model k=6

```
n_neigh = 6
for row in range(len(X_test)):
    euclidian_distance = np.sqrt(np.sum((X_train - X_test[row])**2, axis = 1 ))
    y_pred[row] = y_train[np.argsort(euclidian_distance, axis = 0)[:n_neigh]].mean

#Finding the root mean squared error

RMSE = np.sqrt(np.mean((y_test - y_pred)**2))
print(RMSE)
```

```
0.3168674253970859
```

Linear regression algorithm.

- Note

  Change the file path of class2.csv or class1.csv

  Change the file path regr1.csv or regr2.csv

```
regr_2=pd.read_csv('/content/regr1.csv') # Read the CSV file
print('That shape of data is',regr_2.shape) # check the shape of Data
X=regr_2.drop(['y'],axis=1)  # Features of reg 2 File
y = regr_2['y']
y.values.reshape(-1,1)      # Reshape the label into one coloumn
X=X.to_numpy()
y=y.to_numpy()
```

```
That shape of data is (400, 3)
```

```
X = X.reshape(-1,2)
y = y.reshape(-1,1)
m =y.size
```

```
# Calculating mean and standard deviation for feature normalization
mu = np.mean(X,axis=0)
sigma = np.std(X,axis=0)
mu,sigma
```

```
(array([0.503925, 0.4872  ]), array([0.28708421, 0.29722661]))
```

```
# Feature Normalization
def fnormalize(X,mu,sigma):
    X_norm = np.zeros(X.shape)
    for i in range(X.shape[1]):
        temp_X = X[:,i]
        temp_X = (temp_X-mu[i])/sigma[i]
        X_norm[:,i]=temp_X
    return X_norm
# Feature Normalization
# Adding a columns of ones to normalized X
X_norm = fnormalize(X,mu,sigma)
X_norm = np.hstack((np.ones((m,1)),X_norm))

# Intializing our parameter vector theta to zeros
theta = np.zeros((X_norm.shape[1],1))
theta

# Cost Function
def cost(X,y,theta):
    m = np.size(y)
    J = (1/(2.0*m))*np.sum(np.power((X.dot(theta)-y),2))
    return J
# Finding the cost with theta intialized to zeros
J = cost(X_norm,y,theta)
```

```
J = cost(X_norm,y,theta)
```

Gradient Descent.

```
# Some Gradient Descent Settings
alpha = 0.02
num_iters = 20
# Gradient Descent
def gdescent(X,y,theta,alpha,iters):
    X = np.mat(X); y=np.mat(y); theta = np.mat(theta);
    m = np.size(y)
    J_hist = np.zeros(iters)
    for i in range(0,iters):
        temp = theta - (alpha/m) * (X.T * (X*theta-y))
        theta = temp
        J_hist[i]= cost(X,y,theta)

    return np.asarray(theta),J_hist
```

```
# Getting the optimum parameters using gradient descent
final_theta,cost_hist = gdescent(X_norm,y,theta
                                        ,alpha,num_iters)
final_theta
```

```
array([[ 0.41883058],
       [ 0.18693662],
       [-0.03938644]])
```

```
# Cost at optimum theta
J_final = cost(X_norm,y,final_theta)
J_final
```

```
0.46920518133478767
```

```
# Pre processing the features
tempVal = np.array([0.87,0.2]).reshape(1,-1)
tempVal = fnormalize(tempVal,mu,sigma)
tempVal = np.hstack((np.ones((1,1)),tempVal))
tempVal
```

```
array([[ 1.        ,  1.2751485, -0.9662661]])
```

```
# Actual Prediction
prid = tempVal.dot(final_theta)
print ("Predicted value:\n ",prid)
```

```
Predicted value:
  [[0.69526031]]
```

KNN Regression model value of K

```
# Vectorized approach to find the
# We are setting a range of K values and calculating the RMSE for each of them. Th
k_list = [x for x in range(1,30,1)]

# Calculating the distance matrix using numpy broadcasting technique
distance = np.sqrt(((X_train[:, :, None] - X_test[:, :, None].T) ** 2).sum(1))

#Sorting each data points of the distance matrix to reduce computational effort
sorted_distance = np.argsort(distance, axis = 0)

#The knn function takes in the sorted distance and returns the RMSE of the
def knn(X_train,X_test,y_train,y_test,sorted_distance,k):
    y_pred = np.zeros(y_test.shape)
    for row in range(len(X_test)):

        #Transforming the y_train values to adjust the scale.
        y_pred[row] = y_train[sorted_distance[:,row][:k]].mean() * sigma_y + mu_y

    RMSE = np.sqrt(np.mean((y_test - y_pred)**2))
    return RMSE

#Storing the RMSE values in a list for each k value
rmse_list = []
for i in k_list:
    rmse_list.append(knn(X_train,X_test,y_train,y_test,sorted_distance,i))


print('RMSE value for k= ' , k_list , 'is:', rmse_list)

#Finding the optimal K value
min_rmse_k_value = k_list[rmse_list.index(min(rmse_list))]

#Finding the lowest possible RMSE
optimal_RMSE = knn(X_train,X_test,y_train,y_test,sorted_distance,min_rmse_k_value)
optimal_RMSE
```

```
RMSE value for k=  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17
0.29478895464005733
```

Bulid Model the knn regression

- For file regr2 the value of K is 11
- For file regr1 the value of K is 9

```
n_neigh = 11
for row in range(len(X_test)):
    euclidian_distance = np.sqrt(np.sum((X_train - X_test[row])**2, axis = 1 ))
    y_pred[row] = y_train[np.argsort(euclidian_distance, axis = 0)[:n_neigh]].mean

#Finding the root mean squared error

RMSE = np.sqrt(np.mean((y_test - y_pred)**2))
print(RMSE)
```

```
0.3022456583915291
```

Conclusion

- The predicted value of knn classification is increased by which accuracy is increased
- The RMSE for KNN regression model has been improved
- In last linear regression works better for two class mean for class 1 file classsification model as compare to multi classification model(more than two class) or class 2 file