**Analysis Report**

**1(a).**

**Generic K-Fold Split Function():**

def NfoldSplit(self,Data,n):

      NewData = []

      size = (Data.shape[0]//n)+1

      for i in range(n):

        T = Data[i\*size:i\*size+size]

        NewData.append(T)

      return NewData

**Predict() code scratch :**

def predict(self,X\_test):

      X\_test = X\_test.to\_numpy()

      y\_predicted =(np.sum(X\_test\*self.Coff,axis = 1)) # self.Coff stores the weights

      return y\_predicted

**Scratch Code for Mean Squared Error :**

def MSE(self,ytrue,ypred):

      MSE = np.square(ytrue-ypred)

      MSE = np.sum(MSE)/len(ytrue)

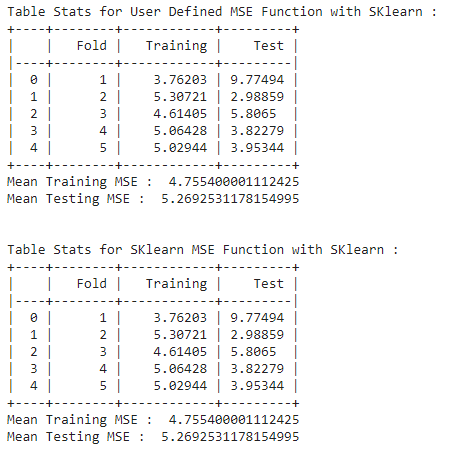
      return MSE

**1(b).**

**Approach :**

* Load the data using the loadmat() function.
* Divide the dataset into a list of folds(as per the number of folds)
* For each fold :
* Train the model using : LinearRegression(fit\_intercept=False).fit(Train[X], Train[Y])
* Predict the values for the training and testing data using the User-defined Predict Method.
* Calculate the MSE over training and testing data using the User Defined Method.
* Calculate the MSE over training and testing data using Sklearn Method.
* Store the above values for each fold.
* Print Table Stats with mean training and testing MSE.

**Output :**



**Observation :**

* The MSE function of Sklearn and user-defined functions give the same results.

**1(c).**

**Approach :**

* Load the data using the loadmat() function.
* Divide the dataset into a list of folds(as per the number of folds)
* For each fold :
* Calculate the regression coefficients using :

Coff = np.matmul(TrainX\_Tr,TrainX)

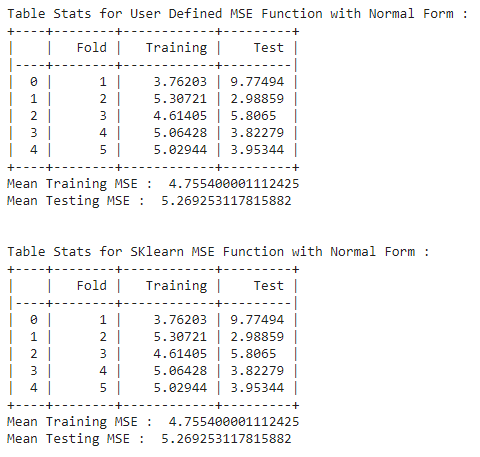
Coff = np.linalg.inv(Coff)

Coff = np.matmul(Coff,TrainX\_Tr)

Coff = np.matmul(Coff,TrainY)

* Predict the values for the training and testing data using the User-defined Predict Method and above calculated coefficients.
* Calculate the MSE over training and testing data using the User Defined Method.
* Calculate the MSE over training and testing data using Sklearn Method.
* Store the above values for each fold.
* Print Table Stats with mean training and testing MSE.

**Output :**



**Observation :**

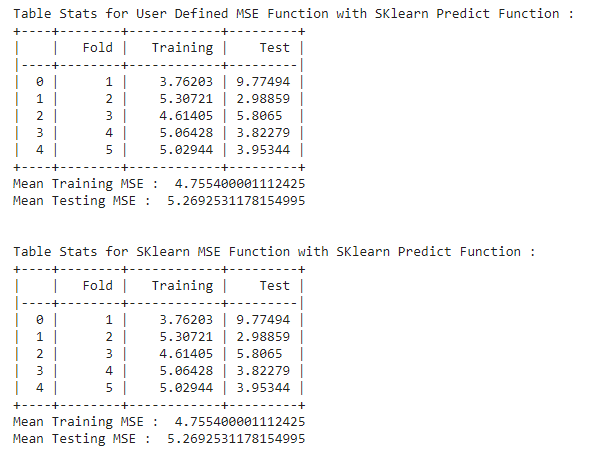
* The performance of both MSE functions is the same.

**1(d).**

**Approach :**

* Load the data using the loadmat() function.
* Divide the dataset into a list of folds(as per the number of folds)
* For each fold :
* Train the model using : LinearRegression(fit\_intercept=False).fit(Train[X], Train[Y])
* Predict the values for the training and testing data using Sklearn Predict Method.
* Calculate the MSE over training and testing data using the User Defined Method.
* Calculate the MSE over training and testing data using Sklearn Method.
* Store the above values for each fold.
* Print Table Stats with mean training and testing MSE.

**Output :**



**Observation :**

* The performance of both methods is the same.
* There is no performance deviation in the three approaches.

**2.1.**

**Scatter Plotting :**

def plot(df):

  plt.figure(figsize=(10,5))

  sn.set(style = 'whitegrid')

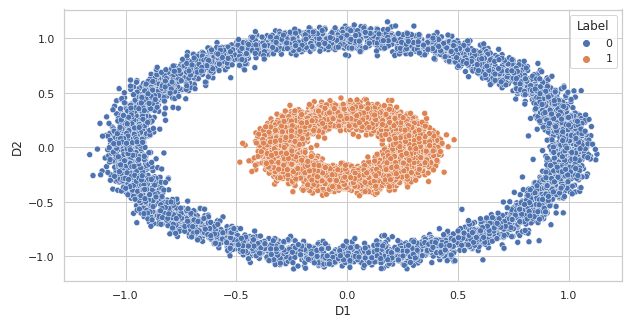
  sn.scatterplot(data= df,x='D1',y='D2',hue='Label')

  plt.xlabel('D1')

  plt.ylabel('D2')

  plt.show()

**Output :**



**Observation :**

* The classes are not linearly separable.
* Both classes belong to the data points that follow the circular distribution.

**2.2.**

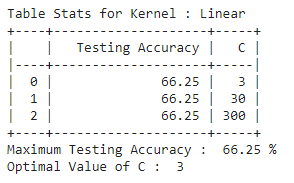
**Approach :**

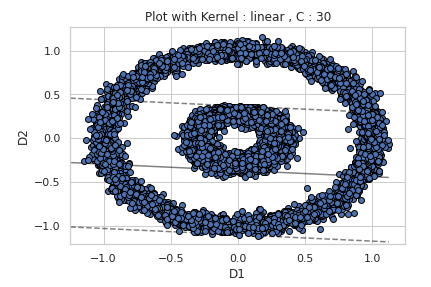
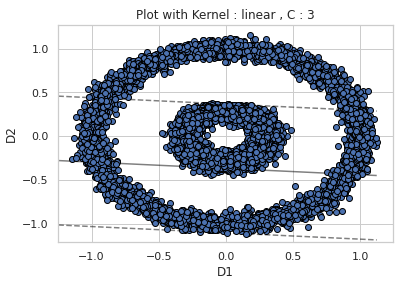
* Load the data using the loadmat() function.
* Perform the dataset splitting into Training data and Testing data with test size = 0.2.
* Create the list of Permissible values of C
* For each c in C :
* Fit the SVM model for the Linear Kernel
* Predict the class labels for the Testing Data using the user-defined Predict Method.
* Calculate the testing accuracy for each value of c and store it.
* Plot the margins, decision boundary, and support vectors for each value of c.
* Print Table Stats and optimal values of C with corresponding testing accuracy.

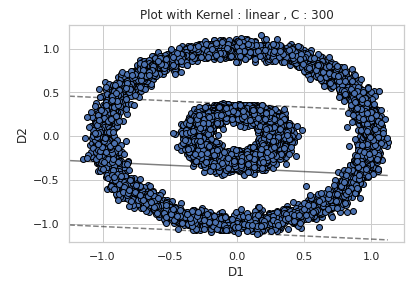
**List of C values taken :**

* 3
* 30
* 300

**Output :**







**Observation :**

* The value of testing accuracy for different values is the same.
* The testing accuracy is approximately equal to 66.25 %.
* Because of the linear kernel, the testing accuracy is low as the data is not linearly separable.

**2.3.**

**Approach :**

* Load the data using the loadmat() function.
* Perform the dataset splitting into Training data and Testing data with test size = 0.2.
* Create the list of Permissible values of C and Gamma.
* For each c in C :
* For each g in Gamma:
* Fit the SVM model for the RBF kernel.
* Predict the class labels for the Testing Data using the user-defined Predict Method.
* Calculate the testing accuracy for a combination of c and g value and store it.
* Plot the margins, decision boundary, and support vectors for each combination of c and g value.
* Print Table Stats and optimal values of C and Gamma with corresponding testing accuracy.

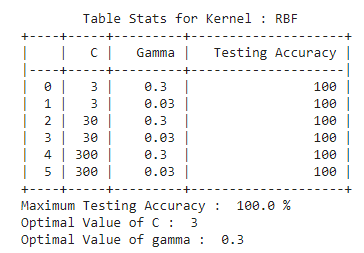
**List of C values taken :**

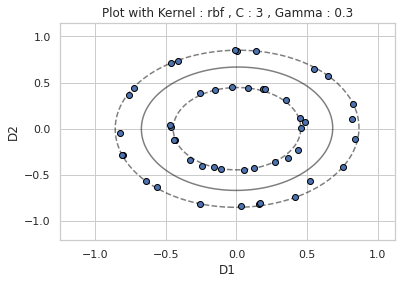
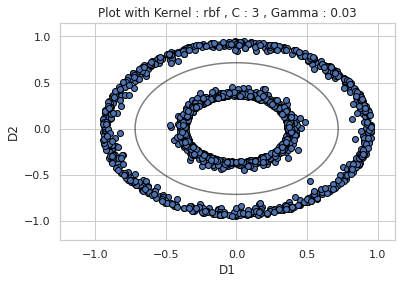
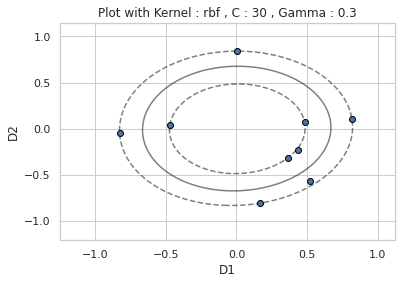
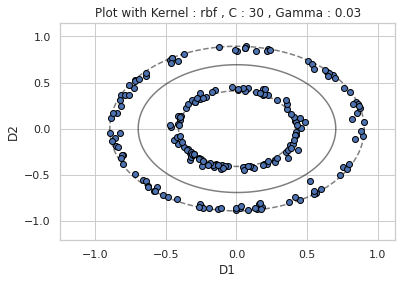
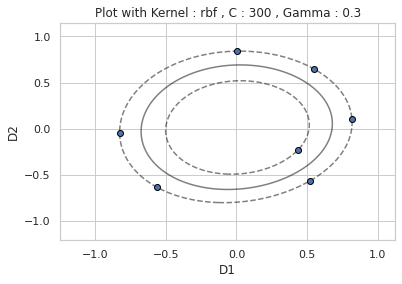
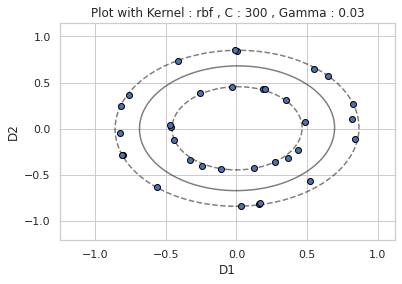
* 3
* 30
* 300

**List of Gamma values taken :**

* 0.3
* 0.03

**Output :**



**     **

**Observation :**

* The value of testing accuracy for a different combination of C and Gamma values is the same, and equal to 100%.
* Because of the use of the RBF kernel, the decision boundary will be circular that entirely separates the data points of both the classes.
* Because of this separation, the testing accuracy is 100%.

**2.4.**

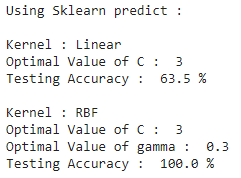
**Approach for Linear Kernal :**

* Load the data using the loadmat() function.
* Perform the dataset splitting into Training data and Testing data with test size = 0.2.
* Set C value equal to the optimal value of C obtained from question 2.2.
* Fit the SVM model with the linear kernel and above C value.
* Predict the class labels for the Testing Data using Sklearn Predict Method.
* Calculate the testing accuracy and print it.

**Approach for RBF Kernal :**

* Load the data using the loadmat() function.
* Perform the dataset splitting into Training data and Testing data with test size = 0.2.
* Set C value equal to the optimal value of C obtained from question 2.3.
* Set g value equal to the optimal value of g obtained from question 2.3.
* Fit the SVM model with the RBF kernel and above C and gamma value.
* Predict the class labels for the Testing Data using Sklearn Predict Method.
* Calculate the testing accuracy and print it.

**Output :**



**Observation :**

* For the linear kernel, there is an approximately 3% drop in testing accuracy for the optimal value of C.
* For the RBF kernel, there is not a performance deviation between the user-defined and the sklearn methods.

**3.1.**

**Scatter Plotting :**

def Scatter\_plot(self,arg):

      Data = self.Load\_DataSet(arg)

      Data = pd.DataFrame(Data, columns =['D1','D2','Label'])

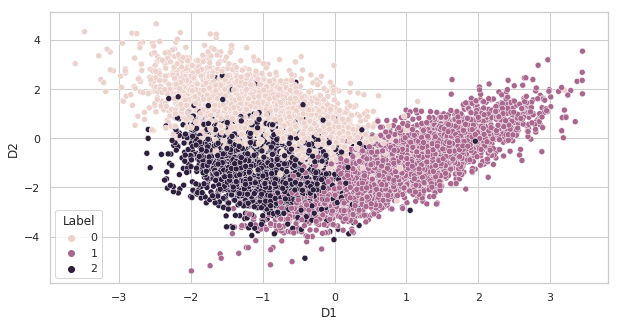
      plt.figure(figsize=(10,5))

      sn.set(style = 'whitegrid')

      sn.scatterplot(data = Data ,x='D1',y='D2',hue='Label')

      plt.show()

**Output :**



**Observation :**

* The dataset contains the three classes.
* Classes are not linearly separable.

**3.2.**

**Approach :**

* Load the data using the loadmat() function.
* Divide the dataset into a list of folds(as per the number of folds).
* Perform the one-hot encoding of the training dataset.
* For each fold :
* For each c in C :
* For each g in Gamma:
* Fit the training data on the C(classes ) number of SVM classifiers and RBF kernel.
* Assign class label for the testing data that has maximum value for the decision function output.
* Report the accuracy testing data using the above-predicted class labels.
* Report the classwise accuracy of testing data using the above-predicted class labels.
* Store the above accuracies values for each fold**.**
* For each fold, report the maximum testing accuracy with the optimal value of C and Gamma.
* Print the classwise accuracy for each fold.
* Report the mean testing accuracy across each fold and class labels.

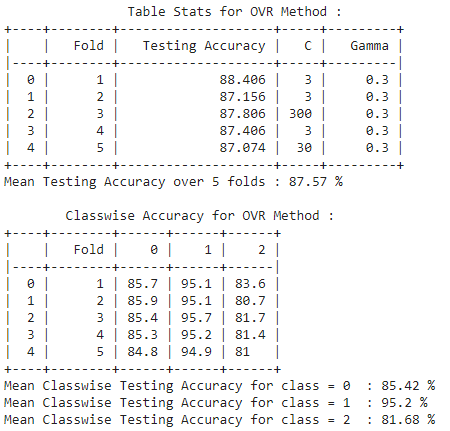
**List of C values taken :**

* 3
* 30
* 300

**List of Gamma values taken :**

* 0.3
* 0.03

**Output :**



**Observations :**

* The testing accuracy across each fold is approximately each to 87 %.
* The classwise testing accuracy for class 1 is high as compared to class 0 and class 2.
* Class 2 has the least classwise testing accuracy.

**3.3.**

**Approach :**

* Load the data using the loadmat() function.
* Create the list of a different combination of classifiers which is needed to be trained.
* Divide the dataset into a list of folds(as per the number of folds).
* For each fold :
* For each c in C :
* For each g in Gamma:
* For each classifier(A, B) in the above list :
* Extract the data points from the Training Data that includes the class label as either A or B and then encode the training data to 1/0 format.
* Fit the above training data using the SVM classier with RBF kernel.
* Report the class labels for the testing data for each classifier.
* Now, assign class labels to testing data using the Maximum Voting Rule.
* Report the accuracy testing data using the above-predicted class labels.
* Report the classwise accuracy of testing data using the above-predicted class labels.
* Store the above accuracies values for each fold**.**
* For each fold, report the maximum testing accuracy with the optimal value of C and Gamma.
* Print the classwise accuracy for each fold.
* Report the mean testing accuracy across each fold and class labels.

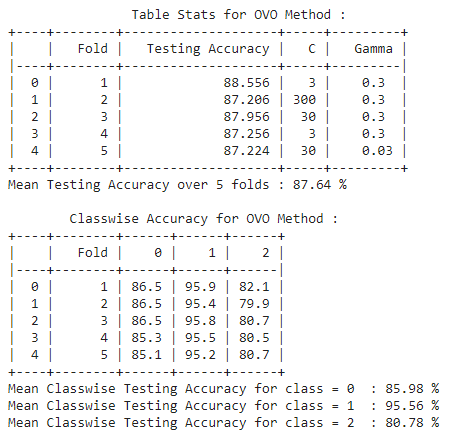
**List of C values taken :**

* 3
* 30
* 300

**List of Gamma values taken :**

* 0.3
* 0.03

**Output :**



**Observation :**

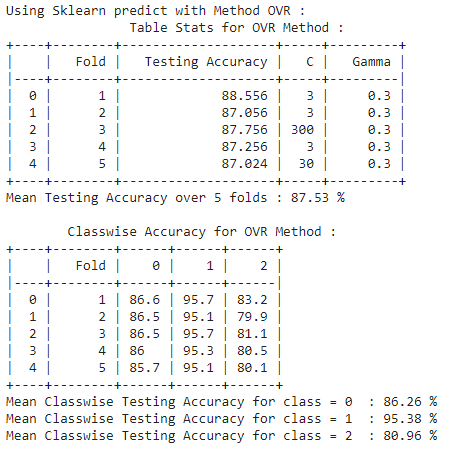
* The testing accuracy across each fold is approximately each to 87.5 %.
* The classwise testing accuracy for class 1 is high as compared to class 0 and class 2.
* Class 2 has the least classwise testing accuracy.

**3.4.**

**Approach for OVR :**

* Load the data using the loadmat() function.
* Divide the dataset into a list of folds(as per the number of folds).
* Set C value equal to the optimal value of C obtained from question 3.2.
* Set g value equal to the optimal value of g obtained from question 3.2.
* For each fold :
* Fit the above training data using a sklearn OVR classifier with RBF Kernal and above C and gamma value.
* Predict the class label for testing data.
* Report the accuracy of testing data using the above-predicted class labels.
* Report the classwise accuracy for testing data using the above-predicted class labels.
* Store the above accuracies for each fold**.**
* Print Table Stats

**Output :**



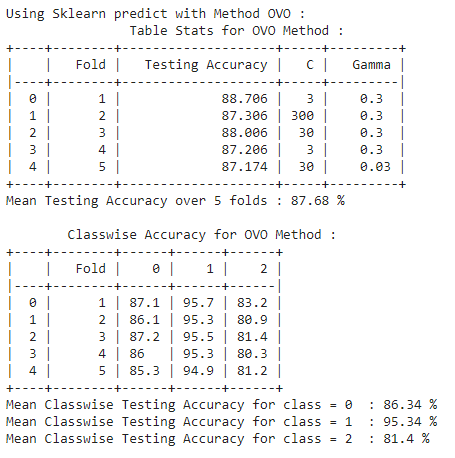
**Observation :**

* As compared to the user-defined ovr method, mean testing across all the folds is approximately the same.
* The classwise accuracies testing data across each fold is also the same(approx).

**Approach for OVO :**

* Load the data using the loadmat() function.
* Divide the dataset into a list of folds(as per the number of folds).
* Set C value equal to the optimal value of C obtained from question 3.3.
* Set g value equal to the optimal value of g obtained from question 3.3.
* For each fold :
* Fit the above training data using a sklearn OVO classifier with RBF Kernal and above C and gamma value.
* Predict the class label for testing data.
* Report the accuracy of testing data using the above-predicted class labels.
* Report the classwise accuracy for testing data using the above-predicted class labels.
* Store the above accuracies for each fold**.**
* Print Table Stats

**Output :**



**Observation :**

* As compared to the user-defined ovo method, the mean testing across all the folds is approximately the same.
* The classwise accuracies testing data across each fold is also the same(approx).