# DSC ASSIGNMENT 10

P MANOHAR RAO

197158

CSE-A

a. Dataset Iris.csv

```
#importing all the libraries
import numpy as np
import pandas as pd

df=pd.read_csv('/content/iris.csv')
print(df.head())
print(df['Species'].unique())

X=df.iloc[:,:-1]
Y=df.iloc[:-1]
```

```
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0            5.1           3.5            1.4           0.2  Iris-setosa
1            4.9           3.0            1.4           0.2  Iris-setosa
2            4.7           3.2            1.3           0.2  Iris-setosa
3            4.6           3.1            1.5           0.2  Iris-setosa
4            5.0           3.6            1.4           0.2  Iris-setosa
['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

b . Group the training data into its respective classes

```
dict={}
for s,rows in df.groupby(['Species']):  #groups the dataframe based on species has species
  rows=np.array(rows)
  rows=rows[:,:-1]                      #removing species column
  dict[s]=rows                          #adds in the dictionary
print(dict.keys())                        #data is saved in dictionary
#dict['Iris-setosa']
```

```
dict_keys(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'])
```

c. Calculate mean vector of given training data of K-dimensions excluding the target class and calculate class-wise mean vector for the given training data

```
feature_means=df.iloc[:,:-1].mean().values
class_wise_mean=df.groupby(['Species']).mean().values
print("feature_means")
```

```
print(df.iloc[:,:-1].mean())
print("class-wise means")
print(df.groupby(['Species']).mean())
#print("class-wise mean vector : " ,class_wise_mean)
#print("feature_means : ",feature_means)
```

```
feature_means
SepalLengthCm    5.843333
SepalWidthCm     3.054000
PetalLengthCm    3.758667
PetalWidthCm     1.198667
dtype: float64
class-wise means
                 SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
Species
Iris-setosa              5.006         3.418          1.464         0.244
Iris-versicolor          5.936         2.770          4.260         1.326
Iris-virginica           6.588         2.974          5.552         2.026
```

d. Calculate scatter matrices needed to maximize the difference between means of given classes and minimize the variance of given classes.

```
SW=np.zeros((4,4))     #scatter within classes for each class i (x-meani)(x-meani)T
i=0
for s in dict.keys():
  cmv=class_wise_mean[i].reshape(-1,1)    #classwise mean
  s1=np.zeros((4,4))                      #initailly all 0s
  for x in dict[s]:
    x=x.reshape(-1,1)
    s1=s1+np.dot((x-cmv),((x-cmv).T))   #shape featurescount x featurescount
  SW=SW+s1
  i+=1
print("Scatter within classes : ")
print(SW)
```

```
Scatter within classes :
[[38.95620000000002 13.683 24.614000000000004 5.6556000000000015]
 [13.683 17.035000000000001 8.12 4.913200000000002]
 [24.614000000000004 8.12 27.220000000000006 6.2536000000000005]
 [5.6556000000000015 4.913200000000002 6.2536000000000005
  6.175599999999999]]
```

```
N=df['Species'].value_counts().values     #number of instances in each class
print(N)
```

```
[50 50 50]
```

```
SB=np.zeros((4,4))    #scatter between classes for each class i Ni*(meani-mean)(meani-mean
for i in range(3):    #for each class
  cmv=class_wise_mean[i].reshape(-1,1)
  fmv=feature_means.reshape(-1,1)
  SB+=N[i]*(np.dot((cmv-fmv),(cmv-fmv).T))
```

```
print(SB)
```

```
[[ 63.21213333 -19.534      165.16466667  71.36306667]
 [-19.534       10.9776     -56.0552     -22.4924    ]
 [165.16466667 -56.0552     436.64373333 186.90813333]
 [ 71.36306667 -22.4924     186.90813333  80.60413333]]
```

e. Calculate eigen values of M and get eigen vector pairs for first n (needed ) dimensions.

```
SW=SW.astype('float64')                       #should set astypecto float64 to avoid ufunc
eigen_values, eigen_vectors = np.linalg.eig(np.linalg.inv(SW).dot(SB)) #np.linalg.eig Comp

print(eigen_values)
print(eigen_vectors)
```

```
[ 3.22719578e+01  2.77566864e-01 -1.65208523e-15  1.05677803e-14]
[[-0.20490976 -0.00898234 -0.69759457  0.50743391]
 [-0.38714331 -0.58899857  0.02388885 -0.44455564]
 [ 0.54648218  0.25428655 -0.03913212 -0.48660978]
 [ 0.71378517 -0.76703217  0.71502434  0.55506039]]
```

f. Selecting Linear Discriminants for the new features subspace

i. Sorting eigen vectors by decreasing eigenvalues

```
print(eigen_values)
inc=eigen_values.argsort()
print(eigen_values[inc])
dec=(-eigen_values).argsort()[:]
print(eigen_values[dec])
```

```
[ 3.22719578e+01  2.77566864e-01 -1.65208523e-15  1.05677803e-14]
[-1.65208523e-15  1.05677803e-14  2.77566864e-01  3.22719578e+01]
[ 3.22719578e+01  2.77566864e-01  1.05677803e-14 -1.65208523e-15]
```

```
sorted_indices=np.argsort(-eigen_values)      #if we negate the values then we get the des
print("BEFORE SORTING:")
print(eigen_values)
sorted_eigenvalues=eigen_values[sorted_indices]
print("AFTER SORTING:")
print(sorted_eigenvalues)
sorted_eigenvectors=eigen_vectors[:,sorted_indices] #applied to all columns
print("SORTED EIGEN VECTORS")
print(sorted_eigenvectors)
```

```
BEFORE SORTING:
[ 3.22719578e+01  2.77566864e-01 -1.65208523e-15  1.05677803e-14]
AFTER SORTING:
[ 3.22719578e+01  2.77566864e-01  1.05677803e-14 -1.65208523e-15]
SORTED EIGEN VECTORS
[[-0.20490976 -0.00898234  0.50743391 -0.69759457]
```

```
 [-0.38714331 -0.58899857 -0.44455564  0.02388885]
 [ 0.54648218  0.25428655 -0.48660978 -0.03913212]
 [ 0.71378517 -0.76703217  0.55506039  0.71502434]]
```

ii. Choosing k eigen vectors with the largest eigenvalues

```
k=2 #lets say 2
w_matrix=sorted_eigenvectors[:,0:k] #choosing K eigen vectors

#first column in our rearranged Eigen vector-matrix
# will be a linear discriminant component that captures the highest variability.

print(w_matrix)
```

```
[[-0.20490976 -0.00898234]
 [-0.38714331 -0.58899857]
 [ 0.54648218  0.25428655]
 [ 0.71378517 -0.76703217]]
```

g. Transforming the samples onto the new subspace.

```
X_lda=np.array(X.dot(w_matrix))
print(X_lda.shape)
#X_lda is composed of the LDA components, or said yet another way, the new feature space.
```

```
(150, 2)
```

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

```
#importing all the libraries
import numpy as np
import pandas as pd

df=pd.read_csv('/content/Wine.csv')
#print(df.info())
print(df.head())
print("UNIQUE Customer_Segment values: ", df['Customer_Segment'].unique())
```

```
   Alcohol  Malic_Acid   Ash  Ash_Alcanity  Magnesium  Total_Phenols  \
0    14.23        1.71  2.43          15.6        127           2.80   
1    13.20        1.78  2.14          11.2        100           2.65   
2    13.16        2.36  2.67          18.6        101           2.80   
3    14.37        1.95  2.50          16.8        113           3.85   
4    13.24        2.59  2.87          21.0        118           2.80   

   Flavanoids  Nonflavanoid_Phenols  Proanthocyanins  Color_Intensity   Hue  \
0        3.06                  0.28             2.29             5.64  1.04   
1        2.76                  0.26             1.28             4.38  1.05   
2        3.24                  0.30             2.81             5.68  1.03   
```

```
3        3.49                    0.24             2.18             7.80  0.86
4        2.69                    0.39             1.82             4.32  1.04

   OD280  Proline  Customer_Segment
0   3.92     1065                 1
1   3.40     1050                 1
2   3.17     1185                 1
3   3.45     1480                 1
4   2.93      735                 1
UNIQUE Customer_Segment values:  [1 2 3]
```

```
#Storing the target and features
X=df.iloc[:,:-1]
Y=df.iloc[:,-1]
print(X.shape)
print(Y.shape)
```

```
(178, 13)
(178,)
```

### b. Feature Scaling

```
from sklearn.preprocessing import StandardScaler      #Preprocessing, Feature scaling
scaler=StandardScaler()
X=scaler.fit_transform(X)
print(X.shape)
```

```
(178, 13)
```

### c. Split the dataset

```
from sklearn.model_selection import train_test_split  #train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,shuffle=True,random_state

print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(142, 13)
(142,)
(36, 13)
(36,)
```

### d. Apply LDA

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda=LDA(n_components=1)                           #transformed features or LDA components
X_train=lda.fit_transform(X_train,Y_train)
X_test=lda.transform(X_test)
```

## e. Train the model with Logistic regression

```
#Building the logistic Regression Model
from sklearn.linear_model import LogisticRegression
clf=LogisticRegression()

#train the model
clf.fit(X_train,Y_train)

#making predictions
y_pred=clf.predict(X_test)
```

## f. Compute the Confusion matrix

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

print("Confusion Matrix")
print(confusion_matrix(Y_test,y_pred))

print("Accuracy : ",str(round(accuracy_score(Y_test,y_pred),2)))
```

```
Confusion Matrix
[[18  0  0]
 [ 0  9  0]
 [ 0  1  8]]
Accuracy :  0.97
```