Q2. For the same dataset (2 classes, male and female)

> **a) Use LDA to reduce the dimension from d to d’. (Here d=128)**

> **b) Choose the direction W to reduce the dimension d' (select appropriate d').**

> **c) Use d’ features to classify the test cases (any classification algorithm will do, Bayes classifier, minimum distance classifier, and so on)**

importing the necessary libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

+ Code + Text

```
from google.colab import files
uploaded = files.upload()
```

Choose files gender_featu...vectors.csv

- **gender_feature_vectors.csv**(text/csv) - 1279817 bytes, last modified: 25/03/2021 - 100% done

```
Saving gender_feature_vectors.csv to gender_feature_vectors (2).csv
```

```
df=pd.read_csv("gender_feature_vectors.csv")
df.head()
```

	Unnamed: 0	Unnamed: 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	1	male	-0.066420	0.151611	0.027740	0.052771	-0.066105	-0.041232	-0.002637	-0.158467	0.130467	-0.044872	0.272529	-0.107907	-0.190014	-0.145586	-0.012682	0.154819	-0.24
1	2	male	-0.030614	0.049667	0.008084	-0.050324	0.007649	-0.063818	-0.019530	-0.119905	0.186553	-0.044821	0.271853	-0.041583	-0.252784	-0.117582	-0.040385	0.112987	-0.19
2	3	male	-0.096178	0.061127	0.035326	-0.035388	-0.090728	-0.018634	-0.024315	-0.139786	0.052211	-0.052085	0.248798	-0.023033	-0.284685	-0.207826	0.078375	0.110781	-0.09
3	4	male	-0.103057	0.085044	0.078333	-0.035873	-0.028163	0.004924	0.007829	-0.017016	0.114907	-0.056267	0.216984	-0.018399	-0.181335	-0.075995	-0.127034	0.093428	-0.17
4	5	male	-0.125815	0.120046	0.023131	-0.042901	0.038215	-0.049677	-0.054258	-0.130758	0.173457	-0.011889	0.175231	-0.039689	-0.141731	-0.143041	-0.017035	0.074751	-0.17

5 rows × 130 columns

```
df.shape
```

```
(800, 130)
```

```
df.drop(['Unnamed: 0'],axis=1,inplace=True)
```

```
df.head()
```

	Unnamed: 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
0	male	-0.066420	0.151611	0.027740	0.052771	-0.066105	-0.041232	-0.002637	-0.158467	0.130467	-0.044872	0.272529	-0.107907	-0.190014	-0.145586	-0.012682	0.154819	-0.241271	-0.16
1	male	-0.030614	0.049667	0.008084	-0.050324	0.007649	-0.063818	-0.019530	-0.119905	0.186553	-0.044821	0.271853	-0.041583	-0.252784	-0.117582	-0.040385	0.112987	-0.199148	-0.05
2	male	-0.096178	0.061127	0.035326	-0.035388	-0.090728	-0.018634	-0.024315	-0.139786	0.052211	-0.052085	0.248798	-0.023033	-0.284685	-0.207826	0.078375	0.110781	-0.099561	-0.15
3	male	-0.103057	0.085044	0.078333	-0.035873	-0.028163	0.004924	0.007829	-0.017016	0.114907	-0.056267	0.216984	-0.018399	-0.181335	-0.075995	-0.127034	0.093428	-0.176822	-0.12
4	male	-0.125815	0.120046	0.023131	-0.042901	0.038215	-0.049677	-0.054258	-0.130758	0.173457	-0.011889	0.175231	-0.039689	-0.141731	-0.143041	-0.017035	0.074751	-0.177354	-0.14

5 rows × 129 columns

```
df2=df.drop(['Unnamed: 1'],axis=1)
Class=np.array(df2)
print(Class)
```

```
[[-0.06641996  0.15161145  0.02773961 ... -0.07640016  0.10749723
   0.00156654]
 [-0.03061386  0.04966652  0.00808374 ...  0.0176384   0.08060966
  -0.01592966]
 [-0.09617768  0.06112669  0.03532604 ...  0.01739147  0.057652
   0.08611634]
 ...
 [-0.20285167  0.0370395   0.07973114 ...  0.03738441 -0.00625749
   0.03997689]
 [-0.08829999  0.06353012  0.04962703 ...  0.00970074 -0.01694169
   0.04807128]
 [-0.15620135  0.05516458  0.14271647 ... -0.0102984  -0.02885648
   0.0753232 ]]
```

```
class_wise=df.groupby('Unnamed: 1')
```

```
class_wise.head()
```

	Unnamed: 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
0	male	-0.066420	0.151611	0.027740	0.052771	-0.066105	-0.041232	-0.002637	-0.158467	0.130467	-0.044872	0.272529	-0.107907	-0.190014	-0.145586	-0.012682	0.154819	-0.241271	-0.
1	male	-0.030614	0.049667	0.008084	-0.050324	0.007649	-0.063818	-0.019530	-0.119905	0.186553	-0.044821	0.271853	-0.041583	-0.252784	-0.117582	-0.040385	0.112987	-0.199148	-0.

splitting the dataset wrt to both the classes

```
class0=Class[:399]
class1=Class[399:]
# class0.head()
print(class0)
```

```
[[-0.06641996  0.15161145  0.02773961 ... -0.07640016  0.10749723
   0.00156654]
 [-0.03061386  0.04966652  0.00808374 ...  0.0176384   0.08060966
  -0.01592966]
 [-0.09617768  0.06112669  0.03532604 ...  0.01739147  0.057652
   0.08611634]
 ...
 [-0.15846041  0.10994754  0.01908765 ...  0.05034712  0.07146505
  -0.02295445]
 [-0.10149928  0.11973877  0.01695069 ... -0.0301937  -0.01864216
   0.03282067]
 [-0.14951558  0.081588    0.09079567 ... -0.07546981  0.06248058
   0.05272722]]
```

```
print(class1)
```

```
[[ 0.03984397  0.07035663  0.13019608 ...  0.08094045  0.01146721
  -0.02199927]
 [ 0.0017469   0.18567775  0.07325977 ... -0.01768997  0.06702832
   0.03645249]
 [-0.09159789  0.09533963  0.07212528 ...  0.01689966 -0.08167572
   0.02280894]
 ...
 [-0.20285167  0.0370395   0.07973114 ...  0.03738441 -0.00625749
   0.03997689]
 [-0.08829999  0.06353012  0.04962703 ...  0.00970074 -0.01694169
   0.04807128]
 [-0.15620135  0.05516458  0.14271647 ... -0.0102984  -0.02885648
   0.0753232 ]]
```

LDA

```
def lda(class0,class1,d_prime):
    mean0=np.average(class0, axis=0)
    mean1=np.average(class1, axis=0)
#     print(mean0)
#     print(mean1)
    Sw = np.zeros((len(mean0), len(mean0)))
```

```
    for row in class0:
        subbed = (row - mean0).reshape((len(mean0), 1))
        dotted = np.dot(subbed, subbed.T)
        Sw += dotted

    for row in class1:
        subbed = (row - mean1).reshape((len(mean1), 1))
        dotted = np.dot(subbed, subbed.T)
        Sw += dotted

    Sb = np.zeros((len(mean0), len(mean0)))
    subbed = (mean0 - np.average(Class, axis = 0)).reshape((len(mean0), 1))
    Sb += len(class0) * np.dot(subbed, subbed.T)

    subbed = (mean1 - np.average(Class, axis = 0)).reshape((len(mean1), 1))
    Sb += len(class1) * np.dot(subbed, subbed.T)

    # finding eigen values and eigen vectors for Sw^-1*Sb.
    e_val, e_vec = np.linalg.eigh(np.dot(np.linalg.inv(Sw), Sb))

    sorted_e_val = np.flip(np.sort(e_val))
    sorted_e_vec = e_vec.copy()
    dummy = 0

    #mapping the eigen vectors to the corresponding sorted eigen values
    for val in sorted_e_val:
        ind = np.argmax(e_val == val * 1)
        sorted_e_vec[:,dummy] = e_vec[:,ind]
        dummy +=1



    new_feat = np.dot(Class, sorted_e_vec)

    return new_feat[:,:d_prime]

# lda(class0,class1,50)
```

```
lda_model=lda(class0,class1,1)
```

```
male=lda_model[:399]
female=lda_model[399:]
male
```

```
    array([[-0.09433745],
           [-0.10278821],
           [-0.14922404],
           [-0.08944469],
           [-0.05134645],
           [-0.06831653],
           [-0.18565844],
           [-0.25259519],
           [-0.1166957 ],
           [-0.12887256],
```

```
        [-0.17158275],
        [-0.15071103],
        [-0.15007123],
        [-0.08988632],
        [-0.16843363],
        [-0.15953118],
        [-0.18257427],
        [-0.118917  ],
        [-0.05086052],
        [-0.19247886],
        [-0.12832977],
        [-0.20866478],
        [-0.12207994],
        [-0.13916045],
        [-0.13278016],
        [-0.151321  ],
        [-0.14201097],
        [-0.08436291],
        [-0.09044554],
        [-0.07374715],
        [-0.17886296],
        [-0.13910243],
        [-0.12341403],
        [-0.06697479],
        [-0.12552517],
        [-0.15364107],
        [-0.15667515],
        [-0.25168539],
        [-0.15790719],
        [-0.13310771],
        [-0.1412294 ],
        [-0.08798135],
        [-0.13910687],
        [-0.2510029 ],
        [-0.07627433],
        [-0.14062105],
        [-0.11835026],
        [-0.30088701],
        [-0.21952465],
        [-0.13459205],
        [-0.20656295],
        [-0.16298588],
        [-0.15995654],
        [-0.18564458],
        [-0.14786435],
        [-0.10983299],
        [-0.0643699 ],
        [-0.15044562],
        [-0.19627824],
```

```
female
```

```
array([[-0.12371005],
       [-0.09419942],
       [-0.38079826],
       [-0.40809649],
       [-0.37224054],
       [-0.4280452 ],
       [-0.33950869],
       [-0.41942729],
       [-0.35731733],
```

```
       [-0.36722844],
       [-0.45103716],
       [-0.30897517],
       [-0.41801003],
       [-0.39304903],
       [-0.41468096],
       [-0.43987833],
       [-0.41406255],
       [-0.37316887],
       [-0.42634567],
       [-0.40262737],
       [-0.3770952 ],
       [-0.43134717],
       [-0.42534638],
       [-0.46594401],
       [-0.39935837],
       [-0.43220905],
       [-0.22930778],
       [-0.44986739],
       [-0.39676864],
       [-0.36199226],
       [-0.36531482],
       [-0.41424389],
       [-0.44742254],
       [-0.10504029],
       [-0.28128666],
       [-0.3885829 ],
       [-0.36950608],
       [-0.36930127],
       [-0.45071132],
       [-0.40507817],
       [-0.30112145],
       [-0.40558333],
       [-0.38366822],
       [-0.43595153],
       [-0.3641957 ],
       [-0.30969825],
       [-0.46627688],
       [-0.25358096],
       [-0.32872946],
       [-0.34052416],
       [-0.38095859],
       [-0.34345312],
       [-0.38106297],
       [-0.37887742],
       [-0.38567129],
       [-0.40487976],
       [-0.43171003],
       [-0.42540979],
       [-0.42592387],
```

Train test split

```
X_train=male[10:]
X_test=male[:10]
Y_train,Y_test=female[10:],female[:10]
```

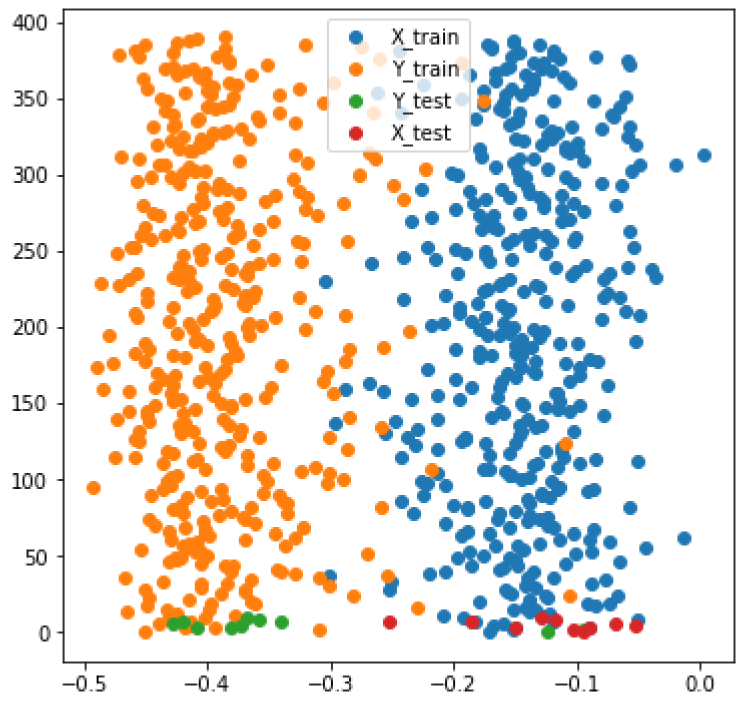
```
X_train
```

```
X_train
```

```
array([[-0.17158275],
       [-0.15071103],
       [-0.15007123],
       [-0.08988632],
       [-0.16843363],
       [-0.15953118],
       [-0.18257427],
       [-0.118917  ],
       [-0.05086052],
       [-0.19247886],
       [-0.12832977],
       [-0.20866478],
       [-0.12207994],
       [-0.13916045],
       [-0.13278016],
       [-0.151321  ],
       [-0.14201097],
       [-0.08436291],
       [-0.09044554],
       [-0.07374715],
       [-0.17886296],
       [-0.13910243],
       [-0.12341403],
       [-0.06697479],
       [-0.12552517],
       [-0.15364107],
       [-0.15667515],
       [-0.25168539],
       [-0.15790719],
       [-0.13310771],
       [-0.1412294 ],
       [-0.08798135],
       [-0.13910687],
       [-0.2510029 ],
       [-0.07627433],
       [-0.14062105],
       [-0.11835026],
       [-0.30088701],
       [-0.21952465],
       [-0.13459205],
       [-0.20656295],
       [-0.16298588],
       [-0.15995654],
       [-0.18564458],
       [-0.14786435],
       [-0.10983299],
       [-0.0643699 ],
       [-0.15044562],
       [-0.19627824],
       [-0.15539006],
       [-0.17533057],
       [-0.10136027],
       [-0.18721703],
       [-0.08843998],
       [-0.14356384],
       [-0.04408384],
       [-0.16542891],
       [-0.14815516],
       [-0.12994921],
```

plot to show clustering

```
plt.figure(figsize= (6, 6))

plt.scatter(X_train[:,0], range(len(X_train[:,0])))
plt.scatter(Y_train[:,0], range(len(Y_train[:,0])))

plt.scatter(Y_test[:,0], range(len(Y_test[:,0])))
plt.scatter(X_test[:,0], range(len(X_test[:,0])))

plt.legend(['X_train', 'Y_train', 'Y_test', 'X_test'])
plt.show()
```



minimum distance classifier

```
print('Predictions for the male test points:\n')

prediction1 = []
for samp in X_test:
    prediction1.append(np.sqrt(np.sum(np.square(X_train - samp))) < np.sqrt(np.sum(np.square(Y_train - samp))))
    print('The prediction is', np.sqrt(np.sum(np.square(X_train - samp))) < np.sqrt(np.sum(np.square(Y_train - samp))))

prediction1 = np.stack(prediction1) * 1 #this is to convert the boolean to numbers

print('\n')
print('\nPredictions for the female test points:\n')

prediction2 = []
for samp in Y_test:
    prediction2.append(np.sqrt(np.sum(np.square(X_train - samp))) > np.sqrt(np.sum(np.square(Y_train - samp))))
    print('The predition is', np.sqrt(np.sum(np.square(X_train - samp))) > np.sqrt(np.sum(np.square(Y_train - samp))))
```

```
    print('The predition is', np.sqrt(np.sum(np.square(X_train - samp))) > np.sqrt(np.sum(np.square(Y_train - samp))))

prediction2 = np.stack(prediction2) * 1

indices_one = prediction2 == 1
indices_zero = prediction2 == 0
prediction2[indices_one] = 0
prediction2[indices_zero] = 1
```

```
Predictions for the male test points:

The prediction is True
The prediction is True
The prediction is True
The prediction is True
The prediction is True
The prediction is True
The prediction is True
The prediction is True
The prediction is True
The prediction is True



Predictions for the female test points:

The predition is False
The predition is False
The predition is True
The predition is True
The predition is True
The predition is True
The predition is True
The predition is True
The predition is True
The predition is True
```

Confusion Matrix

```
def confusion_matrix(pred, true):

    true_pos = 0
    true_neg = 0
    false_pos = 0
    false_neg = 0

    for i in range(len(pred)):

        if pred[i] == true[i]:

            if pred[i] == 1:
                true_pos += 1
            else:
                true_neg += 1
        else:

            if pred[i] == 1 and true[i] == 0:
```

```
            if pred[i] == 1 and true[i] == 0:
                false_pos += 1
            else:
                false_neg += 1

    return true_pos, true_neg, false_pos, false_neg
```

Using different performance metrics

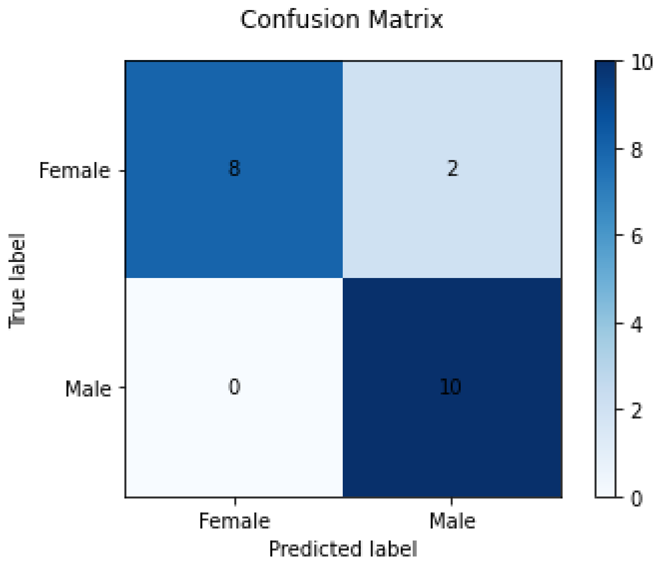
```
predictions=np.concatenate((prediction1,prediction2),axis=0)
x1=np.ones(len(prediction1))
x2=np.zeros(len(prediction2))
x=np.concatenate((x1,x2),axis=0)
true_positives,true_negatives,false_positives,false_negatives=confusion_matrix(predictions,x)
```

```
accuracy = (true_positives + true_negatives)/(true_positives + true_negatives + false_positives + false_negatives)
precision = (true_positives)/(true_positives + false_positives)
recall = (true_positives)/(true_positives + false_negatives)
f1_score = 2 * precision * recall / (precision + recall)

print('Accuracy =', accuracy)
print('Precision =', precision)
print('Recall = ', recall)
print('F1 Score = ', f1_score)
```

```
Accuracy = 0.9
Precision = 0.8333333333333334
Recall =  1.0
F1 Score =  0.9090909090909091
```

```
class_names = ['Female', 'Male']

# randomly generated array
#random_array = np.random.random((2, 2))

matrix = np.array([[true_negatives, false_positives], [false_negatives, true_positives]])

figure = plt.figure()
axes = figure.add_subplot(111)

# using the matshow() function
caxes = axes.matshow(matrix, interpolation ='nearest', cmap = plt.cm.Blues)
figure.colorbar(caxes)

axes.set_xticklabels([''] + class_names)
axes.set_yticklabels([''] + class_names)

plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.gca().xaxis.tick_bottom()

for i in range(len(matrix)):
```

```
  for j in range(len(matrix)):
    plt.text(j, i, str(matrix[i][j]), va = 'center', ha = 'center')

plt.title('Confusion Matrix')
plt.show()
```



0s completed at 8:20 PM