GROUP 10

R SHREJA - COE18B043

RITHIC KUMAR N - COE18B044

SREEDHAR ARUMUGAM - COE18B051

1. Consider the 128- dimensional feature vectors (d=128) given in the “gender_feature_vectors.csv” file. (2 classes, male and female)

a) Use PCA to reduce the dimension from d to d’. (Here d=128)

b) Display the eigenvalue based on increasing order, select the d’ of the corresponding eigenvector which is the appropriate dimension d’ (select d’ S.T first 95% of λ values of the covariance matrix are considered).

c) Use d’ features to classify the test cases (any classification algorithm taught in class like Bayes classifier, minimum distance classifier, and so on)

Dataset Specifications:

- Total number of samples = 800
- Number of classes = 2 (labeled as “male” and “female”)
- Samples from “1 to 400” belongs to class “male”
- Samples from “401 to 800” belongs to class “female”
- Number of samples per class = 400
- Number of dimensions = 128
- Use the following information to design classifier:
- Number of test cases (first 10 in each class) = 20
- Number of training feature vectors (remaining 390 in each class) = 390
- Number of reduced dimensions = d’ (map 128 to d’ features vector)

In [1]:

```
import pandas as pd
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
```

In [2]:

```
df=pd.read_csv('gender_feature_vectors.csv')
```

In [3]:

```
df
```

Out[3]:

	Unnamed: 0	Unnamed: 1	0	1	2	3	4	5	
0	1	male	-0.066420	0.151611	0.027740	0.052771	-0.066105	-0.041232	-(
1	2	male	-0.030614	0.049667	0.008084	-0.050324	0.007649	-0.063818	-(
2	3	male	-0.096178	0.061127	0.035326	-0.035388	-0.090728	-0.018634	-(
3	4	male	-0.103057	0.085044	0.078333	-0.035873	-0.028163	0.004924	(
4	5	male	-0.125815	0.120046	0.023131	-0.042901	0.038215	-0.049677	-(
...	...	...	...	...	...	...	...	...	
795	796	female	-0.164731	0.064301	0.058630	-0.017420	-0.157600	-0.022536	(
796	797	female	-0.095308	0.051095	0.092913	-0.101745	-0.083153	-0.028159	(
797	798	female	-0.202852	0.037039	0.079731	-0.047156	-0.140062	-0.080246	(
798	799	female	-0.088300	0.063530	0.049627	-0.026011	-0.172773	0.086218	(
799	800	female	-0.156201	0.055165	0.142716	-0.115393	-0.128982	-0.139830	-(

800 rows × 130 columns

In [4]:

```
df['Unnamed: 1'].value_counts()
```

Out[4]:

```
female    401
male      399
Name: Unnamed: 1, dtype: int64
```

In [5]:

```
df.drop(columns=df.columns[0],inplace=True)
df
```

Out[5]:

	Unnamed: 1	0	1	2	3	4	5	6	
0	male	-0.066420	0.151611	0.027740	0.052771	-0.066105	-0.041232	-0.002637	-0
1	male	-0.030614	0.049667	0.008084	-0.050324	0.007649	-0.063818	-0.019530	-0
2	male	-0.096178	0.061127	0.035326	-0.035388	-0.090728	-0.018634	-0.024315	-0
3	male	-0.103057	0.085044	0.078333	-0.035873	-0.028163	0.004924	0.007829	-0
4	male	-0.125815	0.120046	0.023131	-0.042901	0.038215	-0.049677	-0.054258	-0
...	...	...	...	...	...	...	...	...	
795	female	-0.164731	0.064301	0.058630	-0.017420	-0.157600	-0.022536	0.002864	-0
796	female	-0.095308	0.051095	0.092913	-0.101745	-0.083153	-0.028159	0.009090	-0
797	female	-0.202852	0.037039	0.079731	-0.047156	-0.140062	-0.080246	0.057668	-0
798	female	-0.088300	0.063530	0.049627	-0.026011	-0.172773	0.086218	0.042710	-0
799	female	-0.156201	0.055165	0.142716	-0.115393	-0.128982	-0.139830	-0.037305	-0

800 rows × 129 columns

In [6]:

```
df.rename(columns={df.columns[0]:"Class"},inplace=True)
df
```

Out[6]:

	Class	0	1	2	3	4	5	6	
0	male	-0.066420	0.151611	0.027740	0.052771	-0.066105	-0.041232	-0.002637	-0.158
1	male	-0.030614	0.049667	0.008084	-0.050324	0.007649	-0.063818	-0.019530	-0.119
2	male	-0.096178	0.061127	0.035326	-0.035388	-0.090728	-0.018634	-0.024315	-0.139
3	male	-0.103057	0.085044	0.078333	-0.035873	-0.028163	0.004924	0.007829	-0.017
4	male	-0.125815	0.120046	0.023131	-0.042901	0.038215	-0.049677	-0.054258	-0.130
...	...	...	...	...	...	...	...	...	
795	female	-0.164731	0.064301	0.058630	-0.017420	-0.157600	-0.022536	0.002864	-0.072
796	female	-0.095308	0.051095	0.092913	-0.101745	-0.083153	-0.028159	0.009090	-0.114
797	female	-0.202852	0.037039	0.079731	-0.047156	-0.140062	-0.080246	0.057668	-0.122
798	female	-0.088300	0.063530	0.049627	-0.026011	-0.172773	0.086218	0.042710	-0.161
799	female	-0.156201	0.055165	0.142716	-0.115393	-0.128982	-0.139830	-0.037305	-0.101

800 rows × 129 columns

In [7]:

```
df_test=df.iloc[np.r_[390:400, 790:800]]
```

In [8]:

```
df_test
```

Out[8]:

	Class	0	1	2	3	4	5	6	
390	male	-0.080683	0.097440	0.006550	0.018112	-0.114999	0.160041	-0.002373	-0.057
391	male	-0.010301	0.135185	0.049710	-0.046424	-0.041742	0.016607	-0.041778	-0.048
392	male	-0.112450	0.080098	0.030571	0.000952	-0.097450	-0.045070	-0.003641	-0.142
393	male	-0.087855	0.100264	0.069775	0.037204	-0.047182	-0.047233	-0.013604	-0.189
394	male	-0.139066	0.141988	0.070456	-0.003518	-0.065637	-0.037767	-0.094195	-0.195
395	male	-0.129449	0.132177	0.055916	-0.009390	-0.080541	-0.072362	-0.067433	-0.192
396	male	-0.158460	0.109948	0.019088	0.015506	-0.069668	0.032311	0.015062	-0.140
397	male	-0.101499	0.119739	0.016951	-0.013677	-0.055524	0.028399	0.028164	-0.152
398	male	-0.149516	0.081588	0.090796	-0.053116	-0.133314	0.001096	0.019941	-0.117
399	female	0.039844	0.070357	0.130196	-0.007683	-0.077825	-0.021298	-0.024133	-0.085
790	female	-0.184166	0.122040	0.064143	-0.116653	-0.140633	0.041252	0.014207	-0.108
791	female	-0.128052	0.082184	0.148978	-0.074984	-0.113145	-0.031003	0.018468	-0.130
792	female	-0.109618	0.094507	0.043757	-0.162004	-0.191951	0.027582	-0.033507	-0.112
793	female	-0.201133	0.068051	0.088596	-0.084936	-0.055628	-0.069027	-0.003922	-0.191
794	female	-0.185053	0.136105	0.096279	-0.139661	-0.244312	0.015234	-0.007418	-0.077
795	female	-0.164731	0.064301	0.058630	-0.017420	-0.157600	-0.022536	0.002864	-0.072
796	female	-0.095308	0.051095	0.092913	-0.101745	-0.083153	-0.028159	0.009090	-0.114
797	female	-0.202852	0.037039	0.079731	-0.047156	-0.140062	-0.080246	0.057668	-0.122
798	female	-0.088300	0.063530	0.049627	-0.026011	-0.172773	0.086218	0.042710	-0.161
799	female	-0.156201	0.055165	0.142716	-0.115393	-0.128982	-0.139830	-0.037305	-0.101

20 rows × 129 columns

In [9]:

```
df_train=df.iloc[np.r_[0:390,400:790]]
```

In [10]:

```
df_train
```

Out[10]:

	Class	0	1	2	3	4	5	6	
0	male	-0.066420	0.151611	0.027740	0.052771	-0.066105	-0.041232	-0.002637	-0.158
1	male	-0.030614	0.049667	0.008084	-0.050324	0.007649	-0.063818	-0.019530	-0.119
2	male	-0.096178	0.061127	0.035326	-0.035388	-0.090728	-0.018634	-0.024315	-0.139
3	male	-0.103057	0.085044	0.078333	-0.035873	-0.028163	0.004924	0.007829	-0.017
4	male	-0.125815	0.120046	0.023131	-0.042901	0.038215	-0.049677	-0.054258	-0.130
...	...	...	...	...	...	...	...	...	
785	female	-0.017931	0.045591	0.059826	-0.059833	-0.126832	0.070365	0.026347	-0.093
786	female	-0.039572	0.049409	0.100950	-0.092940	-0.152356	-0.057714	-0.000538	-0.115
787	female	-0.015170	-0.001116	0.132143	-0.078673	-0.134462	-0.155683	-0.030665	-0.164
788	female	-0.151263	0.096725	0.075206	-0.047269	-0.233372	-0.019918	-0.072370	-0.118
789	female	-0.179743	0.104404	0.052260	-0.074608	-0.166008	0.023287	0.006970	-0.149

780 rows × 129 columns

Covariance

In [11]:

```
df_dim=df_train[[i for i in df.columns[1:]]]
```

In [12]:

```
df_dim.shape
```

Out[12]:

```
(780, 128)
```

In [13]:

```
cov=np.cov(df_dim.T)
cov
```

Out[13]:

```
array([[ 2.95108480e-03, -3.70158184e-04, -2.16257469e-04, ...,
         4.34218359e-04, -1.10121464e-04, -2.23638279e-05],
       [-3.70158184e-04,  2.50417089e-03,  6.23082137e-05, ...,
        -3.84633702e-04, -1.45010282e-04, -2.40364718e-05],
       [-2.16257469e-04,  6.23082137e-05,  2.60834145e-03, ...,
        -3.21716028e-04, -2.00431637e-04,  1.91564799e-05],
       ...,
       [ 4.34218359e-04, -3.84633702e-04, -3.21716028e-04, ...,
         2.36909122e-03,  1.84584670e-06,  1.79111827e-04],
       [-1.10121464e-04, -1.45010282e-04, -2.00431637e-04, ...,
         1.84584670e-06,  2.49192480e-03,  5.51925391e-05],
       [-2.23638279e-05, -2.40364718e-05,  1.91564799e-05, ...,
         1.79111827e-04,  5.51925391e-05,  2.26553560e-03]])
```

In [14]:

```
cov.shape
```

Out[14]:

```
(128, 128)
```

getting eigen values

In [15]:

```
def sorted_eig(A):  # For now we sort 'by convention'. For PCA the sorting is key.
    lambdas, vs = np.linalg.eig(A)
    # Next line just sorts values & vectors together in order of decreasing eigenvalues
    lambdas, vs = zip(*sorted(zip(list(lambdas), list(vs.T)),key=lambda x: x[0], revers
e=True))
    return lambdas, np.array(vs).T  # un-doing the list-casting from the previous line
```

In [16]:

```
lambdas,vs=sorted_eig(cov)
print("Lambda values in sorted order")
lambdas
```

Out[16]:

```
(0.04137574503460549,
 0.02360849711562386,
 0.017077038743770187,
 0.014451291776124938,
 0.012633313809123732,
 0.011714705691204017,
 0.010189552276534721,
 0.009087057091511646,
 0.008482770886136562,
 0.008039865359807597,
 0.00760030905912849,
 0.007145776669589233,
 0.006771403338885542,
 0.006509757015262208,
 0.0062636178632257625,
 0.005842316913291985,
 0.005564646916617724,
 0.00551087741573726 5,
 0.005215948451115262,
 0.005052886212394662,
 0.004814955569113215,
 0.004563226306426189,
 0.0044474704013512225,
 0.004268294677809414,
 0.00422921958039608 1,
 0.003926267119388022,
 0.0038579422169492264,
 0.003691394876422616,
 0.003477780068960983,
 0.0033675627685272884,
 0.003202721657676988,
 0.0030656341277170433,
 0.0030408421988791347,
 0.0028325439794659106,
 0.002775832422228 1914,
 0.00270041537083665 3,
 0.0026746600113390874,
 0.0025363760389129028,
 0.002515055042077224,
 0.0024227300289765487,
 0.002357721538014705,
 0.00232115639334748 1,
 0.0022633127203457093,
 0.002177699886331758,
 0.0020620451101881617,
 0.0020097613332662047,
 0.0019102064874178648,
 0.001856074146743188,
 0.0018327018105405322,
 0.0017736034036216462,
 0.0016454665395786428,
 0.0016042741311068861,
 0.0015387653014735222,
 0.001473453780781087,
 0.0014437964392814296,
 0.0013681760840532897,
 0.001299128550125215 4,
 0.001236013462537223 1,
 0.001179399231489573 8,
```

0.0011756810795875475,
0.0011101014875846825,
0.0010686852603078684,
0.0010080061093842574,
0.0009981204169890476,
0.0009303374948736985,
0.0008861924584951544,
0.0008578262872595781,
0.0008314607222984141,
0.0007511239148571716,
0.0007304558969953307,
0.0007090358458285729,
0.0006463624868829558,
0.0005869581784766859,
0.0005630997907502456,
0.00038966919693531906,
0.0003491875729506964,
0.00017423761449379173,
4.589566755200929e-06,
8.578427835079057e-07,
3.059268608435976e-07,
9.355187331571268e-08,
9.061524745412814e-09,
1.23602304949623e-09,
5.425056669644469e-10,
2.7954655365346586e-10,
9.449981920708562e-11,
2.3908232170159416e-11,
9.426540816467547e-12,
4.18888495467226e-12,
2.5704593491045257e-12,
7.83886053141938e-13,
6.746061771312841e-13,
3.000746804666171e-13,
1.6010753954830057e-13,
7.979081251285144e-14,
5.3879530947440807e-14,
4.361637600750955e-14,
3.0905150060949546e-14,
2.0321363896017388e-14,
1.1463095068504483e-14,
1.0643313860501104e-14,
8.132710422215743e-15,
6.995703750639252 4e-15,
6.65297665108662e-15,
6.286844255079942e-15,
5.6095001472050975e-15,
5.328087543481544e-15,
5.071977414370759e-15,
4.9223449211747615e-15,
4.624848293407667e-15,
4.398807452238238e-15,
4.171949273130997e-15,
3.796127583127979e-15,
3.60845946278965e-15,
3.199356133416859e-15,
2.95836610690943e-15,
2.85193636134112e-15,
2.725376477179258e-15,
2.7001704617578923e-15,
2.4056221409956062e-15,

```
 2.3629505630551914e-15,
 2.177118779513895e-15,
 2.0409126404096537e-15,
 1.9264089736652567e-15,
 1.7448796144443188e-15,
 1.586320349179776e-15,
 1.369477040839034e-15,
 1.142219401624541e-15)
```

In [17]:

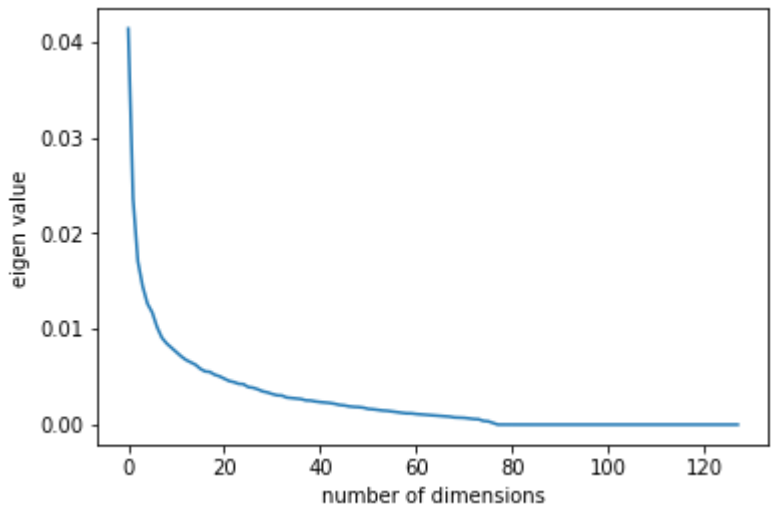
```
vs
```

Out[17]:

```
array([[-0.07127427, -0.06435239, -0.05941313, ...,  0.01756089,
        -0.06551164,  0.03609231],
       [-0.03849229,  0.00910902,  0.02515185, ...,  0.06466022,
        -0.1574927 ,  0.01232222],
       [ 0.09235571,  0.03575751, -0.11693621, ..., -0.03149788,
         0.0080926 ,  0.07226275],
       ...,
       [-0.01535881, -0.00290511,  0.04327725, ...,  0.10022143,
        -0.01055383, -0.12365049],
       [-0.04780239,  0.02650115, -0.08556776, ..., -0.02240944,
        -0.04477904, -0.18795663],
       [-0.03681228, -0.07247092, -0.07462774, ..., -0.02856651,
        -0.08375543, -0.028343  ]])
```

In [18]:

```
plt.plot(lambdas)
plt.xlabel('number of dimensions')
plt.ylabel('eigen value');
```



Getting no of components

In [19]:

```
#consider values which explain 95% of variance

print("Total lambda values:",len(lambdas))

total_eigen=np.sum(lambdas)
r_sum=0

for i in range(len(lambdas)):
    if(r_sum/total_eigen >=.95):
        index=i
        break
    else:
        r_sum+=lambdas[i]

lambda_rd=lambdas[:index]
vs_rd=vs[:index]
```

```
Total lambda values: 128
```

In [20]:

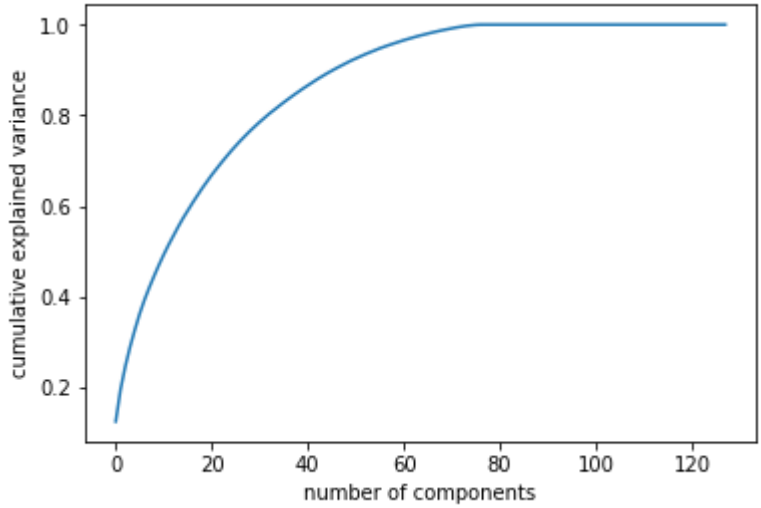
```
print("Reduced lambda ",len(lambda_rd))
```

```
Reduced lambda  57
```

Verifying with inbuilt PCA for no of components

In [21]:

```
from sklearn.decomposition import PCA

pca = PCA().fit(df_dim)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');

#we can see for 0.95, we get around 60
```



Reducing the data

In [22]:

```
red_train_data=df_dim @ vs_rd.T
red_train_data
```

Out[22]:

	0	1	2	3	4	5	6	7	
0	-0.085922	-0.200630	0.193114	0.162222	0.005716	0.034162	0.029775	-0.081066	0.2
1	-0.055755	-0.124344	0.151680	0.157961	0.018568	0.030559	0.088235	-0.114077	0.2
2	-0.116070	-0.119059	0.114342	0.098051	0.045468	0.087185	0.125157	-0.029765	0.2
3	-0.082357	-0.187135	0.187303	0.177185	0.044054	-0.014291	0.103310	-0.039566	0.2
4	-0.076012	-0.152521	0.146189	0.112002	-0.001292	-0.000156	0.106822	-0.039447	0.1
...	...	...	...	...	...	...	...	...	
785	-0.166122	-0.143068	0.166056	0.105643	-0.020830	0.119008	0.096983	-0.106037	0.2
786	-0.018686	-0.175362	0.225020	0.080913	-0.040170	0.181357	0.142460	-0.044167	0.2
787	-0.078697	-0.162946	0.220727	0.122436	0.027959	0.120039	0.052178	-0.042097	0.2
788	-0.141495	-0.130481	0.247952	0.166986	0.015830	0.134146	0.090760	-0.086861	0.2
789	-0.062910	-0.129503	0.198673	0.111234	-0.076534	0.146725	0.146133	-0.050591	0.2

780 rows × 57 columns

In [23]:

```
red_test_data=df_test[[i for i in df_test.columns[1:]]] @ vs_rd.T
red_test_data
```

Out[23]:

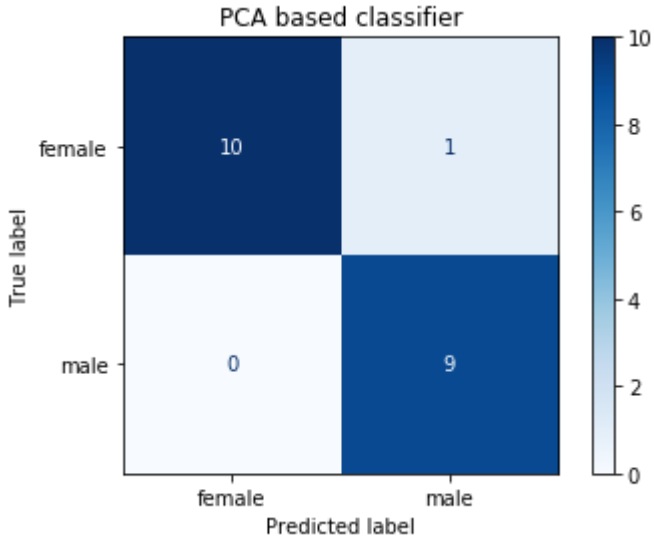
	0	1	2	3	4	5	6	7	
390	-0.177132	-0.175115	0.269655	0.118365	0.038968	0.085124	0.057599	-0.065892	0.1
391	-0.103300	-0.160897	0.167615	0.201968	-0.008014	0.092952	0.045541	0.002151	0.2
392	-0.153435	-0.173594	0.193888	0.138280	0.034524	0.119990	0.119908	-0.088775	0.1
393	-0.115593	-0.050115	0.164323	0.055682	0.003437	0.064277	0.133570	-0.020624	0.1
394	-0.055370	-0.026352	0.158553	0.109222	0.025264	-0.003519	0.089221	0.029834	0.1
395	-0.075354	-0.085509	0.129584	0.130524	0.118713	0.045890	0.123421	-0.066907	0.1
396	-0.133736	-0.087822	0.116717	0.095699	0.107149	0.091118	0.136185	-0.025547	0.1
397	-0.009092	-0.165390	0.148819	0.110606	0.056945	0.022829	0.080098	-0.126040	0.1
398	-0.035924	-0.125528	0.161770	0.197797	-0.043080	0.037714	0.048948	-0.143283	0.1
399	-0.044933	-0.080137	0.083556	0.119618	0.004948	0.084383	0.091632	-0.087539	0.1
790	-0.106879	-0.175464	0.149705	0.078446	-0.025678	0.124027	0.123001	-0.064445	0.2
791	-0.036366	-0.197258	0.199854	0.082745	-0.076543	0.170693	0.068024	-0.083996	0.2
792	-0.126374	-0.273559	0.068183	0.110489	0.033757	0.167480	0.143893	-0.141644	0.2
793	-0.082626	-0.098502	0.064317	0.058600	0.027592	0.065021	0.152353	-0.007970	0.1
794	-0.124669	-0.244321	0.158432	0.100248	-0.001757	0.141562	0.168321	-0.042069	0.1
795	0.034578	-0.137226	0.070334	0.087678	-0.043030	0.057677	0.029719	-0.030424	0.2
796	-0.063201	-0.225409	0.158612	0.092467	0.023689	0.127564	0.171994	-0.043018	0.2
797	0.039070	-0.085903	0.126874	0.071063	-0.034993	0.092489	0.064239	-0.060878	0.1
798	-0.084401	-0.123668	0.107070	0.127319	-0.094750	0.087222	0.028819	-0.110238	0.2
799	-0.088090	-0.212623	0.193325	0.101219	0.022406	0.138563	0.145457	-0.036179	0.1

20 rows × 57 columns

Classification

In [24]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import plot_confusion_matrix

classifier = GaussianNB()
classifier.fit(np.array(red_train_data),df_train['Class'])

disp = plot_confusion_matrix(classifier,np.array(red_test_data) , df_test["Class"].to_l
ist(),
                                 display_labels=["female","male"],
                                 cmap=plt.cm.Blues,
                                 )
disp.ax_.set_title("PCA based classifier")
plt.show()
```



We can see we get good accuracy even after reducing the dimensions (almost half of original dimension)

In [ ]:

```

```

2. 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).

In [1]:

```
import pandas as pd
import numpy as np
import scipy as sp
```

In [2]:

```
df=pd.read_csv('gender_feature_vectors.csv')
```

In [3]:

```
df
```

Out[3]:

	Unnamed: 0	Unnamed: 1	0	1	2	3	4	5	
0	1	male	-0.066420	0.151611	0.027740	0.052771	-0.066105	-0.041232	-0.0
1	2	male	-0.030614	0.049667	0.008084	-0.050324	0.007649	-0.063818	-0.0
2	3	male	-0.096178	0.061127	0.035326	-0.035388	-0.090728	-0.018634	-0.0
3	4	male	-0.103057	0.085044	0.078333	-0.035873	-0.028163	0.004924	0.0
4	5	male	-0.125815	0.120046	0.023131	-0.042901	0.038215	-0.049677	-0.0
...	...	...	...	...	...	...	...	...	
795	796	female	-0.164731	0.064301	0.058630	-0.017420	-0.157600	-0.022536	0.0
796	797	female	-0.095308	0.051095	0.092913	-0.101745	-0.083153	-0.028159	0.0
797	798	female	-0.202852	0.037039	0.079731	-0.047156	-0.140062	-0.080246	0.0
798	799	female	-0.088300	0.063530	0.049627	-0.026011	-0.172773	0.086218	0.0
799	800	female	-0.156201	0.055165	0.142716	-0.115393	-0.128982	-0.139830	-0.0

800 rows × 130 columns

In [4]:

```
df['Unnamed: 1'].value_counts()
```

Out[4]:

```
female    401
male      399
Name: Unnamed: 1, dtype: int64
```

In [5]:

```
df.drop(columns=df.columns[0],inplace=True)
df
```

Out[5]:

	Unnamed: 1	0	1	2	3	4	5	6	
0	male	-0.066420	0.151611	0.027740	0.052771	-0.066105	-0.041232	-0.002637	-0.15
1	male	-0.030614	0.049667	0.008084	-0.050324	0.007649	-0.063818	-0.019530	-0.11
2	male	-0.096178	0.061127	0.035326	-0.035388	-0.090728	-0.018634	-0.024315	-0.13
3	male	-0.103057	0.085044	0.078333	-0.035873	-0.028163	0.004924	0.007829	-0.01
4	male	-0.125815	0.120046	0.023131	-0.042901	0.038215	-0.049677	-0.054258	-0.13
...	...	...	...	...	...	...	...	...	
795	female	-0.164731	0.064301	0.058630	-0.017420	-0.157600	-0.022536	0.002864	-0.07
796	female	-0.095308	0.051095	0.092913	-0.101745	-0.083153	-0.028159	0.009090	-0.11
797	female	-0.202852	0.037039	0.079731	-0.047156	-0.140062	-0.080246	0.057668	-0.12
798	female	-0.088300	0.063530	0.049627	-0.026011	-0.172773	0.086218	0.042710	-0.16
799	female	-0.156201	0.055165	0.142716	-0.115393	-0.128982	-0.139830	-0.037305	-0.10

800 rows × 129 columns

In [6]:

```
df.rename(columns={df.columns[0]:"Class"},inplace=True)
df
```

Out[6]:

	Class	0	1	2	3	4	5	6	
0	male	-0.066420	0.151611	0.027740	0.052771	-0.066105	-0.041232	-0.002637	-0.15846
1	male	-0.030614	0.049667	0.008084	-0.050324	0.007649	-0.063818	-0.019530	-0.11990
2	male	-0.096178	0.061127	0.035326	-0.035388	-0.090728	-0.018634	-0.024315	-0.13978
3	male	-0.103057	0.085044	0.078333	-0.035873	-0.028163	0.004924	0.007829	-0.01701
4	male	-0.125815	0.120046	0.023131	-0.042901	0.038215	-0.049677	-0.054258	-0.13075
...	...	...	...	...	...	...	...	...	.
795	female	-0.164731	0.064301	0.058630	-0.017420	-0.157600	-0.022536	0.002864	-0.07273
796	female	-0.095308	0.051095	0.092913	-0.101745	-0.083153	-0.028159	0.009090	-0.11451
797	female	-0.202852	0.037039	0.079731	-0.047156	-0.140062	-0.080246	0.057668	-0.12208
798	female	-0.088300	0.063530	0.049627	-0.026011	-0.172773	0.086218	0.042710	-0.16185
799	female	-0.156201	0.055165	0.142716	-0.115393	-0.128982	-0.139830	-0.037305	-0.10140

800 rows × 129 columns

In [7]:

```
df_test=df.iloc[np.r_[390:400, 790:800]]
```

In [8]:

```
df_train=df.iloc[np.r_[0:390,400:790]]
```

In [12]:

```
df_train[0:390]
```

Out[12]:

	Class	0	1	2	3	4	5	6	
0	male	-0.066420	0.151611	0.027740	0.052771	-0.066105	-0.041232	-0.002637	-0.15846
1	male	-0.030614	0.049667	0.008084	-0.050324	0.007649	-0.063818	-0.019530	-0.11990
2	male	-0.096178	0.061127	0.035326	-0.035388	-0.090728	-0.018634	-0.024315	-0.13978
3	male	-0.103057	0.085044	0.078333	-0.035873	-0.028163	0.004924	0.007829	-0.01701
4	male	-0.125815	0.120046	0.023131	-0.042901	0.038215	-0.049677	-0.054258	-0.13075
...	...	...	...	...	...	...	...	...	.
385	male	-0.064409	0.111391	0.204528	-0.061478	-0.007144	-0.185951	0.037198	-0.12249
386	male	0.002976	0.060769	-0.016651	0.053049	-0.099903	-0.082844	0.017824	-0.10680
387	male	-0.009311	0.077162	0.031294	-0.089573	-0.038689	-0.031663	-0.012356	-0.06409
388	male	-0.036479	0.063089	0.037658	-0.000695	-0.104741	-0.018407	0.030498	-0.08932
389	male	-0.201210	0.054217	0.173419	0.004309	-0.068915	-0.114538	0.124839	-0.06773

390 rows × 129 columns

In [13]:

```
df_train[390:]
```

Out[13]:

	Class	0	1	2	3	4	5	6	
400	female	0.001747	0.185678	0.073260	0.042142	-0.088674	0.028186	-0.027830	-0.06421
401	female	-0.091598	0.095340	0.072125	-0.092276	-0.079953	0.047782	-0.004701	-0.09200
402	female	-0.018751	0.088572	0.068894	-0.065700	-0.115126	0.024339	-0.028420	-0.15932
403	female	-0.130889	0.093262	0.122244	-0.110014	-0.157625	-0.036781	0.073908	-0.09857
404	female	-0.037433	0.078158	0.118061	-0.117658	-0.194807	-0.045464	-0.014104	-0.15882
...	...	...	...	...	...	...	...	...	.
785	female	-0.017931	0.045591	0.059826	-0.059833	-0.126832	0.070365	0.026347	-0.09353
786	female	-0.039572	0.049409	0.100950	-0.092940	-0.152356	-0.057714	-0.000538	-0.11599
787	female	-0.015170	-0.001116	0.132143	-0.078673	-0.134462	-0.155683	-0.030665	-0.16475
788	female	-0.151263	0.096725	0.075206	-0.047269	-0.233372	-0.019918	-0.072370	-0.11840
789	female	-0.179743	0.104404	0.052260	-0.074608	-0.166008	0.023287	0.006970	-0.14983

390 rows × 129 columns

In [15]:

```
df_test
```

Out[15]:

	Class	0	1	2	3	4	5	6	
390	male	-0.080683	0.097440	0.006550	0.018112	-0.114999	0.160041	-0.002373	-0.05795
391	male	-0.010301	0.135185	0.049710	-0.046424	-0.041742	0.016607	-0.041778	-0.04802
392	male	-0.112450	0.080098	0.030571	0.000952	-0.097450	-0.045070	-0.003641	-0.14282
393	male	-0.087855	0.100264	0.069775	0.037204	-0.047182	-0.047233	-0.013604	-0.18937
394	male	-0.139066	0.141988	0.070456	-0.003518	-0.065637	-0.037767	-0.094195	-0.19566
395	male	-0.129449	0.132177	0.055916	-0.009390	-0.080541	-0.072362	-0.067433	-0.19224
396	male	-0.158460	0.109948	0.019088	0.015506	-0.069668	0.032311	0.015062	-0.14081
397	male	-0.101499	0.119739	0.016951	-0.013677	-0.055524	0.028399	0.028164	-0.15210
398	male	-0.149516	0.081588	0.090796	-0.053116	-0.133314	0.001096	0.019941	-0.11780
399	female	0.039844	0.070357	0.130196	-0.007683	-0.077825	-0.021298	-0.024133	-0.08510
790	female	-0.184166	0.122040	0.064143	-0.116653	-0.140633	0.041252	0.014207	-0.10881
791	female	-0.128052	0.082184	0.148978	-0.074984	-0.113145	-0.031003	0.018468	-0.13028
792	female	-0.109618	0.094507	0.043757	-0.162004	-0.191951	0.027582	-0.033507	-0.11207
793	female	-0.201133	0.068051	0.088596	-0.084936	-0.055628	-0.069027	-0.003922	-0.19139
794	female	-0.185053	0.136105	0.096279	-0.139661	-0.244312	0.015234	-0.007418	-0.07761
795	female	-0.164731	0.064301	0.058630	-0.017420	-0.157600	-0.022536	0.002864	-0.07273
796	female	-0.095308	0.051095	0.092913	-0.101745	-0.083153	-0.028159	0.009090	-0.11451
797	female	-0.202852	0.037039	0.079731	-0.047156	-0.140062	-0.080246	0.057668	-0.12208
798	female	-0.088300	0.063530	0.049627	-0.026011	-0.172773	0.086218	0.042710	-0.16185
799	female	-0.156201	0.055165	0.142716	-0.115393	-0.128982	-0.139830	-0.037305	-0.10140

20 rows × 129 columns

Computing mean and scatter matrix

In [16]:

```
df_male=df_train[[i for i in df.columns[1:]]][:390]
df_male
```

Out[16]:

	0	1	2	3	4	5	6	7	
0	-0.066420	0.151611	0.027740	0.052771	-0.066105	-0.041232	-0.002637	-0.158467	0.13(
1	-0.030614	0.049667	0.008084	-0.050324	0.007649	-0.063818	-0.019530	-0.119905	0.18(
2	-0.096178	0.061127	0.035326	-0.035388	-0.090728	-0.018634	-0.024315	-0.139786	0.05:
3	-0.103057	0.085044	0.078333	-0.035873	-0.028163	0.004924	0.007829	-0.017016	0.11⁴
4	-0.125815	0.120046	0.023131	-0.042901	0.038215	-0.049677	-0.054258	-0.130758	0.17:
...	...	...	...	...	...	...	...	...	
385	-0.064409	0.111391	0.204528	-0.061478	-0.007144	-0.185951	0.037198	-0.122494	0.16(
386	0.002976	0.060769	-0.016651	0.053049	-0.099903	-0.082844	0.017824	-0.106804	0.12'
387	-0.009311	0.077162	0.031294	-0.089573	-0.038689	-0.031663	-0.012356	-0.064096	0.17
388	-0.036479	0.063089	0.037658	-0.000695	-0.104741	-0.018407	0.030498	-0.089321	0.13'
389	-0.201210	0.054217	0.173419	0.004309	-0.068915	-0.114538	0.124839	-0.067737	0.10⁴

390 rows × 128 columns

In [17]:

```
df_female=df_train[[i for i in df.columns[1:]]][390:]
df_female
```

Out[17]:

	0	1	2	3	4	5	6	7	
400	0.001747	0.185678	0.073260	0.042142	-0.088674	0.028186	-0.027830	-0.064211	0.097
401	-0.091598	0.095340	0.072125	-0.092276	-0.079953	0.047782	-0.004701	-0.092005	0.222
402	-0.018751	0.088572	0.068894	-0.065700	-0.115126	0.024339	-0.028420	-0.159320	0.164
403	-0.130889	0.093262	0.122244	-0.110014	-0.157625	-0.036781	0.073908	-0.098571	0.12(
404	-0.037433	0.078158	0.118061	-0.117658	-0.194807	-0.045464	-0.014104	-0.158824	0.159
...	...	...	...	...	...	...	...	...	
785	-0.017931	0.045591	0.059826	-0.059833	-0.126832	0.070365	0.026347	-0.093537	0.278
786	-0.039572	0.049409	0.100950	-0.092940	-0.152356	-0.057714	-0.000538	-0.115991	0.206
787	-0.015170	-0.001116	0.132143	-0.078673	-0.134462	-0.155683	-0.030665	-0.164758	0.105
788	-0.151263	0.096725	0.075206	-0.047269	-0.233372	-0.019918	-0.072370	-0.118408	0.193
789	-0.179743	0.104404	0.052260	-0.074608	-0.166008	0.023287	0.006970	-0.149833	0.159

390 rows × 128 columns

In [18]:

```
mean_male=df_male.mean()
mean_male
```

Out[18]:

```
0     -0.091137
1      0.092309
2      0.043384
3     -0.030786
4     -0.093000
         ...
123   -0.118805
124    0.021776
125   -0.025222
126    0.020244
127    0.040089
Length: 128, dtype: float64
```

In [19]:

```
mean_female=df_female.mean()
mean_female
```

Out[19]:

```
0     -0.116036
1      0.076929
2      0.081975
3     -0.077089
4     -0.122499
         ...
123   -0.086894
124    0.053799
125   -0.031701
126    0.004233
127    0.025536
Length: 128, dtype: float64
```

Sw matrix (within class)

In [20]:

```
#Scatter matrix 1

S1= (df_male.values - np.array(mean_male) ).T @ (df_male.values - np.array(mean_male))
```

In [21]:

```
S1
```

Out[21]:

```
array([[ 1.21590589, -0.22385172, -0.12872454, ...,  0.24384893,
        -0.07807342, -0.03553427],
       [-0.22385172,  0.89023903,  0.14525314, ..., -0.14402007,
        -0.11400393, -0.08026288],
       [-0.12872454,  0.14525314,  0.98592786, ..., -0.1457067 ,
         0.0365962 , -0.02872259],
       ...,
       [ 0.24384893, -0.14402007, -0.1457067 , ...,  0.951065  ,
         0.02352049,  0.00168977],
       [-0.07807342, -0.11400393,  0.0365962 , ...,  0.02352049,
         0.97078087,  0.10619016],
       [-0.03553427, -0.08026288, -0.02872259, ...,  0.00168977,
         0.10619016,  0.68554017]])
```

In [22]:

```
S2 = (df_female.values - np.array(mean_female)).T @ (df_female.values - np.array(mean_femal
```

In [23]:

```
S2
```

Out[23]:

```
array([[ 0.9621001 , -0.13917361,  0.14762725, ...,  0.06294964,
        -0.08545036, -0.05254243],
       [-0.13917361,  1.01438579,  0.01902008, ..., -0.17504062,
        -0.04697787,  0.01789532],
       [ 0.14762725,  0.01902008,  0.75556753, ..., -0.05615372,
        -0.07224368,  0.15315488],
       ...,
       [ 0.06294964, -0.17504062, -0.05615372, ...,  0.88627125,
        -0.04231172,  0.11945256],
       [-0.08545036, -0.04697787, -0.07224368, ..., -0.04231172,
         0.92043746, -0.1086309 ],
       [-0.05254243,  0.01789532,  0.15315488, ...,  0.11945256,
        -0.1086309 ,  1.0380166 ]])
```

In [24]:

```
Sw=S1+S2
```

For Sb (between class) we need mean of whole data

In [25]:

```
mean_whole=np.mean(df_train)
```

In [26]:

```
def Sb_i(mean_whole,mean_class,number_of_points,dim):
    return (number_of_points * np.subtract(mean_class,mean_whole).values.reshape(dim,1) @ r
```

In [27]:

```
Sb1=Sb_i(mean_whole,mean_male,390,128)
```

In [28]:

```
Sb2=Sb_i(mean_whole,mean_female,390,128)
```

In [29]:

```
Sb=Sb1+Sb2
```

In [30]:

```
eigen_vectors, eigen_values,_=np.linalg.svd(np.linalg.inv(Sw) @ Sb)
```

In [31]:

```
eigen_values
```

Out[31]:

```
array([2.72333430e+06, 4.14087275e-04, 3.92718028e-04, 3.47948322e-04,
       3.21549303e-04, 3.08722475e-04, 2.99873459e-04, 2.75389943e-04,
       2.72325531e-04, 2.63408566e-04, 2.50202781e-04, 2.41079159e-04,
       2.33192792e-04, 2.29902802e-04, 2.20424901e-04, 2.12390727e-04,
       2.08723331e-04, 2.02983943e-04, 1.98274952e-04, 1.92096411e-04,
       1.87930711e-04, 1.83259321e-04, 1.79973511e-04, 1.68708968e-04,
       1.64019504e-04, 1.57551063e-04, 1.55982351e-04, 1.52701669e-04,
       1.49555675e-04, 1.44957772e-04, 1.38982285e-04, 1.36554861e-04,
       1.35132546e-04, 1.33338857e-04, 1.28368942e-04, 1.24689578e-04,
       1.19399238e-04, 1.17035283e-04, 1.15960905e-04, 1.11093987e-04,
       1.08886529e-04, 1.04481965e-04, 1.01524049e-04, 1.00881439e-04,
       9.74491377e-05, 9.51172349e-05, 9.35910544e-05, 8.99262536e-05,
       8.88186053e-05, 8.76201235e-05, 7.99192625e-05, 7.83831228e-05,
       7.68043061e-05, 7.57674485e-05, 7.44863358e-05, 7.08509161e-05,
       6.98792249e-05, 6.81283604e-05, 6.65086484e-05, 6.41987831e-05,
       6.04101777e-05, 5.99595235e-05, 5.88472506e-05, 5.64822859e-05,
       5.57394138e-05, 5.39642613e-05, 5.13365336e-05, 5.08517143e-05,
       4.84559338e-05, 4.75352581e-05, 4.73064850e-05, 4.57833601e-05,
       4.35205221e-05, 4.14972441e-05, 4.12446418e-05, 3.94001245e-05,
       3.75535651e-05, 3.70542609e-05, 3.46971925e-05, 3.44804253e-05,
       3.31875332e-05, 3.15001859e-05, 3.06431124e-05, 2.94046780e-05,
       2.80340966e-05, 2.78905436e-05, 2.73364007e-05, 2.41553891e-05,
       2.37390224e-05, 2.28896573e-05, 2.25135106e-05, 2.07032761e-05,
       2.00756602e-05, 1.83019631e-05, 1.74186818e-05, 1.67906905e-05,
       1.61982034e-05, 1.50486336e-05, 1.44613895e-05, 1.40479915e-05,
       1.36479073e-05, 1.17312851e-05, 1.12881342e-05, 1.07155906e-05,
       1.01684656e-05, 9.03868353e-06, 8.60895383e-06, 7.69794773e-06,
       7.07619203e-06, 6.35861567e-06, 5.65950430e-06, 5.08306307e-06,
       4.42827147e-06, 3.77289525e-06, 3.67781457e-06, 3.41245691e-06,
       2.91492001e-06, 2.18146763e-06, 1.79744983e-06, 1.45753050e-06,
       1.14460294e-06, 1.02814399e-06, 7.10385451e-07, 6.55616874e-07,
       4.17230863e-07, 2.33276401e-07, 1.30299101e-07, 9.41889153e-08])
```

In [32]:

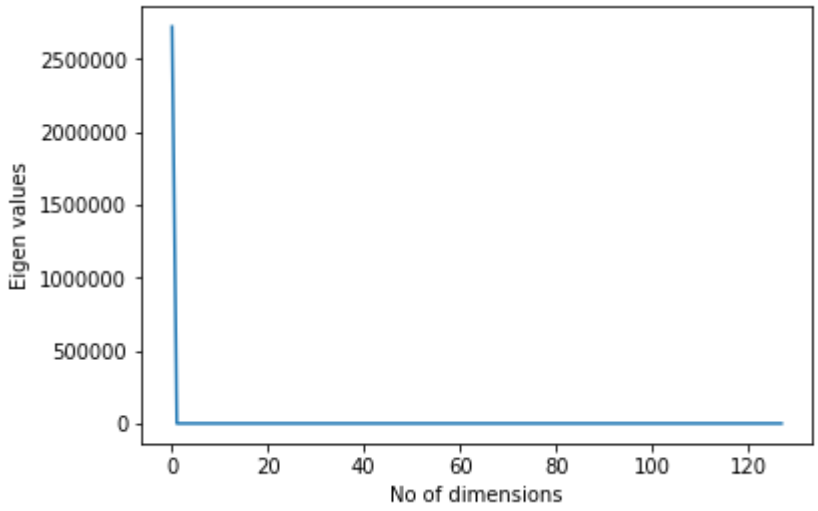
```
import matplotlib.pyplot as plt
```

In [33]:

```
plt.plot(eigen_values)
plt.xlabel("No of dimensions")
plt.ylabel("Eigen values")
```

Out[33]:

```
Text(0, 0.5, 'Eigen values')
```



We take only first eigen value (2 classes --> take (2-1) eigen value).

Reducing data

In [34]:

```
red_train_data=df_train[df.columns[1:]] @ eigen_vectors[0]
red_train_data
```

Out[34]:

```
0     -0.076645
1      0.011543
2      0.075068
3      0.100944
4     -0.005650
         ...
785   -0.060044
786    0.072923
787    0.010113
788    0.093721
789    0.084124
Length: 780, dtype: float64
```

In [35]:

```
red_test_data=df_test[[i for i in df_test.columns[1:]]] @ eigen_vectors[0]
red_test_data
```

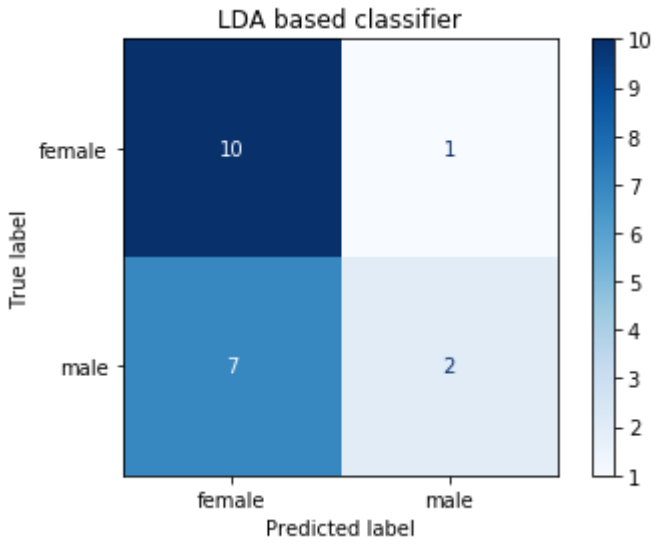
Out[35]:

```
390    0.093422
391   -0.005016
392    0.149489
393    0.084264
394    0.063795
395    0.052293
396    0.160087
397    0.017396
398    0.045768
399    0.040492
790    0.118789
791    0.041542
792    0.105642
793    0.068019
794    0.089727
795    0.098056
796    0.043459
797    0.053029
798    0.056282
799    0.033971
dtype: float64
```

Classification

In [36]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import plot_confusion_matrix

classifier = GaussianNB()
classifier.fit(np.array(red_train_data).reshape(-1,1),df_train['Class'])

disp = plot_confusion_matrix(classifier,np.array(red_test_data).reshape(-1,1) , df_test["Cl
                                 display_labels=["female","male"],
                                 cmap=plt.cm.Blues,
                                 )
disp.ax_.set_title("LDA based classifier")
plt.show()
```



3 Give the comparative study for the above results w.r.t to classification accuracy in terms of the confusion matrix.

Naive Bayes classifier gives 60 percent accuracy for LDA.Whereas for PCA we get 95 percent accuracy. We can see that PCA outperforms LDA in classification.

From the confusion matrix we see PCA based classifier only misclassifies one sample. LDA based classifier misclassifies 8 samples out of 20

Q4. Eigenfaces - Face classification using PCA (40 classes)

a) Use the following "face.csv" file to classify the faces of 40 different people.

b) Do not use in-built function for implementing PCA.

c) Use appropriate classifier taught in class (any classification algorithm taught in class like Bayes classifier, minimum distance clasifier, and so on)

d) Refer to the following link for a description of the dataset

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
```

In [3]:

```
df = pd.read_csv('face.csv')
df.drop(df.columns[-1], axis = 1, inplace = True)
df
```

Out[3]:

	0	1	2	3	4	5	6	7	8
0	0.309917	0.367769	0.417355	0.442149	0.528926	0.607438	0.657025	0.677686	0.690083
1	0.454545	0.471074	0.512397	0.557851	0.595041	0.640496	0.681818	0.702479	0.710744
2	0.318182	0.400826	0.491736	0.528926	0.586777	0.657025	0.681818	0.685950	0.702479
3	0.198347	0.194215	0.194215	0.194215	0.190083	0.190083	0.243802	0.404959	0.483471
4	0.500000	0.545455	0.582645	0.623967	0.648760	0.690083	0.694215	0.714876	0.723140
...	...	...	...	...	...	...	...	...	...
395	0.400826	0.495868	0.570248	0.632231	0.648760	0.640496	0.661157	0.636364	0.665289
396	0.367769	0.367769	0.351240	0.301653	0.247934	0.247934	0.367769	0.512397	0.574380
397	0.500000	0.533058	0.607438	0.628099	0.657025	0.632231	0.657025	0.669422	0.673554
398	0.214876	0.219008	0.219008	0.223140	0.210744	0.202479	0.276859	0.400826	0.487603
399	0.516529	0.462810	0.280992	0.252066	0.247934	0.367769	0.574380	0.615702	0.661157

400 rows × 4096 columns

In [5]:

```
train_data_set = df.iloc[2:10]

for i in range(10, 400, 10):
  train_data_set = train_data_set.append(df.iloc[i + 2:i + 10], ignore_index = True)
```

In [6]:

```
test_data_set = df.iloc[0:2]

for i in range(10, 400, 10):
  test_data_set = test_data_set.append(df.iloc[i:i + 2], ignore_index = True)
```

In [7]:

```
train_data_set = np.transpose(train_data_set)

test_data_set = np.transpose(test_data_set)

d = 4096
```

In [8]:

```
def mean_PCA(train_data_set):
    train_data_set_mean = np.mean(train_data_set, axis = 1)
    return np.array(train_data_set_mean)
```

In [9]:

```
train_data_set_mean = mean_PCA(train_data_set)

train_data_set_covariance_matrix = np.cov(train_data_set)

eigen_values, eigen_vectors = np.linalg.eigh(train_data_set_covariance_matrix)
```

In [10]:

```
eigen_vectors
```

Out[10]:

```
array([[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00, ...,
         4.07541819e-04,  2.90708397e-02, -2.74754798e-03],
       [ 1.41323302e-01,  1.47697747e-02, -3.20600723e-01, ...,
        -1.51739158e-03,  3.42810801e-02, -5.43886813e-03],
       [ 4.52142226e-01,  2.75796727e-01,  3.75162941e-01, ...,
        -2.33185371e-03,  3.90819016e-02, -7.60074745e-03],
       ...,
       [-1.08510497e-02, -2.50947201e-02, -6.58641110e-03, ...,
        -1.30268290e-02, -2.99621494e-02, -2.94621031e-03],
       [ 1.45648223e-03,  2.43254843e-02,  3.17920811e-03, ...,
        -1.02592154e-02, -2.74033230e-02,  8.39296650e-04],
       [-6.55733247e-03,  1.23748159e-02, -1.40301892e-02, ...,
        -8.67573068e-03, -2.50393131e-02,  1.56090205e-03]])
```

In [12]:

```
eigen_pairs = [(np.abs(eigen_values[i]), eigen_vectors[:, i]) for i in range(len(eigen_valu

eigen_pairs.sort(key = lambda x : x[0], reverse = True)

eigen_pairs
```

Out[12]:

```
[(19.105796095321377,
  array([-0.00274755, -0.00543887, -0.00760075, ..., -0.00294621,
          0.0008393 ,  0.0015609 ])),
 (12.207087351051666,
  array([ 0.02907084,  0.03428108,  0.0390819 , ..., -0.02996215,
         -0.02740332, -0.02503931])),
 (6.166659790621014,
  array([ 0.00040754, -0.00151739, -0.00233185, ..., -0.01302683,
         -0.01025922, -0.00867573])),
 (3.766708963990769,
  array([-0.00253901, -0.00561563, -0.01419596, ...,  0.031047  ,
          0.02984297,  0.02726496])),
 (2.730287928495426,
  array([0.02352622, 0.02307565, 0.02036678, ..., 0.03885212, 0.03756989,
         0.0392973 ])),
 (2.491505137016736,
  array([-0.02104077, -0.0211751 , -0.01811579, ...,  0.0269678 ,
          0.02674315,  0.02640275])),
```

In [13]:

```
def get_reduced_dimension(eigen_pairs, eigen_values, percent):
    eigen_values_sum = np.sum(eigen_values)
    eigen_values_sum_reduced_dimension = percent * eigen_values_sum / 100
    sum = 0.0
    count = 0
    for i in eigen_pairs:
        if(sum < eigen_values_sum_reduced_dimension):
            sum += i[0]
            count += 1
        else:
            break
    return count
```

In [14]:

```
d_prime = get_reduced_dimension(eigen_pairs, eigen_values, 95)
d_prime
```

Out[14]:

```
111
```

In [15]:

```
matrix_w = eigen_pairs[0][1].reshape(d, 1)
```

In [16]:

```
for i in range(1, d_prime):
    matrix_w = np.hstack((matrix_w, eigen_pairs[i][1].reshape(d, 1)))
```

In [17]:

```
train_data_set_transformed = np.transpose(matrix_w.T.dot(train_data_set))

test_data_set_transformed = np.transpose(matrix_w.T.dot(test_data_set))
```

In [18]:

```
train_data_set_transformed
```

Out[18]:

```
array([[-3.84278500e+01,  1.05496914e+01,  2.76257385e+00, ...,
        -1.49311834e-01,  2.64169294e-01,  4.94872391e-01],
       [-3.80958302e+01, -2.36530844e+00,  2.77395344e+00, ...,
         1.07242465e-01, -2.50675502e-01,  2.44819087e-01],
       [-3.70319559e+01,  1.22432369e+01,  6.31392191e+00, ...,
        -1.36547366e-01, -4.01670809e-02,  3.49316185e-01],
       ...,
       [-3.11444304e+01,  1.01462209e+01,  5.96225995e+00, ...,
        -6.70871573e-02, -1.75148272e-01,  3.32349524e-01],
       [-3.88338294e+01,  9.01930153e-01,  1.98477303e+00, ...,
        -2.30756990e-02, -3.69142681e-01,  3.00644319e-01],
       [-3.45162687e+01,  6.32423245e+00,  3.80672052e+00, ...,
        -3.53049751e-01,  2.48143830e-01,  2.12767436e-01]])
```

In [19]:

```
test_data_set_transformed
```

Out[19]:

```
array([[-3.95811272e+01,  9.73374957e+00,  1.16472961e+00, ...,
         1.14704260e-02,  2.30335788e-02,  1.73677045e-01],
       [-3.39897567e+01,  1.54595402e+01,  4.52888350e+00, ...,
        -4.59040975e-01, -1.37985001e-01,  4.52161510e-01],
       [-3.42602930e+01,  9.76087593e+00, -1.34650656e+00, ...,
        -1.48750983e-01,  2.39329589e-01,  3.29394425e-01],
       ...,
       [-2.30611479e+01,  9.42401369e+00,  1.71149373e+00, ...,
        -8.64847903e-02, -4.13596812e-01,  5.43861931e-01],
       [-3.18795206e+01,  1.02691521e+01,  5.97999216e+00, ...,
        -2.37681112e-02,  6.24611657e-02,  2.73599124e-01],
       [-3.73080804e+01,  6.39686729e+00,  4.28431021e+00, ...,
        -2.95259371e-01, -4.42425011e-01,  2.70357075e-01]])
```

Bayes Classifier Code

Bayes Classifier with dimesion = 4096

In [20]:

```
train_data_set
```

Out[20]:

	0	1	2	3	4	5	6	7	8
0	0.318182	0.198347	0.500000	0.549587	0.330578	0.128099	0.243802	0.380165	0.657025
1	0.400826	0.194215	0.545455	0.545455	0.305785	0.185950	0.297521	0.442149	0.677686
2	0.491736	0.194215	0.582645	0.541322	0.330578	0.247934	0.367769	0.483471	0.698347
3	0.528926	0.194215	0.623967	0.537190	0.351240	0.314050	0.454545	0.545455	0.706612
4	0.586777	0.190083	0.648760	0.537190	0.425620	0.388430	0.495868	0.582645	0.702479
...	...	...	...	...	...	...	...	...	..
4091	0.148760	0.743802	0.177686	0.661157	0.487603	0.157025	0.177686	0.173554	0.115702
4092	0.144628	0.764463	0.177686	0.690083	0.190083	0.152893	0.190083	0.173554	0.115702
4093	0.140496	0.752066	0.177686	0.714876	0.144628	0.152893	0.190083	0.173554	0.115702
4094	0.148760	0.752066	0.173554	0.706612	0.152893	0.173554	0.181818	0.173554	0.115702
4095	0.152893	0.739669	0.173554	0.702479	0.152893	0.173554	0.190083	0.173554	0.103306

4096 rows × 320 columns

In [21]:

```
test_data_set
```

Out[21]:

	0	1	2	3	4	5	6	7	8
0	0.309917	0.454545	0.541322	0.644628	0.578512	0.628099	0.169422	0.561983	0.454545
1	0.367769	0.471074	0.586777	0.690083	0.603306	0.665289	0.264463	0.665289	0.429752
2	0.417355	0.512397	0.640496	0.702479	0.632231	0.685950	0.219008	0.698347	0.537190
3	0.442149	0.557851	0.661157	0.702479	0.665289	0.694215	0.280992	0.698347	0.611570
4	0.528926	0.595041	0.685950	0.706612	0.677686	0.719008	0.421488	0.731405	0.652893
...	...	...	...	...	...	...	...	...	..
4091	0.132231	0.152893	0.363636	0.111570	0.194215	0.177686	0.363636	0.231405	0.376033
4092	0.148760	0.152893	0.111570	0.115702	0.165289	0.161157	0.198347	0.214876	0.409091
4093	0.152893	0.152893	0.095041	0.111570	0.177686	0.185950	0.210744	0.219008	0.409091
4094	0.161157	0.152893	0.111570	0.107438	0.161157	0.161157	0.235537	0.219008	0.384298
4095	0.157025	0.152893	0.111570	0.119835	0.152893	0.190083	0.214876	0.227273	0.384298

4096 rows × 80 columns

In [22]:

```
train_data_set = np.transpose(train_data_set)
test_data_set = np.transpose(test_data_set)

X_train = train_data_set
y_train = []

for i in range(40):
    for j in range(int(len(train_data_set)/ 40)):
        y_train.append(str(i))
```

In [23]:

```
X_test = test_data_set
y_test = []

for i in range(40):
    for j in range(int(len(test_data_set)/ 40)):
        y_test.append(str(i))
```

In [24]:

```
gnb = GaussianNB()

gnb.fit(X_train, y_train)

y_pred = gnb.predict(X_test)

overall_accuracy = metrics.accuracy_score(y_test, y_pred)
```

In [25]:

```
print("Bayes Classifier with dimension = " + str(d))
print("\nOverall Accuracy = " + str(overall_accuracy))
```

```
Bayes Classifier with dimension = 4096

Overall Accuracy = 0.9
```

Bayes Classifier with dimesion = 111

In [26]:

```
X_train_reduced_dimension = train_data_set_transformed
y_train_reduced_dimension = []

for i in range(40):
    for j in range(int(len(train_data_set_transformed)/ 40)):
        y_train_reduced_dimension.append(str(i))

X_train_reduced_dimension
```

Out[26]:

```
array([[-3.84278500e+01,  1.05496914e+01,  2.76257385e+00, ...,
        -1.49311834e-01,  2.64169294e-01,  4.94872391e-01],
       [-3.80958302e+01, -2.36530844e+00,  2.77395344e+00, ...,
         1.07242465e-01, -2.50675502e-01,  2.44819087e-01],
       [-3.70319559e+01,  1.22432369e+01,  6.31392191e+00, ...,
        -1.36547366e-01, -4.01670809e-02,  3.49316185e-01],
       ...,
       [-3.11444304e+01,  1.01462209e+01,  5.96225995e+00, ...,
        -6.70871573e-02, -1.75148272e-01,  3.32349524e-01],
       [-3.88338294e+01,  9.01930153e-01,  1.98477303e+00, ...,
        -2.30756990e-02, -3.69142681e-01,  3.00644319e-01],
       [-3.45162687e+01,  6.32423245e+00,  3.80672052e+00, ...,
        -3.53049751e-01,  2.48143830e-01,  2.12767436e-01]])
```

In [27]:

```
X_train_reduced_dimension.shape
```

Out[27]:

```
(320, 111)
```

In [28]:

```
X_test_reduced_dimension = test_data_set_transformed
y_test_reduced_dimension = []

for i in range(40):
    for j in range(int(len(test_data_set_transformed)/ 40)):
        y_test_reduced_dimension.append(str(i))

X_test_reduced_dimension
```

Out[28]:

```
array([[-3.95811272e+01,  9.73374957e+00,  1.16472961e+00, ...,
         1.14704260e-02,  2.30335788e-02,  1.73677045e-01],
       [-3.39897567e+01,  1.54595402e+01,  4.52888350e+00, ...,
        -4.59040975e-01, -1.37985001e-01,  4.52161510e-01],
       [-3.42602930e+01,  9.76087593e+00, -1.34650656e+00, ...,
        -1.48750983e-01,  2.39329589e-01,  3.29394425e-01],
       ...,
       [-2.30611479e+01,  9.42401369e+00,  1.71149373e+00, ...,
        -8.64847903e-02, -4.13596812e-01,  5.43861931e-01],
       [-3.18795206e+01,  1.02691521e+01,  5.97999216e+00, ...,
        -2.37681112e-02,  6.24611657e-02,  2.73599124e-01],
       [-3.73080804e+01,  6.39686729e+00,  4.28431021e+00, ...,
        -2.95259371e-01, -4.42425011e-01,  2.70357075e-01]])
```

In [29]:

```
X_test_reduced_dimension.shape
```

Out[29]:

```
(80, 111)
```

In [30]:

```
gnb_reduced_dimension = GaussianNB()

gnb_reduced_dimension.fit(X_train_reduced_dimension, y_train_reduced_dimension)

y_pred_reduced_dimension = gnb_reduced_dimension.predict(X_test_reduced_dimension)

overall_accuracy_reduced_dimension = metrics.accuracy_score(y_test_reduced_dimension, y_pre

print("Bayes Classifier with dimension = " + str(d_prime))
print("\nOverall Accuracy = " + str(overall_accuracy_reduced_dimension))
```

```
Bayes Classifier with dimension = 111

Overall Accuracy = 0.975
```

In [ ]:

```

```

q5

April 5, 2021

1 5. Fisherfaces- Face classification using LDA (40 classes)

Use the following "face.csv" file to classify the faces of 40 different people.

[18]:
```
!pip install scikit-plot
```

```
Collecting scikit-plot
  Downloading https://files.pythonhosted.org/packages/7c/47/32520e259340c140a4ad
27c1b97050dd3254fdc517b1d59974d47037510e/scikit_plot-0.3.7-py3-none-any.whl
Requirement already satisfied: scipy>=0.9 in /usr/local/lib/python3.7/dist-
packages (from scikit-plot) (1.4.1)
Requirement already satisfied: matplotlib>=1.4.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-plot) (3.2.2)
Requirement already satisfied: scikit-learn>=0.18 in
/usr/local/lib/python3.7/dist-packages (from scikit-plot) (0.22.2.post1)
Requirement already satisfied: joblib>=0.10 in /usr/local/lib/python3.7/dist-
packages (from scikit-plot) (1.0.1)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-
packages (from scipy>=0.9->scikit-plot) (1.19.5)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib>=1.4.0->scikit-plot)
(1.3.1)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib>=1.4.0->scikit-plot)
(2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib>=1.4.0->scikit-plot)
(2.4.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-
packages (from matplotlib>=1.4.0->scikit-plot) (0.10.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-
packages (from python-dateutil>=2.1->matplotlib>=1.4.0->scikit-plot) (1.15.0)
Installing collected packages: scikit-plot
Successfully installed scikit-plot-0.3.7
```

[52]:
```
import numpy as np
import pandas as pd
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import scikitplot as skplt
```

2 Function for LDA

[86]:
```
def LDA(X,labels):

        d = X.shape[1]
        classes=np.unique(labels)
        c=len(classes)
        d_=c-1
        class_dict={}
        for i in range(len(classes)):
                class_dict[classes[i]]=i

        class_wise_data=[np.empty((0,)+X[0].shape,float) for i in classes]
        for i in range(len(X)):
                class_wise_data[class_dict[labels[i]]]=np.
↪append(class_wise_data[class_dict[labels[i]]], np.array([X[i],]),axis=0)

        means=[]
        for i in class_wise_data:
                means.append(np.mean(i,axis=0))

        Sw = np.zeros((d,d))
        for i,data in enumerate(class_wise_data):
                z=data-means[i]
                Sw+=(z.T @ z)
        Sw_inv=np.linalg.inv(Sw)

        overall_mean = np.mean(X,axis=0)
        Sb = np.zeros((d,d))
        for i, data in enumerate(means):
                Ni=len(class_wise_data[i])
                z=np.array([means[i]-overall_mean])
                Sb+=(Ni * (z.T @ z))

        M = Sw_inv @ Sb
        eigen_values , eigen_vectors = np.linalg.eigh(M)
        eigen_values , eigen_vectors = eigen_values.astype(np.float64) ,␣
↪eigen_vectors.astype(np.float64)
        sorted_index = np.argsort(eigen_values)[::-1]
        sorted_eigenvectors = eigen_vectors[:,sorted_index]
        sorted_eigenvalue = eigen_values[sorted_index]
```

```
        eigenvector_subset = sorted_eigenvectors[:,0:d_]

        plt.bar(list(range(1,eigen_vectors.shape[0]+1)),sorted_eigenvalue)
        plt.ylabel("eigen values")

        Y=X @ eigenvector_subset
        return Y,eigenvector_subset
```

3 Read data and split it to test and train

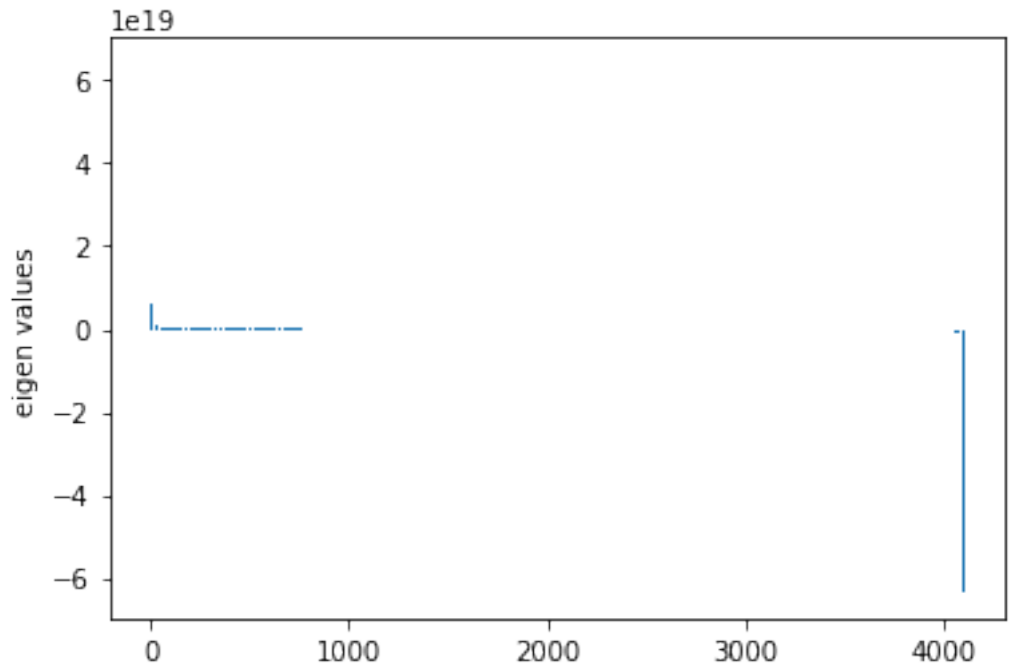
[87]:
```
data=pd.read_csv('face.csv')
```

[88]:
```
x = data.iloc[:,:-1]
target = data.iloc[:,-1]
```

[89]:
```
train_data = pd.concat([data.iloc[i*10+2:(i+1)*10] for i in range(40)])
test_data = pd.concat([data.iloc[i*10:i*10+2] for i in range(40)])
train_data.reset_index(drop=True, inplace=True)
test_data.reset_index(drop=True, inplace=True)
```

4 Use LDA function to reduce dim

[90]:
```
reduced,eigenvector_subset = LDA(np.array(train_data.iloc[:,:
↪-1]),list(train_data['target']))
```



[91]:
```
reduced = pd.DataFrame(reduced)
```

5 Classify test data and find accuracy

[92]:
```
model = GaussianNB()
model.fit(reduced,train_data["target"])
```

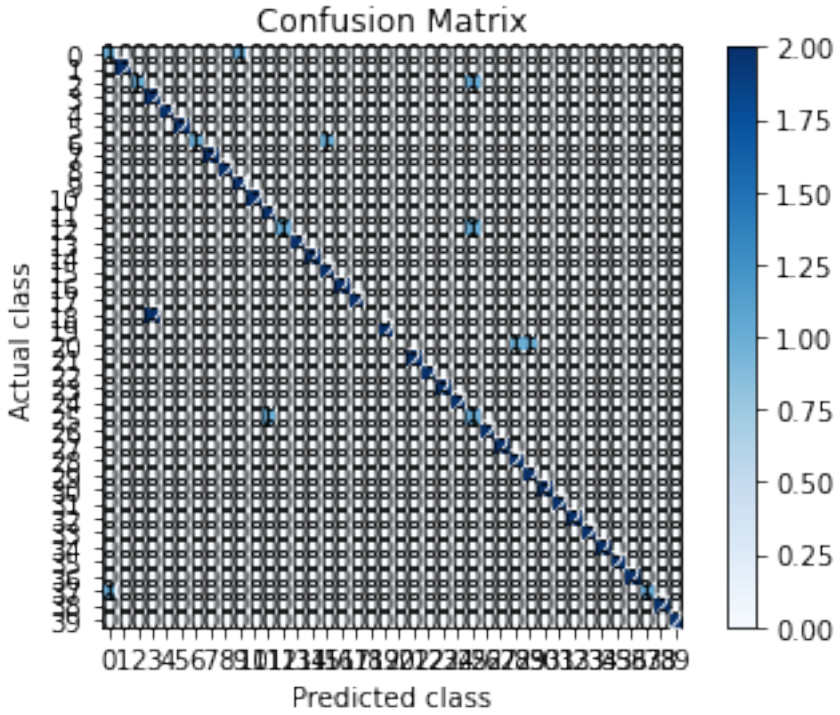
[92]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

[93]:
```
test_reduced=(test_data.iloc[:,:-1]).dot(eigenvector_subset)
predicted= model.predict(test_reduced)
test_reduced['target']=test_data['target']
```

[94]:
```
test_reduced['predicted'] = predicted
correctness=[]
for i in test_reduced.index:
        if test_reduced['target'][i] == test_reduced['predicted'][i]:
                correctness.append("correct")
        else:
                correctness.append("wrong")

test_reduced["correctness"]=correctness
print(test_reduced)
```

```
x=accuracy_score(test_reduced["target"], predicted)
print(f"Accuracy ={x*100}%")

skplt.metrics.plot_confusion_matrix(test_reduced["target"], predicted)#,␣
↪normalize=True)
plt.xlabel('Predicted class')
plt.ylabel('Actual class')
plt.show()
```

```
            0         1         2  …  target  predicted  correctness
0  -14.907826 -4.569430 -0.827318  …       0          0      correct
1  -12.912718 -2.557044  0.112835  …       0          9        wrong
2  -14.155274 -4.687639 -1.662370  …       1          1      correct
3  -13.903795 -4.280577 -1.522141  …       1          1      correct
4  -12.324845 -2.570033 -1.145355  …       2         25        wrong
..        …         …         …  …  …       …          …
75 -13.024525 -2.618749 -0.415230  …      37         37      correct
76  -9.124297 -3.698598 -0.485431  …      38         38      correct
77  -8.225609 -2.829540 -0.746084  …      38         38      correct
78 -11.922242 -1.713910 -1.664928  …      39         39      correct
79 -14.177076 -3.706217 -1.264375  …      39         39      correct

[80 rows x 42 columns]
Accuracy =87.5%
```



[ ]: