PR - Assignment 3

05.04.2021

—

Group 8

Pavendhan N - COE18B041

Santosh Sinduja S - COE18B046

Shanjeev Maruthi - COE18B047

Question 1

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
import scikitplot as skplt
```

In [6]:
```
df = pd.read_csv('gender_feature_vectors.csv')
classes = df['Unnamed: 1']
df.drop(columns=['Unnamed: 1','Unnamed: 0'], axis=1, inplace=True)
df['target'] = classes
df.head(10)
```

Out[6]:

	0	1	2	3	4	5	6	7	8	9	...	
0	-0.066420	0.151611	0.027740	0.052771	-0.066105	-0.041232	-0.002637	-0.158467	0.130467	-0.044872	...	-0.0
1	-0.030614	0.049667	0.008084	-0.050324	0.007649	-0.063818	-0.019530	-0.119905	0.186553	-0.044821	...	-0.0
2	-0.096178	0.061127	0.035326	-0.035388	-0.090728	-0.018634	-0.024315	-0.139786	0.052211	-0.052085	...	0.0
3	-0.103057	0.085044	0.078333	-0.035873	-0.028163	0.004924	0.007829	-0.017016	0.114907	-0.056267	...	-0.0
4	-0.125815	0.120046	0.023131	-0.042901	0.038215	-0.049677	-0.054258	-0.130758	0.173457	-0.011889	...	0.0
5	-0.149119	0.125288	0.142323	-0.009087	-0.031394	-0.123533	0.043598	-0.063999	0.162439	-0.086513	...	0.0
6	-0.139035	0.073513	-0.001770	-0.034225	-0.101610	0.065105	-0.014420	-0.054993	0.134674	-0.058293	...	-0.0
7	-0.074126	-0.000669	0.004166	-0.082413	-0.096091	-0.021992	0.009714	-0.056961	0.174237	-0.056700	...	0.0
8	-0.166220	0.042769	-0.031647	-0.036892	-0.143837	-0.040566	0.042541	-0.122923	0.188971	-0.036112	...	-0.0
9	-0.185770	0.154008	0.073184	-0.070829	-0.144617	-0.019732	-0.019418	-0.004675	0.152325	0.017508	...	0.0

10 rows × 129 columns

In [17]:
```
male = df.query('target == "male"')
female = df.query('target == "female"')

test_data = male.head(10)
test_data = test_data.append(female.head(10))
test_data.reset_index(drop=True,inplace=True)
train_data = df.drop(test_data.index, axis=0)
train_data.reset_index(drop=True,inplace=True)
```

In [18]:

```
X = train_data.iloc[:,:-1]

threshold = 0.95

mean_ = np.mean(X, axis=0)

# Centering the data
X_meaned = X - mean_

# Calculating the covariance matrix of the mean-centered data
cov_mat = np.cov(X_meaned , rowvar = False)

# Calculating Eigenvalues and Eigenvectors of the covariance matrix
eigen_values , eigen_vectors = np.linalg.eigh(cov_mat)

# Sort the eigenvalues in descending order
sorted_index = np.argsort(eigen_values)[::-1]
sorted_eigenvalue = eigen_values[sorted_index].astype(np.float64)

# Similarly sort the eigenvectors
sorted_eigenvectors = eigen_vectors[:,sorted_index]

# calculate the percentage of explained variance per principal component
cumul_eigenvalue = sorted_eigenvalue.cumsum()
cumul_on_total = cumul_eigenvalue / cumul_eigenvalue[-1]

num_components = 0
while(cumul_on_total[num_components] < threshold):
    num_components += 1
num_components += 1

print('The number of features have been reduced from {} to {} for a threshold of {}% va

eigenvector_subset = sorted_eigenvectors[:, 0:num_components]

# Transform the data
X_reduced = np.dot(eigenvector_subset.transpose(), X_meaned.transpose()).transpose()
```

```
The number of features have been reduced from 128 to 57 for a threshold of 95.0% varia
nce.
```

In [19]:

```
principal_df = pd.DataFrame(X_reduced , columns = ['col'+str(i) for i in range(1,num_co
principal_df = pd.concat([principal_df , train_data['target']] , axis = 1)
```

In [20]:

```
classifier = GaussianNB()
classifier.fit(principal_df[['col'+str(i) for i in range(1,num_components+1)]], princip
```

Out[20]:

```
GaussianNB()
```

In [21]:

```
test_reduced = (eigenvector_subset.T @ (test_data.iloc[:,:-1] - mean_).T).T
predicted = classifier.predict(test_reduced)
test_reduced = pd.concat([test_reduced, test_data['target']], axis = 1)
```

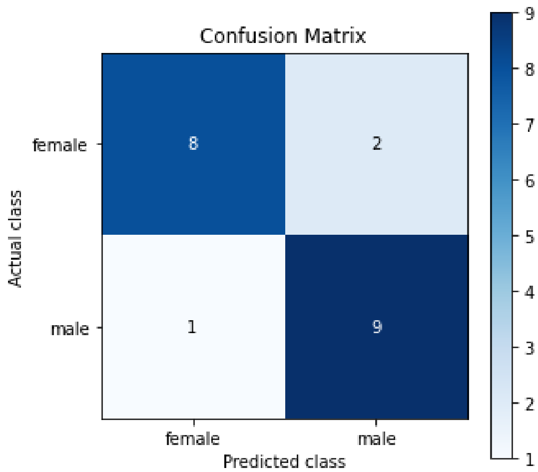
In [22]:

```
test_reduced['predicted'] = predicted
acc = []
for i in range(len(test_reduced)):
    if test_reduced['target'][i] == test_reduced['predicted'][i]:
        acc.append("correct")
    else:
        acc.append("wrong")

test_reduced["correctness"] = acc
x = accuracy_score(test_reduced["target"], predicted)
print("Accuracy =",x*100,"%")
```

```
Accuracy = 85.0 %
```

In [28]:

```
skplt.metrics.plot_confusion_matrix(test_reduced["target"], predicted, figsize=(5,5))
plt.xlabel('Predicted class')
plt.ylabel('Actual class')

plt.savefig("q1_confusion_matrix")
plt.show()
```



In [ ]:

Question 2 ¶

In [20]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.naive_bayes import GaussianNB
import scikitplot as skplt
```

In [21]:
```
df= pd.read_csv(('gender_feature_vectors.csv'),index_col=0)
df.drop(df.columns[df.columns.str.contains('unnamed',case = False)],axis = 1, inplace =
```

First 399 are male category Next 401 are male category

In [22]:
```
df=df*10000
print(df.shape)
df.head(401)
```

```
(800, 128)
```

Out[22]:

	0	1	2	3	4	5	6	7	8	
1	-664.19959	1516.11447	277.39607	527.70555	-661.04963	-412.32228	-26.37491	-1584.66667	1304.66834	
2	-306.13856	496.66520	80.83738	-503.23568	76.49306	-638.18008	-195.30300	-1199.05055	1865.53150	
3	-961.77682	611.26690	353.26038	-353.88201	-907.28119	-186.34144	-243.14573	-1397.85841	522.10610	
4	-1030.57027	850.43512	783.32767	-358.73279	-281.62964	49.24194	78.28606	-170.15841	1149.06780	
5	-1258.15049	1200.45893	231.31274	-429.01006	382.14993	-496.76508	-542.58350	-1307.58137	1734.57026	
...	...	...	...	...	...	...	...	...	...	
397	-1584.60408	1099.47540	190.87646	155.06018	-696.68218	323.11093	150.61509	-1408.17016	1411.32370	
398	-1014.99282	1197.38773	169.50686	-136.76908	-555.23962	283.99080	281.64148	-1520.99669	1098.13653	
399	-1495.15584	815.88000	907.95673	-531.16299	-1333.13820	10.95610	199.41203	-1178.03350	1023.19576	
400	398.43969	703.56630	1301.96080	-76.82588	-778.24667	-212.97958	-241.32527	-851.04927	712.88377	-1
401	17.46896	1856.77752	732.59771	421.42265	-886.73756	281.86083	-278.29867	-642.11033	974.12795	

401 rows × 128 columns

In [24]:
```
# splitting into train and test data
test_male = df[:10].to_numpy()
test_female = df[399:409].to_numpy()
train_male = df[10:399].to_numpy()
train_female = df[409:].to_numpy()
```

In [25]:
```
mean_male = np.mean(train_male,axis=0)
mean_female = np.mean(train_female,axis=0)
```

In [26]:

```
# calculating within class scatter matrix
S_W = np.zeros((128,128))
class_sc_mat = np.zeros((128,128))

for row in train_male:
    row, mv = row.reshape(128,1), mean_male.reshape(128,1) # make column vectors
    class_sc_mat += (row-mv).dot((row-mv).T)
S_W += class_sc_mat                             # sum class scatter matrices

class_sc_mat=np.zeros((128,128))
for row in train_female:
    row, mv = row.reshape(128,1), mean_female.reshape(128,1) # make column vectors
    class_sc_mat += (row-mv).dot((row-mv).T)
S_W += class_sc_mat                             # sum class scatter matrices
```

In [27]:

```
# calculating between class scatter matrix
overall_mean = np.mean(np.concatenate((train_male,train_female), axis=0), axis=0)
overall_mean = np.asarray(overall_mean)
overall_mean = overall_mean.reshape(128,1)
S_B = np.zeros((128,128))

n1 = train_male.shape[0]
n2 = train_female.shape[0]

mean_vec = np.asarray(mean_male)
mean_vec = mean_vec.reshape(128,1) # make column vector
 # make column vector
z = mean_vec - overall_mean
A = n1 * (np.matmul(z,z.T))
S_B = np.add(S_B,A)


mean_vec = np.asarray(mean_female)
mean_vec = mean_vec.reshape(128,1)
z = mean_vec - overall_mean
A = n2 * (np.matmul(z,z.T))
S_B = np.add(S_B,A)
```

Eigen pairs

In [28]:

```
Sw_inv = np.linalg.inv(S_W)
#print(Sw_inv)
M = Sw_inv @ S_B
```

In [32]:

```
eig_vals, eig_vecs = np.linalg.eig(M)
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs = sorted(eig_pairs, key=lambda k: k[0], reverse=True)

eigv_sum = sum(eig_vals)
tot = 0
for i,j in enumerate(eig_pairs):
    tot += (j[0]/eigv_sum).real
    if(tot > 0.9):
        break
```

In [33]:

```
eigen_values , eigen_vectors = eig_vals, eig_vecs
sorted_index = np.argsort(eigen_values)[::-1]
sorted_eigenvectors = eigen_vectors[:,sorted_index]

#print(sum(eigen_values))
sorted_eigenvalue = eigen_values[sorted_index].astype(np.float64)

#We chose only 1
W = sorted_eigenvectors[:,0:1].astype(np.float64)
print(W.shape)
```

```
(128, 1)

/home/sinduja/.local/lib/python3.6/site-packages/ipykernel_launcher.py:6: ComplexWarni
ng: Casting complex values to real discards the imaginary part

/home/sinduja/.local/lib/python3.6/site-packages/ipykernel_launcher.py:9: ComplexWarni
ng: Casting complex values to real discards the imaginary part
  if __name__ == '__main__':
```

In [34]:

```
X_train=np.concatenate((train_male,train_female),axis=0)
X_reduced=X_train @ W
```

In [35]:

```
# let male be 1 and female be 2
y = [1 for i in range(389)]+[2 for i in range(391)]
y = np.array(y)
```

In [36]:

```
classifier=GaussianNB()
classifier.fit(X_reduced.reshape(780,1),y)
```

Out[36]:

```
GaussianNB()
```

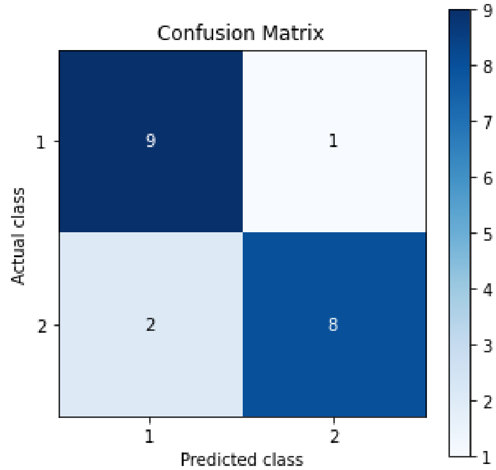
In [37]:

```
correct=0
cases=0
predicted=[]
for row in test_male:
    prediction = classifier.predict((row @ W).reshape(1,1))
    cases+=1
    predicted.append(prediction[0])
    if(prediction[0]==1):
        correct+=1
        print("Actual : Male Predicted : Male")
    else:
        print("Actual : Male Predicted : Female")
for row in test_female:
    prediction = classifier.predict((row @ W).reshape(1,1))
    cases+=1
    predicted.append(prediction[0])
    if(prediction[0]==2):
        correct+=1
        print("Actual : Female Predicted : Female")
    else:
        print("Actual : Female Predicted : Male")

accuracy=correct/cases
print("Accuracy:",accuracy)
```

```
Actual : Male Predicted : Male
Actual : Male Predicted : Male
Actual : Male Predicted : Male
Actual : Male Predicted : Male
Actual : Male Predicted : Male
Actual : Male Predicted : Female
Actual : Male Predicted : Male
Actual : Male Predicted : Male
Actual : Male Predicted : Male
Actual : Male Predicted : Male
Actual : Female Predicted : Male
Actual : Female Predicted : Male
Actual : Female Predicted : Female
Actual : Female Predicted : Female
Actual : Female Predicted : Female
Actual : Female Predicted : Female
Actual : Female Predicted : Female
Actual : Female Predicted : Female
Actual : Female Predicted : Female
Actual : Female Predicted : Female
Accuracy: 0.85
```

In [38]:

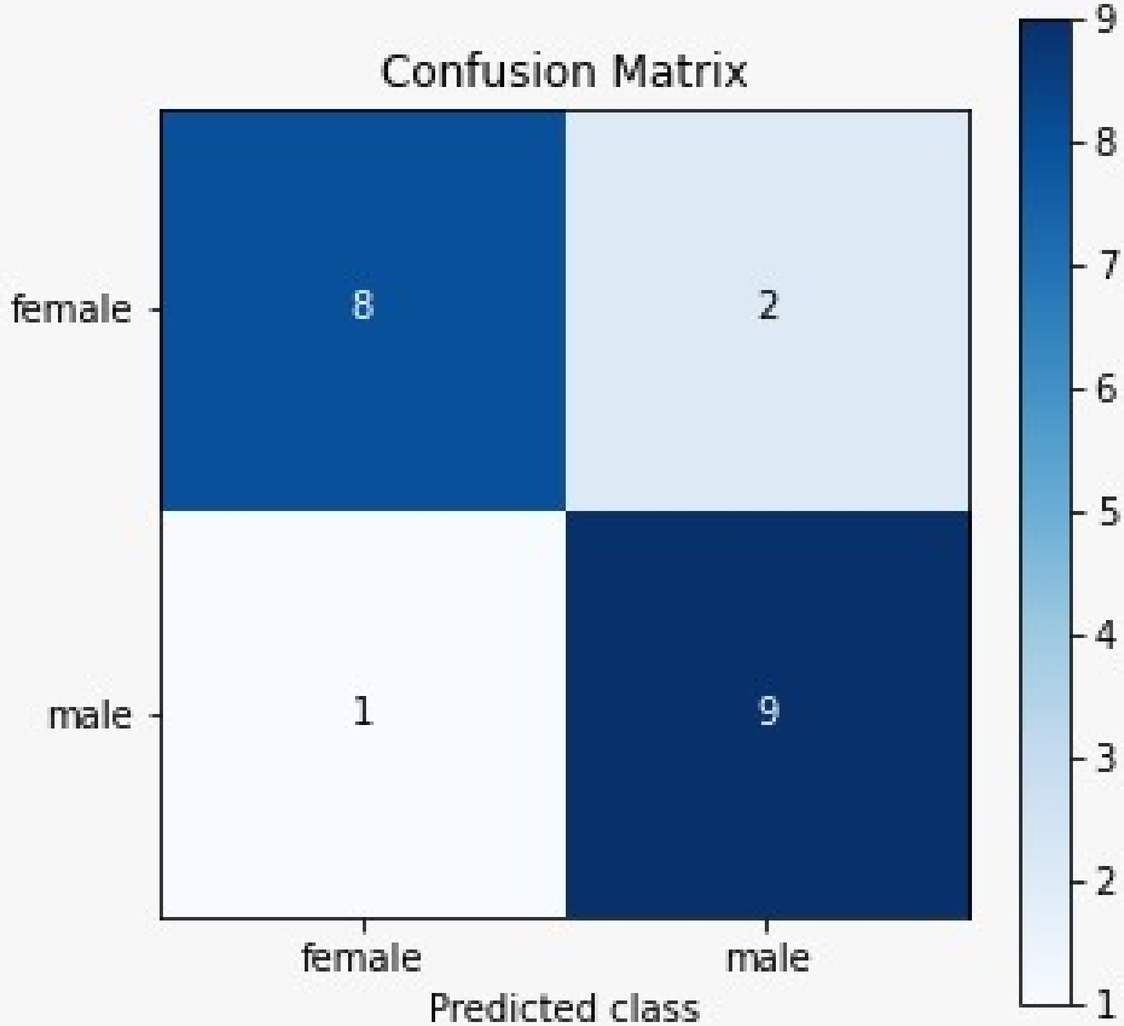
```
expected=[1 for i in range(10)]+[2 for i in range(10)]
skplt.metrics.plot_confusion_matrix(expected, predicted, figsize=(5,5))
plt.xlabel('Predicted class')
plt.ylabel('Actual class')
plt.savefig("q2_confusion_matrix")
plt.show()
```

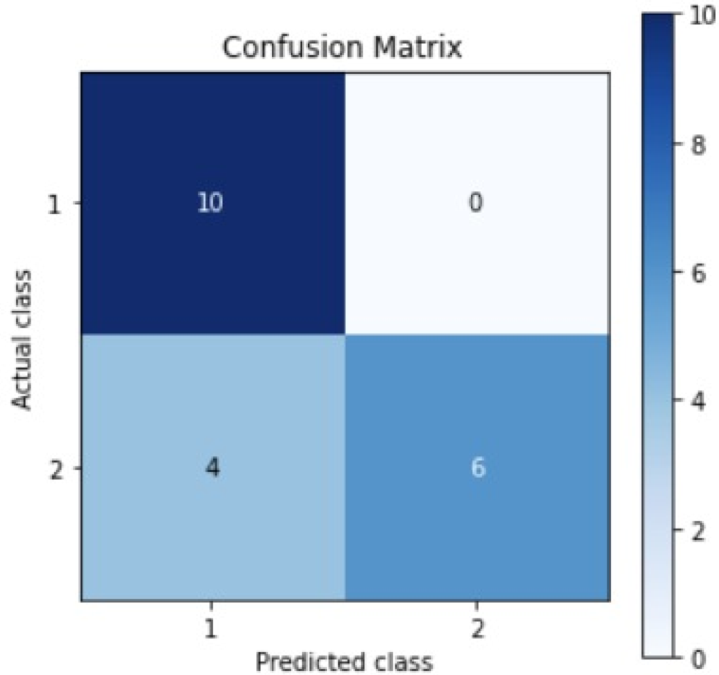


Question 3

In [42]:
```
from IPython.display import Image
```

In [47]:
```
display(Image(filename="1_cm.jpg"))
display(Image(filename="2_cm.jpeg"))
```




Confusion Matrix
10
0
4
6
1
2
Actual class
1
2
Predicted class
10
8
6
4
2
0

In [ ]:

Question 4

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
import scikitplot as skplt
```

In [2]:
```
df = pd.read_csv('face.csv')
df.head(10)
```

Out[2]:

	0	1	2	3	4	5	6	7	8	9	...	4087	4088
0	0.309917	0.367769	0.417355	0.442149	0.528926	0.607438	0.657025	0.677686	0.690083	0.685950	...	0.669422	0.652893
1	0.454545	0.471074	0.512397	0.557851	0.595041	0.640496	0.681818	0.702479	0.710744	0.702479	...	0.157025	0.136364
2	0.318182	0.400826	0.491736	0.528926	0.586777	0.657025	0.681818	0.685950	0.702479	0.698347	...	0.132231	0.181818
3	0.198347	0.194215	0.194215	0.194215	0.190083	0.190083	0.243802	0.404959	0.483471	0.516529	...	0.636364	0.657025
4	0.500000	0.545455	0.582645	0.623967	0.648760	0.690083	0.694215	0.714876	0.723140	0.731405	...	0.161157	0.177686
5	0.549587	0.545455	0.541322	0.537190	0.537190	0.533058	0.528926	0.533058	0.590909	0.611570	...	0.619835	0.623967
6	0.330578	0.305785	0.330578	0.351240	0.425620	0.500000	0.603306	0.632231	0.644628	0.644628	...	0.541322	0.541322
7	0.128099	0.185950	0.247934	0.314050	0.388430	0.462810	0.520661	0.557851	0.590909	0.623967	...	0.157025	0.165289
8	0.243802	0.297521	0.367769	0.454545	0.495868	0.537190	0.578512	0.603306	0.611570	0.632231	...	0.669422	0.537190
9	0.380165	0.442149	0.483471	0.545455	0.582645	0.628099	0.648760	0.677686	0.690083	0.710744	...	0.157025	0.165289

10 rows × 4097 columns

In [3]:
```
train_data = pd.concat([df.iloc[i*10+2:(i+1)*10] for i in range(40)])
train_data.reset_index(drop=True,inplace=True)

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

In [4]:
```
# remove target column
X = train_data.iloc[:,:-1]

threshold = 0.95

mean_ = np.mean(X, axis=0)

# Centering the data
X_meaned = X - mean_

# Calculating the covariance matrix of the mean-centered data
cov_mat = np.cov(X_meaned , rowvar = False)

# Calculating Eigenvalues and Eigenvectors of the covariance matrix
eigen_values , eigen_vectors = np.linalg.eigh(cov_mat)

# Sort the eigenvalues in descending order
sorted_index = np.argsort(eigen_values)[::-1]
sorted_eigenvalue = eigen_values[sorted_index].astype(np.float64)

# Similarly sort the eigenvectors
sorted_eigenvectors = eigen_vectors[:,sorted_index]
```

```
# calculate the percentage of explained variance per principal component
cumul_eigenvalue = sorted_eigenvalue.cumsum()
cumul_on_total = cumul_eigenvalue / cumul_eigenvalue[-1]

num_components = 0
while(cumul_on_total[num_components] < threshold):
    num_components += 1
num_components += 1

print('The number of features have been reduced from {} to {} for a threshold of {}% variance.'.form

eigenvector_subset = sorted_eigenvectors[:, 0:num_components]

# Transform the data
X_reduced = np.dot(eigenvector_subset.transpose(), X_meaned.transpose()).transpose()
```

```
The number of features have been reduced from 4096 to 111 for a threshold of 95.0% variance.
```

In [5]:

```
principal_df = pd.DataFrame(X_reduced , columns = ['col'+str(i) for i in range(1,num_components+1)])
principal_df = pd.concat([principal_df , train_data['target']] , axis = 1)
```

In [6]:

```
classifier = GaussianNB()
classifier.fit(principal_df[['col'+str(i) for i in range(1,num_components+1)]], principal_df["target
```

Out[6]:

```
GaussianNB()
```

In [7]:

```
test_reduced = (eigenvector_subset.T @ (test_data.iloc[:,:-1] - mean_).T).T
predicted = classifier.predict(test_reduced)
test_reduced = pd.concat([test_reduced, test_data['target']], axis = 1)
```

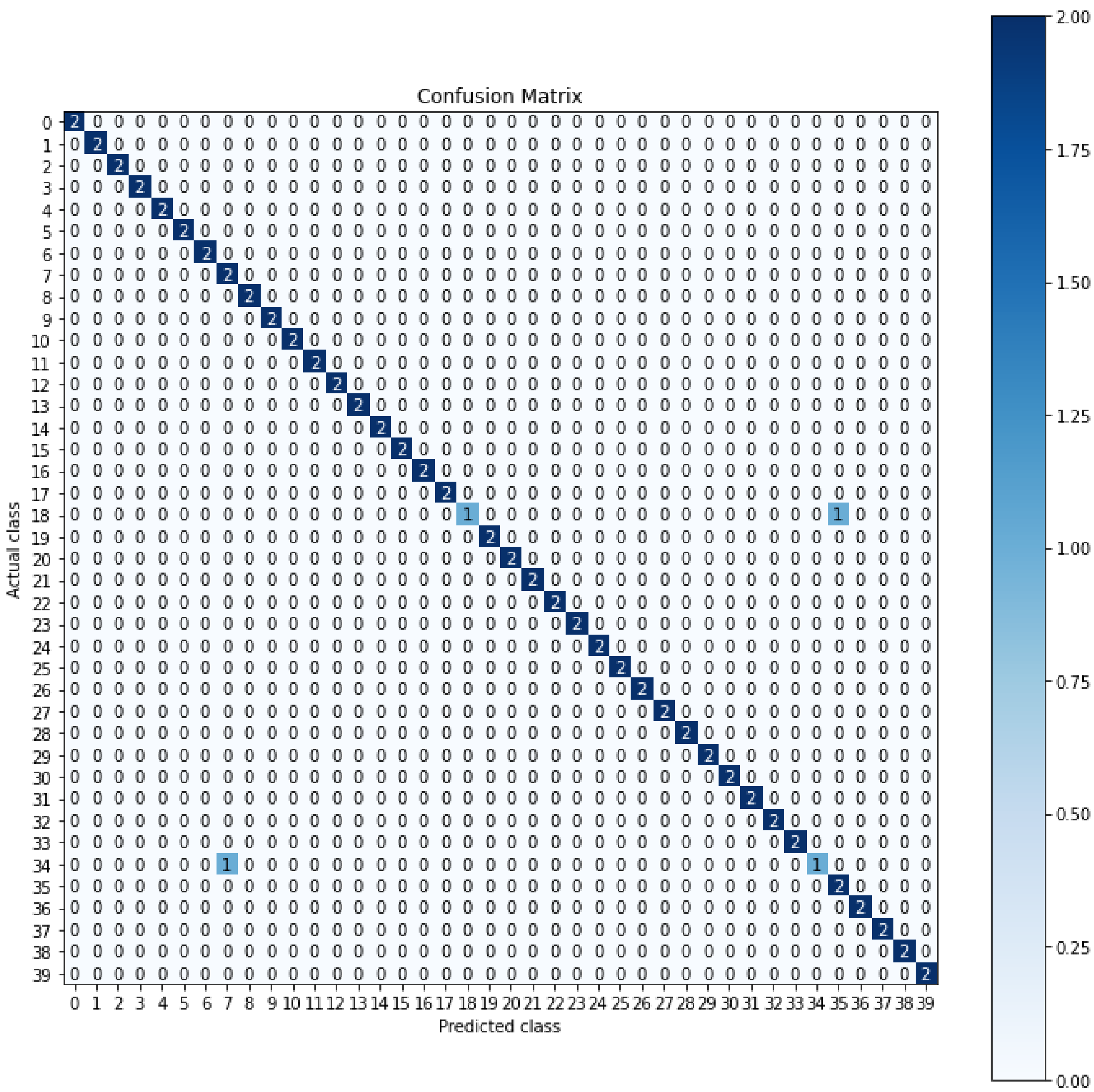
In [8]:

```
test_reduced['predicted'] = predicted
acc = []
for i in range(len(test_reduced)):
    if test_reduced['target'][i] == test_reduced['predicted'][i]:
        acc.append("correct")
    else:
        acc.append("wrong")

test_reduced["correctness"] = acc
x = accuracy_score(test_reduced["target"], predicted)
print("Accuracy =",x*100,"%")
```

```
Accuracy = 97.5 %
```

In [13]:

```
skplt.metrics.plot_confusion_matrix(test_reduced["target"], predicted, figsize=(12,12))
plt.xlabel('Predicted class')
plt.ylabel('Actual class')
plt.show()
```


Confusion Matrix
Actual class
Predicted class
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

Question 5

In [8]:

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
import scikitplot as skplt
```

In [24]:

```
data = pd.read_csv('face.csv')
print(data.shape)
data.head(401)
```

```
(400, 4097)
```

Out[24]:

	0	1	2	3	4	5	6	7	8	9	...	4087	408
0	0.309917	0.367769	0.417355	0.442149	0.528926	0.607438	0.657025	0.677686	0.690083	0.685950	...	0.669422	0.6528
1	0.454545	0.471074	0.512397	0.557851	0.595041	0.640496	0.681818	0.702479	0.710744	0.702479	...	0.157025	0.1363
2	0.318182	0.400826	0.491736	0.528926	0.586777	0.657025	0.681818	0.685950	0.702479	0.698347	...	0.132231	0.1818
3	0.198347	0.194215	0.194215	0.194215	0.190083	0.190083	0.243802	0.404959	0.483471	0.516529	...	0.636364	0.6570
4	0.500000	0.545455	0.582645	0.623967	0.648760	0.690083	0.694215	0.714876	0.723140	0.731405	...	0.161157	0.1776
...	...	...	...	...	...	...	...	...	...	...	...	...	
395	0.400826	0.495868	0.570248	0.632231	0.648760	0.640496	0.661157	0.636364	0.665289	0.698347	...	0.396694	0.2644
396	0.367769	0.367769	0.351240	0.301653	0.247934	0.247934	0.367769	0.512397	0.574380	0.628099	...	0.334711	0.2892
397	0.500000	0.533058	0.607438	0.628099	0.657025	0.632231	0.657025	0.669422	0.673554	0.702479	...	0.148760	0.1528
398	0.214876	0.219008	0.219008	0.223141	0.210744	0.202479	0.276859	0.400826	0.487603	0.549587	...	0.392562	0.3677
399	0.516529	0.462810	0.280992	0.252066	0.247934	0.367769	0.574380	0.615702	0.661157	0.615702	...	0.264463	0.2933

400 rows × 4097 columns

In [25]:

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

labels = list(train_data['target'])
X = np.array(train_data.iloc[:,:-1])

height,width = X.shape
classes = np.unique(labels)
```

In [26]:

```
c = len(classes)
d_= c-1
d = {}
for i in range(len(classes)):
    d[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[d[labels[i]]]=np.append(class_wise_data[d[labels[i]]], np.array([X[i],]),axis=0)

# calculating class wise means
means=[]
for i in class_wise_data:
```

```python
    means.append(np.mean(i,axis=0))

# calculating within class scatter matrix
Sw = np.zeros((width,width))
for i,data in enumerate(class_wise_data):
    z = data-means[i]
    Sw += (z.T @ z)

Sw_inv = np.linalg.inv(Sw)
```

In [27]:

```python
# calculating between class scatter matrix
overall_mean = np.mean(X,axis=0)
Sb = np.zeros((width,width))

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)
sorted_index = np.argsort(eigen_values)[::-1]
sorted_eigenvectors = eigen_vectors[:,sorted_index]

sorted_eigenvalue = eigen_values[sorted_index].astype(np.float64)

eigenvector_subset = sorted_eigenvectors[:,0:d_]
Y = X @ eigenvector_subset
```

In [28]:

```python
reduced = Y
reduced = pd.DataFrame(reduced)
```

In [29]:

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

Out[29]: GaussianNB()

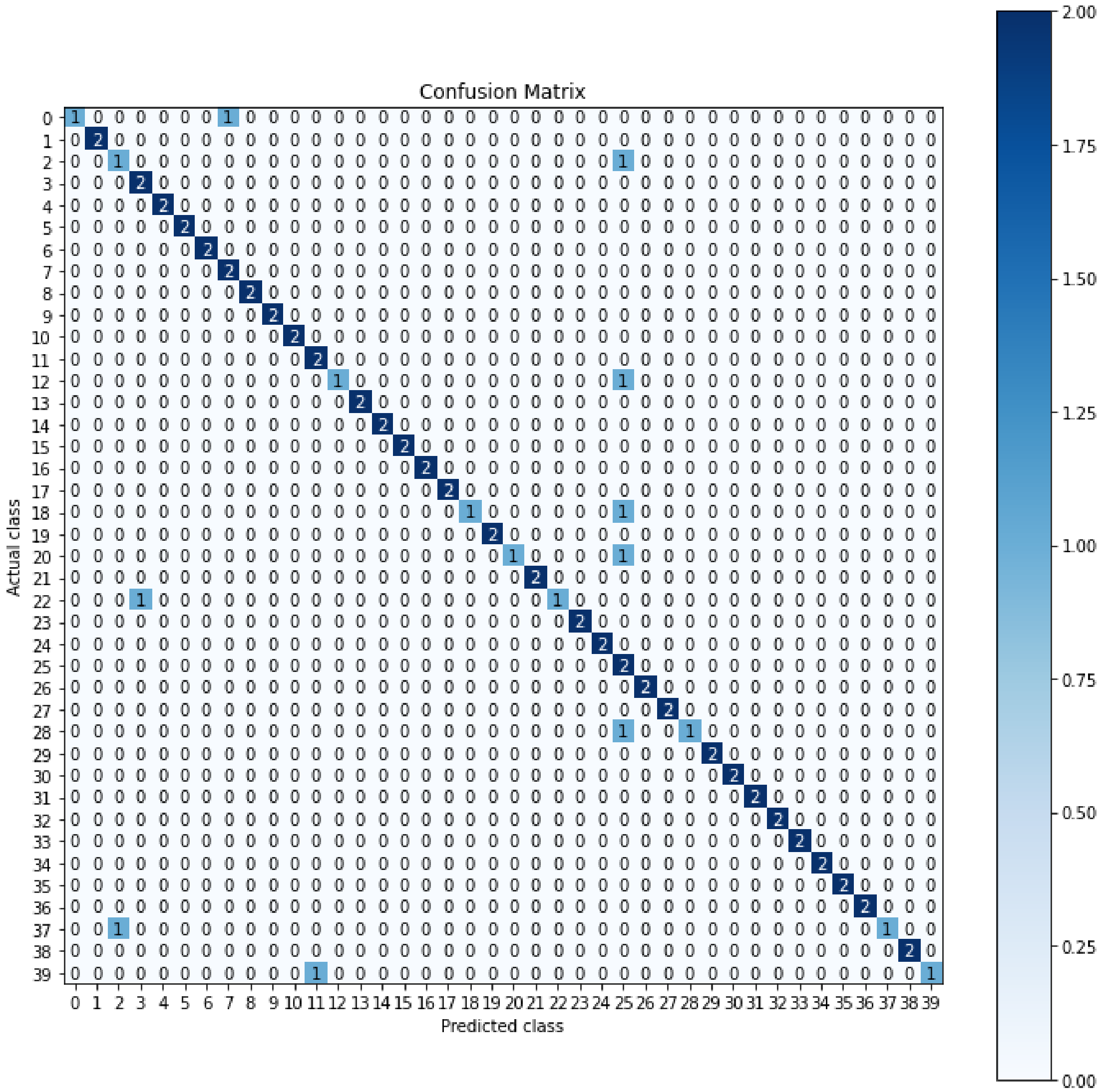
In [30]:

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

In [33]:

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

test_reduced["correctness"] = acc

x = accuracy_score(test_reduced["target"], predicted)
print("Accuracy =",x*100)

skplt.metrics.plot_confusion_matrix(test_reduced["target"], predicted, figsize=(12,12))
plt.xlabel('Predicted class')
plt.ylabel('Actual class')

plt.show()
```

Accuracy = 88.75


Confusion Matrix
Actual class
Predicted class
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

In [ ]: