Q5. Fisherfaces - Face classification using LDA (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 LDA.**

> **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**

> **https://towardsdatascience.com/eigenfaces-face-classification-in-python-7b8d2af3d3ea**

Importing the necessary libraries

```
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
```

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

Choose files face.csv

- **face.csv**(application/vnd.ms-excel) - 17088890 bytes, last modified: 25/03/2021 - 100% done

```
Saving face.csv to face (1).csv
```

Importing the dataset

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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0.309917	0.367769	0.417355	0.442149	0.528926	0.607438	0.657025	0.677686	0.690083	0.685950	0.702479	0.698347	0.694215	0.698347	0.690083	0.694215	0.690083	0.698347	0.702479	0.702479	0.698347
1	0.454545	0.471074	0.512397	0.557851	0.595041	0.640496	0.681818	0.702479	0.710744	0.702479	0.710744	0.698347	0.702479	0.706612	0.706612	0.714876	0.714876	0.706612	0.698347	0.714876	0.698347
2	0.318182	0.400826	0.491736	0.528926	0.586777	0.657025	0.681818	0.685950	0.702479	0.698347	0.702479	0.706612	0.706612	0.714876	0.710744	0.706612	0.706612	0.706612	0.723140	0.719008	0.719008
3	0.198347	0.194215	0.194215	0.194215	0.190083	0.190083	0.243802	0.404959	0.483471	0.516529	0.537190	0.566116	0.574380	0.586777	0.611570	0.632231	0.640496	0.657025	0.673554	0.694215	0.702479
4	0.500000	0.545455	0.582645	0.623967	0.648760	0.690083	0.694215	0.714876	0.723140	0.731405	0.739669	0.739669	0.764463	0.756198	0.764463	0.785124	0.793388	0.797521	0.814050	0.809917	0.809917

5 rows × 4097 columns

```
df.shape
```

```
(400, 4097)
```

LDA function

```
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)
    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
```

Preprocessing

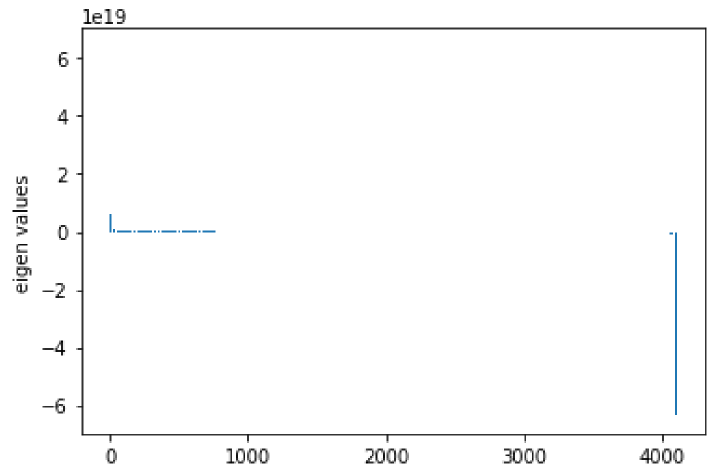
```
X = df.iloc[:,:-1]
target = df.iloc[:,-1]
```

Test and Train Split

```
train_data = pd.concat([df.iloc[i*10+2:(i+1)*10] for i in range(40)])
test_data = pd.concat([df.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)
```

Eigen Vectors

```
reduced, eigen_vec_subset = LDA(np.array(train_data.iloc[:,:-1]),list(train_data['target']))
reduced = pd.DataFrame(reduced)
```




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

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

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

```
            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

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

```
Accuracy = 87.5%
```

0s completed at 11:11 AM