15/11/2021 **ML LAB 11** 2019103573

Implement PCA and LDA. Compare the performance wrt to time and accuracy of MLP and K-means clustering before and after applying dimensionality reduction on any dataset of your choice from UCI repository.

PRNCIPAL COMPONENTS ANALYSIS (PCA)

In [2]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
%matplotlib inline
from sklearn.datasets import load_breast_cancer
```

In [3]:
```
cancer=load_breast_cancer()
X=cancer.data
y=cancer.target
df=pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
df.head()
```

Out[3]:

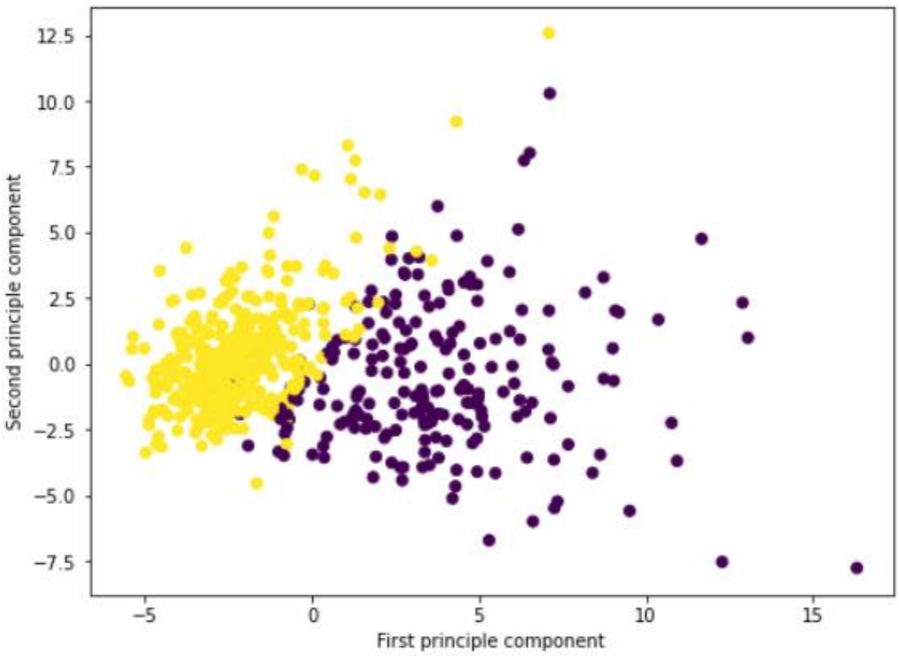
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.1622
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.1238
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.1444
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.2098
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.1374

5 rows × 30 columns

In [4]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(df)
scaled_data=scaler.transform(df)
```

In [5]:
```
from sklearn.decomposition import PCA
pca=PCA(n_components=2)
pca.fit(scaled_data)
x_pca=pca.transform(scaled_data)
#print(scaled_data.shape)
#print(x_pca.shape)
#print(scaled_data)
#print(x_pca)
plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0],x_pca[:,1],c=cancer['target'])
plt.xlabel('First principle component')
plt.ylabel('Second principle component')
```

Out[5]: `Text(0, 0.5, 'Second principle component')`



In [5]:
```
X_train_new, X_test_new, y_train, y_test = train_test_split(x_pca, y, test_size = 0.3,random_state=20, stratify=y)
knn_pca = KNeighborsClassifier(7)
knn_pca.fit(X_train_new,y_train)
print("Train Accuracy ",knn_pca.score(X_train_new,y_train)*100,"%")
print("Test Accuracy ",knn_pca.score(X_test_new,y_test) *100,"%")
```

```
Train Accuracy  95.7286432160804 %
Test Accuracy  92.98245614035088 %
```

In [11]:
```
from sklearn import metrics
knn_pca_y_pred = knn_pca.predict(X_test_new)
confusion = metrics.confusion_matrix(y_true = y_test, y_pred = knn_pca_y_pred)
print(confusion)
class_wise = metrics.classification_report(y_true=y_test, y_pred=knn_pca_y_pred)
print(class_wise)
```

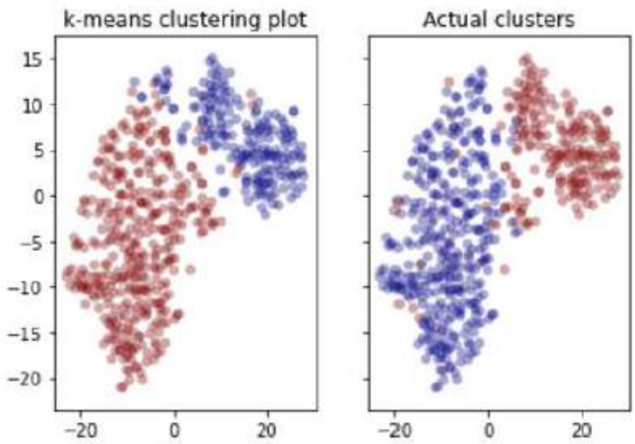
```
[[ 59   5]
 [  7 100]]
              precision    recall  f1-score   support

           0       0.89      0.92      0.91        64
           1       0.95      0.93      0.94       107

    accuracy                           0.93       171
   macro avg       0.92      0.93      0.93       171
weighted avg       0.93      0.93      0.93       171
```

MLP prediction 99.5%

K means Clustering plot of Breast cancer dataset:



So we get a clearer picture of the data clusters after dimentionality reduction clearly as it reduces the number of features in consideration

LINEAR DISCRIMINANT ANALYSIS (LDA)

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
#Load dataset
n_components = 2
data = pd.read_csv('/content/bmd.csv')
data = data[['age','weight_kg','height_cm','bmd','fracture']]
data
```

Out[1]:

	age	weight_kg	height_cm	bmd	fracture
0	57.052768	64.0	155.5	0.8793	no fracture
1	75.741225	78.0	162.0	0.7946	no fracture
2	70.778900	73.0	170.5	0.9067	no fracture
3	78.247175	60.0	148.0	0.7112	no fracture
4	54.191877	55.0	161.0	0.7909	no fracture
...	...	...	...	...	...
164	77.982543	74.0	164.0	0.7941	fracture
165	50.285303	59.0	161.0	0.7971	fracture
166	46.359721	67.0	169.0	0.8037	fracture
167	54.788368	70.0	166.0	0.8072	fracture
168	69.994822	68.5	165.0	0.8664	fracture

In [2]:
```python
#Normalizing the attributes and encoding labels
from sklearn.preprocessing import StandardScaler
stdsc = StandardScaler()
X_train_std = stdsc.fit_transform(data.iloc[:,range(0,4)].values)
from sklearn.preprocessing import LabelEncoder
class_le = LabelEncoder()
y = class_le.fit_transform(data['fracture'].values)
#Between class variance
S_W = np.zeros((4,4))
for i in range(2):
    S_W += np.cov(X_train_std[y==i].T)
S_W
```

Out[2]:
```
array([[ 1.9725685 , -0.04759746, -0.42736814, -0.30524193],
       [-0.04759746,  1.78794962,  0.71452081,  0.59460869],
       [-0.42736814,  0.71452081,  2.03028906,  0.56144434],
       [-0.30524193,  0.59460869,  0.56144434,  1.0964439 ]])
```

In [3]:
```python
#Distance between mean and sample of class
N=np.bincount(y)
vecs=[]
[vecs.append(np.mean(X_train_std[y==i],axis=0)) for i in range(2)]
mean_overall = np.mean(X_train_std, axis=0)
S_B=np.zeros((4,4))
for i in range(2):
    S_B += N[i]*(((vecs[i]-mean_overall).reshape(4,1)).dot(((vecs[i]-mean_overall).reshape(1,4))))
S_B #Display eigen values
```

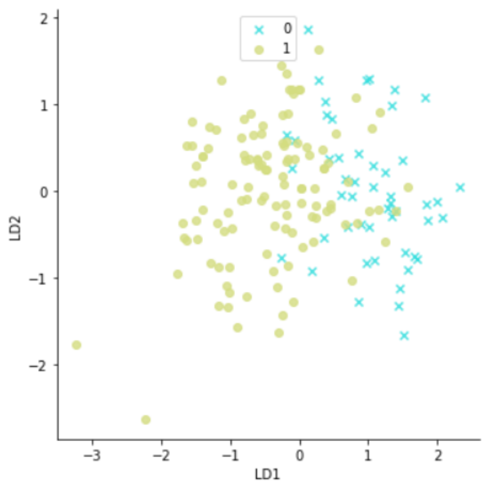
Out[3]:
```
array([[ 17.63488395, -19.24443281,  -4.31743302, -34.05701777],
       [-19.24443281,  21.00088638,   4.71148831,  37.16542689],
       [ -4.31743302,   4.71148831,   1.05700882,   8.33795637],
       [-34.05701777,  37.16542689,   8.33795637,  65.77193605]])
```

In [4]:
```python
eigen_vals, eigen_vecs = np.linalg.eig(np.linalg.inv(S_W).dot(S_B))
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:,i]) for i in range(len(eigen_vals))]
eigen_pairs = sorted(eigen_pairs,key=lambda k: k[0], reverse=True)
print('Eigenvalues in decreasing order:\n')
for eigen_val in eigen_pairs:
    print(eigen_val[0])
```

```
Eigenvalues in decreasing order:

69.77622956845525
2.9078998724795312e-15
2.9078998724795312e-15
0.0
```

In [6]:
```python
#Visualizing the data after LDA
import seaborn as sns
markers = ['x','o']
sns.lmplot(x="LD1", y="LD2", data=data, markers=markers,fit_reg=False, hue='class',legend=False, palette='rainbow')
plt.legend(loc='upper center')
plt.show()
```



In [5]:
```python
#Finding LD1 & LD2
tot = sum(eigen_vals.real)
discr = [(i / tot) for i in sorted(eigen_vals.real, reverse=True)]
cum_discr = np.cumsum(discr)
W=np.hstack((eigen_pairs[0][1][:,].reshape(4,1),eigen_pairs[1][1][:, ].reshape(4,1))).real
X_train_lda = X_train_std.dot(W)
#Adding LD1 & LD2 Value to dataframe
data=pd.DataFrame(X_train_lda)
data['class']=y
data.columns=["LD1","LD2","class"]
data.head()
```

Out[5]:

	LD1	LD2	class
0	-0.794078	0.361718	1
1	0.041572	-0.642138	1
2	-0.325557	-1.110186	1
3	0.233302	-0.170556	1
4	-0.061762	0.367884	1

In [9]:
```python
#KNN Classifier
X_train, X_test, y_train, y_test = train_test_split(data[['LD1','LD2']], data['class'],
test_size=0.20)
knn_lda= KNeighborsClassifier(n_neighbors=3)
knn_lda.fit(X_train, y_train)
y_pred = knn_lda.predict(X_test)
print("Train accuracy ",knn_lda.score(X_train,y_train)*100,"%")
print("Test accuracy ",knn_lda.score(X_test,y_test)*100,"%")
```

```
Train accuracy  90.37037037037037 %
Test accuracy  91.17647058823529 %
```

In [11]:
```python
from sklearn import metrics
knn_lda_y_pred = knn_lda.predict(X_test)
confusion = metrics.confusion_matrix(y_true = y_test, y_pred = knn_lda_y_pred)
print(confusion)
class_wise = metrics.classification_report(y_true=y_test, y_pred=knn_lda_y_pred)
print(class_wise)
```

```
[[11  2]
 [ 1 20]]
              precision    recall  f1-score   support

           0       0.92      0.85      0.88        13
           1       0.91      0.95      0.93        21

    accuracy                           0.91        34
   macro avg       0.91      0.90      0.91        34
weighted avg       0.91      0.91      0.91        34
```