Lab program no 10

Build k-Means algorithm to cluster a set of data stored in a .CSV file.

Program:

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np
# import some data to play with
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns =  ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']

# Build the K Means Model
model = KMeans(n_clusters=3)
model.fit(X) # model.labels_ : Gives cluster no for which samples belongs to
```

Iris Versicolor   Iris Setosa   Iris Virginica

Plot the results

```
# # Visualise the clustering results
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])
# Plot the Original Classifications using Petal features
plt.subplot(2, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
# Plot the Models Classifications
plt.subplot(2, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
```

Lab program no 11

Implement Dimensionality reduction using Principle Component Analysis (PCA) method.

Libraries

In [21]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
```

The Data

Let's work with the cancer data set again since it had so many features.

In [22]:
```
from sklearn.datasets import load_breast_cancer
```

In [23]:
```
cancer = load_breast_cancer()
```

In [24]:
```
cancer.keys()
```

Out[24]:
```
dict_keys(['DESCR', 'data', 'feature_names', 'target_names', 'target'])
```

In [25]:
```
print(cancer['DESCR'])
```

In [26]:
```
df = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
#(['DESCR', 'data', 'feature_names', 'target_names', 'target'])
```

In [27]:
```
df.head()
```

Out[27]:

	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	wors smoothnes
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	0.162
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	0.123
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	0.144
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	0.209
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	0.137

5 rows × 30 columns

PCA Visualization

As we've noticed before it is difficult to visualize high dimensional data, we can use PCA to find the first two principal components, and visualize the data in this new, two-dimensional space, with a single scatter-plot. Before we do this though, we'll need to scale our data so that each feature has a single unit variance.

In [30]:
```
from sklearn.preprocessing import StandardScaler
```

In [32]:
```
scaler = StandardScaler()
scaler.fit(df)
```

Out[32]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [33]:
```
scaled_data = scaler.transform(df)
```

PCA with Scikit Learn uses a very similar process to other preprocessing functions that come with SciKit Learn. We instantiate a PCA object, find the principal components using the fit method, then apply the rotation and dimensionality reduction by calling transform().

We can also specify how many components we want to keep when creating the PCA object.

In [34]:
```
from sklearn.decomposition import PCA
```

In [35]:
```
pca = PCA(n_components=2)
```

In [36]:
```
pca.fit(scaled_data)
```

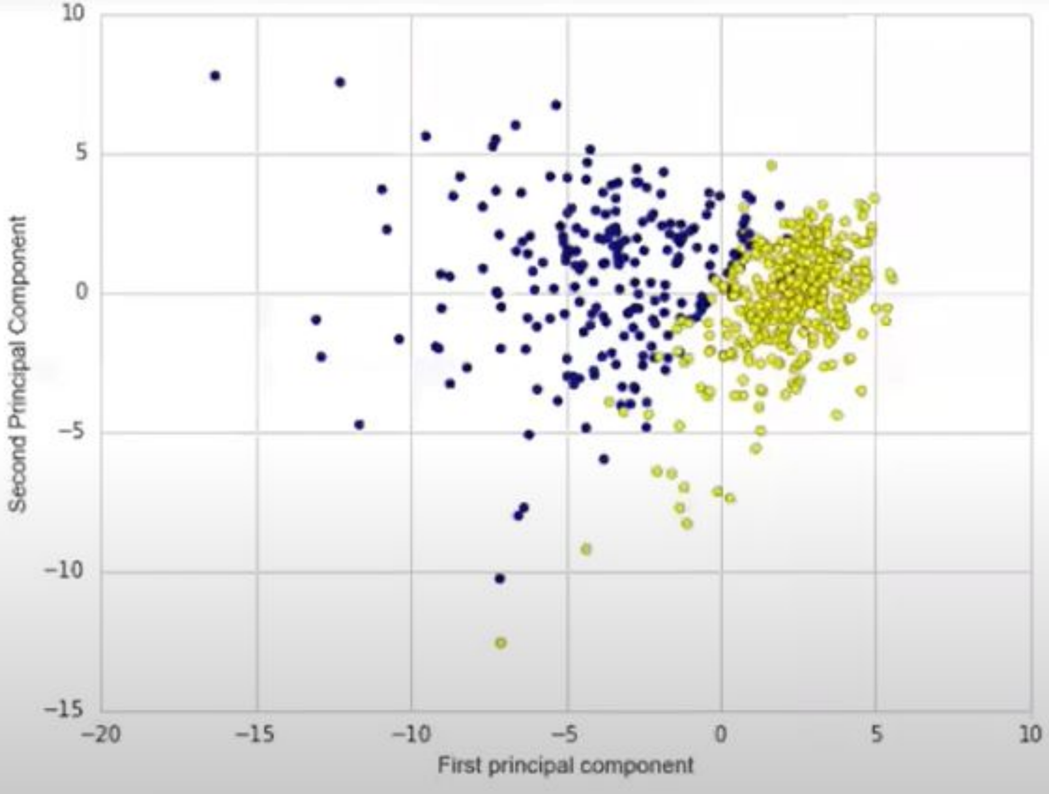
Out[36]:
```
PCA(copy=True, n_components=2, whiten=False)
```

Now we can transform this data to its first 2 principal components.

```
In [37]: x_pca = pca.transform(scaled_data)

In [38]: scaled_data.shape

Out[38]: (569, 30)

In [39]: x_pca.shape

Out[39]: (569, 2)
```

Great! We've reduced 30 dimensions to just 2! Let's plot these two dimensions out!

```
In [52]: plt.figure(figsize=(8,6))
         plt.scatter(x_pca[:,0],x_pca[:,1],c=cancer['target'],cmap='plasma')
         plt.xlabel('First principal component')
         plt.ylabel('Second Principal Component')

Out[52]: <matplotlib.text.Text at 0x11eb56908>
```



Clearly by using these two components we can easily separate these two classes.