Project By-Prince k Singh

Data Mining and Neural Networks

Breast cancer diagnosis – 2

Use Breast Cancer Wisconsin (Diagnostic) Data Set you have used in the first task and perform principal component analysis and k-means clustering.

1.Princple component analysis of data

first we import data set and know about data before PCA analysis Hence import required library

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
df = pd.read_csv("breast cancer wisconsin.csv")
df.head()
```

Out[1]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter_worst	area_worst
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	17.33	184.60	2019.0
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	23.41	158.80	1956.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	25.53	152.50	1709.0
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	26.50	98.87	567.7
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	16.67	152.20	1575.0

5 rows × 33 columns

As from data it is clear that it unsupervised kind of problem in which we have to diagnosis breast cancer which is MALAGNINE OR BENINE means cancerous or not cancerous so be separate this featurs from whole data set and also convert M AND B into 0 and 1 means numerical label with the help of label encoder

Now it's an unsupervised learning dataset

(Because we've removed the diagnosis label) - Use this version.

In [2]:
```
df_new = df.drop(['diagnosis','id','Unnamed: 32'], axis=1)
df_new_label=df['diagnosis']
df_new.head()
```

Out[2]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	...	radius_worst	texture_
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	

5 rows × 30 columns

Perform PCA on your dataset

.we have need to standardize your data before PCA.

In [3]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

In [4]:
```
#Standardize the data:
scaler = StandardScaler()
norm_data = scaler.fit_transform(df_new)
norm_data
```

Out[4]:
```
array([[ 1.09706398, -2.07333501,  1.26993369, ...,  2.29607613,
         2.75062224,  1.93701461],
       [ 1.82982061, -0.35363241,  1.68595471, ...,  1.0870843 ,
        -0.24388967,  0.28118999],
       [ 1.57988811,  0.45618695,  1.56650313, ...,  1.95500035,
         1.152255  ,  0.20139121],
       ...,
       [ 0.70228425,  2.0455738 ,  0.67267578, ...,  0.41406869,
        -1.10454895, -0.31840916],
       [ 1.83834103,  2.33645719,  1.98252415, ...,  2.28998549,
         1.91908301,  2.21963528],
       [-1.80840125,  1.22179204, -1.81438851, ..., -1.74506282,
        -0.04813821, -0.75120669]])
```

now we have to calculate Principle components and corresponding eigenvalue of correalation matrix

In [5]:
```
#Create pca instance
# The parameter = 30 because I  want my all principle components as total features is 30
pca = PCA(30)

#Fit on data
df_pca=pca.fit(norm_data)

#access value and vectors:
eigenvectors= pca.components_
eigenvalues = pca.explained_variance_

#transform data:
#Right now we have the eigenvectors and values calculated
#for the values fed in, but we haven't actually generated
#principal component variables. Here is when we create them:

pca_vals = pca.transform(norm_data)
pca_vals
df_pca
```

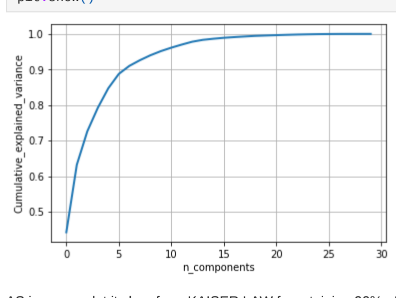
Out[5]: PCA(n_components=30)

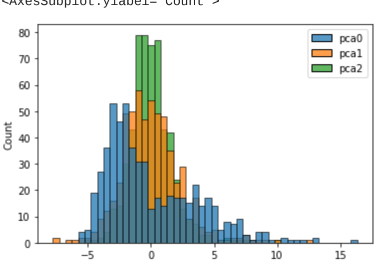
EIGENVALUE

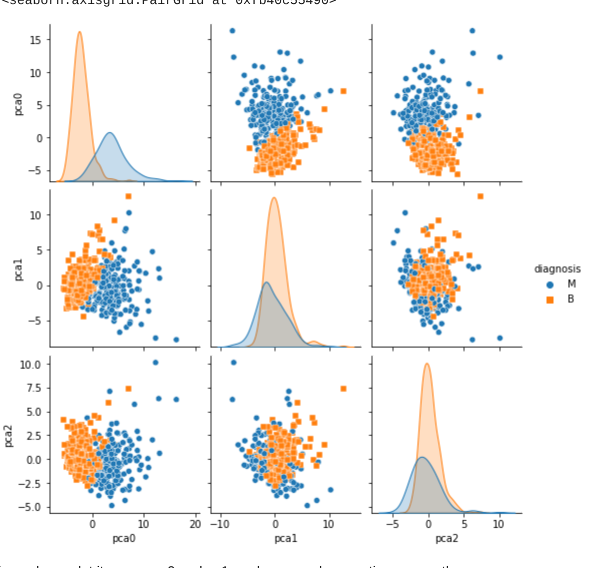
Hence i present all eigenvalues in the form of list

In [6]:
```
list(eigenvalues)
```

Out[6]:
```
[13.304990794374538,
 5.701374603726146,
 2.8229101550062246,
 1.9841275177302025,
 1.6516332423301185,
 1.2094822398029674,
 0.6764088817009056,
 0.4774562546895078,
 0.4176287821078163,
 0.35131087488173296,
 0.29443315349116456,
 0.26162116136612074,
 0.24178242132831318,
 0.15728614921759293,
 0.09430069560105578,
 0.08000340447737674,
 0.05950361353043181,
 0.05271142221014774,
 0.04956470021298151,
 0.03121426055306633,
 0.030025663090428315,
 0.027487711338904375,
 0.02438369135459111,
 0.01808679398430535,
 0.015508527134418559,
 0.008192037117607276,
 0.006912612579184491,
```







from above plot it seems pc0 and pc1 are have good seperation among them

coparison of one component predictors with one attribute predictors by plotting pair plots of general attribute

In [13]:
```
df_new2 = df.drop(['id','Unnamed: 32'], axis=1)
df_new2.head()
```

Out[13]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst	perimet
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	25.38	17.33	
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	24.99	23.41	
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	...	23.57	25.53	
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	...	14.91	26.50	
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	...	22.54	16.67	

5 rows × 31 columns

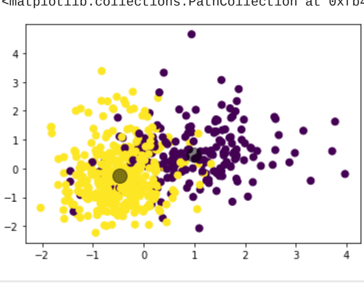
In [14]:
```
#sns.pairplot(df_new2, hue='diagnosis', markers=["o", "s"], corner=False)
```

3.K-means clustering

now performing K-MEANS clustering for k=2,3 and 5

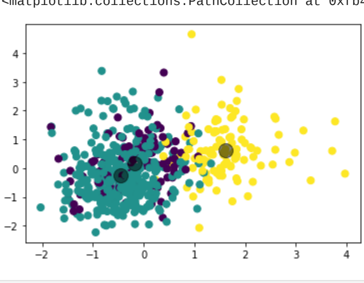
In [15]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2)
kmeans.fit(norm_data )
y_kmeans = kmeans.predict(norm_data )
plt.scatter(norm_data [:, 0], norm_data [:, 1], c=y_kmeans, s=50, cmap='viridis')
centers1 = kmeans.cluster_centers_
plt.scatter(centers1[:, 0], centers1[:, 1], c='black', s=200, alpha=0.5)
```

Out[15]: <matplotlib.collections.PathCollection at 0xfb410c3d30>



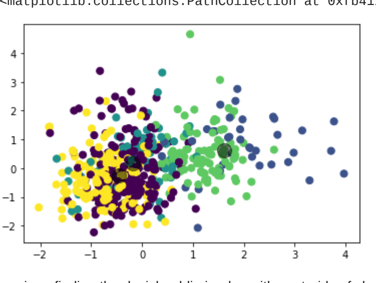
In [16]:
```
kmeans = KMeans(n_clusters=3)
kmeans.fit(norm_data )
y_kmeans = kmeans.predict(norm_data )
plt.scatter(norm_data [:, 0], norm_data [:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
```

Out[16]: <matplotlib.collections.PathCollection at 0xfb411333a0>



In [17]:
```
kmeans = KMeans(n_clusters=5)
kmeans.fit(norm_data )
y_kmeans = kmeans.predict(norm_data )
plt.scatter(norm_data [:, 0], norm_data [:, 1], c=y_kmeans, s=50, cmap='viridis')
centroids = kmeans.cluster_centers_
labels=kmeans.labels_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
```

Out[17]: <matplotlib.collections.PathCollection at 0xfb4118ab50>



now i am finding the davisbouldin inndex with centroids of clusters so first define a function of davisbouldin fith it parameter of satandarised data predicted label and centroids hence we find individual index and cenrroids table as well

now i plot first two principle components and plot it with diffrent colour i used facegrid plot for representation

In [18]:
```
pca = PCA(2)

#Fit on data
df_pca=pca.fit(norm_data)
pca_data = pca.fit_transform(norm_data)
pca_df = pd.DataFrame(pca_data, columns=['1st_principal', '2nd_principal'])
pca_df['diagnosis'] = df['diagnosis']
print('Shape of PCA dataset is {}'.format(pca_df.shape))
pca_df.head()
```

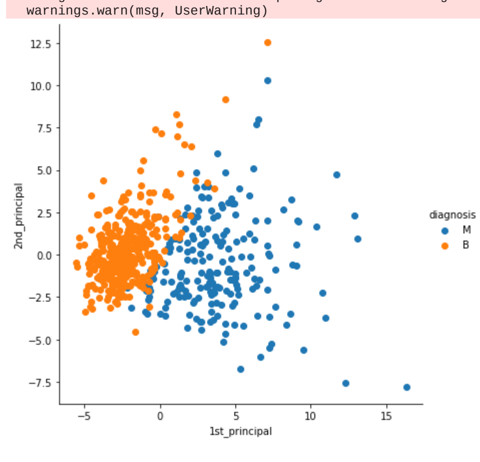
Shape of PCA dataset is (569, 3)

Out[18]:

	1st_principal	2nd_principal	diagnosis
0	9.192837	1.948583	M
1	2.387802	-3.768172	M
2	5.733896	-1.075174	M
3	7.122953	10.275589	M
4	3.935302	-1.948072	M

In [19]:
```
import seaborn as sns
sns.FacetGrid(pca_df, hue="diagnosis", size=6).map(plt.scatter, '1st_principal', '2nd_principal').add_legend()
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\axisgrid.py:316: UserWarning: The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



daviesbouldin index

we calculate daviesbouldin index and first define function and get out put of davies index list and distance mean

In [22]:
```
 def daviesbouldin(X, labels, centroids):
    from scipy.spatial.distance import pdist, euclidean

    nbre_of_clusters = len(centroids) #Get the number of clusters
    distances = [[] for e in range(nbre_of_clusters)] #Store intra-cluster distances by cluster
    distances_means = [] #Store the mean of these distances
    DB_indexes = [] #Store Davies_Boulin index of each pair of cluster
    second_cluster_idx = [] #Store index of the second cluster of each pair
    first_cluster_idx = 0 #Set index of first cluster of each pair to 0

    # Step 1: Compute euclidean distances between each point of a cluster to their centroid
    for cluster in range(nbre_of_clusters):
        for point in range(X[labels == cluster].shape[0]):
            distances[cluster].append(euclidean(X[labels == cluster][point], centroids[cluster]))

    # Step 2: Compute the mean of these distances
    for e in distances:
        distances_means.append(np.mean(e))

    # Step 3: Compute euclidean distances between each pair of centroid
    ctrds_distance = pdist(centroids)

    # Tricky step 4: Compute Davies-Bouldin index of each pair of cluster
    for i, e in enumerate(e for start in range(1, nbre_of_clusters) for e in range(start, nbre_of_clusters)):
        second_cluster_idx.append(e)
        if second_cluster_idx[i-1] == nbre_of_clusters - 1:
            first_cluster_idx += 1
        DB_indexes.append((distances_means[first_cluster_idx] + distances_means[e]) / ctrds_distance[i])

    # Step 5: Compute the mean of all DB_indexes
    print("NO of DB_indexes selected randomly accordingly k-means:" ,DB_indexes)


    print("NO of distance mean selected randomly accordingly k-means:" , distances_means)
```

In [23]:
```
kmeans = KMeans(n_clusters=5)
kmeans.fit(norm_data )
y_kmeans = kmeans.predict(norm_data )
centroids = kmeans.cluster_centers_
labels=kmeans.labels_
daviesbouldin(norm_data,kmeans.labels_,kmeans.cluster_centers_)
```

```
NO of DB_indexes selected randomly accordingly k-means: [0.8271299460651033, 2.097197579392939, 1.1743153586291681, 1.4784865620042233, 0.7320604422841057, 1.5354351608420593, 1.3726490727257177, 1.064488835063282, 1.0190611003617975, 1.4874870729275882]
NO of distance mean selected randomly accordingly k-means: [3.3167818860954976, 5.478378131343587, 2.8670397471503164, 3.4492761656104274, 5.231531246891376]
```

In [24]:
```
centroids=kmeans.cluster_centers_
centroids
```

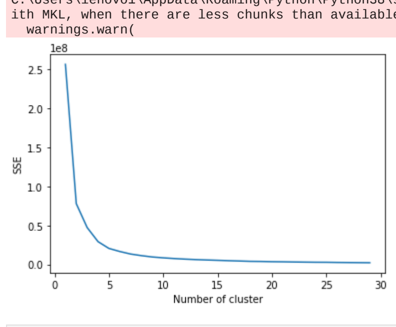
Out[24]:
```
array([[-7.12937056e-01, -3.41133570e-01, -6.96841381e-01,
        -6.63433755e-01,  3.75621627e-01, -1.56232478e-01,
        -4.18464074e-01, -4.35731124e-01,  9.39518151e-02,
         4.58260603e-01, -3.97749856e-01,  1.08097298e-01,
        -3.79150225e-01, -4.24342764e-01,  5.45377150e-01,
        -5.18416167e-02, -1.25964135e-01, -7.08211441e-02,
         1.73267523e-01,  1.01931089e-01, -7.03178268e-01,
        -3.60641120e-01, -6.84743232e-01, -6.40978285e-01,
         3.02143306e-01, -2.60687735e-01, -3.86591954e-01,
        -4.26995878e-01, -1.34684389e-01,  6.84106738e-02],
       [ 2.03323618e+00,  7.96379008e-01,  2.11978771e+00,
         2.21216632e+00,  8.90971322e-01,  1.97017715e+00,
         2.16770918e+00,  2.29850288e+00,  1.08730134e+00,
         4.40485733e-01,  2.43894161e+00,  2.60884175e-01,
         2.54672924e+00,  2.46878691e+00,  1.20372271e-01,
         1.22710599e+00,  9.71986922e-01,  1.27870725e+00,
         5.60544207e-01,  6.33740260e-01,  2.11722285e+00,
         5.89682639e-01,  2.22382288e+00,  2.26915289e+00,
         4.97077707e-01,  1.36061575e+00,  1.46204220e+00,
         1.80705543e+00,  6.58263504e-01,  6.37963010e-01],
       [-3.44617898e-01, -1.78245033e-01, -3.84605706e-01,
        -3.74797240e-01, -8.09801020e-01, -7.98611325e-01,
        -7.14177235e-01, -7.15387521e-01, -6.10586106e-01,
        -5.89872819e-01, -4.65400893e-01, -1.15354650e-01,
        -4.78978777e-01, -3.98962847e-01, -4.62132919e-01,
        -6.11985915e-01, -5.03440866e-01, -6.49484125e-01,
        -2.71256078e-01, -5.16659857e-01, -4.11906641e-01,
        -1.93183887e-01, -4.46775241e-01, -4.26008477e-01,
        -7.81993741e-01, -6.64055724e-01, -6.68970950e-01,
        -7.14645387e-01, -4.50772071e-01, -6.42297377e-01],
       [ 1.15349171e+00,  4.66614847e-01,  1.12548043e+00,
         1.09784923e+00,  1.76597845e-01,  3.63828298e-01,
         6.57646267e-01,  9.08731862e-01,  1.15191580e-01,
        -5.72212726e-01,  6.67052222e-01, -9.37205035e-02,
         5.82397545e-01,  6.38593316e-01, -2.29502275e-01,
         4.25341977e-02,  1.20424452e-01,  4.04658744e-01,
        -1.98253107e-01, -1.74387464e-01,  1.18426959e+00,
         4.96910672e-01,  1.13239898e+00,  1.10281971e+00,
         2.99386332e-01,  3.68649288e-01,  5.79953645e-01,
         9.12583180e-01,  2.78409277e-01, -8.87785578e-02],
       [-2.08926332e-01,  2.16659525e-01, -1.12977573e-01,
        -2.48337626e-01,  9.97228689e-01,  1.38428236e+00,
         1.13591527e+00,  6.45294077e-01,  1.03851208e+00,
         1.63407862e+00, -7.54071390e-03,  1.24195006e-01,
         7.63504902e-02, -1.41073668e-01,  5.12832420e-01,
         1.48683322e+00,  1.31929190e+00,  9.79254295e-01,
         5.07201340e-01,  1.49675084e+00, -1.03034301e-01,
         4.00458305e-01, -1.12997915e-03, -1.70797024e-01,
         1.12532896e+00,  1.56581705e+00,  1.47780003e+00,
         9.25144181e-01,  1.06481117e+00,  1.88221983e+00]])
```

4. Clustering and Classification

now to find purity which is kind of accuracy in kmeans 1st of all plot sum of squared error with and it seems after 15 cluster the the rate of decrease of error becomes constant hence we take no of cluster 15 and then find purity as it is random hence each time purity can be cange as well

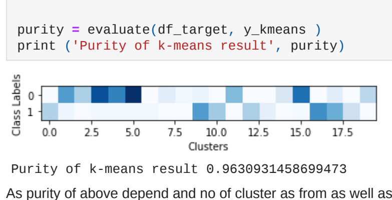
In [25]:
```
sse = {}
for k in range(1, 30):
    kmeans = KMeans(n_clusters=k, max_iter=1000).fit(df_new)
    y_kmeans = kmeans.labels_
    #print(data["clusters"])
    sse[k] = kmeans.inertia_ # Inertia: Sum of distances of samples to their closest cluster center
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.ylabel("SSE")
plt.show()
```

```
C:\Users\lenovo1\AppData\Roaming\Python\Python38\site-packages\sklearn\cluster\_kmeans.py:881: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=3.
  warnings.warn(
```



In [38]:
```
kmeans = KMeans(n_clusters=20)
kmeans.fit(norm_data )
y_kmeans = kmeans.predict(norm_data )
def evaluate(y_true, y_clusters):
    cm = np.zeros((np.max(y_true)+1, np.max(y_clusters)+1))
    for i in range(y_true.size):
        cm[y_true[i], y_clusters[i]]+=1
    plt.imshow(cm, interpolation='none', cmap=plt.cm.Blues)
    plt.ylabel('Class Labels')
    plt.xlabel('Clusters')
    plt.show()
    purity = 0.
    for cluster in range(cm.shape[1]):  # clusters are along columns
        purity += np.max(cm[:, cluster])
    return purity/y_true.size


purity = evaluate(df_target, y_kmeans )
print ('Purity of k-means result', purity)
```




```
Purity of k-means result 0.9630931458699473
```

As purity of above depend and no of cluster as from as well as it is also change with each run and because of randomnes in clustering process it shold be ideally 1 but in real practice it is closer to 1 because

As sum of square error (SSE) is in above graph ideally should we 0 or touches 0 but in mathematically it takes infinite iteartion so we loss of information from target attribute over categorical attribute or in other word we ralativaly gain information from categorical to target

relative information gain

information gain= information(target) - information(categorical)

relative informatiopn gain = log(purity value target) - log(purity value categorical)

here in above case purity value target =1, purity value categorical = 0.963

In [41]:
```
gain= np.log2(1)-np.log2(0.963)
```

In [43]:
```
gain*100
```

Out[43]: 5.439229681862779