L13 fork from PCA - IRIS

February 17, 2023

[ ]:
```
from sklearn import datasets
#we use iris data
iris = datasets.load_iris()
iris_X=iris.data
```

[ ]:
```
len(iris_X[0]) #first let us see the dimension of the feature
```

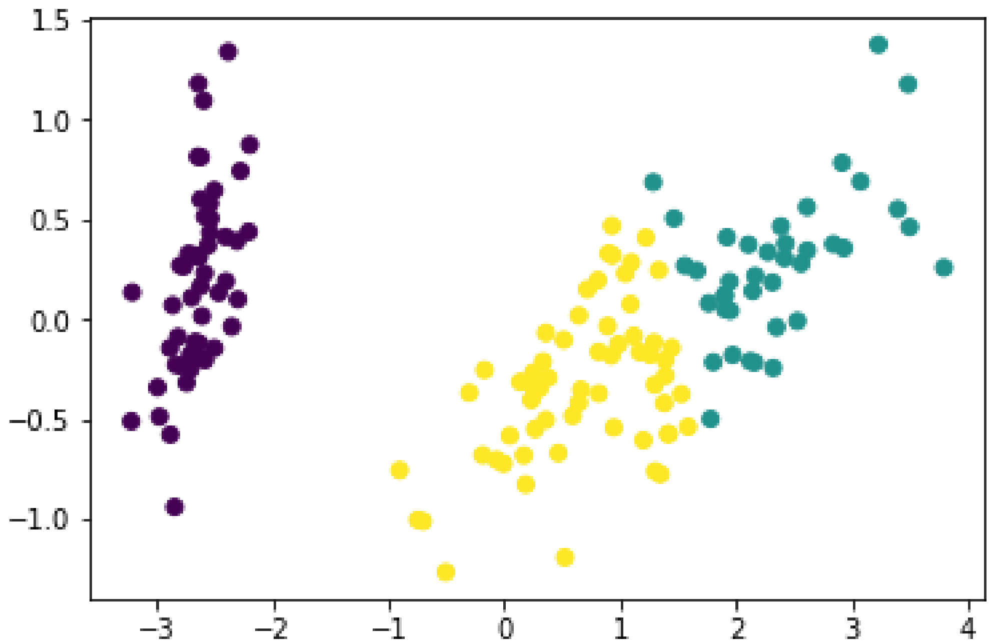
[ ]:
```
4
```

[ ]:
```
from sklearn.decomposition import PCA
#let us use PCA to compress the data to 2 dimensions
pca_2 = PCA(n_components=2) #define the PCA model
pca_iris=pca_2.fit_transform(iris_X) #use the PCA model to fit and transform␣
↪the original data to 2-dimension data
```

[ ]:
```
from sklearn.cluster import KMeans
#we use clustering to cluster the transformed data
kmeans = KMeans(n_clusters=3, random_state=0) #define the cluster model
cluster_pca=kmeans.fit(pca_iris) #fit the model with transformed data
labels_pca=kmeans.labels_ #generate the label
```

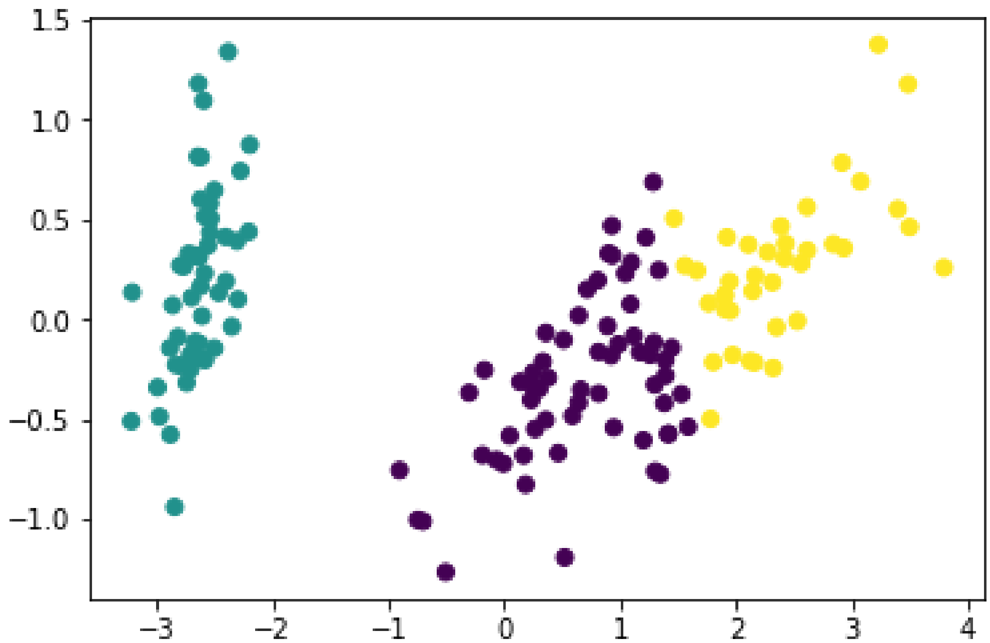
[ ]:
```
import matplotlib.pyplot as plt
%matplotlib inline

#let use plot the clustered data
plt.scatter(pca_iris[:, 0], pca_iris[:, 1], c=labels_pca)
```

[ ]:
```
<matplotlib.collections.PathCollection at 0x1a259590cd0>
```



[ ]:
```
#we want to cluster the original data
cluster_pca=kmeans.fit(iris.data)
labels_ori=kmeans.labels_
plt.scatter(pca_iris[:, 0], pca_iris[:, 1], c=labels_ori)
#very similar to the result with PCA
```

[ ]: <matplotlib.collections.PathCollection at 0x1a259693970>



[ ]:
```python
#check the label
print(labels_pca)
print(labels_ori)
print(iris.target)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1 1 2 1 1 1 1
 1 1 2 2 1 1 1 1 2 1 2 1 2 1 1 2 2 1 1 1 1 1 2 1 1 1 1 2 1 1 1 2 1 1 1 2 1
 1 2]
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

[ ]:
```python
#reorganize the label
import numpy as np

clu_pca_0=np.where(labels_pca==0)
```

```python
clu_ori_0=np.where(labels_ori==1)
clu_pca_1=np.where(labels_pca==2)
clu_ori_1=np.where(labels_ori==0)
clu_pca_2=np.where(labels_pca==1)
clu_ori_2=np.where(labels_ori==2)
```

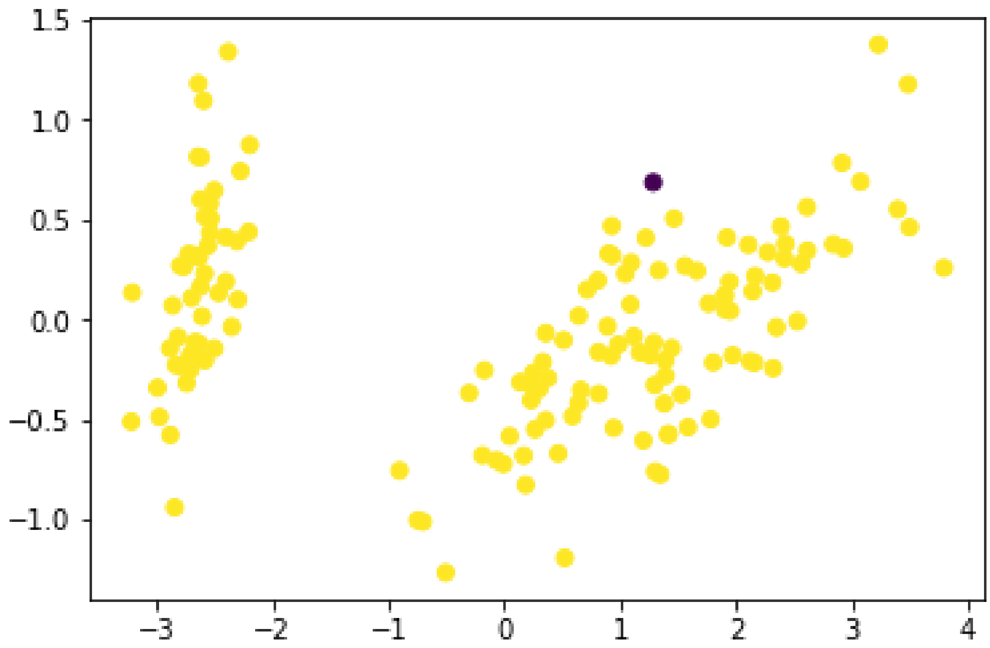
[ ]:
```python
labels_pca[clu_pca_0]=0
labels_pca[clu_pca_1]=1
labels_pca[clu_pca_2]=2
labels_ori[clu_ori_0]=0
labels_ori[clu_ori_1]=1
labels_ori[clu_ori_2]=2
```

[ ]:
```python
print(labels_pca)
print(labels_ori)
print(iris.target)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

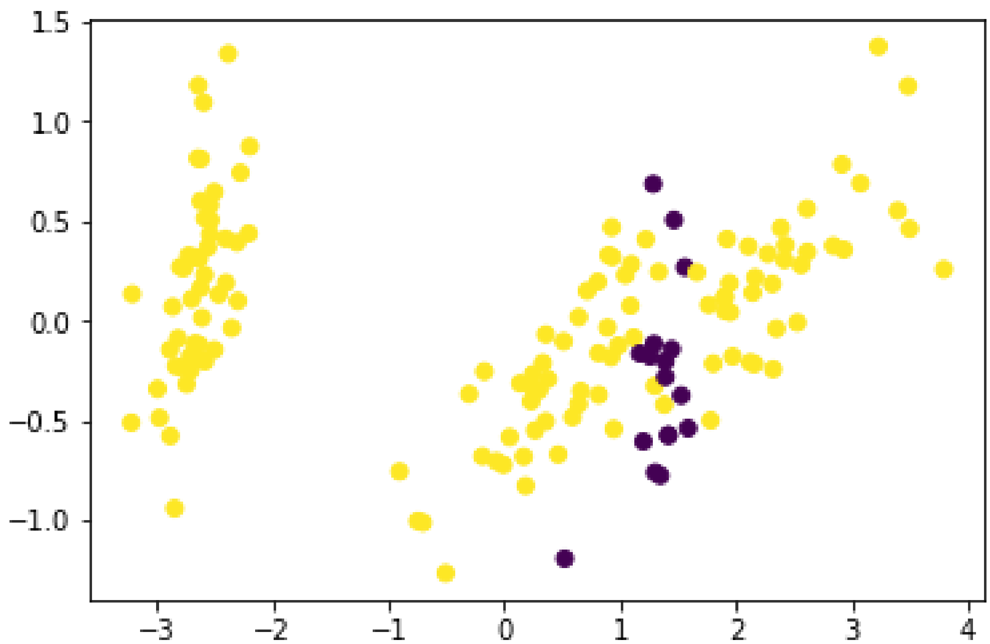
[ ]:
```python
same_diff=(labels_ori==labels_pca) #let us check which data points have␣
↪different label with/without PCA
plt.scatter(pca_iris[:, 0], pca_iris[:, 1], c=same_diff)
#only one data point
```

[ ]: <matplotlib.collections.PathCollection at 0x1a259704eb0>



[ ]:
```
#let us see how many data points are clustered correctly after PCA
pca_true=(iris.target==labels_pca)
plt.scatter(pca_iris[:, 0], pca_iris[:, 1], c=pca_true)
```

[ ]: <matplotlib.collections.PathCollection at 0x1a25976fa60>



[ ]:
```python
#new part
#obtain the eigenvalues and eigenvectors
#let us conduct PCA without dimension reduction
pca_4 = PCA(n_components=4)
pca_4.fit(iris_X)
```

[ ]: PCA(n_components=4)

[ ]:
```python
#get the covariance matrix, the eigenvalues and eigenvector
cov_M=pca_4.get_covariance()
e_M=pca_4.components_
lambda_V=pca_4.explained_variance_
```

[ ]:
```python
e_M
```

[ ]:
```
array([[ 0.36138659, -0.08452251,  0.85667061,  0.3582892 ],
       [ 0.65658877,  0.73016143, -0.17337266, -0.07548102],
       [-0.58202985,  0.59791083,  0.07623608,  0.54583143],
       [-0.31548719,  0.3197231 ,  0.47983899, -0.75365743]])
```

[ ]:
```python
lambda_V
```

[ ]:
```
array([4.22824171, 0.24267075, 0.0782095 , 0.02383509])
```

[ ]:
```python
#let us check Q e_i= lambda_i e_i
print(np.dot(cov_M,e_M[0,:]))
print(np.dot(lambda_V[0],e_M[0,:]))
```

```
[ 1.52802986 -0.35738162  3.62221038  1.51493333]
[ 1.52802986 -0.35738162  3.62221038  1.51493333]
```

[ ]:
```python
#calculate the covariance matrix by ourself
mat_X=np.mat(iris_X-np.mean(iris_X,0))
Q=np.dot(mat_X.T,mat_X) #here we do not devide it by N-1, then we will see it␣
↪will not influence the results in general
```

[ ]:
```python
eigval, eigvec = np.linalg.eig(Q) #solve the eigenvalues and eigenvectors
```

[ ]:
```python
eigvec #same as pca_4.components_.T
```

[ ]:
```
matrix([[ 0.36138659, -0.65658877, -0.58202985,  0.31548719],
        [-0.08452251, -0.73016143,  0.59791083, -0.3197231 ],
        [ 0.85667061,  0.17337266,  0.07623608, -0.47983899],
        [ 0.3582892 ,  0.07548102,  0.54583143,  0.75365743]])
```

[ ]:
```python
eigval #different from pca_4.components_
```

[ ]: array([630.0080142 , 36.15794144, 11.65321551,   3.55142885])

[ ]:
```
eigval/149 #devided by N-1, it becomes same as pca_4.components_
#Whether to devide by N-1 will not influence the results
```

[ ]: array([4.22824171, 0.24267075, 0.0782095 , 0.02383509])

[ ]:
```
print(np.dot(Q,eigvec[:,0])) #Qe_1
print(eigval[0]*eigvec[:,0]) #lambda_1e_1
```

```
[[227.67644905]
 [-53.24986124]
 [539.70934728]
 [225.72506561]]
[[227.67644905]
 [-53.24986124]
 [539.70934728]
 [225.72506561]]
```

[ ]: