Principal Components Analysis

Tutorial 4 Yang

Objectives

- Understand the principles of principal components analysis (PCA)
- Know the principal components analysis method
- Study the PCA function of sickit-learn.decomposition
- Process the data set by the PCA of sickit-learn
- Learn to apply PCA in a reality example

Principal Components Analysis

- Method:
  - Subtract the mean
  - Calculate the covariance matrix
  - Calculate the eigenvectors and eigenvalues of the covariance matrix
  - Choosing components and forming a feature vector
  - Deriving the new data set

Example 1

- $Data =$

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

sklearn.decomposition.PCA

- It uses the LAPACK implementation of the full SVD or a randomized truncated SVD by the method of Halko et al. 2009, depending on the shape of the input data and the number of components to extract.
- It can also use the scipy.sparse.linalg ARPACK implementation of the truncated SVD.
- Notice that this class does not upport sparse input.

Parameters of PCA

```
class sklearn.decomposition. PCA (n_components=None, copy=True, whiten=False, svd_solver='auto', tol=0.0,
iterated_power='auto', random_state=None)
```
[source]

n_components: Number of components to keep.
if n_components is not set: n_components == min (n_samples, n_features), default=None

if n_components == 'mle' and svd_solver == 'full', Minka's MLE is used to guess the dimension
if 0 < n_components < 1 and svd_solver == 'full', select the number of components such that the amount of variance that needs to be explained is greater than the percentage specified by n_components;n_components cannot be equal to n_features for svd_solver == 'arpack'.

svd_solver:
auto: default, if the input data is larger than 500x500 and the number of components to extract is lower than 80% of the smallest dimension of the data, then the more efficient 'randomized' method is enabled. Otherwise the exact full SVD is computed and optionally truncated afterwards.
full: run exact full SVD calling the standard LAPACK solver via scipy.linalg.svd and select the components by postprocessing.
arpack: run SVD truncated to n_components calling ARPACK solver via scipy.sparse.linalg.svds. It requires strictly 0 <n_components< X.shape[1]
randomized: run randomized SVD by the method of Halko et al.

http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

How to use PCA

```
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
s=np.array([[2.5,2.4], [0.5,0.7], [2.2,2.9],
[1.9,2.2], [3.1,3.0], [2.3,2.7], [2, 1.6], [1, 1.1],
[1.5, 1.6], [1.1, 0.9]])
pca = PCA(n_components=1)
s1=pca.fit_transform(s)

print (s1)
print (pca.components_) eigenvectors
print (pca.explained_variance_) eigenvalues
print (pca.explained_variance_ratio_)
```

```
import math as m
sv=0
for i in range(len(s1)):      length of the vector
   sv=sv+s1[i]**2
print(m.sqrt(sv))
```

```
print (pca.singular_values_)
print (pca.mean_)
print (pca.n_components_)
print (pca.noise_variance_)
```

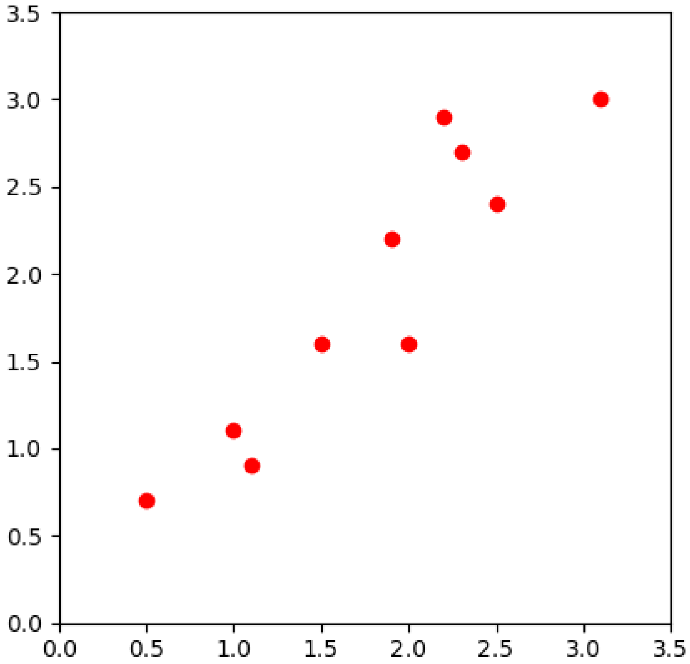
average of (min(n_features, n_samples) - n_components) smallest eigenvalues =0.04908383/(2-1)

$$\lambda = [1.28402771\ 0.04908383]$$

$$[0.96318131\ \ 0.03681869]$$

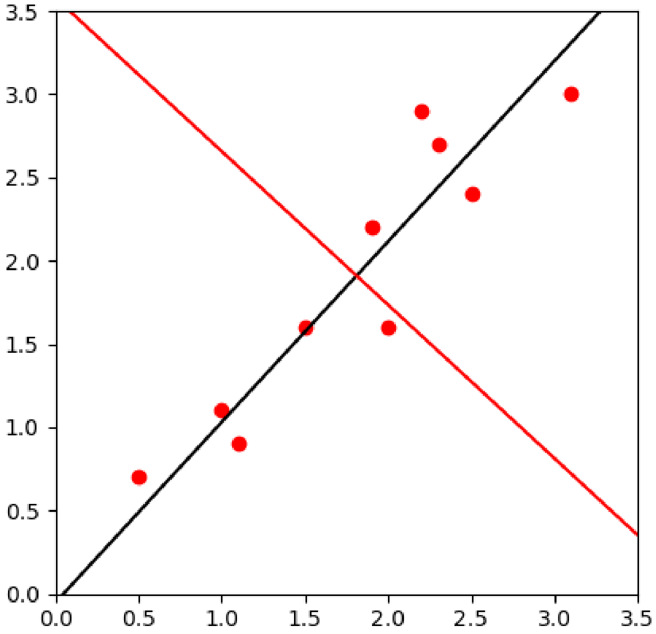
How to choose PC

```
pca = PCA(n_components=1)
s1=pca.fit_transform(s)
print (s1)

plt.xlim(0,3.5)
plt.ylim(0,3.5)
plt.gca().set_aspect('equal', adjustable='box')
plt.plot(s[:,0],s[:,1],'ro')
```



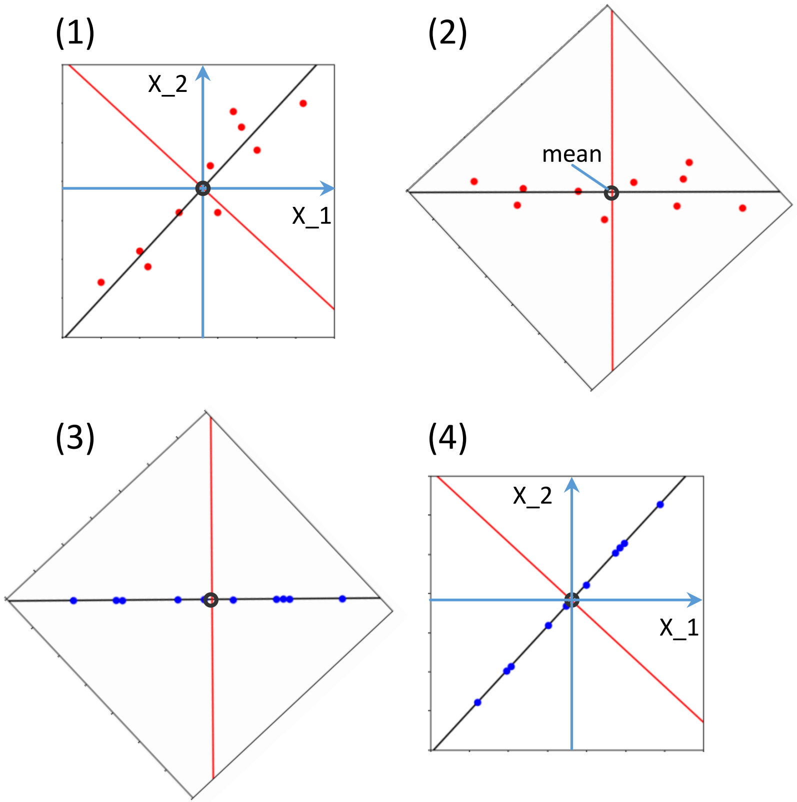
How to choose important PC

```
pca1 = PCA(n_components=2)
pca1.fit(s)

x=np.linspace(0,3.5)
y=pca1.components_[1][0]/pca1.components_[0][0]
*x+pca1.mean_[1]-pca1.components_[1][0]/
pca1.components_[0][0]*pca1.mean_[0]
plt.plot(x, y, 'k-')

y=pca1.components_[1][1]/pca1.components_[0][1]
*x+pca1.mean_[1]-pca1.components_[1][1]/
pca1.components_[0][1]*pca1.mean_[0]
plt.plot(x, y,'r-')
```



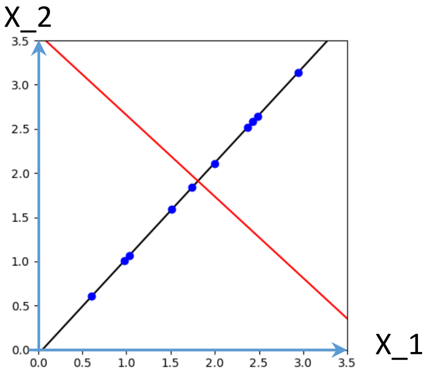
Exercise 1: Projection of PC

```
 #step 1
s2=np.zeros([10,2])
for i in range(len(s)):
    s2[i]=s[i]-pca1.mean_

#step 2
c=np.dot(s2, pca1.components_)

#step 3
c[:, 1] = 0

#step 4
c1=np.dot(c, pca1.components_.transpose())
```

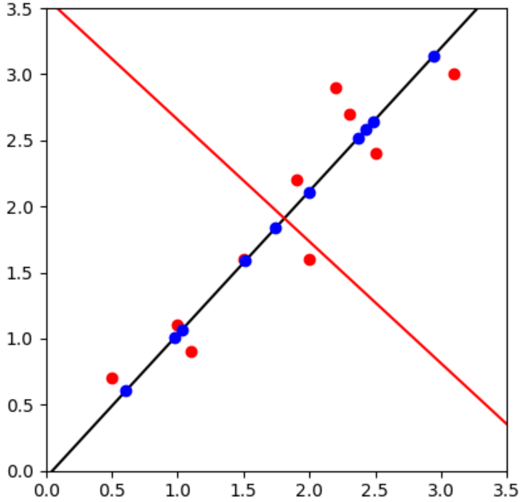


Exercise 1: Projection of PC

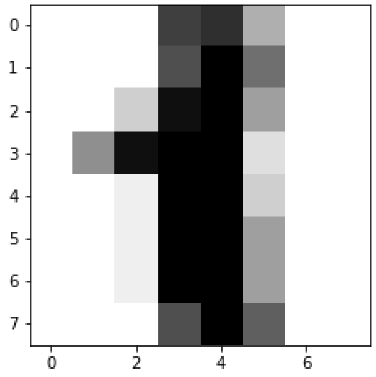
```
#step 5
plt.plot(c1[:,0]+pca1.mean_[0],c1[:,1]+pca1.mean_[1], ’bo’)

plt.show()
```

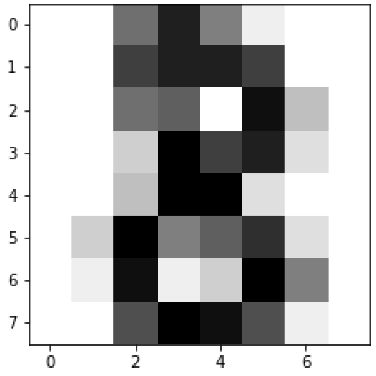



```
#step 1 ~5 equal to one function in sklearn.PCA

c1=pca.inverse_transform(s1)
plt.plot(c1[:, 0], c1[:, 1], 'bo')
```



Exercise 2: visualization of the images





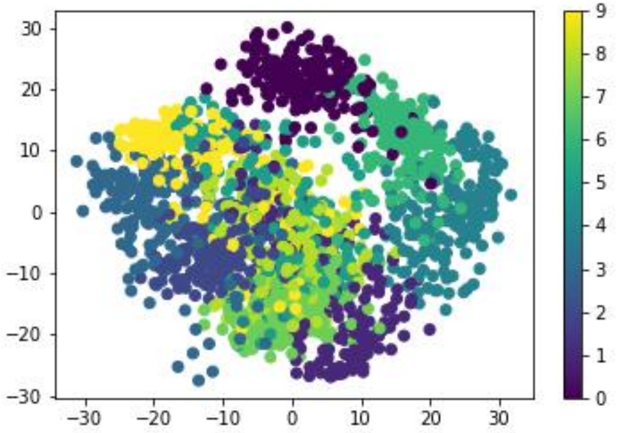
Input: the dataset made up of 1797 8x8 images, each is of a hand-written digit, first transform it into a feature vector with length 64

How to show the data of the images in a two-dimension figure?

Exercise 2: visualization of the images

```
import numpy as np
from sklearn.decomposition import PCA
from sklearn import datasets
import matplotlib.pyplot as plt

# load the handwriting data from the database
digits=datasets.load_digits()
print (digits.keys())
print (digits.data.shape)

#assignment
X,y=digits.data, digits.target
#define the pca
pca = PCA(n_components=2)
#reduce the features to 2 components
X_proj=pca.fit_transform(X)
```

```
#only retain about 28% of the variance by 2 PC
print (np.sum(pca.explained_variance_ratio_))

#plot the PC as a scatter plot
plt.scatter(X_proj[:,0], X_proj[:,1], c=y)
plt.colorbar()
plt.show()
```



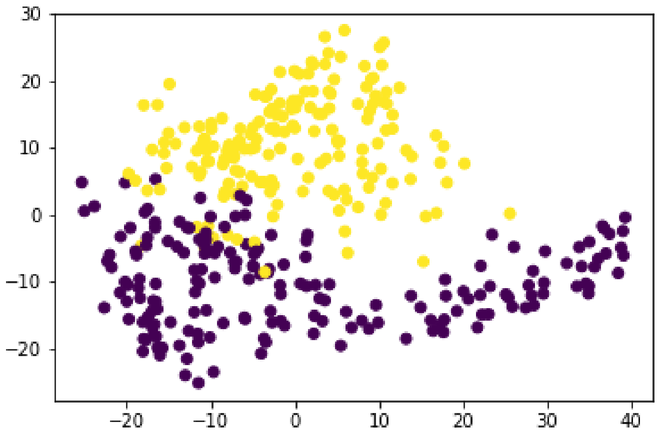
Exercise 3: Preprocess the dataset

- Load the dataset of hand-written digits of one and eight
- Preprocess the dataset by PCA
- Plot the PC as a scatter plot (2-dimension)
- Print the amount of variance
- Change the n_components to values in range of (0,1), then observe the amount of variance and its estimated number of components

Exercise 3: Preprocess the dataset

```
import numpy as np
from sklearn.decomposition import PCA
from sklearn import datasets
import matplotlib.pyplot as plt

#load the dataset
digits=datasets.load_digits()
data, target=digits.data,digits.target
X=data[np.logical_or(target==1,target==8), :]
y=target[np.logical_or(target==1,target==8)]
```

```
#define the PCA
pca = PCA(n_components=2)
#plot the PC as a scatter plot
X_proj=pca.fit_transform(X)
plt.scatter(X_proj[:,0], X_proj[:,1], c=y)
plt.show()
```



Parameters of PCA

```
class sklearn.decomposition. PCA (n_components=None, copy=True, whiten=False, svd_solver='auto', tol=0.0,
iterated_power='auto', random_state=None)                                                      [source]
```

n_components: Number of components to keep.
if n_components is not set: n_components == min (n_samples, n_features), default=None

if n_components == 'mle' and svd_solver == 'full', Minka's MLE is used to guess the dimension
if 0 < n_components < 1 and svd_solver == 'full', select the number of components such that the amount of variance that needs to be explained is greater than the percentage specified by n_components;n_components cannot be equal to n_features for svd_solver == 'arpack'.

svd_solver:
auto: default, if the input data is larger than 500x500 and the number of components to extract is lower than 80% of the smallest dimension of the data, then the more efficient 'randomized' method is enabled. Otherwise the exact full SVD is computed and optionally truncated afterwards.
full: run exact full SVD calling the standard LAPACK solver via scipy.linalg.svd and select the components by postprocessing.
arpack: run SVD truncated to n_components calling ARPACK solver via scipy.sparse.linalg.svds. It requires strictly 0 <n_components< X.shape[1]
randomized: run randomized SVD by the method of Halko et al.

http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

Exercise 3: Preprocess the dataset

```
#print the amount of variance
print (np.sum(pca.explained_variance_ratio_))

#change the n_components
pca = PCA(n_components=0.50)
#reduce the feature dimensions
x=pca.fit_transform(X)

#print the estimated number of components
print (pca.n_components_)
#print the amount of variance
print (np.sum(pca.explained_variance_ratio_))
```

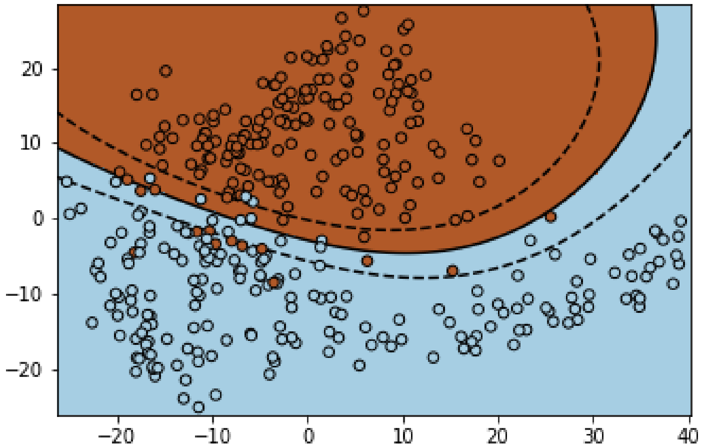
Output1:
0.39154700078274873

Output2:
3
Output3:
0.5084657414976048

estimate the number of components to retain the amount of variance by n_components (in the range of (0,1))

Exercise 4:Application

```
from sklearn import svm, model_selection
clf = svm.SVC(kernel='rbf', gamma=0.001)
scores = model_selection.cross_val_score(clf, x, y, cv=6)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(),
scores.std() * 2))

clf.fit(x,y)
plt.scatter(x[:, 0], x[:, 1], c=y, zorder=10,
cmap=plt.cm.Paired, edgecolor='k', s=30)
x_min, x_max = x [:, 0].min()-1, x [:, 0].max()+1
y_min, y_max= x [:, 1].min()-1, x [:, 1].max()+1

# create a mesh to plot in
xx, yy = np.mgrid[x_min:x_max:200j, y_min:y_max:200j]
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
```




```
# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.pcolormesh(xx, yy, Z>0,
cmap=plt.cm.Paired)
plt.contour(xx, yy, Z, colors=['k', 'k',
'k'],linestyles=['--', '-', '--'], levels=[-0.5, 0, 0.5])
plt.show()
```

Exercise 5: Image compression

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.decomposition import PCA

img=plt.imread("sample_Bw.png")
print (img.shape)
plt.imshow(img, cmap=plt.cm.gray)
plt.show()
```


```
pca = PCA(n_components=100, svd_solver='full')
pca.fit(img)
nd=pca.transform(img)
ni=pca.inverse_transform(nd)
plt.imshow(ni, cmap=plt.cm.gray)
plt.show()
print (np.shape(nd))
print (ni.shape)
print (ni)
```

Exercise 5: Image compression

```
for i in range(317):
    img[i,:]=img[i,:]-pca.mean_

U, S, V = np.linalg.svd(img)
z=np.dot(np.eye(100) *S[:100], V[:100,:])
Z=np.dot(U[:,:100], z)
for i in range(317):
    Z[i,:]=Z[i,:]+pca.mean_

plt.imshow(Z, cmap=plt.cm.gray)
plt.show()

print (Z)
print (Z.shape)
```