AI and Machine Learning

Zhiyun Lin

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Principal Components Analysis (PCA)

- Principal Components Analysis
- Autoencoders

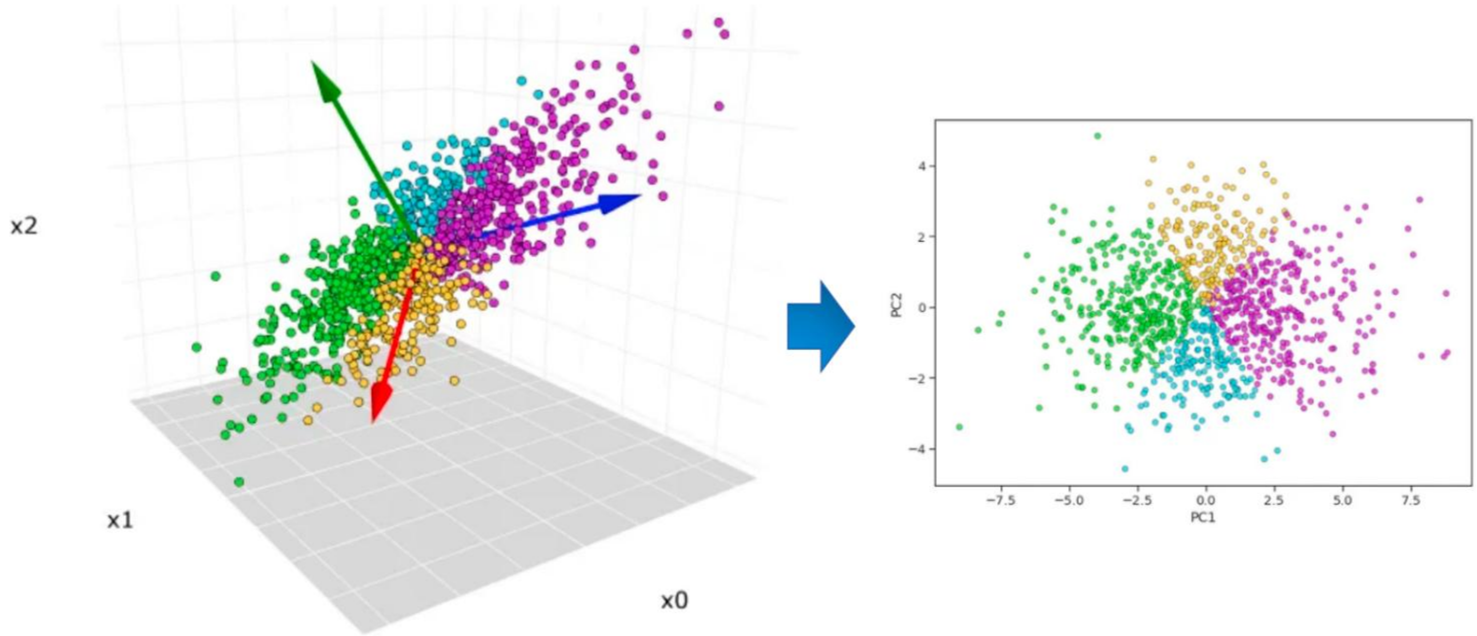
Principal Components Analysis

- PCA: most popular instance of second main class of unsupervised learning methods, **projection methods**, aka **dimensionality-reduction** methods
- Aim: find a **small number** of “directions"” in input space that explain variation in input data; re-represent data by projecting along those directions
- Important assumption: **variation contains information**.
- Data is assumed to be continuous:
  - **linear relationship** between data and the learned representation

PCA: Common Tool

- Handles high-dimensional data
  - If data has thousands of dimensions, can be difficult for a classifier to deal with
- Often can be described by much lower dimensional representation
- Useful for:
  - Visualization
  - Preprocessing
  - Modeling - prior for new data
  - Compression

PCA: Intuition



PCA: Intuition

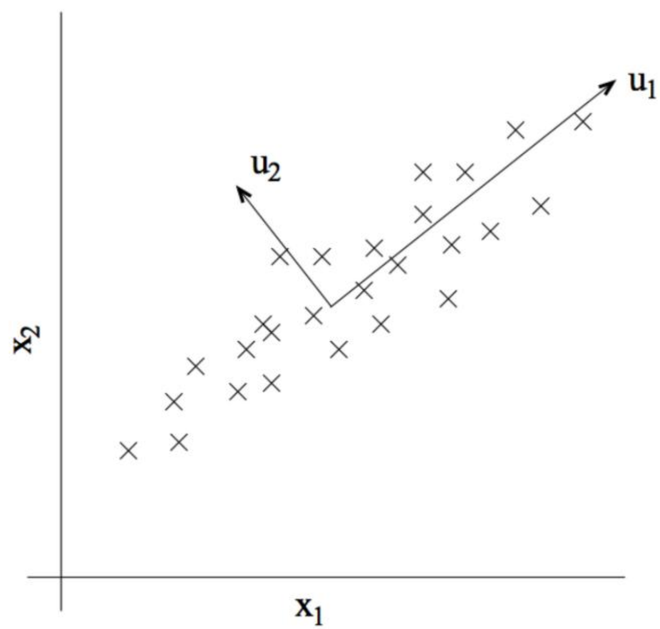
- As in the previous lecture, training data has N vectors, $\{x^{(n)}\}_{n=1}^{N}$, , of dimensionality D, so $x^{(n)} \in \mathbb{R}^D$
- Aim to reduce dimensionality
  - linearly project to a much lower dimensional space, $M << D$::

$$x \approx U_{pca} z + a$$

> where U_{pca} is a $D \times M$ matrix and z is an M-dimensional vector

PCA: Intuition

- Search for orthogonal directions in space with the **highest variance**
    - project data onto this subspace
- Structure of data vectors is encoded in sample covariance



Finding Principal Components

- To find the principal component directions,we center the data (**subtract the sample mean from each variable**)
- Calculate the **empirical covariance matrix**:

$$C = \frac{1}{N} \sum_{n=1}^{N} (x^{(n)} - \bar{x})(x^{(n)} - \bar{x})^T$$

> with $\bar{x}$ the mean

- What's the dimensionality of C?

Finding Principal Components

- Find the M eigenvectors with largest eigenvalues of C: these are the **principal components**
- Assemble these eigenvectors into a $D \times M$ matrix U_{pca}
- We can now express D-dimensional vectors x by projecting them to M-dimensional

$$z = U_{pca}^T x$$

Standard PCA

- **Algorithm:** to find M components underlying D-dimensional data

1. Select the top M eigenvectors of C (data covariance matrix):

$$C = \frac{1}{N}\sum_{n=1}^{N}(x^{(n)} - \bar{x})(x^{(n)} - \bar{x})^T = U\Sigma U^T \approx U_{1:M}\Sigma_{1:M}U_{1:M}^T$$

> where U is orthogonal, columns are unit-length eigenvectors
>
> $$U^T U = UU^T = I$$
>
> and Σ is a matrix with eigenvalues on the diagonal, representing the variance in the direction of each eigenvector

Standard PCA

- Matrix form of C:

$$C = \frac{1}{N}(\mathbf{X} - \bar{x})(\mathbf{X} - \bar{x})^T$$

> where $\mathbf{X}$ is a D-by-N matrix of data vectors, with each column being a data sample

Standard PCA

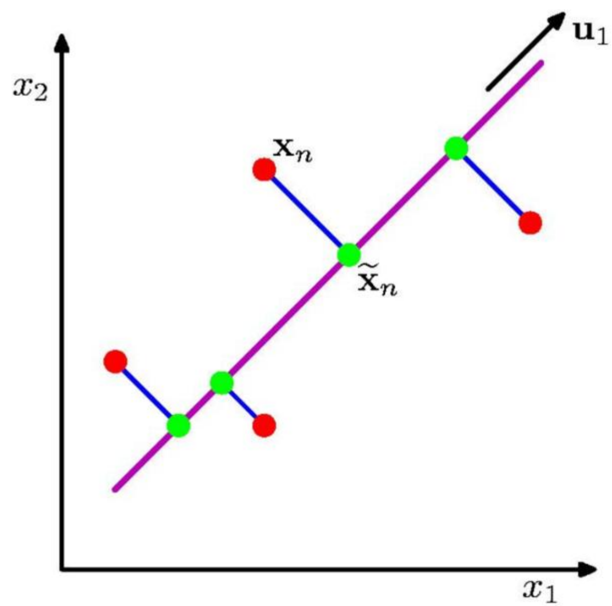
2. Project each input vector x into this subspace, e.g.,

$$z_j = u_j^T x; \qquad z = U_{1:M}^T x$$

> Let $U_{pca} = U_{1:M}$. Then we have the principal components
>
> $$z = U_{pca}^T x$$

Two Derivations of PCA

- Two views/derivations:
  - Maximize variance (scatter of green points)
  - Minimize error (red-green distance per datapoint)



PCA: Minimizing Reconstruction Error

- We can think of PCA as projecting the data onto a lower-dimensional subspace
- One derivation is that we want to find the projection such that the best linear reconstruction of the data is as close as possible to the original data

$$J(u, z, b) = \sum_n ||x^{(n)} - \tilde{x}^{(n)}||^2$$

> where
>
> $$\tilde{x}^{(n)} = \sum_{j=1}^{M} z_j^{(n)} u_j + \sum_{j=M+1}^{D} b_j u_j$$

PCA: Minimizing Reconstruction Error

- Objective minimized when first M components are the eigenvectors with the maximal eigenvalues

$$z_j^{(n)} = u_j^T x^{(n)}, \ \forall j = 1, \dots, M;$$

$$b_j = \bar{x}^T u_j, \ \forall j = M+1, \dots, D.$$

> In the maxtirx form:
>
> $$x^{(n)} \approx \tilde{x}^{(n)} = U_{1:M} z^{(n)} + a$$

> where $\quad a = U_{M+1:D} b, \qquad b = U_{M+1:D}^T \bar{x}$

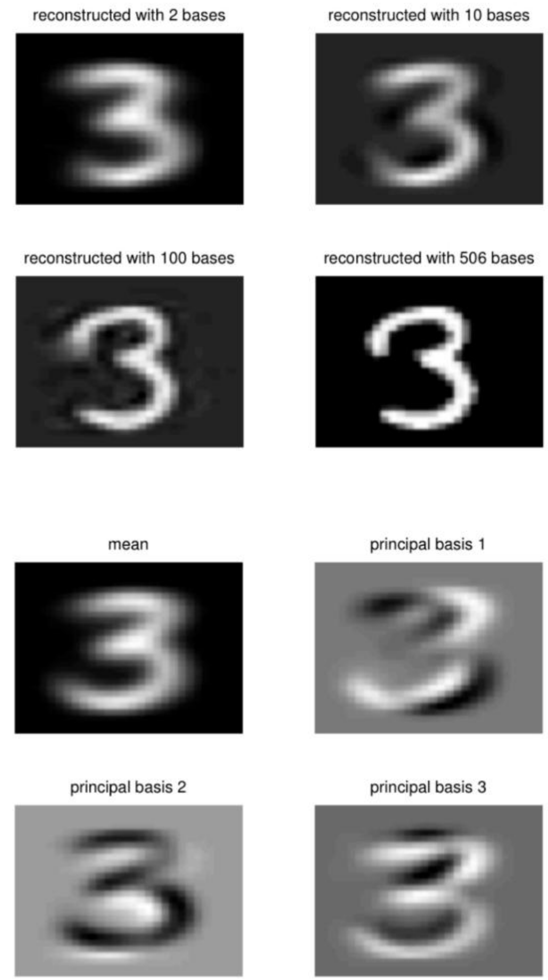
- If the **mean $\bar{x}$ is zero**, then $b = 0$ and $a = 0$.

Applying PCA to Faces

- Run PCA on 2429 19×19 grayscale images
- Compresses the data: can get good reconstructions with only 3 components

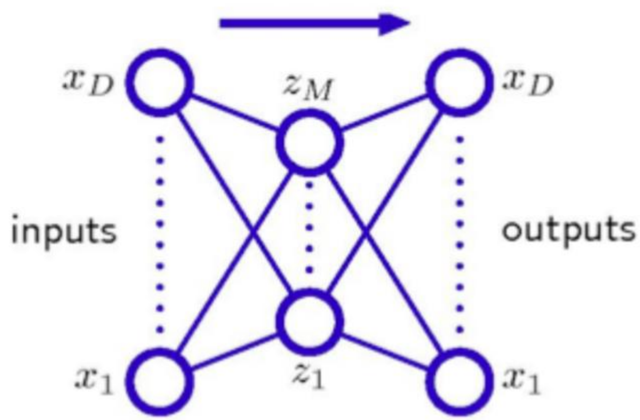
- PCA for pre-processing: can apply classifier to latent representation
  - PCA with 3 components obtains **79% accuracy** on face/non-face discrimination on test data vs.76.8% for GMM with 84 states
- Can also be good for visualization

Applying PCA to Digits



Relation to Neural Networks

- PCA is closely related to a particular form of neural network
- An **autoencoder** is a neural network whose outputs are its own inputs



- The goal is to minimize **reconstruction error**

Autoencoders

- Define

$$z = f(Wx); \quad \hat{x} = g(Vz)$$

- Goal:

$$\min_{W,V} \frac{1}{2N} \sum_{n=1}^{N} ||x^{(n)} - \hat{x}^{(n)}||^2$$

- If g and f are linear

$$\min_{W,V} \frac{1}{2N} \sum_{n=1}^{N} ||x^{(n)} - VWx^{(n)}||^2$$

- In other words, the **optimal solution is PCA** for the case when the mean of the data is 0.
- What if the mean of the data is not 0?

Autoencoders: Nonlinear PCA

- What if $g()$ is not linear?
- Then we are basically doing **nonlinear PCA**
- Some subtleties but in general this is an accurate description

Comparing Reconstructions

Real data

30-d deep autoencoder

30-d logistic PCA

30-d PCA

Python Codes for PCA

```
import pandas as pd

df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/'
                      'machine-learning-databases/wine/wine.data',
                      header=None)
df_wine.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

Python Codes for PCA

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


# split into training and testing sets
X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3,
    stratify=y, random_state=0
)
# standardize the features
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

Python Codes for PCA

```
import numpy as np


cov_mat = np.cov(X_train_std.T)
eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)
```

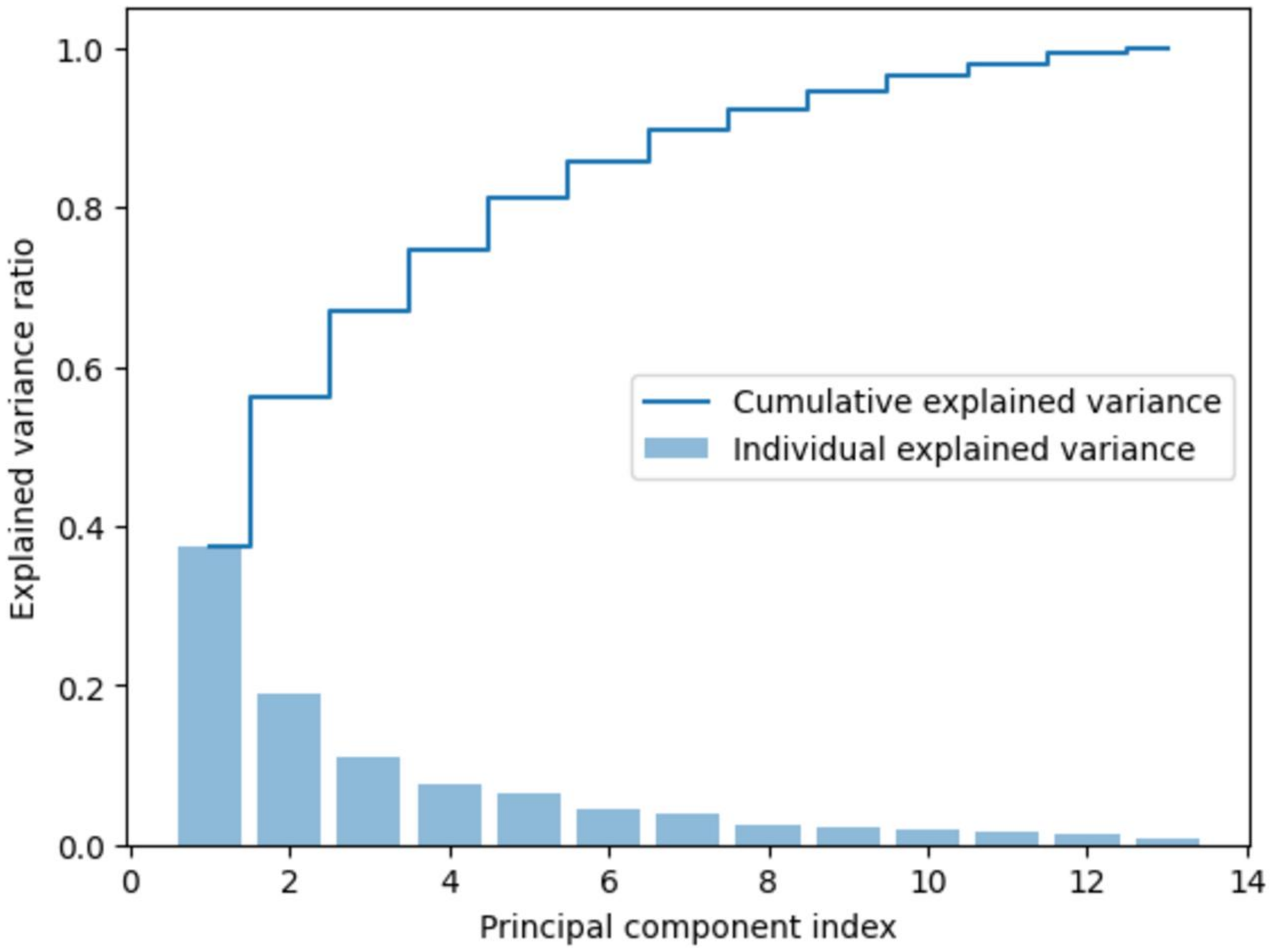
Python Codes for PCA

```
import matplotlib.pyplot as plt

# 累加解释方差（特征值）之和
tot = sum(eigen_vals)
var_exp = [(i / tot) for i in sorted(eigen_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)

# 绘制解释方差
plt.bar(range(1,14), var_exp, alpha=0.5,
        align='center', label='individual explained variance')
plt.step(range(1,14), cum_var_exp, where='mid',
         label='cumulative explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal component index')
plt.legend(loc='best')
plt.show()
```

Python Codes for PCA



Python Codes for PCA

```python
# 生成(特征值, 特征向量)的特征对, tuple类型
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:, i]) for i in range(len(eigen_vals))]

# 特征对按特征值降序排序
eigen_pairs.sort(key=lambda k: k[0], reverse=True)
```

Python Codes for PCA

```
Eigenvalues in descending order:
[(4.892308303273746, array([ 0.14669811, -0.24224554, -0.02993442, -0.25519002,  0.12079772,
        0.38934455, 0.42326486, -0.30634956,  0.30572219, -0.09869191, 0.30032535,  0.36821154, 0.29259713])),
(2.4663503157592244, array([ 0.50417079,  0.24216889,  0.28698484, -0.06468718,  0.22995385,
        0.09363991,  0.01088622,  0.01870216,  0.03040352,  0.54527081, -0.27924322, -0.174365,  0.36315461])),
(1.4280997275048464, array([-0.11723515,  0.14994658,  0.65639439,  0.58428234,  0.08226275,
        0.18080442,  0.14295933,  0.17223475,  0.1583621 , -0.14242171, 0.09323872,  0.19607741, -0.09731711])),
(1.0123346209044963, array([ 0.20625461,  0.1304893 ,  0.01515363, -0.09042209, -0.83912835,
        0.19317948,  0.14045955,  0.33733262, -0.1147529 ,  0.07878571, 0.02417403,  0.18402864,  0.05676778])),
(0.8490645933450235, array([-0.18781595,  0.56863978, -0.29920943, -0.04124995, -0.02719713,
        0.14064543,  0.09268665, -0.08584168,  0.56510524,  0.01323461, -0.37261081,  0.08937967, -0.21752948])),
(0.6018151434229905, array([-0.14885132, -0.26905276, -0.09333861, -0.10134239,  0.11256735,
        0.01222488, -0.05503452,  0.69534088,  0.49835441,  0.15945216, 0.21651535, -0.23517236,  0.10562138])),
(0.5225154620639972, array([-0.17926366, -0.59263673,  0.06073346,  0.25032387, -0.28524056,
        0.05314553,  0.07989941, -0.29737172,  0.20251913,  0.39736411, -0.38465475, -0.08629033, -0.13029829])),
(0.33051429173094055, array([-0.40305492, -0.10183371,  0.35184142, -0.50045728,  0.08373917,
        0.13511146,  0.00336017,  0.19012076, -0.17602994, -0.21493067, -0.51725944,  0.13645604,  0.16775843])),
(0.29595018365934656, array([-0.41719758,  0.21710149,  0.12854985,  0.04733441, -0.27891878,
       -0.28098565, -0.0391443 , -0.27862219,  0.14853946, -0.00410241, 0.19781412, -0.23813815,  0.63735021])),
(0.23995530477949092, array([ 4.13320786e-04, -0.0878560762, -0.452518598,  0.486169765, 0.114764951,
        0.0945645138, -0.100444099, 0.200128778, -0.139942067, -0.115349466, -0.302254353,  0.318414303, 0.503247839])),
(0.21432211869872336, array([ 0.40356719, -0.152475   ,  0.16837606, -0.06709029, -0.10239686,
        -0.61860015, -0.13968028,  0.00163324,  0.38856849, -0.3083459, -0.20045639,  0.28410033,  0.03755468])),
(0.16831253504096216, array([ 0.27566086, -0.0813845 , -0.01297513,  0.0989088 , -0.09592977,
        0.28389764,  0.11672921, -0.03965663,  0.08606027, -0.57165189, -0.19884453, -0.65086971,  0.07123771])),
(0.08414845672679461, array([-0.05546872,  0.03327316, -0.10061857,  0.05616586,  0.09584239,
       -0.42126512,  0.8472247 ,  0.1662568 , -0.16619747,  0.03961736, -0.10538369, -0.09950556, -0.01606618]))]
```

Python Codes for PCA

```
w = np.hstack((eigen_pairs[0][1][:, np.newaxis], eigen_pairs[1][1][:, np.newaxis]))
print('Matrix W:\n', w)
```

```
Matrix U:
 [[ 0.14669811  0.50417079]
 [-0.24224554  0.24216889]
 [-0.02993442  0.28698484]
 [-0.25519002 -0.06468718]
 [ 0.12079772  0.22995385]
 [ 0.38934455  0.09363991]
 [ 0.42326486  0.01088622]
 [-0.30634956  0.01870216]
 [ 0.30572219  0.03040352]
 [-0.09869191  0.54527081]
 [ 0.30032535 -0.27924322]
 [ 0.36821154 -0.174365  ]
 [ 0.29259713  0.36315461]]
```

Python Codes for PCA

- In math

$$z = U^T x$$

- In python codes

$$z = x^T U$$

```
X_train_0_pca = X_train_std[0].dot(U)
print(X_train_0_pca)
```

```
[2.59891628 0.00484089]
```

```
X_train_pca = X_train_std.dot(U)
```

Python Codes for PCA

```
colors = ['r', 'b', 'g']
markers = ['s', 'x', 'o']
for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_pca[y_train==l, 0],
                X_train_pca[y_train==l, 1],
                c=c, label=l, marker=m)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.show()
```

Python Codes for PCA