智能设备采集的用户行为数据的分析

一、课前准备

- 了解无监督学习相关算法的基本知识。
- 了解numpy/pandas/scikit-learn等工具的使用。

二、课堂主题

本节代码实践课主要是学习数据降维PCA和聚类K-Means的应用场景，以及Python的代码实现。

三、课堂目标

1. 用于数据降维的PCA
2. 聚类K-Means的应用
3. 数据处理与数据可视化

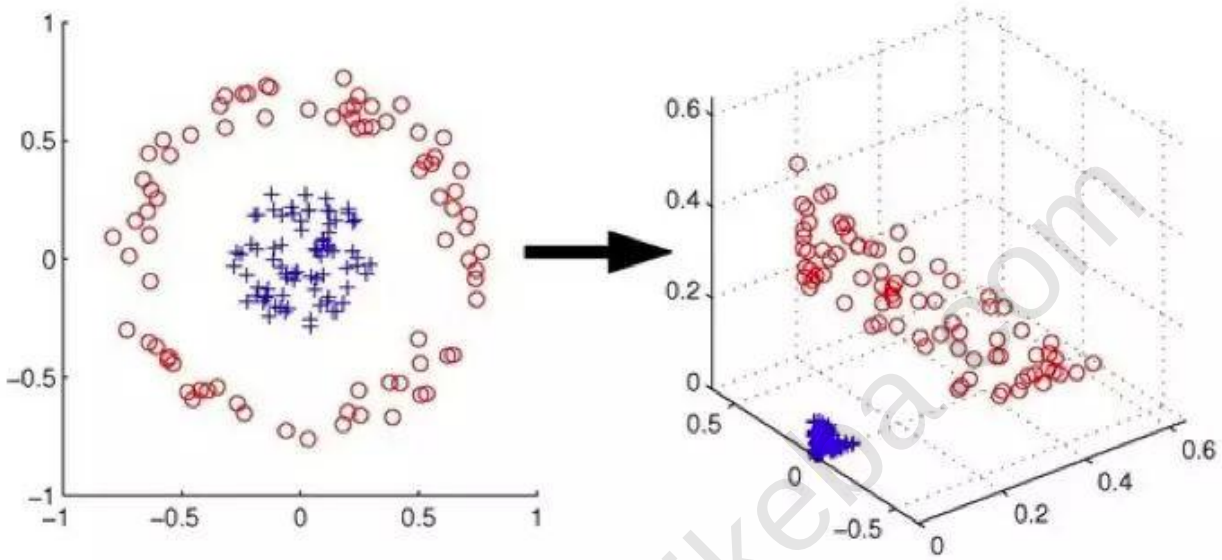
四、知识要点

4.1 PCA降维

4.1.1 降维的概念

在机器学习所训练的数据中，通常会存在着很多的特征，这也就意味着我们所要处理的数据的维度是很大的，由于维度大的数据处理起来非常困难，各种各样的降维算法也就随之产生了。

降维就是指采用某种映射方法，将原高维空间中的数据点映射到低维度的空间中。降维的本质是学习一个映射函数 f : x->y，其中x是原始数据点的表达，目前最多使用向量表达形式。y是数据点映射后的低维向量表达，通常y的维度小于x的维度（当然提高维度也是可以的）。f可能是显式的或隐式的、线性的或非线性的。



4.1.2 方差

方差：指的是一组数据中的各个数减去这组数据的平均数的平方和的平均数。

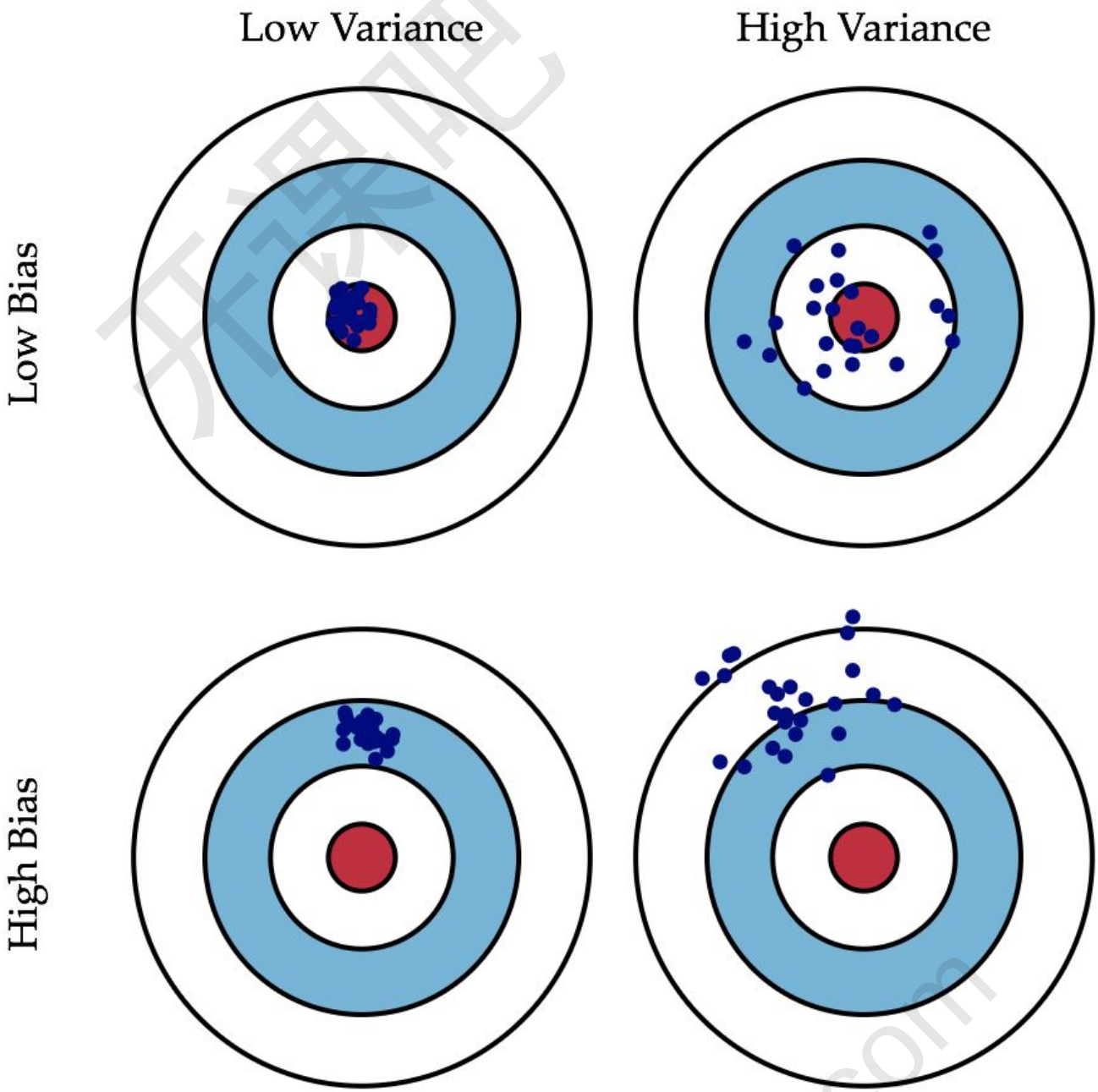
例如有一组数据：{1, 2, 3, 4, 5}

可以计算出平均数为：(1+2+3+4+5)/5=3

各个数与平均数差的平方和为：10

方差为：10/5=2

本质：**方差用于测算数值型数据的离散程度。**



4.1.3 协方差

协方差本质：**用于度量各个维度偏离其均值的程度。**

数学期望

- 离散

$E(x) = \sum_{i=1} x_i p_i$

- 连续

$E(x) = \int_{-\infty}^{\infty} x f(x) dx$

协方差的计算方式如下（以二维为例）：

$cov(X, Y) = E[(x - E[x])(y - E[y])]$

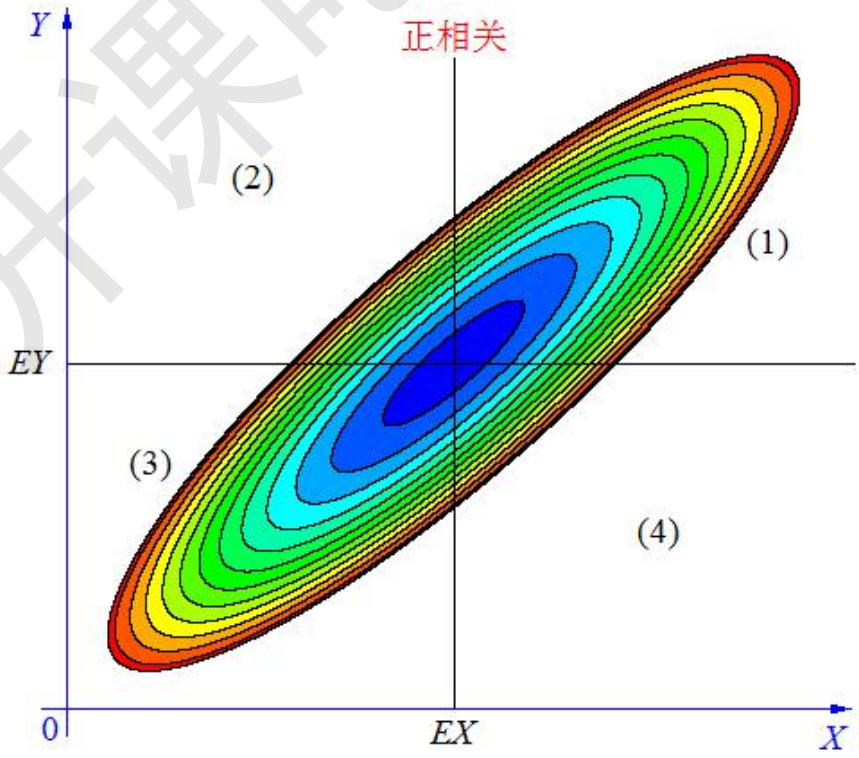
也可以写成如下的形式：

$cov(X,Y) = E[(x-\bar{x})(y-\bar{y})]$

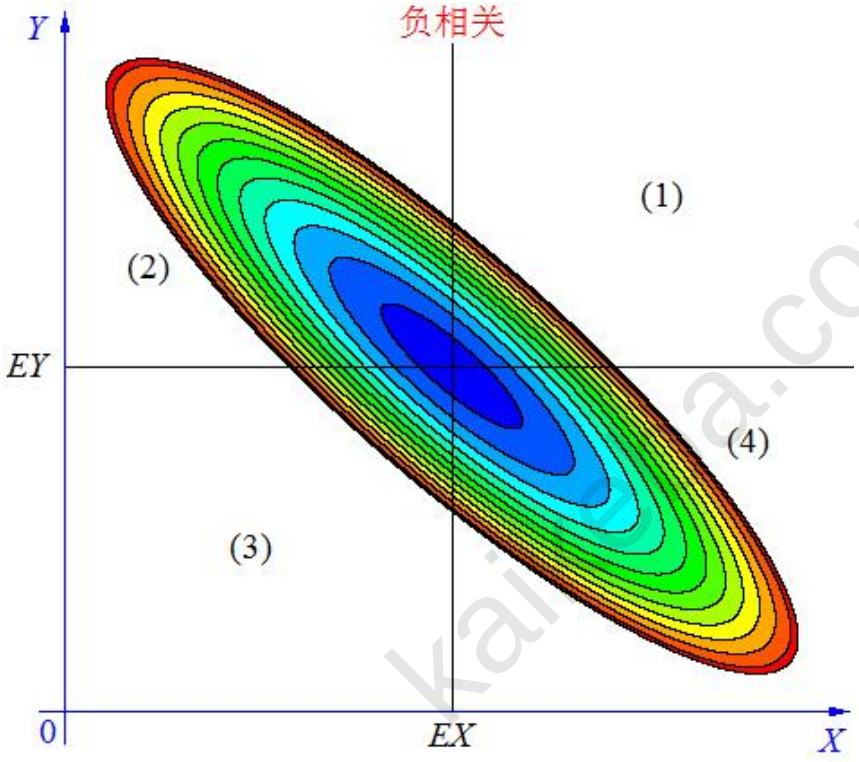
这里的E表示期望，$\bar{x},\bar{y}$表示均值。

计算出了协方差的结果后，有三种解释的方式，来看一下：

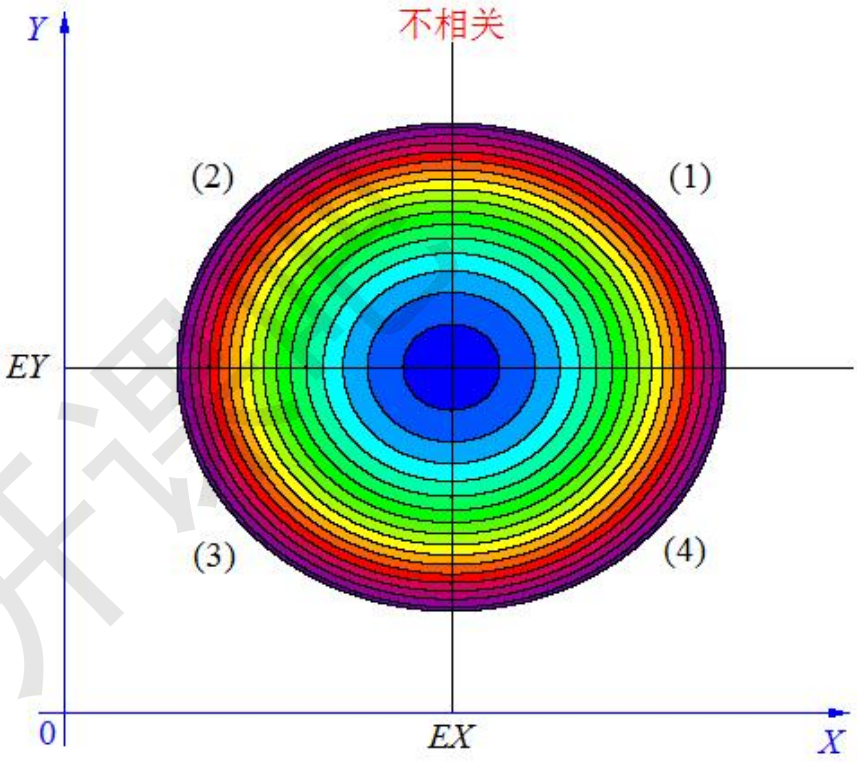
当cov(X,Y)>0时，表明X和Y正相关：



当cov(X,Y)<0时，表明X和Y负相关：



当cov(X,Y)=0时，表明X和Y不相关：



协方差为0，两个随机变量不一定相互独立，而两个随机变量相互独立，协方差一定为0。

4.1.4 主成分分析（PCA）

Principal Component Analysis(PCA)是最常用的线性降维方法，它的目标是通过某种线性投影，将高维的数据映射到低维的空间中表示，并期望在所投影的维度上数据的方差最大，以此使用较少的数据维度，同时保留住较多的原数据点的特性。

PCA的几个步骤如下：

一、标准化

为了让每一个维度对分析的结果造成同样的影响，需要对连续的初始变量的范围作标准化。

具体一点说就是因为我们后续的结果对数据的方差十分敏感，取值范围较大的维度会比相对较小的维度造成更大的影响（例如一个在1-100之间变化的维度对结果的影响，比一个0-1的更大），会导致一个偏差较大的结果，所以，将数据转化到相似的范围可以预防这个问题。

数据标准化的方法如下(其中standard deviation表示标准差)：

$$z = \frac{value - mean}{standard \cdot deviation}$$

二、计算协方差矩阵

这一步是为了理解数据集中的变量是如何从平均值变化过来的，同时可以查看不同的特征之间又有什么关系，此时要计算协方差矩阵。

协方差矩阵是一个P*P的对称矩阵（P是维度的数量）它涵盖了数据集中所有元组对初始值的协方差，例如一个拥有三个变量x,y,z和三个维度的数据集，协方差矩阵将是一个3*3的矩阵：

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

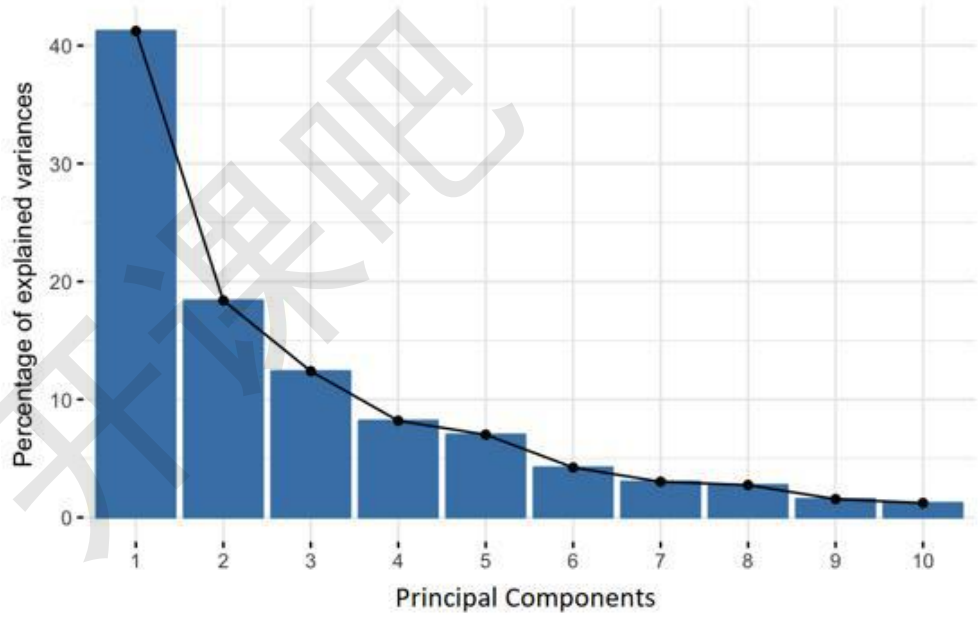
由于变量与自身的协方差等于他的方差，在主对角线上我们已将计算出了各个变量初始值的方差。协方差矩阵的每一个元组关于主对角线对称，这意味着上三角部分和下三角部分是相等的。

三、计算主成分

- 主成分是什么?

主成分是一个新的变量，他是初始变量的线性组合。新的变量之间是不相关的，第一个主成分中包含了初始变量的大部分信息，是初始变量的压缩和提取。

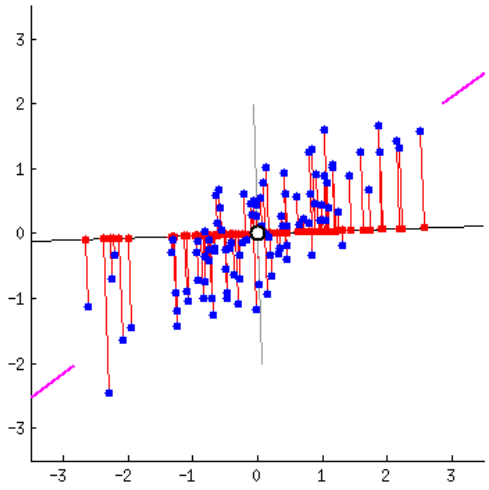
例如：虽然在一个 10 维的数据集中算出了 10 个主成分，但大部分的信息都会被压缩在第一主成分中，剩下的大部分信息又被压缩到第二主成分中，以此类推，得到了下面这张图：



从理论方面来说，主成分代表着蕴含 最大方差的方向。对于主成分来说，变量的方差越大，空间中点就越分散，空间中的点越分散，那么它包含的信息就越多。简单的讲，主成分就是一条更好的阐述数据信息的新坐标轴，因此更容易从中观测到差异。

- 怎么计算主成分

有多少个变量就有多少个主成分，对于第一主成分来说沿着对应的坐标轴变化意味着有最大的方差，例如用下列的散点图表示：



主成分应该大致是图中紫色线的方向。（因为它穿过了原点，而且数据映射在这条线上后，有着最大方差（各点与原点距离的均方））

第二个主成分也是这样计算的，它与第一主成分不相关（即为互相垂直）表示了下一个最大方差的方向。

重复上面的步骤，直到我们从原始数据中计算出所有的主成分。

- 特征值和特征向量

特征值和特征向量通常成对出现，每一个特征向量对应一个特征值，他们各自的数量相等，等于原始数据的维度，例如有三个变量就会有三个特征向量与三个特征值。

协方差矩阵的特征向量其实就是一些列的坐标轴，将数据映射到这些坐标轴之后，将会得到最大的方差（这意味着更多的信息），这就是主成分，特征值其实就是特征向量的系数，它代表了每个特征向量包含了多少信息量。

四、主成分向量

主成分向量仅仅是一个矩阵，里面有我们决定保留的特征向量。这是数据降维的第一步，我们只是要在n个变量中保留p个特征向量（成分）我们把数据映射到新的坐标轴上时，最后数据将只有p个维度。

五、将数据映射到新的主成分坐标系中

将使用从协方差矩阵中算出来的特征向量形成主成分矩阵，并将原始数据映射到主成分矩阵对应的坐标轴上，这就叫做主成分分析。具体的做法便是用原数据矩阵的转置乘以主成分矩阵的转置。

PCA的流程总结如下：

- 1）将原始数据按列组成n行m列矩阵X
- 2）将X的每一行（代表一个属性字段）进行零均值化，即减去这一行的均值
- 3）求出协方差矩阵
- 4）求出协方差矩阵的特征值及对应的特征向量
- 5）将特征向量按对应特征值大小从上到下按行排列成矩阵，取前k行组成矩阵P
- 6）Y=PX即为降维到k维后的数据

六、PCA的计算过程

```
# 样本集合

X = np.array([[10, 15, 29],
              [15, 46, 13],
              [23, 21, 30],
              [11, 9, 35],
              [42, 45, 11],
              [9, 48, 5],
              [11, 21, 14],
              [8, 5, 15],
              [11, 12, 21],
              [21, 20, 25]])
```

```
# 中心化（未除）
# 均值
[ 16.1  24.2  19.8]

# 样本集的中心化(每个元素将去当前维度特征的均值)：
[[ -6.1  -9.2   9.2]
 [ -1.1  21.8  -6.8]
 [  6.9  -3.2  10.2]
 [ -5.1 -15.2  15.2]
 [ 25.9  20.8  -8.8]
 [ -7.1  23.8 -14.8]
 [ -5.1  -3.2  -5.8]
 [ -8.1 -19.2  -4.8]
 [ -5.1 -12.2   1.2]
 [  4.9  -4.2   5.2]]
```

```
# 计算协方差矩阵

C =[[ 108.32222222   74.53333333  -10.08888889]
    [  74.53333333  260.62222222 -106.4       ]
    [ -10.08888889 -106.4          94.17777778]]
```

```
# 计算协方差矩阵特征值

[ 335.15738485   95.32771231   32.63712506]

# 得到特征向量

# 样本矩阵X的协方差矩阵C的特征向量：
[[-0.30253213 -0.87499307 -0.37797014]
 [-0.86718533  0.08811216  0.49012839]
 [ 0.39555518 -0.47604975  0.78543792]]
```

```
# 按照特征值最大原理选择特征向量（例如TOP2）

p1 = [-0.30253213,  -0.86718533,  0.39555518]
p2 = [-0.87499307, 0.08811216, -0.47604975]

# 得到降维矩阵P

P = np.array([[-0.30253213, -0.87499307],
              [-0.86718533,  0.08811216],
              [ 0.39555518, -0.47604975]])
# 根据样本X和降维矩阵P进行降维

Y = PX
Y = np.dot(X, P)
```

```
X = [[10, 15, 29],
     [15, 46, 13],
     [23, 21, 30],
     [11, 9, 35],
     [42, 45, 11],
     [9, 48, 5],
     [11, 21, 14],
     [8, 5, 15],
     [11, 12, 21],
     [21, 20, 25]]

# 10*3 dot 3*2 = 10*2

P = ([[-0.30253213, -0.87499307],
      [-0.86718533,  0.08811216],
      [ 0.39555518, -0.47604975]])
```

```
Y = [[ -4.56200104 -21.2336912 ]
     [-39.28629002 -15.26038349]
     [-13.30247561 -32.55597794]
     [  2.71190993 -25.49365577]
     [-47.37858268 -38.02120912]
     [-42.36990935  -6.0258027 ]
     [-16.00097294 -14.43926499]
     [ -0.822856   -13.7001301 ]
     [ -5.42741864 -18.56462272]
     [-13.80800193 -28.51385518]]
```

设A是一个n阶矩阵，若数λ和n维非零列向量满足$Ax = \lambda x$，数λ称为A的特征值，x称为A对应于特征值λ的特征向量，此时$|\lambda E - A|$叫做特征多项式，若特征多项式为0则称为A的特征方程（齐次线性方程组），求解特征值的过程其实就是求解特征方程。

示例如下：

给定条件：

$E = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$（单位矩阵）

$A = \begin{bmatrix} 4 & 2 & -5 \\ 6 & 4 & -9 \\ 5 & 3 & -7 \end{bmatrix}$

第一步（根据条件得到特征方程）：

$Ax = \lambda x \Rightarrow Ax = \lambda Ex \Rightarrow (A - \lambda E)x = 0$

第二步（展开特征方程）：

$$|\lambda E - A| = \begin{bmatrix} \lambda - a_{11} & -a_{12} & \dots & -a_{1n} \\ -a_{21} & \lambda - a_{22} & \dots & -a_{2n} \\ \dots & \dots & \dots & \dots \\ -a_{m1} & -a_{m2} & \dots & \lambda - a_{mn} \end{bmatrix} = 0$$

第三步（带入数据）：

$$|\lambda E - A| = \begin{bmatrix} \lambda - 4 & -2 & 5 \\ -6 & \lambda - 4 & 9 \\ -5 & -3 & \lambda + 7 \end{bmatrix} = 0$$

第四步（矩阵展开计算行列式）：

$$(\lambda - 4)(\lambda - 4)(\lambda + 7) + (-2 * 9 * -5) + (5 * -6 * -3) - (5 * (\lambda - 4) * -5) - (-2 * -6 * (\lambda + 7)) - ((\lambda - 4) * 9 * -3) = 0$$

第五步（行列式化简得到）：

$\lambda^2 * (\lambda - 1) = 0$

第六步（解得特征值）：

$$\begin{cases} \lambda_1 = 1 \\ \lambda_2 = \lambda_3 = 0 \end{cases}$$

第七步（$\lambda = 1$时求解）：

$(E - A)x = 0$

$$E - A = \begin{bmatrix} -3 & -2 & 5 \\ -6 & -3 & 9 \\ -5 & -3 & 8 \end{bmatrix} \overset{\text{化简}}{\rightarrow} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix}$$

第八步（计算特征矩阵（向量））：

$$(E - A)x = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0$$

$$\Rightarrow \begin{cases} x_1 - x_3 = 1 \\ x_2 - x_3 = 0 \end{cases}$$

令x1=1，解得：

$$X_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

同理可以计算出$\lambda_2 = \lambda_3 = 0$时的结果如下：

$$X_2 = X_3 = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$$

至此特征值和特征向量就成功的计算出来了，结合上面的内容就是我们PCA要做的事情了。

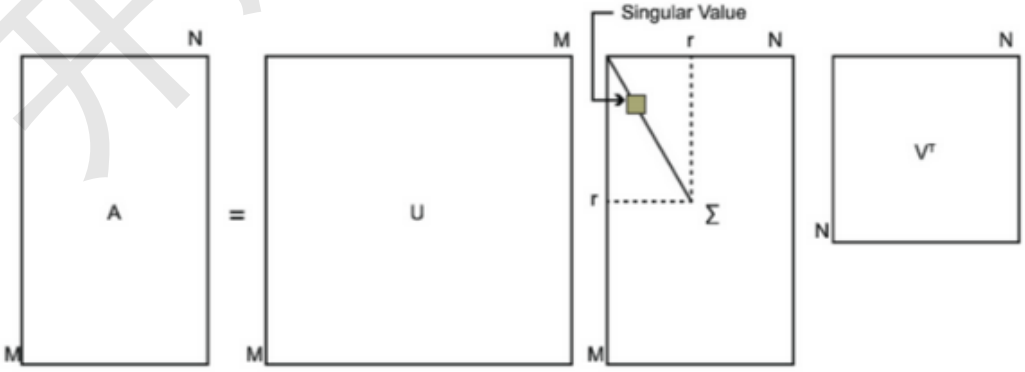
4.1.5 奇异值分解（SVD）

假设矩阵A是一个M*N的矩阵，对于矩阵A的SVD我们有如下的定义：

$A = U \sum V^T$

其中U是一个m*m的矩阵，$\sum$是一个m*n的矩阵（除了主对角线，其余都为0），对角线上的元素称之为奇异值，V是一个n*n的矩阵。

其中$U^T U = I, V^T V = I$



对于V矩阵，内部的特征值和特征向量满足下式：

$(A^T A)v_i = \lambda_i v_i$

对于U矩阵，内部的特征值和特征向量满足下式：

$(AA^T)u_i = \lambda_i u_i$

奇异值矩阵的求解方式如下：

$A = U \sum V^T$
$AV = U \sum V^T V$
$AV = U \sum$

对于每一个特征值有如下的式子：

$Av_i = \sigma_i u_i$

求解σ_i即可。

```
# 样本集合

A = [[0, 1],
     [1, 1],
     [1, 0]]
```

```
# 求出A^TA和AA^T

A.TA = [[2, 1],
        [1, 2]]

AA.T = [[1, 1, 0],
        [1, 2, 1],
        [0, 1, 1]]
```

```
# 求解特征值和特征向量
# 对于v
lambda1 = 3
v1 = [0.707, 0.707]

lambda2 = 1
v2 = [-0.707, 0.707]

# 对于u
lambda1 = 3
u1 = [0.408, 0.816, 0.408]
lambda2 = 1
u2 = [0.707, 0, -0.707]
lambda3 = 0
u3 = [0.577, -0.577, 0.577]
```

```
# 求解奇异值为

sigma1 = 1.732
sigma2 = 1

# 最终结果
sigma = [[1.732, 0]
         [0,     1],
         [0,     0]]

V = [[0.707,  0.707],
     [-0.707, 0.707]]

U = [[0.408, 0.707,  0.577],
     [0.816, 0,     -0.577],
     [0.408, -0.707, 0.577]]

A = U*sigma*V
```

4.1.6 PCA和SVD的区别

- PCA不能进行稀疏矩阵的降维，SVD可以。
- PCA只能获得单个方向上的主成分，SVD可以获取两个方向。
- 对比于PCA来说SVD更稳定。

4.2 智能设备采集项目

4.2.1 项目背景：

样本数据为某公司通过智能手机与手环采集到的用户行为与动作数据（特征值X）；

一条记录包含多种特征、分别对应用户具体的活动类型（标签值y）。

活动类型（y）包括：行走、上楼梯、下楼梯、坐着、站立、躺下。

> 1 – 行走
>
> 2 – 上楼梯（躯体向上移动）
>
> 3 – 下楼梯（躯体向下移动）
>
> 4 – 坐着
>
> 5 – 站立
>
> 6 – 躺下

接下来的代码实操中，我们先不使用标签值，即先尝试基于各个特征值对样本进行无监督学习（降维、聚类），然后对比通过聚类的样本划分结果与样本真正的分类标签，从而了解无监督学习下的聚类结果与实际的分类有什么区别与联系。

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, AgglomerativeClustering, SpectralClustering
from sklearn import metrics

from matplotlib import pyplot as plt
import os
import numpy as np
import pandas as pd
import seaborn as sns
from tqdm import tqdm_notebook
import warnings
warnings.filterwarnings('ignore')

plt.style.use(['seaborn-darkgrid'])
plt.rcParams['figure.figsize'] = (12, 9)
```

4.2.2 数据读取与处理

```
# 标签值为用户的活动类型
X_train = np.loadtxt("1.0X_train.txt")
y_train = np.loadtxt("1.0y_train.txt").astype(int)

X_test = np.loadtxt("1.0X_test.txt")
y_test = np.loadtxt("1.0y_test.txt").astype(int)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((7352, 561), (2947, 561), (7352,), (2947,))
```

由于我们现在进行的是无监督学习，没有实际的标签值来判断准确率，所以可以直接合并训练集和测试集

```
X = np.vstack([X_train, X_test])
y = np.hstack([y_train, y_test])

X.shape, y.shape
```

```
((10299, 561), (10299,))
```

scipy.sparse -- 稀疏矩阵 scipy.sparse.hstack scipy.sparse.vstack

pd.concatenate pd.merge -- pd.dataframe

np.vstack np.hstack np.concatenate -- np.array

4.2.3 知识扩展

> np.vstack

> np.vstack等同于np.concatenate([], axis=0)

> np.hstack

> np.hstack等同于np.concatenate([], axis=1)

```
# np.vstack:按垂直方向（行顺序）堆叠数组构成一个新的数组
a = np.array([[1,2,3], [4,5,6]])

b = np.array([[7,8,9], [10,11,12]])

c = np.vstack((a,b)) # 将两个（2,3）形状的数组按垂直方向叠加
print(c.shape)
c
```

```
(4, 3)
```

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

```
# np.vstack等同于np.concatenate([], axis=0)
np.concatenate([a, b], axis=0)
```

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

```
# np.hstack:按水平方向（列顺序）堆叠数组构成一个新的数组
a = np.array([[1,2,3], [4,5,6]])

b = np.array([[7,8,9], [10,11,12]])

c2 = np.hstack((a,b)) # 将两个（2,3）形状的数组按水平方向叠加
print(c2.shape)
c2
```

```
(2, 6)
```

```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
```

```
# np.hstack等同于np.concatenate([], axis=1)
np.concatenate([a, b], axis=1)
```

```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
```

4.2.4 PCA降维，缩减特征的数量

回到项目正题

```
# 查看标签数据中唯一值有哪些，并得到数据集目标类别的数量 n_classes
n_classes = np.unique(y).size
np.unique(y)
```

```
array([1, 2, 3, 4, 5, 6])
```

用户具体的活动类型（y）

> 1 - 行走

> 2 - 上楼梯（躯体向上移动）

> 3 - 下楼梯（躯体向下移动）

> 4 - 坐着

> 5 - 站立

> 6 - 躺下

特征数有561个，因此要先使用PCA降维

```
# 使用 StandardScaler 完成特征数据的规范化，注意是对每一个特征维度去均值+方差归一化
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled.shape
```

```
(10299, 561)
```

主成分分析 PCA 降维来缩减特征的数量

主成分分析（PCA）是一种常见的数据降维方法，其目的是在“信息”损失较小的前提下，将高维的数据转换到低维，从而减小计算量

```
RANDOM_STATE=5  # 设定好随机种子，避免后续重跑出现不同的结果
pca = PCA(n_components=0.9, random_state=RANDOM_STATE).fit(X_scaled)
# n_components 可以是小数、也可以是整数，小数-设置降维后的数据所能保留的信息比例，整数-设置主成分的个数
# PCA主成分数量k的选择，是一个数据压缩的问题。
# 通常我们直接将参数n_components设置为float数据，来间接解决k值选取的问题

X_pca = pca.transform(X_scaled)
X_pca.shape
```

```
(10299, 65)
```

问题：降维后的第一主成分涵盖了多大比例的方差?

主成分对应的方差比例越高，说明主成分所含的信息量就越大。

```
round(float(pca.explained_variance_ratio_[0] * 100))
```

```
51
```

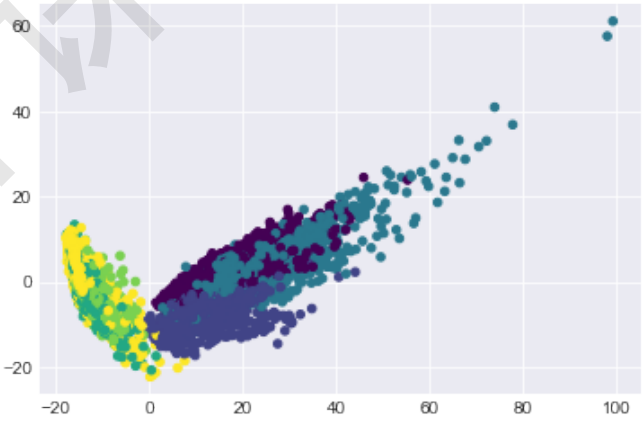
```
round(float(pca.explained_variance_ratio_[1] * 100))
```

```
6
```

4.2.5 数据可视化

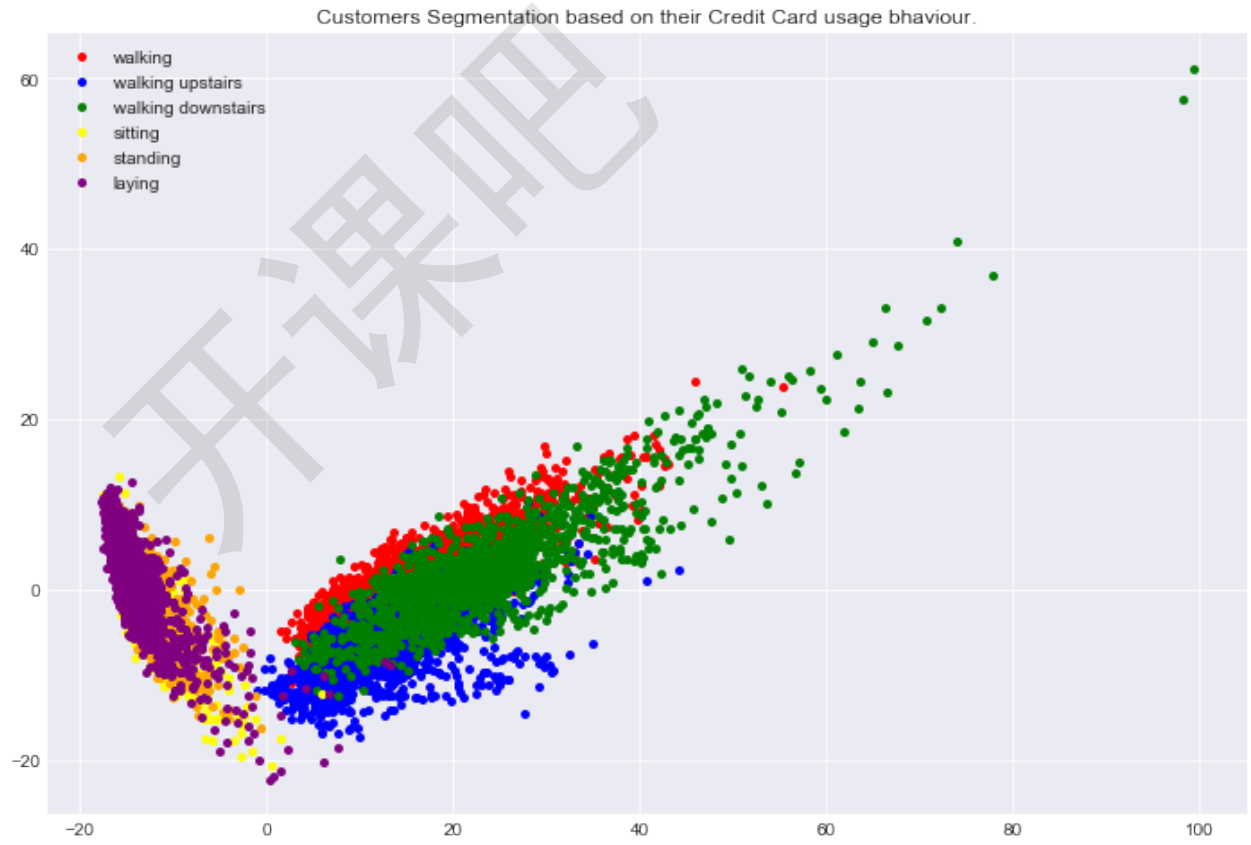
绘制前两个主成分特征的二维散点图

```
# 将第一主成分与第二主成分的分布呈现在二维图中，点的颜色对应实际所属的类型（y）
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, s=20, cmap='viridis')
plt.show()
```




```
# 更好的画图方法，就是不同类型（y）的样本点对应不同的颜色
xx1, xx2 = X_pca[:, 0], X_pca[:, 1]

colors = {1: 'red',
          2: 'blue',
          3: 'green',
          4: 'yellow',
          5: 'orange',
          6: 'purple'}

names = {1: 'walking',
         2: 'walking upstairs',
         3: 'walking downstairs',
         4: 'sitting',
         5: 'standing',
         6: 'laying'}

df = pd.DataFrame({'x': xx1, 'y':xx2, 'label':y})
groups = df.groupby('label')            # 所以先要得到一个dataframe，前两列分别是第一、第二主成分，第三列是实际类别

fig, ax = plt.subplots(figsize=(12, 8))

for name, group in groups:              # 遍历每一类，这样的话，label与color一一对应，例如类别为walking的样本点就是红色的
    ax.plot(group.x, group.y, marker='o', linestyle='', ms=5,
            color=colors[name], label=names[name], mec='none')
    ax.set_aspect('auto')
    ax.tick_params(axis='x',which='both',bottom='off',top='off',labelbottom='off')
    ax.tick_params(axis= 'y',which='both',left='off',top='off',labelleft='off')
```

```
ax.legend()
ax.set_title("Customers Segmentation based on their Credit Card usage bhaviour.")
plt.show()
```



通过肉眼观察，上图大致分为几个聚类簇？不同簇中又包含哪些类型的活动?

【1】 2 个簇：walking, walking upstairs, walking downstairs 和 sitting, standing, laying。

4.2.6 K-Means聚类

使用 KMeans 聚类方法对 PCA 降维后的数据进行聚类操作。于此同时，这里建议聚集为 6 类，实际上在正常情况下我们一般并不会知道聚为几类。

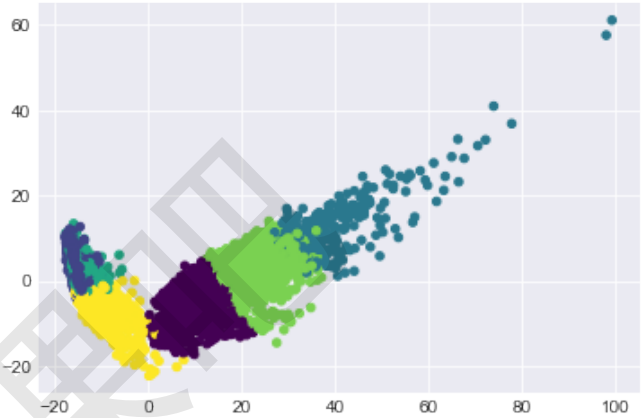
```
kmeans = KMeans(n_clusters=n_classes, n_init=100,
                random_state=RANDOM_STATE, n_jobs=1)
kmeans.fit(X_pca)
# n_init 用不同的初始化质心运行算法的次数。
# 由于K-Means是结果受初始值影响的局部最优的迭代算法，因此需要多跑几次以选择一个较好的聚类效果，默认是10

cluster_labels = kmeans.labels_
# 每个样本点对应的分类
```

sklearn中K-means算法的属性

- cluster_centers_：向量，形状为(n_clusters, n_features)；是聚类各类质心的坐标。
- labels_: 每个样本点对应的分类
- inertia_：float，每个点到其簇质心的距离之和，即类内离差平方和。

使用 PCA 前面 2 个主成分特征绘制二维图像，但这一次使用 KMeans 聚类标签进行着色。

```
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=cluster_labels, s=20, cmap='viridis')
plt.show()
```



结果对比

通过聚类的样本划分结果与样本对应的真正标签。

```
# 实际标签值
print(len(y))
y[:50]
```

```
10299
```

```
array([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4])
```

```
# 聚类标签值
print(len(cluster_labels))
cluster_labels [:50]
# 注意聚类标签值的0-5与实际标签值1-6在数值上没有对应关系，聚类标签值的0-5只是用来区分那6个簇的
```

```
10299
```

```
array([3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 5, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 5, 5, 3, 5, 5, 5, 5, 1, 1, 1, 1, 5, 5, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3], dtype=int32)
```

接下来，我们分别查看原始类别下的样本对应的聚类后所属的簇，从而查看原始的标签分布和 K-Means 聚类的标签分布有什么异同点。

```
# 使用pd.crosstab按照原始活动类别标签和聚类标签统计分组的频数
tab = pd.crosstab(y, cluster_labels, margins=True)
tab.index = ['walking', 'going up the stairs',
             'going down the stairs', 'sitting', 'standing', 'laying', 'all']
tab.columns = ['cluster' + str(i + 1) for i in range(6)] + ['all']
tab
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	cluster1	cluster2	cluster3	cluster4	cluster5	cluster6	all
walking	904	0	78	0	740	0	1722
going up the stairs	1242	0	5	0	295	2	1544
going down the stairs	320	0	196	0	890	0	1406
sitting	1	90	0	1235	0	451	1777
standing	0	0	0	1344	0	562	1906
laying	5	1556	0	53	0	330	1944
all	2472	1646	279	2632	1925	1345	10299

表格行名称是原始的活动类别标签，而列名称则对应了 K-Means 聚类后所属的簇序号。

可以看到，几乎每一个原始类别下的样本都被重新聚为了几个分散的簇，例如原始类别为“walking”的样本有一部分在簇1，有一部分在簇3，有一部分在簇5。虽然一些原始类别下的样本会有很大比例聚集在某一个簇里，例如“going up the stairs”的样本很大比例聚集在了簇1。但是我们依然可以得出结论，无监督学习聚类后的类与样本的原始分类还是有一定的不同的，因为聚类算法它没有标签值（y）的输入，它对类别的划分基于的并不是特征值和标签值的学习，而只是特征值。

> 这里，我们想定量地看看各原始类别下的样本聚类后的分散程度，这里推荐使用某一原始类别 K-Means 聚类后的最大数量簇的样本数，除以原始类别下的样本总数，来表征聚类的分散程度。得到的数值越小，即代表聚类后的样本越分散。

> 例如，walking 被重新划分为 cluster1, cluster3, cluster5。

> 其中，数量最多的是 cluster1，cluster1是walking类别下样本聚类后的最大数量簇，样本数量是904；而原始walking类别下的样本数为1722

> 则分散程度：904/1722=0.524

按照这个方法，计算各个原始类别聚类后的分散程度，并按照分散程度由大到小排序。

```
pd.Series(tab.iloc[:-1, :-1].max(axis=1).values /
          tab.iloc[:-1, -1].values, index=tab.index[:-1]).sort_values()
```

```
walking                  0.524971
going down the stairs    0.633001
sitting                  0.694992
standing                 0.705142
laying                   0.800412
going up the stairs      0.804404
dtype: float64
```

实际类别为going up the stairs(上楼梯)的样本，聚类后分散程度最小，聚类效果最好；其次是躺下与站立。

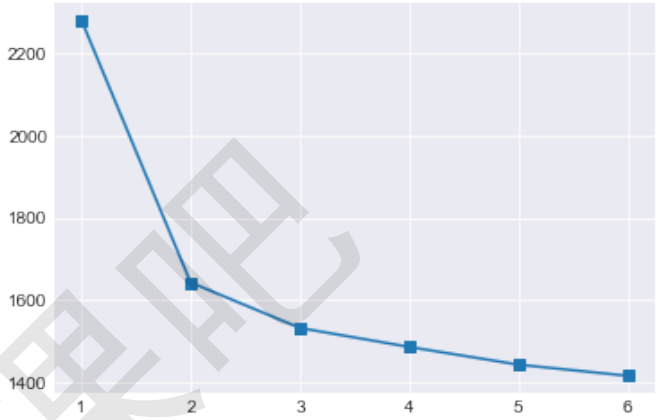
```
ts=pd.DataFrame([[1,2,3],[4,5,6]])
print(ts.max(axis=1))   # 每行的max
print(ts.max(axis=0))   # 每列的max
print(ts.max(axis=1).values/ts.iloc[:, -1].values)   # 相当于每行的max分别除以每行的最后一个数
```

```
0    3
1    6
dtype: int64
0    4
1    5
2    6
dtype: int64
[1. 1.]
```

接下来，通过求解与观察不同的K值下，各样本点到其簇质心的距离平方和，并基于肘部法则来确定适合该数据集的较优 K 取值。

```
# 聚类的 K值遍历1-6，通过观察 K-类内离差平方和的开方根 的折线图与肘部法则，选择较合适的 K值
# tqdm_notebook可以以进度条的形式查看计算的过程
inertia = []
for k in tqdm_notebook(range(1, n_classes + 1)):
    kmeans = KMeans(n_clusters=k, n_init=100,
                    random_state=RANDOM_STATE, n_jobs=1).fit(X_pca)
    inertia.append(np.sqrt(kmeans.inertia_))     # kmeans.inertia_每个点到其簇质心的距离平方和，即类内离差平方和；越小越好


plt.plot(range(1, 7), inertia, marker='s')
plt.show()
```

```
HBox(children=(IntProgress(value=0, max=6), HTML(value='')))
```



这里通过肘部法则，选择拐点对应的K值，取K=2

下面我们使用K=2来进行聚类，并且将聚类结果可视化，观察效果

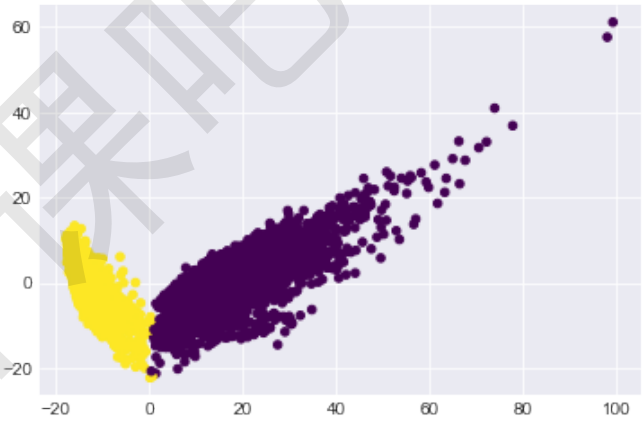
```
kmeans2 = KMeans(n_clusters=2, n_init=100,
                 random_state=RANDOM_STATE, n_jobs=1).fit(X_pca)
cluster_labels2 = kmeans2.labels_

# 查看原始类别下的样本聚类后的分散程度
tab2 = pd.crosstab(y, cluster_labels2, margins=True)
tab2.index = ['walking', 'going up the stairs',
              'going down the stairs', 'sitting', 'standing', 'laying', 'all']
tab2.columns = ['cluster' + str(i + 1) for i in range(2)] + ['all']
tab2

# 这样就很清晰了，前3大类的样本基本均聚集在第2个簇，后3大类的样本基本均聚集在第1个簇；
# 虽然一个簇里包含3个原始大类的样本，但目前就不存在原始大类下的样本分散在不同簇里的情况了。
# 所以目前这个聚类结果，虽然不能精确到每个原始大类，但是可以较准确地将前3大类的样本和后3大类的样本分开，比上面的K=6的聚类结果要来得清晰。
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	cluster1	cluster2	all
walking	1722	0	1722
going up the stairs	1536	8	1544
going down the stairs	1406	0	1406
sitting	3	1774	1777
standing	0	1906	1906
laying	12	1932	1944
all	4679	5620	10299

聚类结果的可视化

```
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=cluster_labels2, s=20, cmap='viridis')
plt.show()
```



较优K的取值方法补充

（1）肘部法则（基于图）

（2）(K=i的类内离差平方和 - K=i+1的类内离差平方和)/(K=i-1的类内离差平方和 - K=i的类内离差平方和)，选择比值最小时对应的i作为最佳的k

```
d = {}
for k in range(2, 6):
    i = k - 1
    d[k] = (inertia[i] - inertia[i + 1]) / (inertia[i - 1] - inertia[i])
```

```
d
```

```
{2: 0.17344753560094492,
 3: 0.4168830093548634,
 4: 0.9332261323524175,
 5: 0.6296969962681888}
```

这里的最佳聚类 K 值应该为 2，也就是将数据聚集为 2 类。

4.3 聚类和分类的比较

聚类	分类
无标签数据	有标签数据
不高度重视训练集	高度重视训练集
目标：找到数据的相似之处	目标：却认数据属于哪个类别
步骤：训练（聚类）	步骤：训练、测试
复杂度不高	通常很复杂

五、总结

- PCA的作用
- 协方差的计算方式
- 特征值的计算方式
- 智能设备采集项目的制作

六、作业

- 总结降维方法的使用
- 总结聚类算法