華中師範大學

自然语言处理
第七章 文本聚类

主讲教师：曾江峰

华中师范大学 信息管理学院

jfzeng@ccnu.edu.cn

文本聚类

- 本章首先介绍了文本聚类的相关概念，给出了基于 K-Means 机器学习算法的文本聚类步骤，重点介绍了 K-Means 聚类算法的原理和步骤，主成分分析方法用于数据降维，并介绍了 K-Means 的 ARI 和轮廓系数两个评估指标。最后给出英文文本和中文文本的聚类聚类实例。

语料清洗

1 概述

2 K-Means算法

3 主成分分析

4 K-Means评估指标

5 K-means英文文本聚类

6 K-Means中文文本聚类

1. 概述

- 监督学习：分类和回归

监督学习是通过现有训练数据集进行建模，再用模型对新的数据样本进行分类或者回归分析的机器学习方法。这种学习方法类似学生通过研究问题和参考答案来学习，在掌握问题和答案之间的对应关系后，就可以解决相似新问题。

1. 概述

- 监督学习：分类和回归

监督学习是指“喂”给算法的数据提前带有正确答案。正确答案在机器学习领域被称为标签（label），需要进行标注。监督学习的“输出”不同，当算法输出的是连续值，就是回归问题（Regression）。若输出是离散值，则是分类问题（classification）。

1. 概述

- 无监督学习：聚类算法

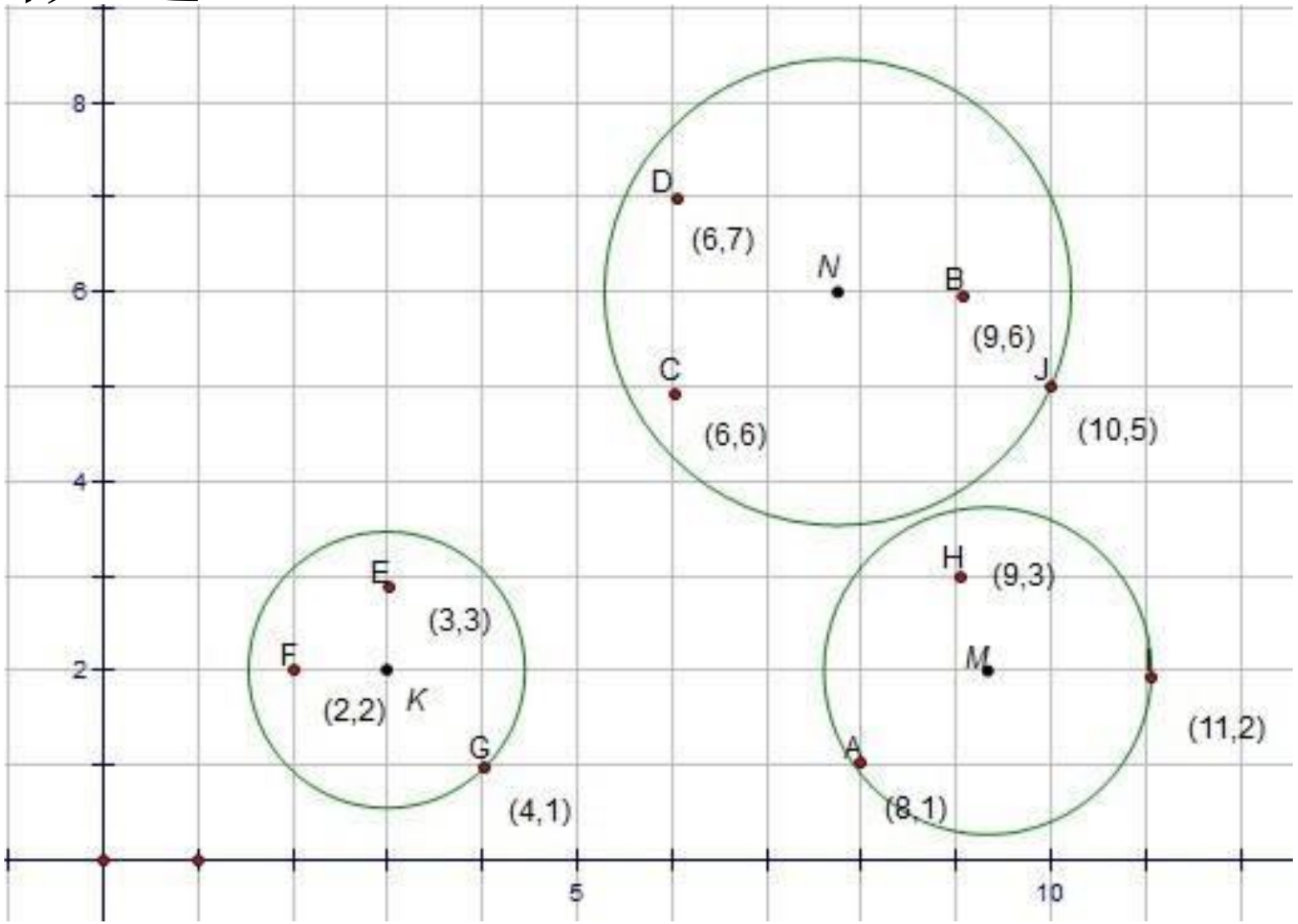
无监督学习又称为非监督学习，着重于发现数据本身的分布特点，是在没有训练数据集的情况下，对没有标签的数据进行分析并建立合适的模型，以便给出问题解决方案的方法。与监督学习不同，无监督学习不需要对数据进行标记，没有目标，因此无法从事预测任务，而适合对数据进行分析。

1. 概述

- 文本聚类算法

文本聚类（Text clustering）是将文档由原有的自然语言文字信息转化成数学信息，以高维空间点的形式展现出来，通过计算哪些点距离比较近，从而将那些点聚成一个簇，簇的中心叫做簇心。一个好的聚类要保证簇内点的距离尽量的近，但簇与簇之间的点要尽量的远。如图所示，以 K、M、N 三个点为聚类的簇心，将结果聚为三类，使得簇内点的距离尽量近，但簇与簇之间的点尽量远。

1. 概述



如图所示，以K、M、N 三个点为聚类的簇心，将结果聚为三类，使得簇内点的距离尽量近，但簇与簇之间的点尽量远。

1. 概述

- 流程

1. 语料清洗。通过Pandas等进行数据处理，采用Matplotlib进行数据可视化。

2. 特征工程。由于文本是非结构化数据，需要将向量空间模型转化成结构化形式，进行独热编码。

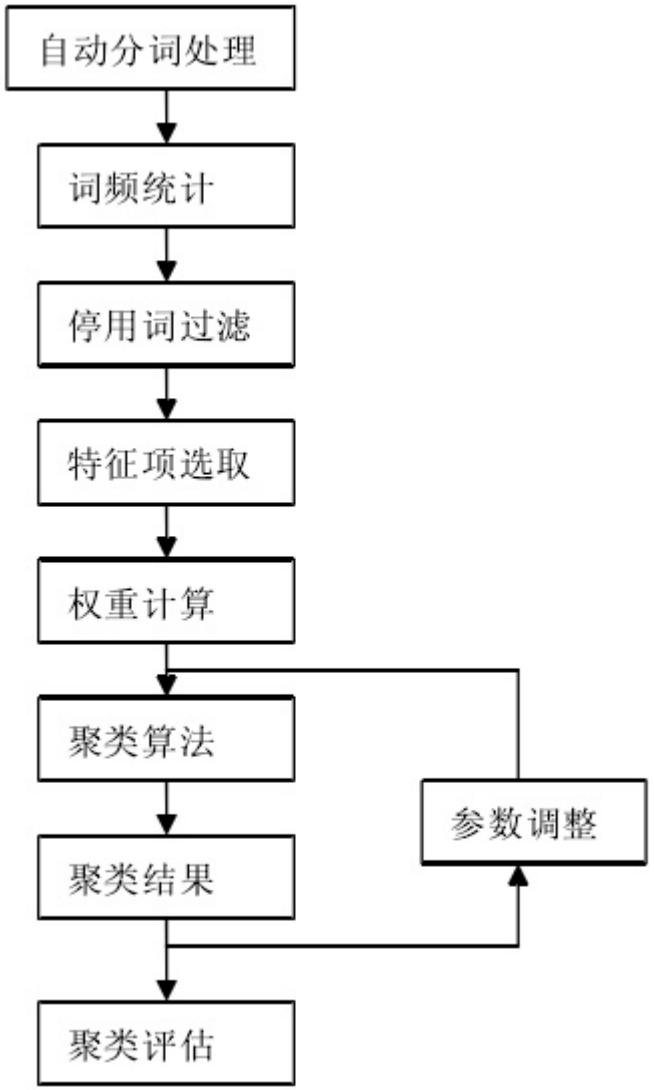
3. 分词处理。采用jieba等分词库对文本进行分词，采用停止词过滤无用信息。

4. 聚类算法。采用K-Means算法进行聚类，由于数据高维聚类效果较差，因此需采用主成分分析进行降维处理。

5. 聚类评估。通过计算ARI、轮廓系数等，进行聚类效果的评估。

1. 概述

- 流程



《自然语言处理入门》

2. k-means算法

2.1 算法原理

K-Means 由 Stuart Lloyd 于1957年提出，是一种迭代求解的聚类分析算法。K-Means 通过样本之间的距离的均值，把相似度高的样本聚成一簇，形成不同的类别。该算法最大的特点是简单，便于理解，运算速度快，但是只能应用于连续型数据，并且必须在聚类前指定分类数。

2. k-means算法

2.1 算法原理

实现 K-Means 聚类算法的步骤如下。

步骤1：确定最终聚类数，给出 K 值。

步骤2：随机选定 K 个值，计算每一个样本到 K 个值的距离，将样本点归到最相似的类中，分成 K 个簇，产生“质心”——簇中所有数据的均值。

步骤3：反复计算 K 个簇的质心，直到质心不再改变，最终确定每个样本所属的类别以及每个类的质心。

2. k-means算法

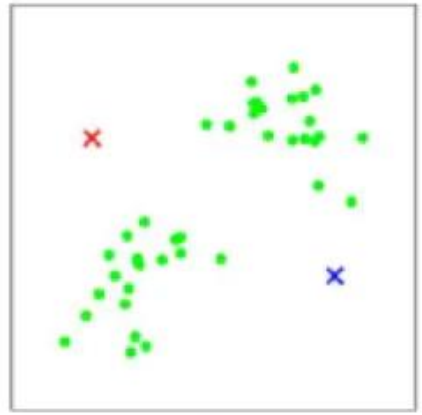
2.2 数学理论实现

K-Means 算法运行步骤如下所示。

步骤1：初始数据集如图(a)所示，确定 k=2。

步骤2：随机选择两个点作为质心——圆点和叉号，如图(b)所示。

(a)



(b)

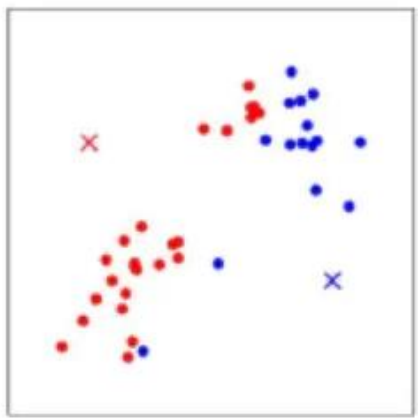
《自然语言处理入门》

2. k-means算法

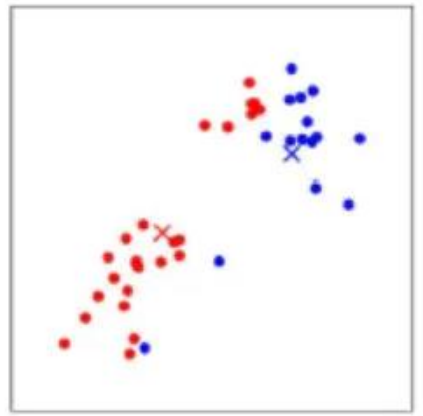
2.2 数学理论实现

步骤3：计算样本与圆点质心和叉号质心的距离，标记每个样本的类别，如图(c)所示。

步骤4：反复迭代，标记圆点和叉号各自新的质心，如图(d)所示。



(c)



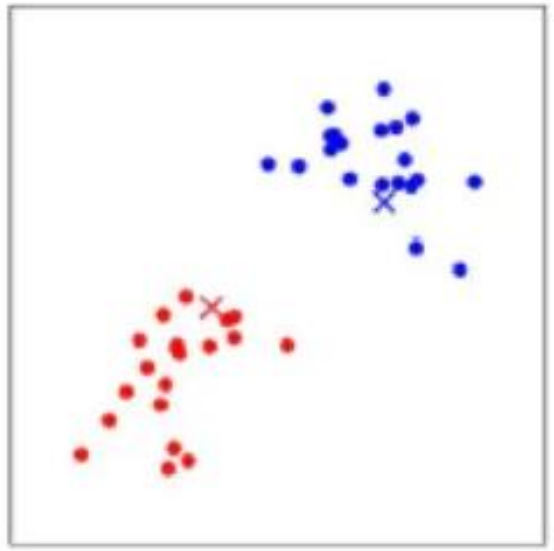
(d)

《自然语言处理入门》

2. k-means算法

2.2 数学理论实现

步骤5：直到质心不再改变，最终得到两个类别，如图(e)所示。



(e)

《自然语言处理入门》

2. k-means算法

2.2 数学理论实现

【例1】K-Means实现过程

平面上分别有O_1, O_2, O_3, O_4, O_5 五个点，其坐标x值与y值如右所示。

坐标点	x	y
O_1	0	2
O_2	0	0
O_3	1.5	0
O_4	5	0
O_5	5	2

设定聚类 K=2，即类别分别为C_1和C_2，其计算步骤如下。

2. k-means算法

2.2 数学理论实现

【例1】K-Means实现过程

步骤1：随意选择两个点为簇中心。

不妨选择$O_1(0,2)$和$O_2(0,0)$为中心，$M_1 = O_1$，$M_2 = O_2$

坐标点	x	y
O_1	0	2
O_2	0	0
O_3	1.5	0
O_4	5	0
O_5	5	2

2. k-means算法

2.2 数学理论实现

【例1】K-Means实现过程

步骤2：计算每个点与簇中心的距离，将其赋给最近的簇。

对于O_3：
$$\begin{cases} d(M_1, O_3) = \sqrt{(0-1.5)^2+(2-0)^2} = 2.5 \\ d(M_2, O_3) = \sqrt{(0-1.5)^2+(0-0)^2} = 1.5 \end{cases}$$

$d(M_1, O_3) \geq d(M_2, O_3)$，故将$O_3$分配给$C_2$

坐标点	x	y
O_1	0	2
O_2	0	0
O_3	1.5	0
O_4	5	0
O_5	5	2

2. k-means算法

2.2 数学理论实现

【例1】K-Means实现过程

步骤2：计算每个点与簇中心的距离，将其赋给最近的簇。

对于O_4：
$$\begin{cases} d(M_1, O_4) = \sqrt{(0-5)^2+(2-0)^2} = \sqrt{29} \\ d(M_2, O_4) = \sqrt{(0-5)^2+(0-0)^2} = 5 \end{cases}$$

$d(M_1, O_4) \geq d(M_2, O_4)$，故将$O_4$分配给$C_2$

坐标点	x	y
O_1	0	2
O_2	0	0
O_3	1.5	0
O_4	5	0
O_5	5	2

2. k-means算法

2.2 数学理论实现

【例1】K-Means实现过程

步骤2：计算每个点与簇中心的距离，将其赋给最近的簇。

对于O_5：
$$\begin{cases} d(M_1, O_5) = \sqrt{(0-5)^2+(2-2)^2} = 5 \\ d(M_2, O_5) = \sqrt{(0-5)^2+(0-2)^2} = \sqrt{29} \end{cases}$$

$d(M_1, O_5) \leq d(M_2, O_4)$，故将$O_5$分配给$C_1$

坐标点	x	y
O_1	0	2
O_2	0	0
O_3	1.5	0
O_4	5	0
O_5	5	2

2. k-means算法

2.2 数学理论实现

【例1】K-Means实现过程

步骤2：计算每个点与簇中心的距离，将其赋给最近的簇。

此时，得到新簇$C_1 = \{O_1, O_5\}$，$C_2 = \{O_2, O_3, O_4\}$

坐标点	x	y
O_1	0	2
O_2	0	0
O_3	1.5	0
O_4	5	0
O_5	5	2

2. k-means算法

2.2 数学理论实现

【例1】K-Means实现过程

步骤3：计算新的簇中心。

新簇$C_1 = \{O_1, O_5\}$

$$C_2 = \{O_2, O_3, O_4\}$$

$$M_1 = (\frac{0+5}{2}, \frac{2+2}{2}) = (2.5, 2)$$

$$M_2 = (\frac{0+1.5+5}{3}, \frac{0+0+0}{2}) = (2.17, 0)$$

坐标点	x	y
O_1	0	2
O_2	0	0
O_3	1.5	0
O_4	5	0
O_5	5	2

2. k-means算法

2.2 数学理论实现

【例1】K-Means实现过程

步骤4：计算每个点与新簇中心的距离，将其赋给最近的簇，重复步骤2和步骤3。

对于O_3：
$$\begin{cases} d(M_1, O_3) = \sqrt{(2.5-1.5)^2+(2-0)^2} = \sqrt{5} \\ d(M_2, O_3) = \sqrt{(2.17-1.5)^2+(0-0)^2} = \sqrt{0.4489} \end{cases}$$

$d(M_1, O_3) \geq d(M_2, O_3)$，故将$O_3$分配给$C_2$

以此类推.....

坐标点	x	y
O_1	0	2
O_2	0	0
O_3	1.5	0
O_4	5	0
O_5	5	2

2. k-means算法

2.2 数学理论实现

【例1】K-Means实现过程

由于迭代的簇中心不变，聚类为

$$C_1 = \{O_1, O_5\}$$

$$C_2 = \{O_2, O_3, O_4\}$$

坐标点	x	y
O_1	0	2
O_2	0	0
O_3	1.5	0
O_4	5	0
O_5	5	2

2. k-means算法

2.2 Python实现

【例2】Python实现 K-Means 算法举例

```
# 例2: Python实现K-Means算法举例
import numpy as np
import matplotlib.pyplot as plt
import random
def get_distance(p1, p2):
    diff = [x-y for x, y in zip(p1, p2)]
    distance = np.sqrt(sum(map(lambda x: x**2, diff)))
    return distance
# 计算多个点的中心
# cluster = [[1,2,3], [-2,1,2], [9, 0 ,4], [2,10,4]]
def calc_center_point(cluster):
    N = len(cluster)
    m = np.matrix(cluster).transpose().tolist()
    center_point = [sum(x)/N for x in m]
    return center_point
```

```
# 检查两个点是否有差别
def check_center_diff(center, new_center):
    n = len(center)
    for c, nc in zip(center, new_center):
        if c != nc:
            return False
    return True
```


2. k-means算法

2.2 Python实现

【例2】Python实现 K-Means 算法举例

```
# K-means算法的实现
def K_means(points, center_points):
    N = len(points)   # 样本个数
    n = len(points[0])   # 单个样本的维度
    k = len(center_points)   # k值大小
    tot = 0
    while True:   # 迭代
        temp_center_points = []   # 记录中心点
        clusters = []   # 记录聚类的结果
        for c in range(0, k):
            clusters.append([])   # 初始化
        # 针对每个点，寻找距离其最近的中心点（寻找组织）
        for i, data in enumerate(points):
            distances = []
            for center_point in center_points:
                distances.append(get_distance(data, center_point))
            index = distances.index(min(distances))   # 找到最小距离的那个中心点的索引
            clusters[index].append(data)    # 中心点代表的簇，里面增加一个样本
        tot += 1
        # print(tot, '次迭代 ', clusters)
        k = len(clusters)
        colors = ['r.', 'g.', 'b.', 'k.', 'y.']    # 颜色和点的样式
```


2. k-means算法

2.2 Python实现

【例2】Python实现 K-Means 算法举例

```
        for i, cluster in enumerate(clusters):
            data = np.array(cluster)
            data_x = [x[0] for x in data]
            data_y = [x[1] for x in data]
            plt.subplot(2, 3, tot)
            plt.plot(data_x, data_y, colors[i])
            plt.axis([0, 1000, 0, 1000])
```

```
        # 重新计算中心点
        for cluster in clusters:
            temp_center_points.append(calc_center_point(cluster))
        # 在计算中心点的时候，需要将原来的中心点算进去
        for j in range(0, k):
            if len(clusters[j]) == 0:
                temp_center_points[j] = center_points[j]
        # 判断中心点是否发生变化
        for c, nc in zip(center_points, temp_center_points):
            if not check_center_diff(c, nc):
                center_points = temp_center_points[:]   # 复制一份
                break
        else:   # 如果没有变化，退出迭代，聚类结束
            break
    plt.show()
    return clusters   # 返回聚类的结果
```


2. k-means算法

2.2 Python实现

【例2】Python实现 K-Means 算法举例

```
# 随机获取一个样本集，用于测试K-Means算法
def get_test_data():
    N = 1000
    # 产生点的区域
    area_1 = [0, N / 4, N / 4, N / 2]
    area_2 = [N / 2, 3 * N / 4, 0, N / 4]
    area_3 = [N / 4, N / 2, N / 2, 3 * N / 4]
    area_4 = [3 * N / 4, N, 3 * N / 4, N]
    area_5 = [3 * N / 4, N, N / 4, N / 2]
    areas = [area_1, area_2, area_3, area_4, area_5]
    k = len(areas)
    # 在各个区域内，随机产生一些点
    points = []
    for area in areas:
        rnd_num_of_points = random.randint(50, 200)
        for r in range(0, rnd_num_of_points):
            rnd_add = random.randint(0, 100)
            rnd_x = random.randint(area[0] + rnd_add, area[1] - rnd_add)
            rnd_y = random.randint(area[2], area[3] - rnd_add)
            points.append([rnd_x, rnd_y])
    # 自定义中心点，目标聚类个数为5，因此选定5个中心点
    center_points = [[0, 250], [500, 500], [500, 250], [500, 250], [500, 750]]
    return points, center_points
```


2. k-means算法

2.2 Python实现

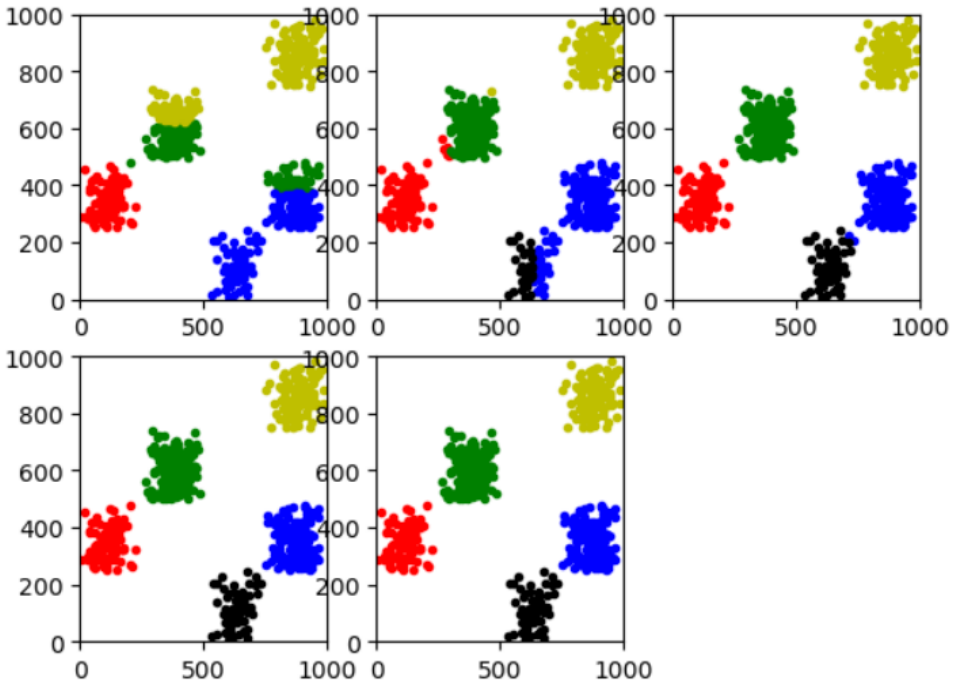
【例2】Python实现 K-Means 算法举例

```
if __name__ == '__main__':
    points, center_points = get_test_data()
    clusters = K_means(points, center_points)
    # print('########最终结果###########')
    # for i, cluster in enumerate(clusters):
    #     print('cluster ', i, ' ', cluster)
```


2. k-means算法

2.2 Python实现

【例2】Python实现 K-Means 算法举例



《自然语言处理入门》

3. 主成分分析

3.1 算法原理

在多变量的问题中，变量之间往往存在信息重叠，通过正交变换将相关性转换为线性不相关，转换后的变量称为主成分。主成分分析（Principal Component Analysis, PCA）是找出数据中最主要的特征代替原有数据，保持原有数据的方差信息，通常用于高维数据的降维、数据压缩和预处理。

3. 主成分分析

3.1 算法原理

PCA 具有如下优点。

- 仅以方差衡量信息量。
- 各主成分之间正交，可消除原始数据成分间相互影响的因素。
- 计算方法简单，主要运算是特征值分解，易于实现。

3. 主成分分析

3.1 算法原理

PCA 算法的主要缺点如下。

- 主成分各个特征维度的含义具有一定的模糊性。
- 非主成分也可能含有对样本差异的重要信息。

3. 主成分分析

3.2 components参数

Sklearn提供 decomposition.PCA 用于主成分分析。

PCA(n_components=n)

- n_components参数取值有小数和整数之分。小数表示保留百分之多少的信息。整数表示减少到多少特征。

3. 主成分分析

3.2 components参数

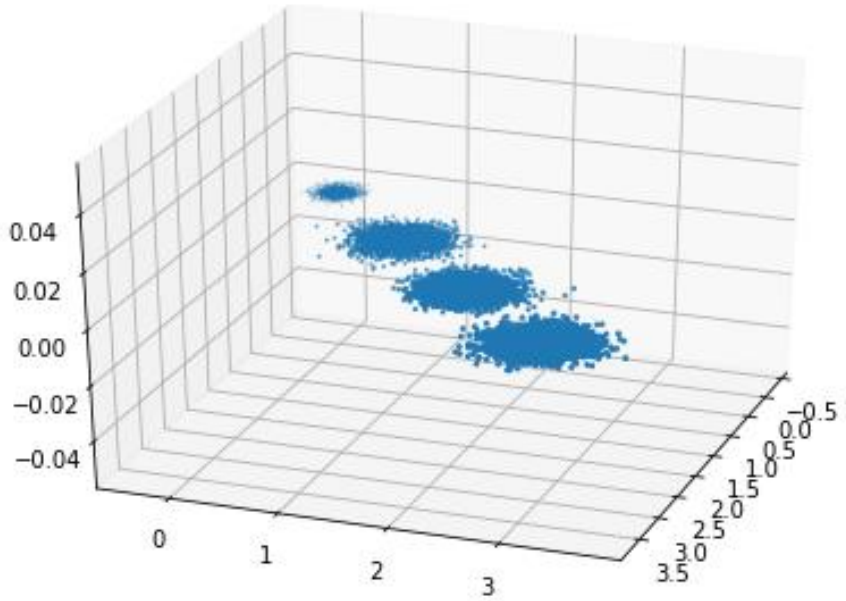
【例3】n_components举例

```
# 例3：n_components举例
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.datasets import make_blobs
# make_blobs：多类单标签数据集，为每个类分配一个或多个正态分布的点集
# X为样本特征，Y为样本簇类别，共1000个样本，每个样本3个特征，共4个簇
X, y = make_blobs(n_samples=10000,
                  n_features=3,
                  centers=[[3,3,3],[0,0,0], [1,1,1], [2,2,2]],
                  cluster_std=[0.2, 0.1, 0.2, 0.2],
                  random_state=9)
fig = plt.figure()
ax = Axes3D(fig, rect=[0, 0, 1, 1], elev=30, azim=20)
plt.scatter(X[:, 0], X[:, 1], X[:, 2],marker='o')
```


3. 主成分分析

3.2 components参数

【例3】n_components举例



3. 主成分分析

3.2 components参数

【例3】n_components举例

```
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
pca.fit(X)
print(pca.explained_variance_ratio_)
print(pca.explained_variance_)
```

```
[0.98318212 0.00850037 0.00831751]
[3.78521638 0.03272613 0.03202212]
```


3. 主成分分析

3.2 components参数

【例3】n_components举例

```
# 下面采用PCA进行特征降维
```

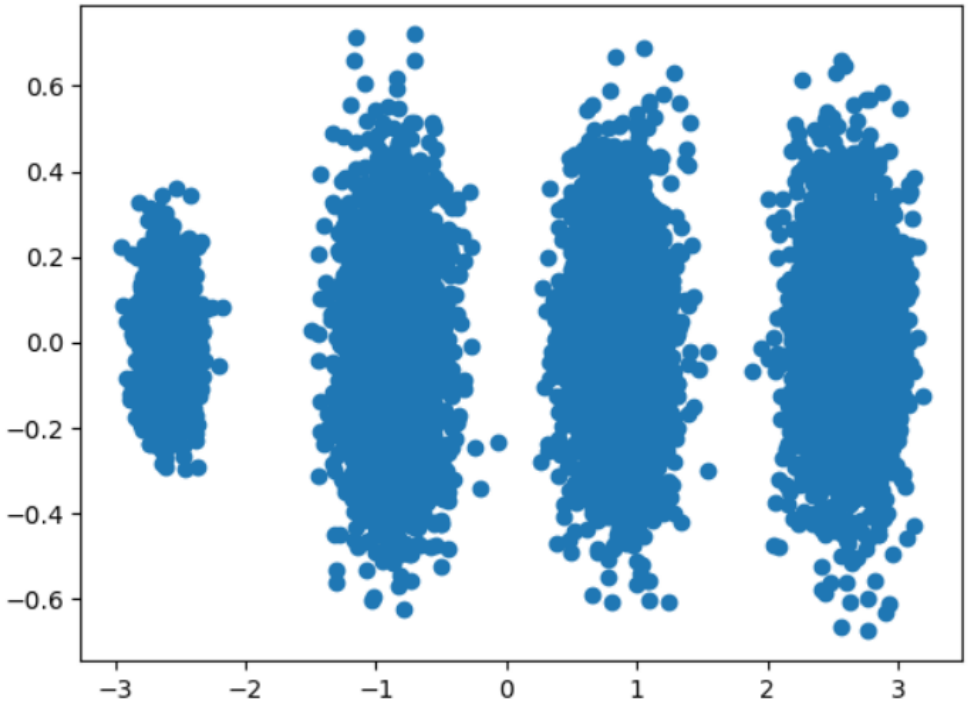
```
# 情况一：n_components取整数
# 从3维降到2维，选择前两个特征，而抛弃第三个特征。
from sklearn.decomposition import PCA
pca = PCA(n_components=2)    #减少到2个特征
pca.fit(X)
print(pca.explained_variance_ratio_)
print(pca.explained_variance_)
X_new = pca.transform(X)
plt.scatter(X_new[:, 0], X_new[:, 1],marker='o')    # 将转化后的数据分布可视化
plt.show()
```

```
[0.98318212 0.00850037]
[3.78521638 0.03272613]
```


3. 主成分分析

3.2 components参数

【例3】n_components举例



《自然语言处理入门》

3. 主成分分析

3.2 components参数

【例3】n_components举例

```
# 情况二：n_components取小数
# 保存95%的信息。
pca = PCA(n_components=0.95)     #保留95%的信息
pca.fit(X)
print(pca.explained_variance_ratio_)
print(pca.explained_variance_)
print(pca.n_components_)
```

```
[0.98318212]
[3.78521638]
1
```


3. 主成分分析

3.3 对鸢尾花数据进行降维

【例4】使用Sklearn实现对鸢尾花数据集降维，将原先四维特征数据集降维为二维

```
# 例4:使用Sklearn实现对鸢尾花数据集降维，将原先四维特征数据集降维为二维
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn import datasets
iris = datasets.load_iris()
x = iris.data
y = iris.target
# print(x)

# 加载PCA算法，设置降维后主成分数目为2
pca = PCA(n_components=2)
reduced_x = pca.fit_transform(x)
# print(reduced_x)
red_x,red_y = [],[]
blue_x,blue_y = [],[]
green_x,green_y = [],[]
```

3. 主成分分析

3.3 对鸢尾花数据进行降维

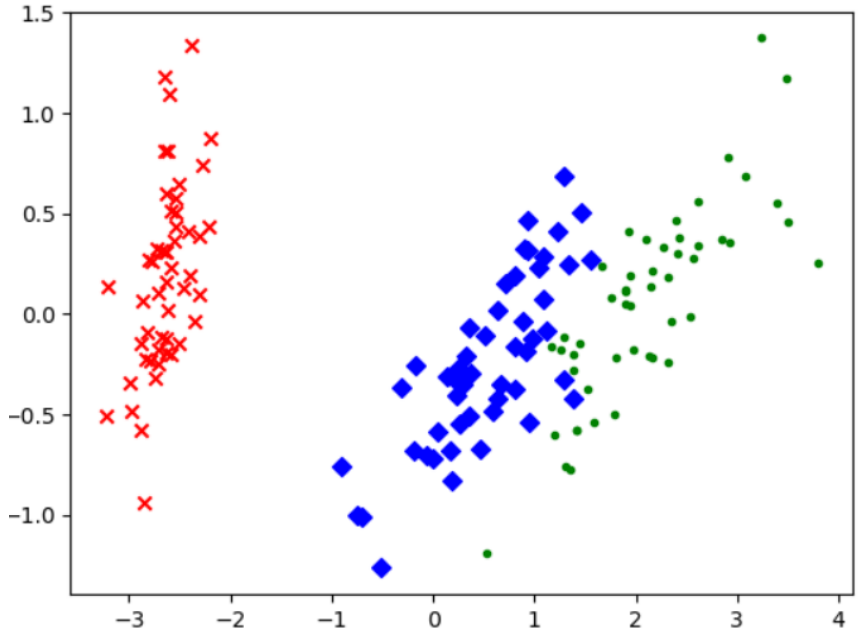
【例4】使用Sklearn实现对鸢尾花数据集降维，将原先四维特征数据集降维为二维

```
for i  in range(len(reduced_x)):
    if  y[i] == 0:
        red_x.append(reduced_x[i][0])
        red_y.append(reduced_x[i][1])
    elif y[i] == 1:
        blue_x.append(reduced_x[i][0])
        blue_y.append(reduced_x[i][1])
    else:
        green_x.append(reduced_x[i][0])
        green_y.append(reduced_x[i][1])
plt.scatter(red_x,red_y,c = 'r',marker = 'x')
plt.scatter(blue_x,blue_y,c = 'b',marker = 'D')
plt.scatter(green_x,green_y,c = 'g',marker = '.')
plt.show()
```


3. 主成分分析

3.3 对鸢尾花数据进行降维

【例4】使用Sklearn实现对鸢尾花数据集降维，将原先四维特征数据集降维为二维



《自然语言处理入门》

4. K-Means评估指标

Sklearn 提供调整兰德系数（Adjusted Rand Index, ARI）和轮廓系数用于评价 K-Means 的性能，确定最佳 K 值。

4. K-Means评估指标

4.1 调整兰德系数

当数据具有所属类别，采用调整兰德系数（Adjusted Rand Index,ARI）指标来评价 K-Means 的性能。ARI 取值范围为[-1,1]，值越大意味着聚类结果与真实情况越吻合。

Sklearn 提供了adjusted_rand_score()函数计算 ARI。

adjusted_rand_score(y_test,y_pred)

- y_true ： 真实值。
- y_pred ： 预测值。

4. K-Means评估指标

4.1 调整兰德系数

【例5】ARI举例

```
# 例5: ARI举例
from sklearn.metrics import adjusted_rand_score
y_true = [3,-0.5,2,7]
y_pred = [2.5,0.0,2,8]
print(adjusted_rand_score(y_true,y_pred))
```

```
1.0
```


4. K-Means评估指标

4.2 轮廓系数

当数据没有所属类别时，使用轮廓系数（Silhouette Coefficient）度量聚类效果。轮廓系数兼顾聚类的凝聚度和分离度，取值范围为[-1,1]，数值越大，聚类效果越好。

对于任意点 i 的轮廓系数，数学公式如下。

$$S(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}$$

4. K-Means评估指标

4.2 轮廓系数

$$S(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}$$

$a(i)$是指点 i 到所有簇内其他点的距离的平均值。

$b(i)$是指点 i 到与相邻最近的一簇内的所有点的平均距离的最小值。

4. K-Means评估指标

4.2 轮廓系数

步骤1：对于已聚类数据中的第 i 个样本 $X(i)$ ，计算 $X(i)$ 与其同一个类簇中的所有其他样本距离的平均值，记作 $a(i)$ ，用于量化簇内的凝聚度。

步骤2：选取 $X(i)$ 外的一个簇 b ，计算 $X(i)$ 与簇 b 中所有样本的平均距离，遍历所有其他簇，找到最近平均距离的那个簇，记作 $b(i)$ ，用于量化簇间的分离度。

步骤3：对于样本 $X(i)$ ，计算轮廓系数 $S(i)$ 。

4. K-Means评估指标

4.2 轮廓系数

由轮廓系数 $S(i)$ 的计算公式可知，如果 $S(i)$ 小于0，说明 $X(i)$ 与其簇内元素的平均距离大于最近的其他簇，表示簇类效果不好；如果 $a(i)$ 趋于0，或者 $b(i)$ 足够大，那么 $S(i)$ 趋近1，说明聚类效果比较好。

$$S(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}$$

4. K-Means评估指标

4.2 轮廓系数

Sklearn提供了silhouette_score()计算所有点的平均轮廓系数。

silhouette_score(X,labels)

- X：特征值。
- labels：被聚类标记的目标值。

4. K-Means评估指标

4.2 轮廓系数

【例6】轮廓系数

```
# 例6：轮廓系数
# 生成数据模块
from sklearn.datasets import make_blobs
# k-means模块
from sklearn.cluster import KMeans
# 评估指标—轮廓系数,前者为所有点的平均轮廓系数，后者返回每个点的轮廓系数
from sklearn.metrics import silhouette_score, silhouette_samples
import numpy as np
import matplotlib.pyplot as plt
# 生成数据
x_true, y_true = make_blobs(n_samples= 600 , n_features= 2, centers= 4, random_state= 1)

# 绘制出所生成的数据
plt.figure(figsize= (6, 6))
plt.scatter(x_true[:, 0], x_true[:, 1], c= y_true, s= 10)
plt.title("Origin data")
plt.show()
```


4. K-Means评估指标

4.2 轮廓系数

【例6】轮廓系数

```
# 根据不同的n_centers进行聚类
n_clusters = [x for x in range(3, 6)]
for i in range(len(n_clusters)):
    # 实例化k-means分类器
    clf = KMeans(n_clusters= n_clusters[i])
    y_predict = clf.fit_predict(x_true)

    # 绘制分类结果
    plt.figure(figsize= (6, 6))
    plt.scatter(x_true[:, 0], x_true[:, 1], c= y_predict, s= 10)
    plt.title("n_clusters= {}".format(n_clusters[i]))
    ex = 0.5
    step = 0.01
    xx, yy = np.meshgrid(np.arange(x_true[:, 0].min() - ex, x_true[:, 0].max() + ex, step),
                         np.arange(x_true[:, 1].min() - ex, x_true[:, 1].max() + ex, step))
    zz = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    zz.shape = xx.shape

    plt.contourf(xx, yy, zz, alpha= 0.1)
    plt.show()
```


4. K-Means评估指标

4.2 轮廓系数

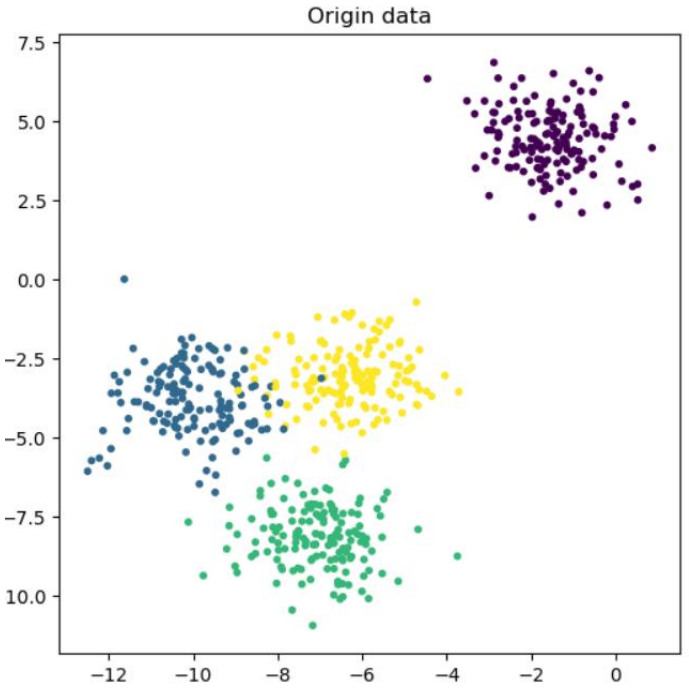
【例6】轮廓系数

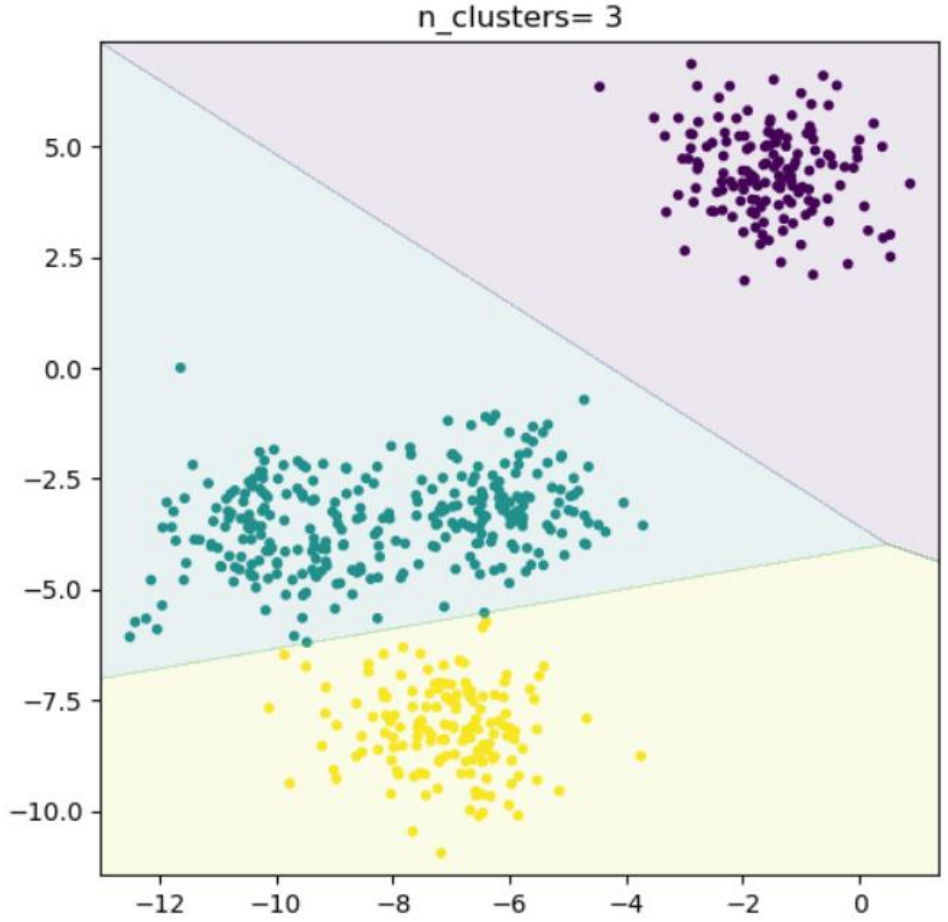
```
    # 打印平均轮廓系数
    s = silhouette_score(x_true, y_predict)
    print("When cluster = {}\nThe silhouette_score = {}".format(n_clusters[i], s))

# silhouette_samples()返回每个点的轮廓系数，计算轮廓系数为正的点的个数。
    n_s_bigger_than_zero = (silhouette_samples(x_true, y_predict) > 0).sum()
    print("{}/{}\n".format(n_s_bigger_than_zero, x_true.shape[0]))
```


4. K-Means评估指标

4.2 轮廓系数

【例6】轮廓系数



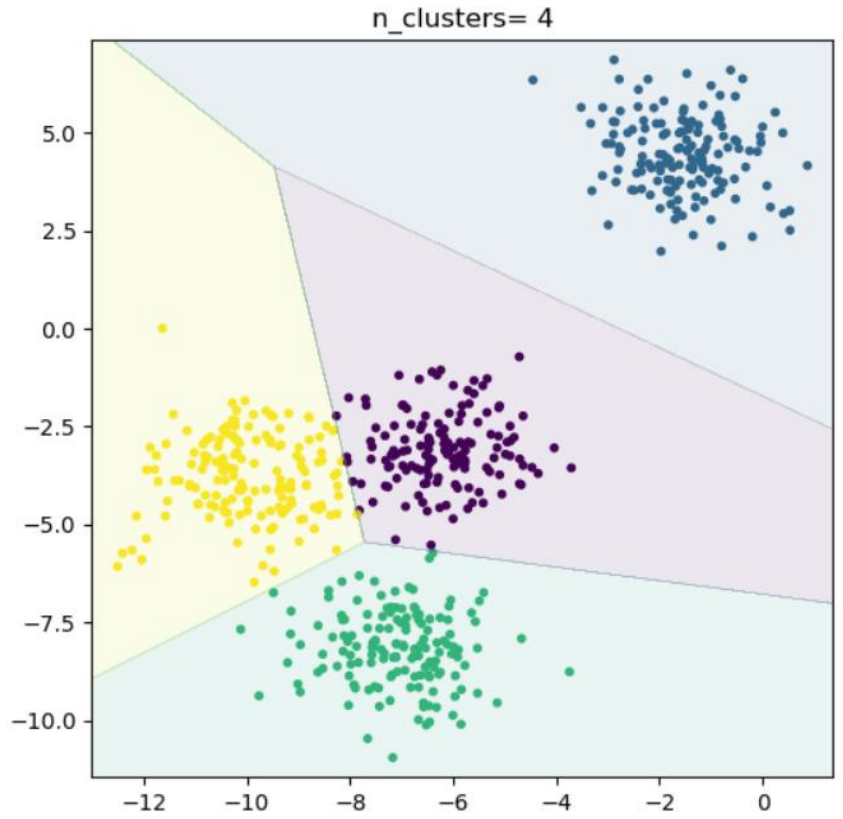



```
When cluster = 3
The silhouette_score = 0.6009420412542107
595/600
```

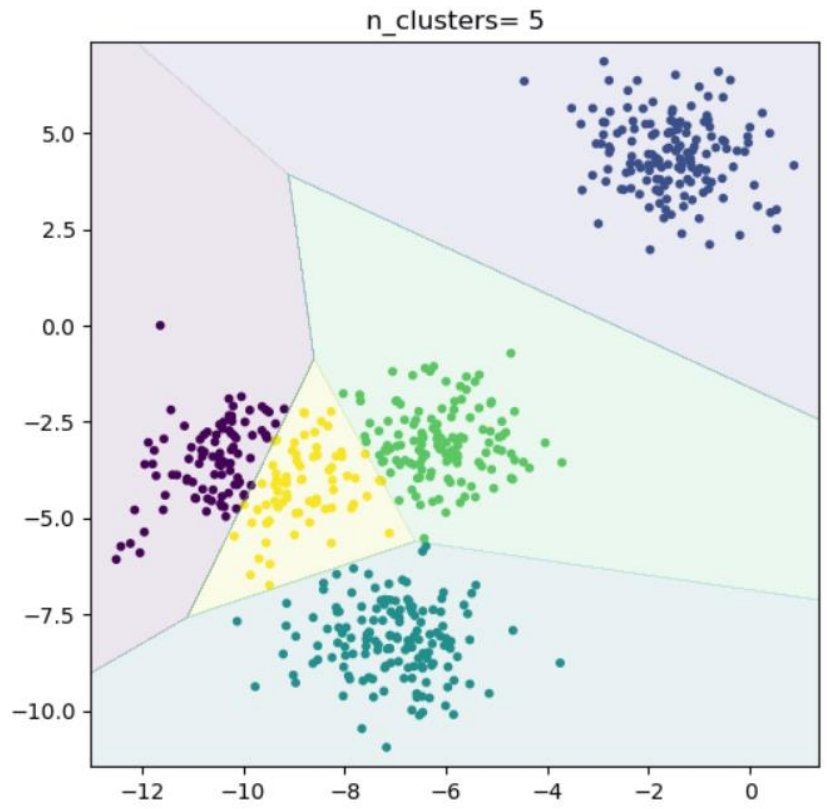
4. K-Means评估指标

4.2 轮廓系数

【例6】轮廓系数




```
When cluster = 4
The silhouette_score = 0.637556444143356
599/600
```




```
When cluster = 5
The silhouette_score = 0.5604812245680646
598/600
```


5.K-Means英文文本聚类

【例7】K-Means英文文本聚类

第一步：使用 Pandas 的 DataFrame 进行数据格式化。

第二步：使用 NLTK 的语料库对英文文本进行分词和去除停止词。

第三步：使用 np.vectorizer 进行向量化处理。

第四步：使用 Tfidfvectorizer 构造词袋模型。

第五步：使用 cosine_similarity 计算余弦相似度，构造相关性矩阵。

第六步：使用 K-Means 算法进行聚类操作。

5. K-Means英文文本聚类

【例7】K-Means英文文本聚类

```
import pandas as pd
import numpy as np
import re       #正则表达式
import nltk
corpus = ['The sky is blue and beautiful.',
          'Love this blue and beautiful sky!',
          'The quick brown fox jumps over the lazy dog.',
          'The brown fox is quick and the blue dog is lazy!',
          'The sky is very blue and the sky is very beautiful today',
          'The dog is lazy but the brown fox is quick!'
]
labels = ['weather', 'weather', 'animals', 'animals', 'weather', 'animals']

corpus = np.array(corpus)
corpus_df = pd.DataFrame({'Document': corpus, 'categoray': labels})
```


5. K-Means英文文本聚类

【例7】K-Means英文文本聚类

```
# 载入英文的停用词表
stopwords = nltk.corpus.stopwords.words('english')
# 建立词分割模型
cut_model = nltk.WordPunctTokenizer()
# 定义分词和停用词去除的函数
def Normalize_corpus(doc):
    # 去除字符串中结尾的标点符号
    doc = re.sub(r'[^a-zA-Z0-9\s]', '', string = doc)
    # 字符串变小写格式
    doc = doc.lower()
    # 去除字符串两边的空格
    doc = doc.strip()
    # 进行分词操作
    tokens = cut_model.tokenize(doc)
    # 使用停止用词表去除停用词
    doc = [token for token in tokens if token not in stopwords]
    # 将去除停用词后的字符串使用' '连接，为了接下来的词袋模型做准备
    doc = ' '.join(doc)
    return doc
```

《自然语言处理入门》

5. K-Means英文文本聚类

【例7】K-Means英文文本聚类

```
Normalize_corpus = np.vectorize(Normalize_corpus)
corpus_norm = Normalize_corpus(corpus)

# TF-idf模型
from sklearn.feature_extraction.text import TfidfVectorizer
Tf = TfidfVectorizer(use_idf=True)
Tf.fit(corpus_norm)
vocs = Tf.get_feature_names_out()
corpus_array = Tf.transform(corpus_norm).toarray()
corpus_norm_df = pd.DataFrame(corpus_array, columns=vocs)
# print(corpus_norm_df)
```


5. K-Means英文文本聚类

【例7】K-Means英文文本聚类

```
#计算余弦相似度
#使用cosine_similarity计算余弦相似度 ， 构造相关性矩阵
from sklearn.metrics.pairwise import cosine_similarity
similarity_matrix = cosine_similarity(corpus_array)
similarity_matrix_df = pd.DataFrame(similarity_matrix)

# Kmeans聚类
from sklearn.cluster import KMeans
model = KMeans(n_clusters=2)      #聚类为2
model.fit(np.array(similarity_matrix))
print(model.labels_)
corpus_norm_df['k_labels'] = np.array(model.labels_)
print(corpus_norm_df)
```


5. K-Means英文文本聚类

【例7】K-Means英文文本聚类

```
[1 1 0 0 1 0]
   beautiful      blue     brown       dog       fox     jumps      lazy  \
0   0.604749  0.518224  0.000000  0.000000  0.000000  0.000000  0.000000
1   0.455454  0.390289  0.000000  0.000000  0.000000  0.000000  0.000000
2   0.000000  0.000000  0.375653  0.375653  0.375653  0.542607  0.375653
3   0.000000  0.357850  0.417599  0.417599  0.417599  0.000000  0.417599
4   0.357583  0.306421  0.000000  0.000000  0.000000  0.000000  0.000000
5   0.000000  0.000000  0.447214  0.447214  0.447214  0.000000  0.447214

       love     quick       sky     today  k_labels
0  0.000000  0.000000  0.604749  0.000000         1
1  0.657873  0.000000  0.455454  0.000000         1
2  0.000000  0.375653  0.000000  0.000000         0
3  0.000000  0.417599  0.000000  0.000000         0
4  0.000000  0.000000  0.715166  0.516505         1
5  0.000000  0.447214  0.000000  0.000000         0
```


6. K-Means中文文本聚类

【例8】K-Means中文文本聚类

“搜狗新闻语料库”具有15个类别，本例只取出语料库中前两类4000条数据作为聚类样本，保存在sougou_train_data.txt，聚成两类。

6.K-Means中文文本聚类

【例8】K-Means中文文本聚类

步骤1：先对语料库进行分词

步骤2：读取语料库分词结果

步骤3：tfidf向量化，svd降维

步骤4：K-Means聚类

6. K-Means中文文本聚类

【例8】K-Means中文文本聚类

```
# 数据加载，加载语料库，停用词，语料库已分词文本
import pandas as pd
import numpy as np
print("读取数据集")
train_df = pd.read_csv("C:\\Users\Administrator\\自然语言处理\\sougou_train_data.txt",
                       error_bad_lines = False,
                       sep = '\t',
                       header = None,
                       index_col = 0)
train_df = train_df[:4000]   # 选取语料库中十二类文本中前两类作为聚类语料库
# 分词
print('开始分词')
import jieba
import time
train_df = pd.DataFrame(columns = ['label', '文章'])
stopword_list = [k.strip() for k in open("C:\\Users\\Administrator\\自然语言处理\\stopwords-master\\hit_stopwords.txt",
                                         encoding = 'utf8',).readlines() if k.strip() != '']
cutWords_list = []
i = 0
startTime = time.time()
```


6. K-Means中文文本聚类

【例8】K-Means中文文本聚类

```
for article in train_df['文章']:
    cutWords = [k for k in jieba.cut(article) if k not in stopword_list]
    i += 1
    if i % 1000 == 0:
        print('前%d篇文章分词共花费%.2f秒' %(i, time.time()-startTime))
    cutWords_list.append(cutWords)

# 读取分词结果
print('读取分词结果')
with open("C:\\Users\\Administrator\\自然语言处理\\cutWords_list.txt", encoding='utf8') as file:
    cutWords_list = [k.split() for k in file.readlines()]
with open("C:\\Users\\Administrator\\自然语言处理\\stopwords-master\\hit_stopwords.txt", encoding='utf8') as file:
    stopWord_list = [k.strip() for k in file.readlines()]
```


6. K-Means中文文本聚类

【例8】K-Means中文文本聚类

```
# 处理分词格式
print('处理分词格式')
out = ''
i = 0
cutwords_list = []
for cutWords in cutWords_list[:4000]:    # 选取语料库中十二类文本中前两类作为聚类语料库
    out = ''
    for word in cutWords:
        if word not in stopWord_list:
            out += word
            out += " "
    cutwords_list.append(out)
    i = i + 1

cutWords_list = cutwords_list

a = cutWords_list
```


6. K-Means中文文本聚类

【例8】K-Means中文文本聚类

```
# 对分词文本进行tfidf向量化
print('对分词文本进行tfidf向量化')
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(stop_words = stopWord_list, min_df = 40, max_df = 0.3)
tfidf_model = tfidf.fit(a)   # 低级失误train_df[1]
X = tfidf_model.transform(a)
print('词表大小:', len(tfidf.vocabulary_))
print(X.shape)
sentence_vec_sif = X
```


6. K-Means中文文本聚类

【例8】K-Means中文文本聚类

```
# 对tfidf得到的向量矩阵进行压缩，降维（潜在语义分析）
print('对tfidf得到的向量矩阵进行压缩，降维（潜在语义分析）')
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Normalizer
from sklearn.decomposition import TruncatedSVD  # 奇异值分解(SVD)用于降维
svd = TruncatedSVD(n_components = 2000)   # will extract 10 "topics"
normalizer = Normalizer()   # will give each document a unit norm
lsa = Pipeline(steps = [('svd', svd), ('normalizer', normalizer)])   # 先降维，再正则化
lsa_sentences = lsa.fit_transform(sentence_vec_sif)
print(lsa_sentences.shape)
sentence_vec_sif = lsa_sentences

# 碎石图用于查看降维为2000后信息量损失情况
import matplotlib.pyplot as plt
import numpy as np
plt.plot(np.cumsum(svd.explained_variance_ratio_))
plt.savefig("碎石图.png")
```


6. K-Means中文文本聚类

【例8】K-Means中文文本聚类

```
# 将dataframe转换为list，后面打印每一类得用
train_data = np.array(train_df)   # 先将数据框转换为数组
train_data_list = train_data.tolist()   # 其次转换为列表

# 使用KMeans对已降维的文本向量进行聚类
print('使用KMeans对已降维的文本向量进行聚类')
from sklearn.cluster import KMeans
kmean_model = KMeans(n_clusters = 2,init = 'k-means++',n_init = 10,random_state = 10)
kmean_model.fit(sentence_vec_sif)
print('打印出每条数据分类后的label: ',kmean_model.labels_)   # 打印出每条数据分类后的label

from collections import Counter
result = Counter(kmean_model.labels_)
print('统计label每一类的个数: ',result)   # 统计label每一类的个数
```


6. K-Means中文文本聚类

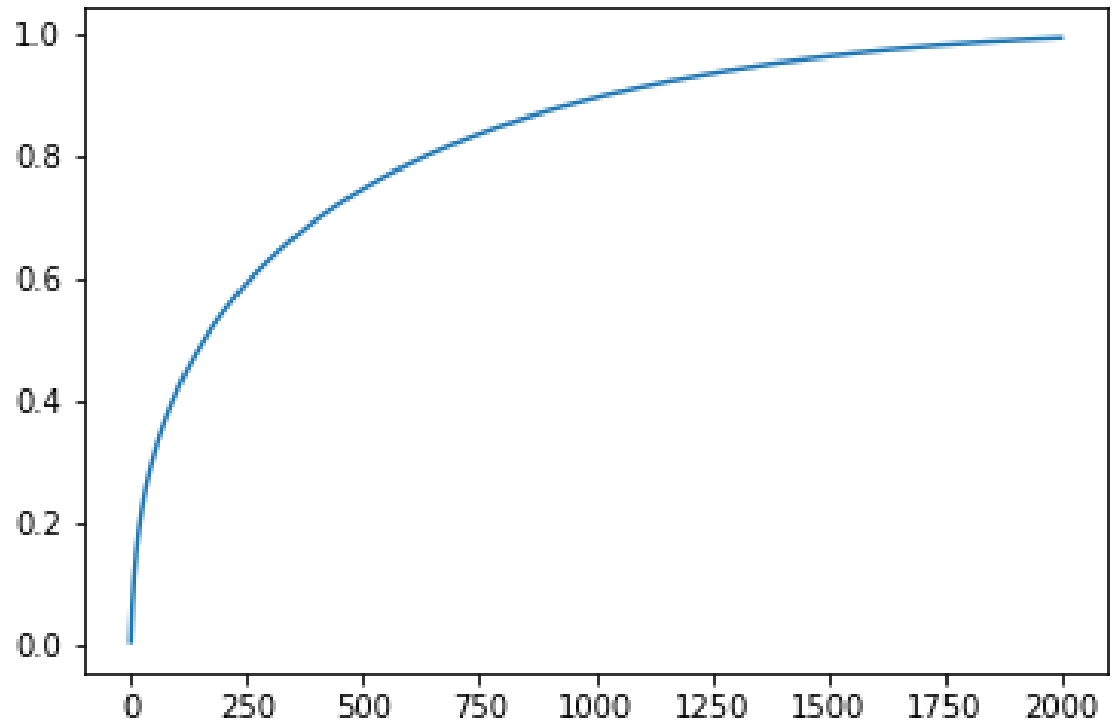
【例8】K-Means中文文本聚类

```
读取数据集
开始分词
前 1000 篇文章分词共花费 92.23 秒
前 2000 篇文章分词共花费 187.45 秒
前 3000 篇文章分词共花费 385.87 秒
前 4000 篇文章分词共花费 570.12 秒
读取分词结果
处理分词格式
对分词文本进行tfidf向量化
词表大小: 7831
(4000,7831)
对tfidf得到的向量矩阵进行压缩，降维（潜在语义分析）
(4000,2000)
使用KMeans对已降维的文本向量进行聚类
打印出每条数据分类后的label: [1 1 1 ... 0 0 0]
统计label每一类的个数: Counter({1: 2080, 0: 1920})
```

《自然语言处理入门》

6. K-Means中文文本聚类

【例8】K-Means中文文本聚类



数据有4000个样本，使用K-Means算法聚为2类。其中，2080个为1类，另一类有1920个样本。数据最初7831个特征，降维后是 2000个特征，碎石图显示维度数量与误差性关系的图表，当特征为2000维时，信息保留为98%左右，符合降维要求。

《自然语言处理入门》

Thank you!