交通大数据

聚类分析

- 刘志远教授
- Email: zhiyuanl@seu.edu.cn

我们在之前的学习中，学习了线性回归模型，决策树，支持向量机等经典的机器学习算法，这些方法有一个共同的特点：**训练数据有标签**。每一条训练数据的特征都对应了事先给定的标签，然而**在实际应用场景中，有标签的训练数据通常是难以获取的**。

例如： 新闻网站每天从互联网上搜索爬取海量新闻，这些数据本身不包含具体的分类标签（例如 军事，社会，娱乐），那么**这些网站如何对新闻进行合理的分类**?

大家打开淘宝等购物软件时，会发现每个人的推荐商品都差别很大。实际上，**面对海量的历史购物数据，购物网站难以通过人工给每一条数据，每一个用户打标签**。那么，购物网站是如何根据用户特点进行定制化的商品推送呢？

在很多实际应用场景中，我们需要对无标签的数据进行处理，在这种情况下，我们通常进行无监督学习。不同于监督学习，无监督学习的目标，**是通过对无标记训练样本的学习来揭示数据的内在性质及规律，为进一步数据分析提供基础**。而此类学习任务中研究最多、应用最广的算法，就是“聚类”（clustering）。

本节课我们将具体学习聚类分析算法

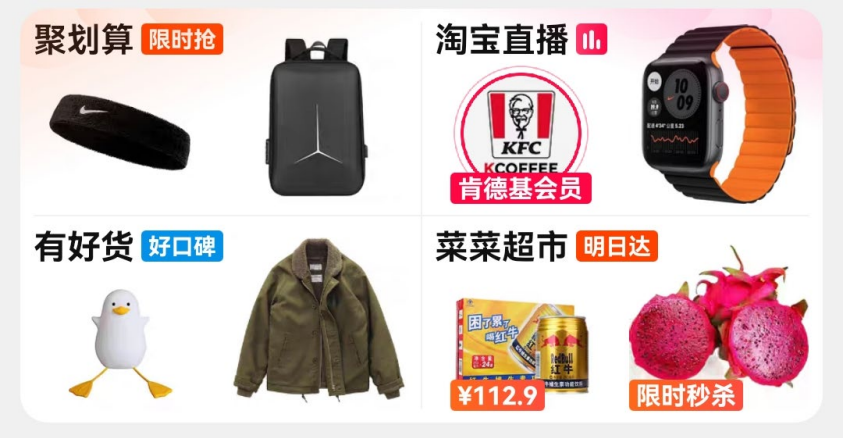
聚类分析

❑ 学习目标

- 了解聚类算法的适用场景
- 掌握聚类分析的一般建模流程与常用相似程度计算方式
- 掌握K-均值聚类、高斯混合聚类、层次聚类、DBSCAN等经典聚类算法

聚类分析

我们在浏览电商网站时通常会收到一些商品推送，但是我们之前可能并没有关注过推送的商品，那么电商网站是如何决定给每个人推销商品的呢？



分类：“学生” “运动” “休闲”

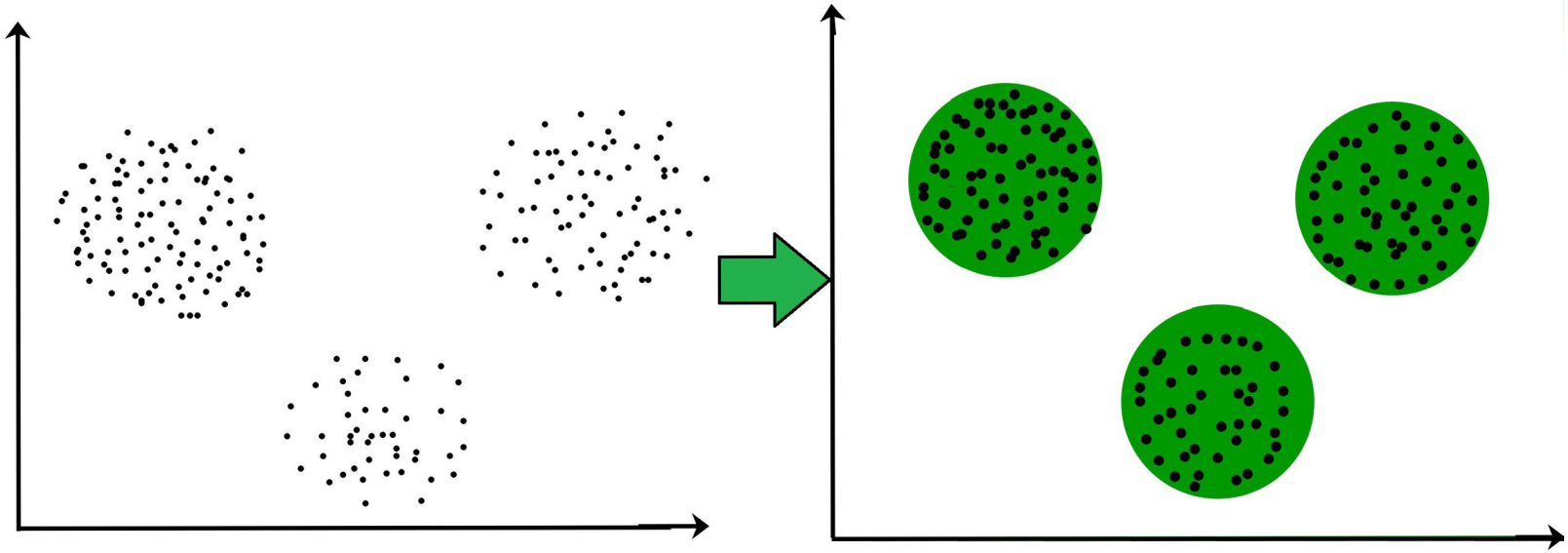
电商可以根据用户的年龄、性别、居住地、购买记录等信息将用户划分为“白领”，“大学生”，“高收入者”等等类别，并据此推送商品。这是**聚类分析算法**的一个典型应用。

1 聚类分析的一般建模流程

- 聚类分析的建模原理
- 聚类分析算法的性能度量
- 相似程度计算

聚类分析的建模原理

聚类模型是**建立在无类别标记数据上，通过学习数据内部的相似关系，进而获取数据分类的一种无监督学习算法**。聚类算法会基于某个特定的规则，将数据集划分成多个不相交的簇（cluster）



聚类分析的建模原理

聚类算法有很多不同的类型，常用的聚类方法包括K-均值（K-means）聚类算法、高斯混合（Gaussian mixture model）聚类算法、层次聚类算法（Hierarchical clustering）、基于密度的DBSCAN聚类算法等。我们将在后续的学习中依次介绍各类聚类算法。

算法分类	算法名称	算法特点
原型聚类	K-均值聚类	K-均值聚类也称为快速聚类法，在最小化误差函数的基础上将**数据划分为预定的类数 K**。该算法原理简单并便于处理大规模数据。
	高斯混合聚类	与K-均值用原型向量刻画聚类结构不同，高斯混合聚类**采用概率模型表达聚类原型**。假设所有样本服从一个整体分布，由 个独立高斯分布混合组成。每个高斯分布均代表一个簇，每个观测样本均有一定概率由不同的高斯分布产生，相对于每个观测样本完全由同一个簇产生的**硬聚类**，该方法属于**软聚类**。

聚类分析的建模原理

算法分类	算法名称	算法特点
层次聚类	AGNES	层次聚类也称为系统聚类，试图**在不同层次对数据集进行划分，从而形成树形的聚类结构**。数据集的划分可采用“自底向上”的聚合策略，也可采用“自顶向下”的分拆策略。AGNES的策略为前者。树状结构所处的位置越低，其所包含的对象就越少，共同特征越多。该聚类算法只适合少量数据时使用，数据量过大时聚类速度较慢。
密度聚类	DBSCAN	DBSCAN是应用最广泛的密度聚类算法，它**基于一组“邻域”参数来刻画样本分布的紧密程度**。从样本密度的角度来考察样本之间的可连接性，并基于可连接样本不断扩展聚类簇以获得最终的聚类结果。

聚类分析算法的性能度量

思考：在处理无类别标记的数据时，**如何科学度量聚类分析结果的优劣**？

聚类算法的度量方法大致有两类。

1. 是将聚类结果与某个“参考模型”（reference model）进行比较（例如将领域专家给出的划分结果作为参考模型），这种度量称为“**外部指标**”（external index）
2. 是直接考察聚类结果而不利用任何参考模型，常称之为“**内部指标**”（internal index）。

一般来说，聚类任务很难找到可靠的参考模型，因此，**本课程着重介绍常用的内部指标**。

聚类分析算法的性能度量

用 $dist(\cdot)$ 表示两个样本点之间的距离（相似程度）。将聚类结果的簇划分记为$\{E_l|l=1,\ldots,K\}$，定义以下指标：

（1）簇内两个不同样本之间距离的均值：

$$avg(E_l)=\frac{2}{|E_l|(|E_l|-1)}\sum_{1\le s<t\le|E_l|}\mathrm{dist}(\boldsymbol{x}_s,\boldsymbol{x}_t)$$

（2）簇内两不同样本间的最远距离

$$diam(E_l)=\max_{1\le s<t\le|E|}\mathrm{dist}(\boldsymbol{x}_s,\boldsymbol{x}_t)$$

（3）两不同簇之间最近的样本距离

$$d_{\min}(E_l,E_p)=\min_{\boldsymbol{x}_s\in E_l,\boldsymbol{x}_t\in E_p}\mathrm{dist}(\boldsymbol{x}_s,\boldsymbol{x}_t)$$

（4）两不同簇中心点的距离

$$d_{cen}(E_l,E_p)=\mathrm{dist}(\boldsymbol{\mu}_l,\boldsymbol{\mu}_p)$$ ➡ 簇p的中心点

聚类分析算法的性能度量

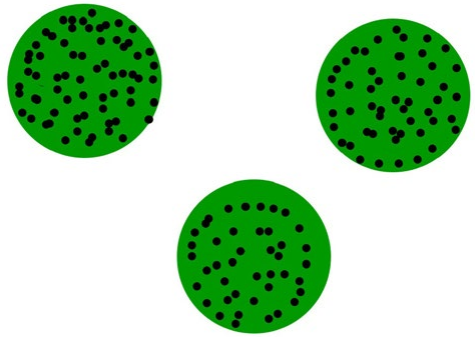
基于以上定义，常用的聚类性能度量内部指标可以总结如下

（1）DB指数（Davies-Bouldin Index, DBI）

DB指数的构建思想是：在聚类结果中，对于任意一个簇l，找到与其**最相似的簇p，**如下公式所示，计算二者的**相似度**（二者的簇内平均距离的和，与簇间中心点距离的比值），并对所有相似度取平均。若均值越高，说明聚类结果中不同簇的相似度偏高、聚类结果差。

$$\mathrm{DBI}=\frac{1}{k}\sum_{l=1}^{k}\max_{p\neq l}\left(\frac{avg(E_l)+avg(E_p)}{d_{cen}(E_l,E_p)}\right)$$

因此，DB指数值越小聚类效果越好。



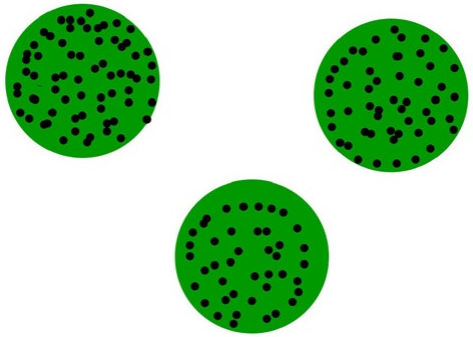
聚类分析算法的性能度量

（2）Dunn指数（Dunn Index, DI)

DI的构建思路是：用**任意两个簇**之间最近的距离的最小值，除以**任意一个簇**内距离最远的两个点的距离的最大值

$$\mathrm{DI} = \frac{\min_{1\le l,p\le k, l\ne p} d_{\min}(E_l, E_p)}{\max_{1\le q\le k} diam(E_q)}$$

该指数值越大聚类效果越好。



聚类分析算法的性能度量

（3）轮廓系数（Silhouette Coefficient, SC）

假设样本点 $\boldsymbol{x}_i$ 属于簇 E_l, 定义：

样本点$\boldsymbol{x}_i$到簇E_l内其他点的平均距离：

$$a(\boldsymbol{x}_i)=\frac{1}{|E_l|-1}\sum_{\boldsymbol{x}_s\in E_l,\boldsymbol{x}_s\neq\boldsymbol{x}_i} dist(\boldsymbol{x}_i,\boldsymbol{x}_s)$$

样本点$\boldsymbol{x}_i$到其他簇E_q内所有样本点的**平均距离最小值**：

$$b(\boldsymbol{x}_i)=\min_{q\neq l}\frac{1}{|E_q|}\sum_{\boldsymbol{x}_s\in E_q} dist(\boldsymbol{x}_i,\boldsymbol{x}_s)$$

聚类分析算法的性能度量

（3）轮廓系数（Silhouette Coefficient, SC）

样本点 x_i 的轮廓系数定义为：

$$SC_i = \frac{b(\boldsymbol{x}_i) - a(\boldsymbol{x}_i)}{\max\left(b(\boldsymbol{x}_i), a(\boldsymbol{x}_i)\right)} = \begin{cases} 1 - a(\boldsymbol{x}_i)/b(\boldsymbol{x}_i) & a(\boldsymbol{x}_i) < b(\boldsymbol{x}_i) \\ 0 & a(\boldsymbol{x}_i) = b(\boldsymbol{x}_i) \\ b(\boldsymbol{x}_i)/a(\boldsymbol{x}_i) - 1 & a(\boldsymbol{x}_i) > b(\boldsymbol{x}_i) \end{cases}$$

可以看出，样本点轮廓系数介于-1和1之间，值越大表明聚类效果越好（**簇内间距小，簇间间距大**）。

整个样本集的轮廓系数*SC*是所有样本点轮廓系数的均值：

$$SC = \frac{1}{m}\sum_{i=1}^{m} SC_i$$

聚类分析算法的性能度量

以上距离度量指标都是基于样本点间的距离（相似程度）$dist(\cdot)$进行计算的，在聚类算法中，样本点间相似度的度量方式往往是多样化的。

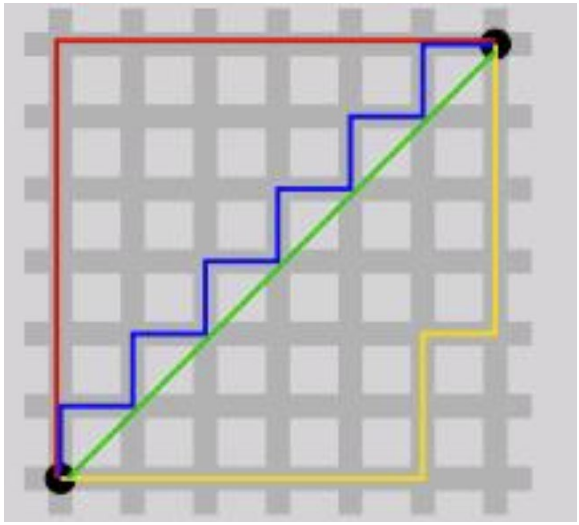
度量样本之间的相似性最常用的是欧几里得距离、曼哈顿距离和闵可夫斯基距离。假设每个样本包含d个维度，那么包含m个样本点的数据集D可以用如下矩阵表示：

$$\begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{md} \end{bmatrix}$$

（1）欧几里得距离（Euclidean distance）

$$dist(\boldsymbol{x}_t, \boldsymbol{x}_s) = \sqrt{\sum_{j=1}^{d} (x_{tj} - x_{sj})^2}$$

聚类分析算法的性能度量



（2）曼哈顿距离（Manhattan distance）

$$dist(\boldsymbol{x}_t, \boldsymbol{x}_s) = \sum_{j=1}^{d} | x_{tj} - x_{sj} |$$

右图中白色表示高楼大厦，灰色表示街道。哪个是欧几里得距离，哪个是曼哈顿距离？

（3）闵可夫斯基距离（Minkowski distance）

$$dist(\boldsymbol{x}_t, \boldsymbol{x}_s) = \sqrt[q]{\sum_{j=1}^{d} \left| x_{tj} - x_{sj} \right|^q}$$

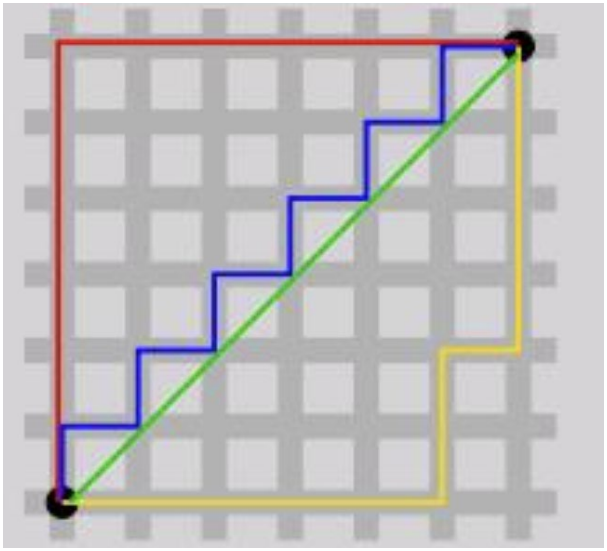
式中，p最常用的取值是 2 和 1, 前者是欧几里得距离，后者是曼哈顿距离。

聚类分析算法的性能度量

请思考，

（1）上述三个距离度量指标有什么缺陷，如何进行改进？

（2）在交通问题的实际应用中，计算两个交通小区在空间位置上的相似程度，什么距离度量方式最为合理呢？



2 K-均值聚类

- 目标函数与算法流程
- 算法变体：K-Medoids
- K-均值算法实例应用

目标函数与算法流程

K-均值聚类算法**在最小化误差函数的基础上，将数据划分为预定的簇数k**，K-均值聚类算法采用距离作为相似程度的判断指标，认为两个对象的距离越近，其相似度越大。

K-均值聚类算法的核心思想是对于给定数据集$D = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$，针对聚类所得簇划分$\{E_l | l = 1, \ldots, K\}$最小化误差的平方和（SSE），计算方式如下：

$$SSE = \sum_{l=1}^{K} \sum_{\boldsymbol{x}_i \in E_l} dist(\boldsymbol{\mu}_l, \boldsymbol{x}_i)^2$$

其中，K为聚类簇的个数，E_l 为第l个簇，$\boldsymbol{\mu}_l$为簇E_l 的聚类中心

目标函数与算法流程

K-均值聚类算法的主要过程如下：

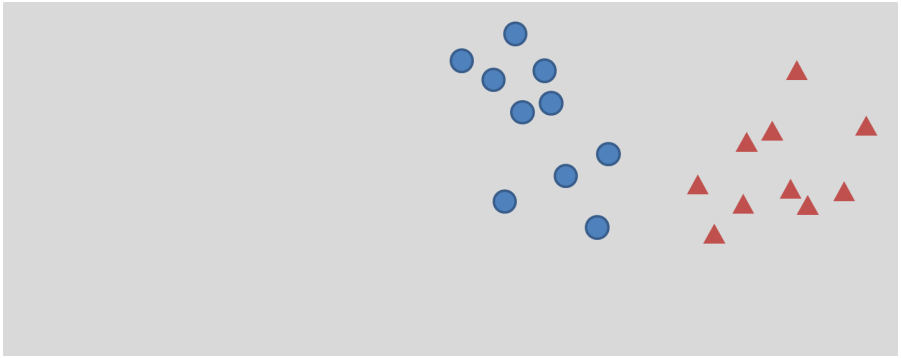
K均值聚类算法流程	
步骤一	从n个样本数据中随机选取K个对象作为初始的聚类中心
步骤二	分别计算每个样本到各个聚类中心的距离，将对象分配到距离最近的簇中；
步骤三	所有对象分配完成后，重新计算k个簇的中心；
步骤四	与前一次计算得到的k个簇的中心比较，如果任何一个簇的聚类中心发生变化，转至**步骤二**，否则转至**步骤五**
步骤五	算法终止并输出聚类结果。

请思考：根据上述算法流程，K均值聚类算法的主要缺陷是什么？

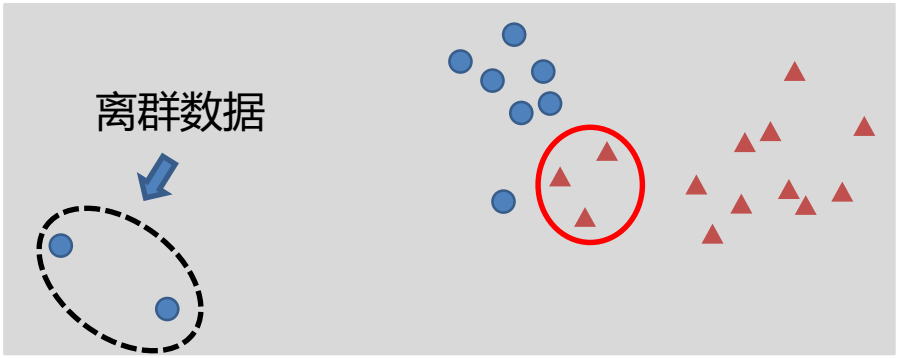
目标函数与算法流程

请思考：根据上述算法流程，K均值聚类算法的主要缺陷是什么？

- 聚类结果容易受到初始聚类中心的影响
- K值需要人为确定，然而在实际应用中很难给出合适的取值
- 对数据中的离群点敏感，离群点会扭曲数据分布和聚类结果



图a 原始数据聚类结果



图b 添加离群数据后聚类结果

算法变体：K-Medoids

K-Medoids算法是为了减弱K-均值算法对离群点的敏感性提出的，其与K-均值聚类算法类似，但主要区别在于中心点的选取方法：

1. K-均值算法将中心点选取为簇内**所有样本点的均值**（相当于寻找质心）
2. K-Medoids算法将中心点选取为簇内一个样本点，**该样本点距离该簇内所有样本点的距离平方和最小**。

当含有离群点时，K-Medoids算法的鲁棒性更好。

算法变体：K-Medoids

对于簇E_l 与样本点$\boldsymbol{x}_1, \ldots \boldsymbol{x}_n \in E_l$ ， K-均值聚类算法与K-Medoids算法的中心点$\boldsymbol{\mu}_l$选取方式可见下表

聚类算法	**中心点选取方式**
K-均值聚类算法	$\boldsymbol{\mu}_l = \sum_{i=1}^{n} \boldsymbol{x}_i / n$
K-Medoids聚类算法	$\boldsymbol{\mu}_l = \underset{\boldsymbol{x}_s}{\operatorname{argmin}} \sum_{i=1}^{n} dist(\boldsymbol{x}_i, \boldsymbol{x}_s)^2$

试一试

利用K-均值算法，对无真实标签下的网格拥堵状态进行判别。选取“平均速度”和“平均停车次数”作为聚类特征，在50个样本的小数据集上进行聚类分析，**来判断网格的拥堵状态并进行可视化**。

表 数据示例

编号	平均速度	平均停车次数	编号	平均速度	平均停车次数
1	4.99	3.26	26	20.83	0
2	7.11	0.66	27	21.01	1
3	3.64	0.72	28	25.61	0
4	10.17	3.32	29	12.18	0.29
5	8.23	0.48	30	22.74	2.5

试一试

(1) 引入package

```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.preprocessing import StandardScaler
```

(2) 数据读取与标准化

```
# 读取数据
data_ori = pd.read_csv('聚类数据集.csv')
# 选择特征
feature = ['stopNum', 'aveSpeed']
# 数据标准化
scaler = StandardScaler()
scaler.fit(data_ori[feature])
data_ori_nor = scaler.transform(data_ori[feature])
```

试一试

(3) 模型训练与结果输出

```
# K均值聚类
n = 3
labels = KMeans(n_clusters=n, random_state=0).fit(data_ori_nor).labels_
# 输出数据集
output_data = pd.concat((data_ori,
                         pd.DataFrame(labels, columns = ['labels'])),
                         axis=1)
output_data.to_csv('kmeans聚类结果.csv', index=False)
```

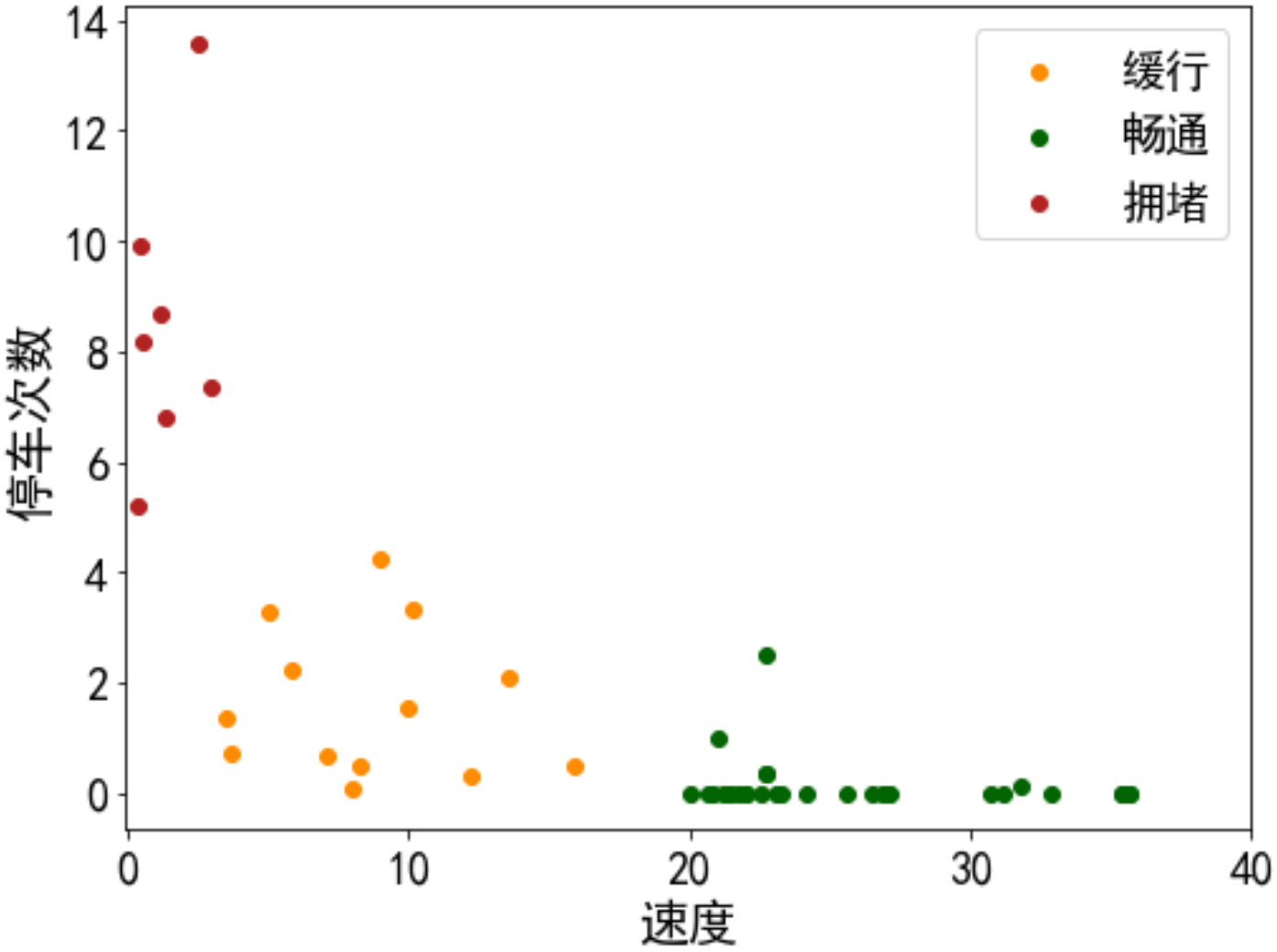
(4) 结果可视化

```
# 可视化
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
import numpy as np
plt.rcParams['axes.unicode_minus'] = False
plt.rcParams['font.sans-serif'] = 'SimHei'
# 读取数据
df = pd.read_csv('kmeans聚类结果.csv')
# 判断各簇实际意义
grouped = df.groupby(['labels']).mean()['aveSpeed']
congested = int(grouped.idxmin(axis=0))
clear = int(grouped.idxmax(axis=0))
slow = [x for x in [0,1,2] if x not in [congested, clear]][0]
```

尝试改变K值，观察聚类结果变化情况（提示：可视化部分代码需要对应调整）

试一试

结果可视化：



3 高斯混合聚类

- 基本概念
- 关键参数推导
- EM算法

基本概念

高斯混合聚类采用概率模型来表达聚类原型。**假设所有样本服从一个整体分布，该分布由k个混合成分组成**，每个混合成分对应一个高斯分布，也就是说，每个样本均有一定概率由不同高斯分布产生。这里的每个“成分”即对应聚类的一个“簇”。

假设d维随机变量$\boldsymbol{y}$服从高斯分布$N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, 其概率密度函数记为：

$$f(\boldsymbol{y}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} exp\left(-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{y}-\boldsymbol{\mu})\right)$$

根据上式可以看出，**$\boldsymbol{\mu}$ 与 $\boldsymbol{\Sigma}$ 唯一决定了高斯分布的概率密度函数。**

基本概念

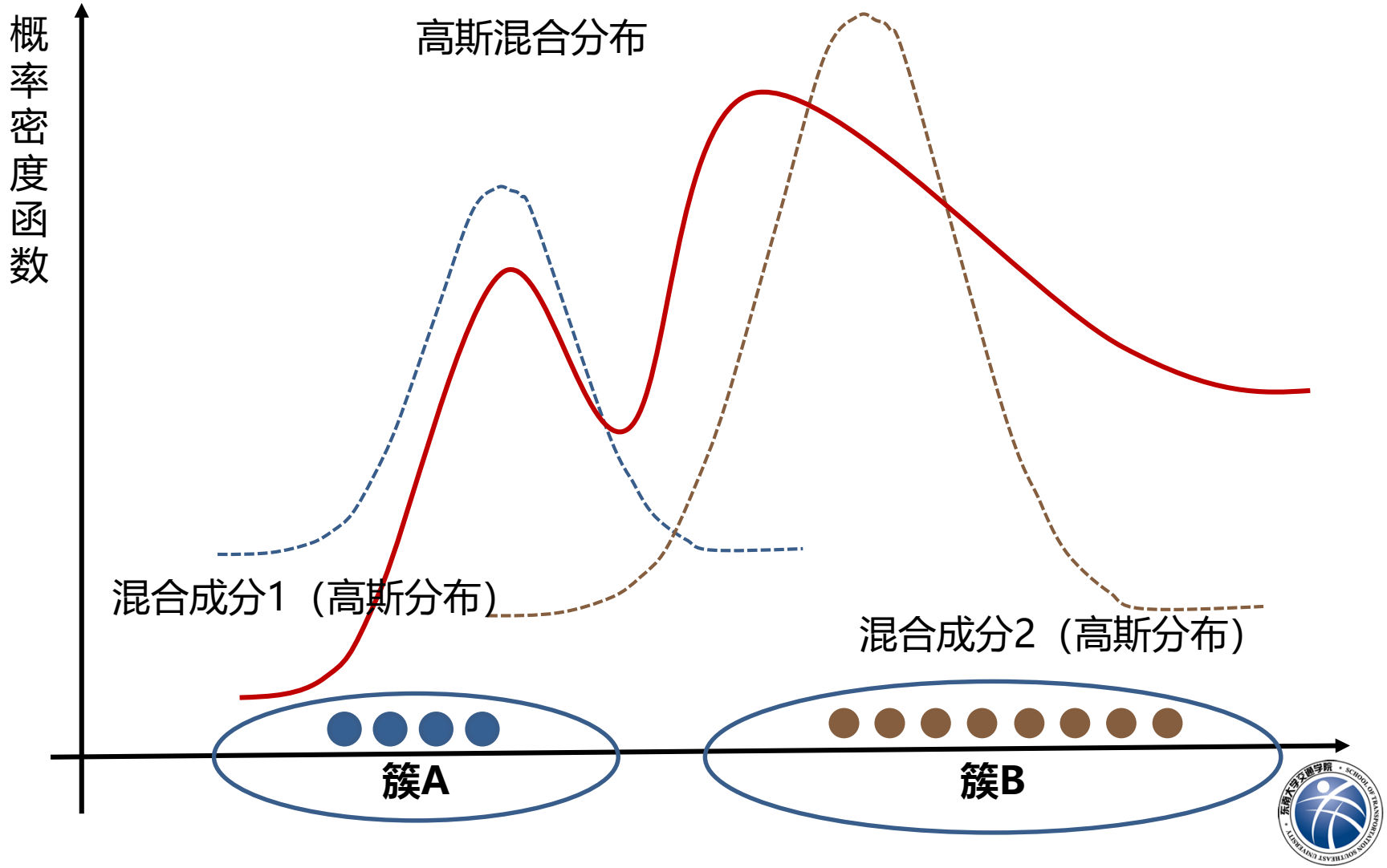
假设d维样本空间中的随机变量$\boldsymbol{x}$服从高斯混合分布，其概率密度函数定义为：

$$f_M(\boldsymbol{x}) = \sum_{i=1}^{k} \alpha_i \times f(\boldsymbol{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$$

该分布由k个混合成分组成，每个混合成分对应一个高斯分布。其中 $\boldsymbol{\mu}_i$ 和 $\boldsymbol{\Sigma}_i$ 是第i个高斯混合成分的参数，而α_i为相应的“混合系数”（mixture coefficient），有 $\sum_{i=1}^{k} \alpha_i = 1$。

高斯混合聚类假设所有样本都源于高斯混合分布的采样，并通过判断样本点“最有可能”由哪个高斯分布生成进行聚类

基本概念



基本概念

高斯混合聚类**假设所有样本都源于高斯混合分布的采样**，对于数据集$D = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$中的任意一个样本$\boldsymbol{x}_t$，引入随机变量$z_t \in \{1, \dots, k\}$表示该样本的高斯混合成分（例如，$z_t = 1$表示该样本由混合成分1生成）。

将z_t的**先验分布**（与数据无关，根据先验知识定义的分布）定义为：

$$f(z_t = i) = \alpha_i$$

根据贝叶斯定理，z_t的**后验分布**（融合先验知识和数据的分布）可以根据下式计算

$$f_M(z_t = i|\boldsymbol{x}_t) = \frac{f_M(\boldsymbol{x}_t|z_t = i)f(z_t = i)}{f_M(\boldsymbol{x}_t)} = \frac{\alpha_i \times f(\boldsymbol{x}_t|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{l=1}^{k} \alpha_l \times f(\boldsymbol{x}_t|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}$$

基本概念

$$f_M(z_t = i|\boldsymbol{x}_t) = \frac{f_M(\boldsymbol{x}_t|z_t = i)f(z_t = i)}{f_M(\boldsymbol{x}_t)} = \frac{\alpha_i \times f(\boldsymbol{x}_t|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{l=1}^{k} \alpha_l \times f(\boldsymbol{x}_t|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}$$

$f_M(z_t = i|\boldsymbol{x}_t)$给出了**样本$\boldsymbol{x}_t$由第$i$个混合成分生成的后验概率**。为方便说明，我们将$f_M(z_t = i|\boldsymbol{x}_t)$记为：$\gamma_{ti}, i \in \{1, \dots, k\}$ 。即：

$$\gamma_{ti} = \frac{\alpha_i \times f(\boldsymbol{x}_t|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{l=1}^{k} \alpha_l \times f(\boldsymbol{x}_t|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}$$

当高斯混合分布参数（$\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i$）已知时，样本$\boldsymbol{x}_t$被划分到的簇$l_t$为后验概率最大的簇：

$$l_t = \underset{i \in \{1,\dots,k\}}{\operatorname{argmax}} \gamma_{ti}$$

关键参数推导（阅读内容）

上述结论是在高斯混合分布参数$\{(\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)|i = 1, \dots k\}$已知的情况下得到的。接下来我们讨论**如何根据数据估计高斯混合分布的参数**。

我们常用极大似然估计对概率模型的参数进行估计。首先构造对数似然函数：

$$LL(D) = ln \prod_{t=1}^{n} f_M(\boldsymbol{x}_t) = \sum_{t=1}^{n} ln \sum_{l=1}^{k} \alpha_l \times f(\boldsymbol{x}_t|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)$$

最优参数$\{(\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)|i = 1, \dots k\}$ 使对数似然函数取极大值，有：

$$\frac{\partial LL(D)}{\partial \boldsymbol{\mu}_i} = 0, \qquad \frac{\partial LL(D)}{\partial \boldsymbol{\Sigma}_i} = 0, \qquad \frac{\partial LL(D)}{\partial \alpha_i} = 0, \qquad i = 1, \dots k$$

关键参数推导（阅读内容）

根据 $\frac{\partial LL(D)}{\partial \boldsymbol{\mu}_i} = 0$ 可以得到

$$\sum_{t=1}^{n} \frac{\alpha_i \times f(\boldsymbol{x}_t|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{l=1}^{k} \alpha_l \times f(\boldsymbol{x}_t|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)} (\boldsymbol{x}_t - \boldsymbol{\mu}_i) = 0$$

即：

$$\sum_{t=1}^{n} \gamma_{ti} (\boldsymbol{x}_t - \boldsymbol{\mu}_i) = 0$$

可以得到

$$\boldsymbol{\mu}_i = \frac{\sum_{t=1}^{n} \gamma_{ti} \boldsymbol{x}_t}{\sum_{t=1}^{n} \gamma_{ti}}$$

关键参数推导（阅读内容）

类似的，根据 $\frac{\partial LL(D)}{\partial \boldsymbol{\Sigma}_i} = 0$ 可以得到

$$\boldsymbol{\Sigma}_i = \frac{\sum_{t=1}^{n} \gamma_{ti} (\boldsymbol{x}_t - \boldsymbol{\mu}_i)^t (\boldsymbol{x}_t - \boldsymbol{\mu}_i)}{\sum_{t=1}^{n} \gamma_{ti}}$$

对于混合系数α_i，除了要求对数似然函数对混合系数偏导数等于零，还需要满足以下约束：

$$\alpha_i > 0, \qquad \sum_{i=1}^{k} \alpha_i = 1$$

因此构造拉格朗日函数：

$$LA(\alpha_1, \ldots \alpha_k, \lambda) = LL(D) + \lambda \left(\sum_{i=1}^{k} \alpha_i - 1 \right)$$

关键参数推导（阅读内容）

对于混合系数α_i，除了要求对数似然函数对混合系数偏导数等于零，还需要满足以下约束：

$$\alpha_i > 0, \qquad \sum_{i=1}^{k} \alpha_i = 1$$

因此构造拉格朗日函数：

$$LA(\alpha_1, \ldots \alpha_k, \lambda) = LL(D) + \lambda \left(\sum_{i=1}^{k} \alpha_i - 1 \right)$$

令拉格朗日函数对α_i 偏导数等于0，可以得到：

$$\sum_{t=1}^{n} \frac{f(\boldsymbol{x}_t | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{l=1}^{k} \alpha_l \times f(\boldsymbol{x}_t | \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)} + \lambda = 0$$

关键参数推导（阅读内容）

令拉格朗日函数对α_i 偏导数等于0，可以得到：

$$\sum_{t=1}^{n} \frac{f(\boldsymbol{x}_t|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{l=1}^{k} \alpha_l \times f(\boldsymbol{x}_t|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)} + \lambda = 0$$

对上式两侧同时乘以α_i, 并对 i 求和，可以得到

$$\lambda = -n$$

$$\alpha_i = \frac{\sum_{t=1}^{n} \gamma_{ti}}{n}$$

请思考，$\lambda = -n$ 是如何计算得到的?

关键参数推导（阅读内容）

请思考，$\lambda = -n$ 是如何计算得到的?

$$\sum_{t=1}^{n} \frac{f(\boldsymbol{x}_t|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{l=1}^{k} \alpha_l \times f(\boldsymbol{x}_t|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)} + \lambda = 0$$

等式两边同时乘以α_i，得到:

$$\sum_{t=1}^{n} \frac{\alpha_i f(\boldsymbol{x}_t|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{l=1}^{k} \alpha_l \times f(\boldsymbol{x}_t|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)} + \alpha_i\lambda = 0$$

将 $\frac{\alpha_i f(\boldsymbol{x}_t|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{l=1}^{k} \alpha_l \times f(\boldsymbol{x}_t|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}$ 记为 m_{it}，显然，$\sum_{i=1}^{k} m_{it} = 1$，那么有:

$$\sum_{i=1}^{k}\sum_{t=1}^{n} m_{it} + \sum_{i=1}^{k} \alpha_i\lambda = \sum_{t=1}^{n}\sum_{i=1}^{k} m_{it} + \lambda = n + \lambda = 0$$

关键参数推导

综上，高斯混合分布的三个关键参数可以通过如下计算得到：

$$\boldsymbol{\mu}_i = \frac{\sum_{t=1}^{n} \gamma_{ti}\, \boldsymbol{x}_t}{\sum_{t=1}^{n} \gamma_{ti}}$$

$$\boldsymbol{\Sigma}_i = \frac{\sum_{t=1}^{n} \gamma_{ti} (\boldsymbol{x}_t - \boldsymbol{\mu}_i)^t (\boldsymbol{x}_t - \boldsymbol{\mu}_i)}{\sum_{t=1}^{n} \gamma_{ti}}$$

$$\alpha_i = \frac{\sum_{t=1}^{n} \gamma_{ti}}{n}$$

可以看出，高斯混合分布参数$\{(\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) | i = 1, \dots k\}$均与$\gamma_{ti}$有关，然而$\gamma_{ti} = \frac{\alpha_i \times f(\boldsymbol{x}_t | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{l=1}^{k} \alpha_l \times f(\boldsymbol{x}_t | \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}$，与高斯混合分布参数有关。为了对$\gamma$取值进行估计，高斯混合聚类采用**Expectation Maximization（EM）算法**进行估计。

EM算法

EM算法是在概率模型中寻找参数最大似然估计或者最大后验估计的算法，其中**概率模型依赖于无法观测的隐性变量**。

一个例子：

投掷两枚硬币A和B，每次实验进行10次投掷，共进行5次实验，结果如下：

实验	硬币A	硬币B
1	3正7负	
2	4正6负	
3		7正3负
4	2正8负	
5		7正3负

我们可以计算得到硬币A和硬币B正面向上的概率：

$$\theta_A = \frac{3+4+2}{30} = 0.3 \qquad \theta_B = \frac{7+7}{20} = 0.7$$

EM算法

但是如果“掩盖”投掷两枚硬币A和B的信息，根据如下实验结果估计两枚硬币正面向上的概率，就需要引入一组隐变量$\{z_t\}$ $t = 1, \ldots, 5$表示各次实验使用硬币的情况。

实验	实验结果
1	3正7负
2	4正6负
3	7正3负
4	2正8负
5	7正3负

显然，估计θ_A和θ_B需要确定隐变量$\{z_t\}$的取值，而估计隐变量$\{z_t\}$又需要知道θ_A和θ_B的取值（鸡生蛋、蛋生鸡）。**EM算法就是解决这类问题的算法。**

EM算法的思路也很简单：先初始化θ_A和θ_B，估计隐变量$\{z_t\}$，之后固定隐变量，用最大似然估计更新θ_A和θ_B，不断迭代直至收敛

EM算法

用EM算法解决上述问题，需要通过多轮迭代。初始化$\theta_A = 0.4$和$\theta_B = 0.6$，下面将不断更新这两个概率。

以第1轮迭代的实验1为例，根据贝叶斯定理可以算出$P(选A|投掷结果)$：

$$P(选A|投掷结果) = \frac{P(选A, 投掷结果)}{P(选A, 投掷结果) + P(选B, 投掷结果)}$$

$$= \frac{\theta_A^{正面次数} \times (1-\theta_A)^{反面次数}}{\theta_A^{正面次数} \times (1-\theta_A)^{反面次数} + \theta_B^{正面次数} \times (1-\theta_B)^{反面次数}} = \frac{0.4^3 \times 0.6^7}{0.4^3 \times 0.6^7 + 0.6^3 \times 0.4^7} = 0.835$$

$$P(选B|投掷结果) = 1 - P(选A|投掷结果) = 0.165$$

$$硬币A为正面期望次数 = 正面次数 \times 选硬币A概率 = 3 \times 0.835 = 2.505$$

$$硬币A为反面期望次数 = 反面次数 \times 选硬币A概率 = 7 \times 0.835 = 5.845$$

$$硬币B为正面期望次数 = 正面次数 \times 选硬币B概率 = 3 \times 0.165 = 0.495$$

$$硬币B为反面期望次数 = 反面次数 \times 选硬币B概率 = 7 \times 0.165 = 1.155$$

EM算法

用EM算法解决上述问题，需要通过多轮迭代。初始化$\theta_A = 0.4$和$\theta_B = 0.6$，下面将不断更新这两个概率。

第一轮迭代的完整计算表如下：

z_t

实验	选硬币A概率	选硬币B概率	硬币A为正面期望次数	硬币A为反面期望次数	硬币B为正面期望次数	硬币B为反面期望次数
1	0.835	0.165	2.505	5.845	0.495	1.155
2	0.692	0.308	2.769	4.154	1.231	1.846
3	0.165	0.835	1.155	0.495	5.845	2.505
4	0.919	0.081	1.839	7.354	0.161	0.646
5	0.165	0.835	1.155	0.495	5.845	2.505
合并			9.422	18.343	13.578	8.657

需要指出的是，隐变量$\{z_t\}$ $t = 1, \ldots, 5$表示各次实验使用硬币的情况，共有两个可能的取值（使用A或使用B），符合伯努利分布，因此表格中2、3列的选硬币A概率和选硬币B概率即为隐变量$\{z_t\}$的概率分布。

EM算法

用EM算法解决上述问题，需要通过多轮迭代。初始化$\theta_A = 0.4$和$\theta_B = 0.6$，下面将不断更新这两个概率。

第一轮迭代完后，新的$\theta_A = \frac{\text{硬币}A\text{为正面期望次数总和}}{\text{硬币}A\text{为正面期望次数总和+硬币}A\text{为反面期望次数总和}} = \frac{9.422}{9.422+18.343} = 0.339$，$\theta_B = \frac{\text{硬币}B\text{为正面期望次数总和}}{\text{硬币}B\text{为正面期望次数总和+硬币}B\text{为反面期望次数总和}} = \frac{13.578}{13.578+8.657} = 0.611$，如此迭代，直到达到收敛条件。设收敛条件迭代后为$\theta_A$和$\theta_B$变化的绝对值都小于0.001，则迭代6轮后算法收敛，最终$\theta_A = 0.302$，$\theta_B = 0.648$。

注意：

1. 初始化概率不影响最终结果，结果最终会收敛到同一个值，但是由于算法存在对称性，因此需要保证初始化时A的概率大于B的概率，不然最终结果会反过来。

2. 以上结果之所以不是0.3和0.7，是因为实验次数过少，增加实验次数就可以解决这个问题。

EM算法

EM算法包含两个步骤：

E 步：根据数据、模型和当前参数，估计隐性变量

M步：在给定的隐性变量下，计算使极大似然估计函数取值最大的参数

采用EM算法解决高斯混合聚类的参数估计问题，算法流程如下：

EM算法流程

步骤一	初始化参数$\{(\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\|i = 1, \dots k\}$
步骤二（E步）	根据当前参数计算$\gamma_{ti}, t \in \{1, \dots, n\}, i \in \{1, \dots, k\}$；
步骤三（M步）	根据步骤二中计算所得γ_{ti}更新参数$\{(\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\|i = 1, \dots k\}$
步骤五	若EM算法的停止条件满足，例如：已经达到最大迭代次数，则根据高斯混合分布确定簇划分，否则返回**步骤二。**

高斯混合聚类

以拥堵判断数据为例说明高斯混合聚类中应用EM算法进行参数计算步骤

- 数据输入：

数据编号	停车次数	平均速度
1	3.24	4.99
2	5.31	0.34
…	…	…
50	1.03	5.34

- 参数初始化：

聚类类别：$k = 3,$

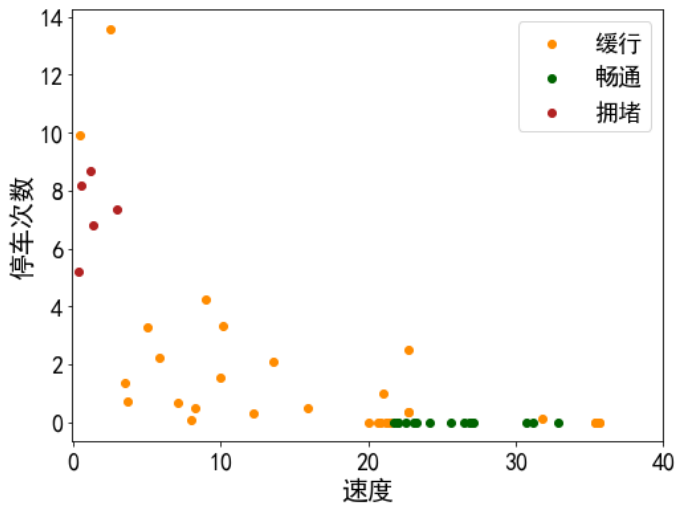
聚类参数：$\alpha_1 = \alpha_2 = \alpha_3 = 1/3$，$\boldsymbol{\mu}_1 = \boldsymbol{\mu}_2 = \boldsymbol{\mu}_3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$，$\boldsymbol{\Sigma}_i$由python package指定

高斯混合聚类

- 第一轮迭代:
  - 基于$\{(\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)|i = 1, \dots 3\}$计算$\gamma_{ti}, t \in \{1, \dots, 50\}, i \in \{1, \dots, 3\}$。以数据点1为例，其$\gamma_{11} = 0.307$，$\gamma_{12} = 0.43$，$\gamma_{13} = 0.263$
  - 基于$\gamma_{ti}, t \in \{1, \dots, 50\}, i \in \{1, \dots, 3\}$，更新$\{(\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)|i = 1, \dots 3\}$，得到参数:

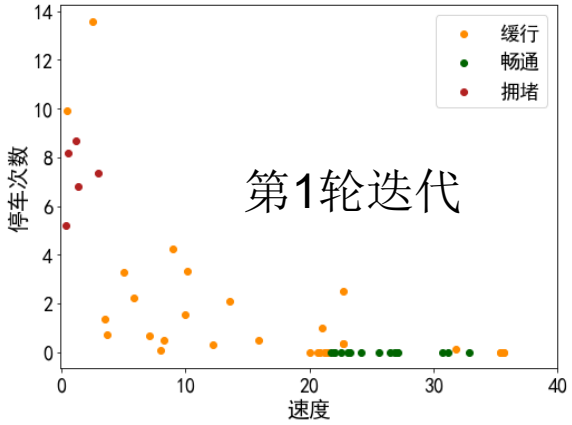
$\alpha_1 = 0.120$，$\alpha_2 = 0.125$，$\alpha_3 = 0.754$，$\boldsymbol{\mu}_1 = \begin{pmatrix} 1.13 \\ -1.23 \end{pmatrix}$，$\boldsymbol{\mu}_2 = \begin{pmatrix} -0.55 \\ 0.879 \end{pmatrix}$，$\boldsymbol{\mu}_3 = \begin{pmatrix} -0.88 \\ 0.05 \end{pmatrix}$，$\boldsymbol{\Sigma}_1 = \begin{pmatrix} 0.29 & -0.12 \\ -0.12 & 0.11 \end{pmatrix}$，$\boldsymbol{\Sigma}_2 = \begin{pmatrix} 0 & 0 \\ 0 & 0.03 \end{pmatrix}$，$\boldsymbol{\Sigma}_2 = \begin{pmatrix} 1.02 & -0.6 \\ -0.6 & 0.93 \end{pmatrix}$

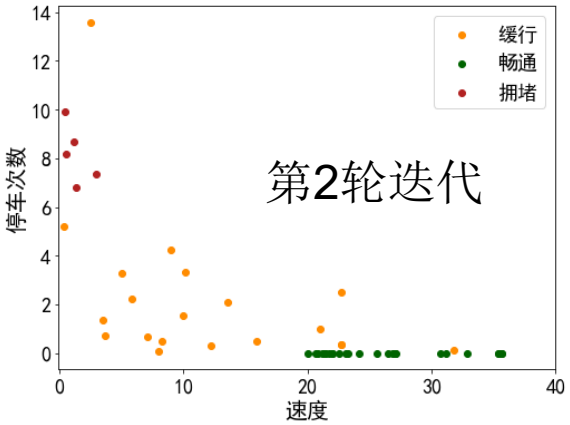
  - 聚类结果:

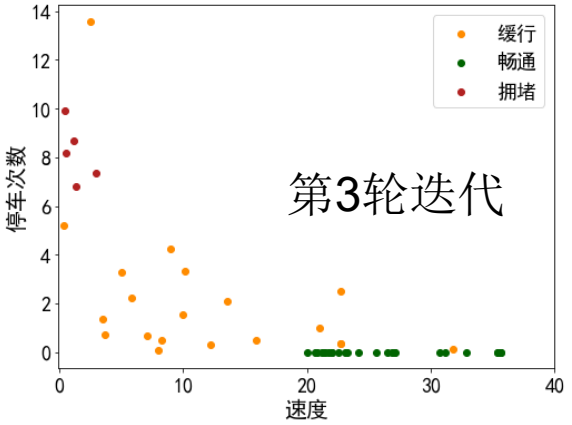


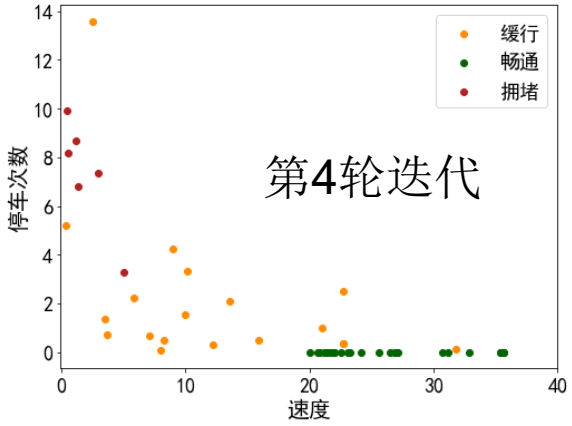
高斯混合聚类

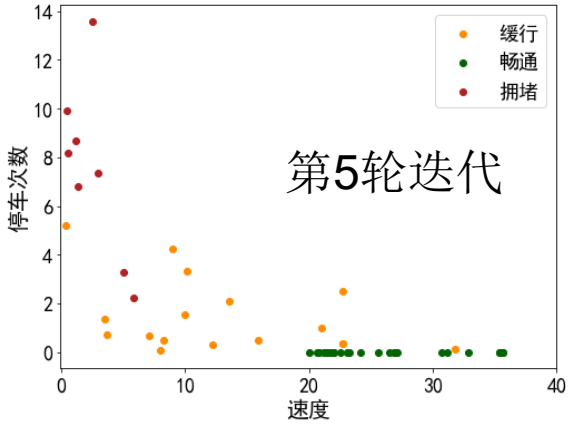
- 第二轮迭代:
  - 基于$\{(\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)|i = 1, \dots 3\}$计算$\gamma_{ti}, t \in \{1, \dots, 50\}, i \in \{1, \dots, 3\}$
  - 基于$\gamma_{ti}, t \in \{1, \dots, 50\}, i \in \{1, \dots, 3\}$，更新$\{(\alpha_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)|i = 1, \dots 3\}$
- 第三轮迭代…

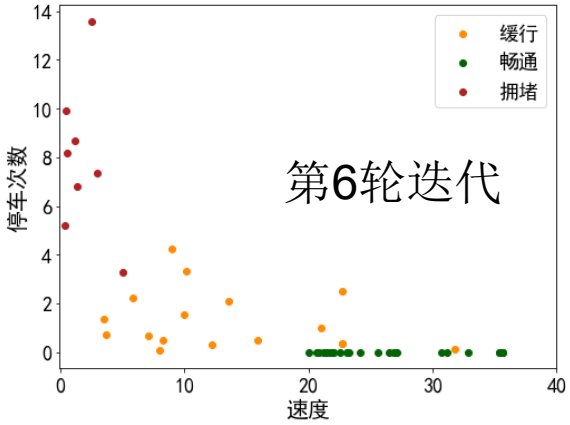

第1轮迭代
缓行
畅通
拥堵
停车次数
速度

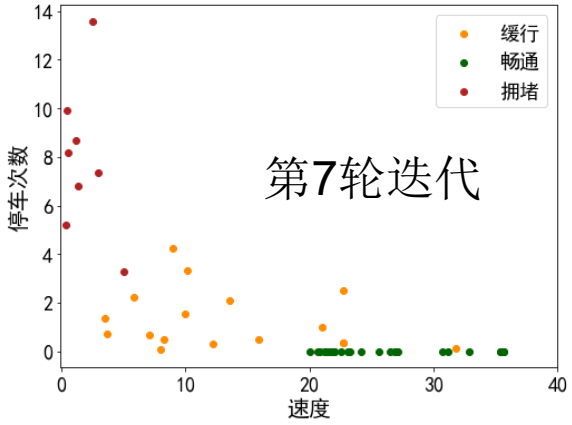

第2轮迭代
缓行
畅通
拥堵
停车次数
速度

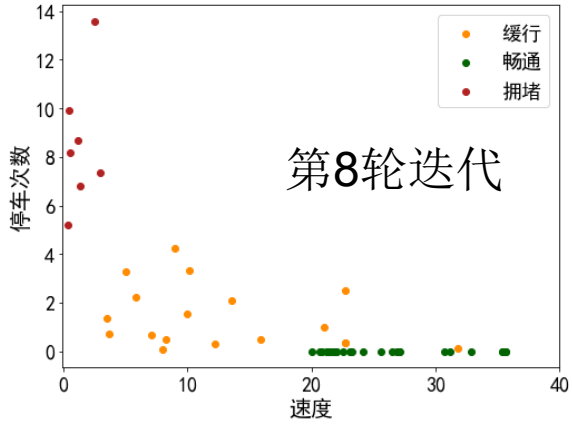

第3轮迭代
缓行
畅通
拥堵
停车次数
速度

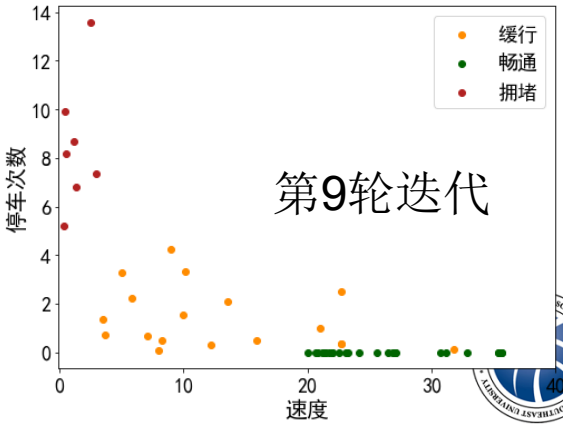

第4轮迭代
缓行
畅通
拥堵
停车次数
速度


第5轮迭代
缓行
畅通
拥堵
停车次数
速度


第6轮迭代
缓行
畅通
拥堵
停车次数
速度


第7轮迭代
缓行
畅通
拥堵
停车次数
速度


第8轮迭代
缓行
畅通
拥堵
停车次数
速度


第9轮迭代
缓行
畅通
拥堵
停车次数
速度

试一试

利用高斯混合聚类算法，对无真实标签下的网格拥堵状态进行判别。选取“平均速度”和“平均停车次数”作为聚类特征，在50个样本的小数据集上进行聚类分析，**判断网格的拥堵状态并进行结果可视化**

表 数据示例

编号	平均速度	平均停车次数	编号	平均速度	平均停车次数
1	4.99	3.26	26	20.83	0
2	7.11	0.66	27	21.01	1
3	3.64	0.72	28	25.61	0
4	10.17	3.32	29	12.18	0.29
5	8.23	0.48	30	22.74	2.5

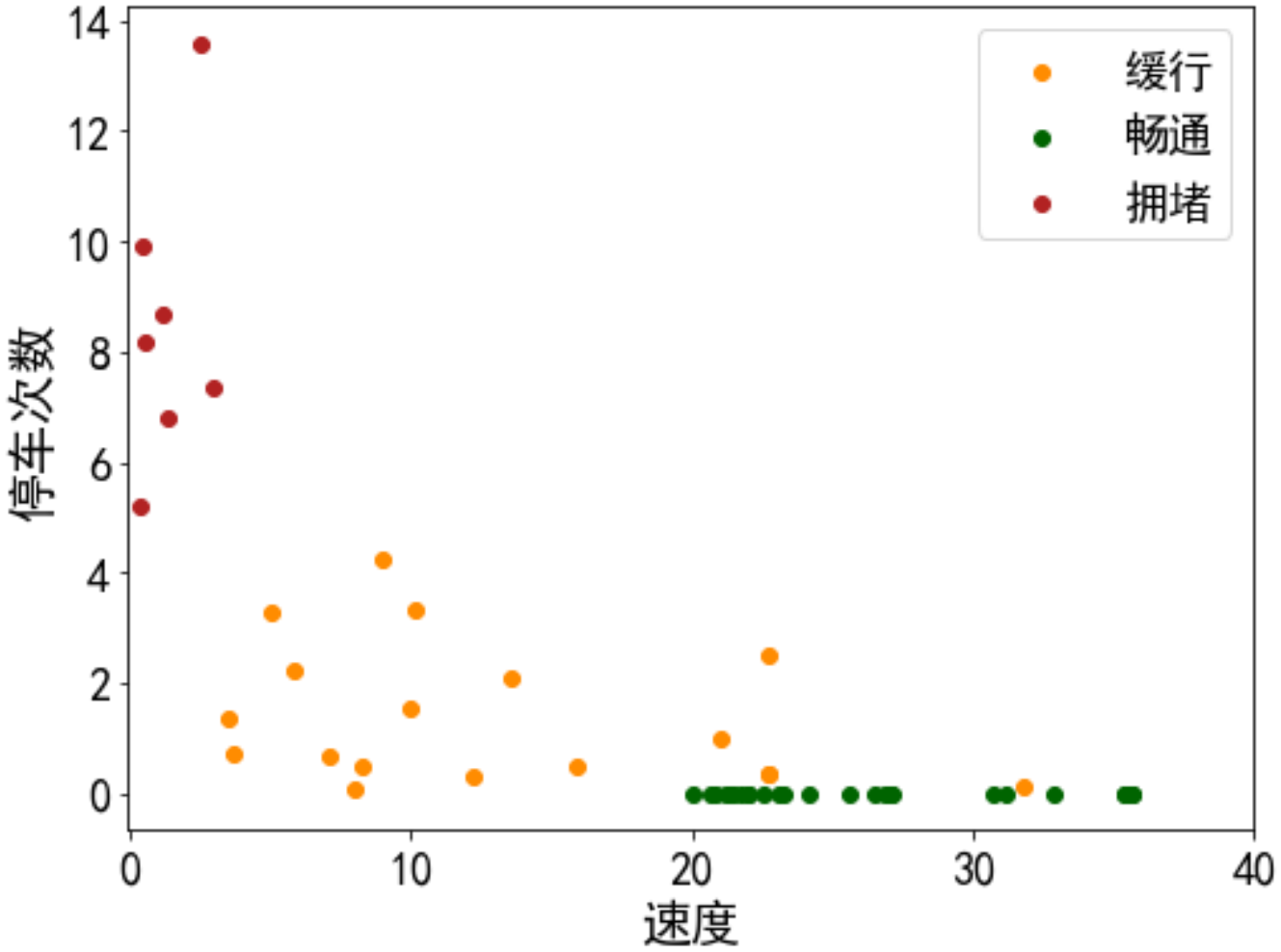
试一试

数据预处理与可视化部分代码与K-means算法相同，模型训练代码如下：

```
# 高斯混合聚类
n = 3
GMM = GaussianMixture(n, random_state=0).fit(data_ori_nor)
labels = GMM.predict(data_ori_nor)
num_iter = GMM.n_iter_
print (num_iter)
# 输出计算过程中的迭代轮数，这里是：11轮
# 输出数据集
output_data = pd.concat((data_ori,
                         pd.DataFrame(labels, columns = ['labels'])),
                         axis=1)
output_data.to_csv('GMM聚类结果.csv', index=False)
```

试一试

结果可视化



t	γ_{t1}	γ_{t2}	γ_{t3}
0	2.4910E-06	0.0000E+00	1.0000E+00
1	1.2966E-15	0.0000E+00	1.0000E+00
2	1.9093E-05	0.0000E+00	9.9998E-01
3	2.9654E-26	0.0000E+00	1.0000E+00
4	1.9618E-20	0.0000E+00	1.0000E+00
5	2.4636E-144	9.9944E-01	5.5978E-04
6	9.8735E-01	0.0000E+00	1.2648E-02
7	7.7448E-05	0.0000E+00	9.9992E-01
8	5.5187E-28	0.0000E+00	1.0000E+00
9	9.5315E-20	7.3893E-83	1.0000E+00
10	6.8149E-19	0.0000E+00	1.0000E+00
11	1.1695E-196	9.9976E-01	2.4089E-04
12	1.0000E+00	0.0000E+00	9.1456E-22
13	4.1254E-128	9.9920E-01	7.9818E-04
14	3.6717E-271	9.9988E-01	1.2439E-04
15	3.2239E-130	9.9924E-01	7.5988E-04
16	9.5759E-71	0.0000E+00	1.0000E+00
17	2.5176E-121	9.9906E-01	9.4155E-04
18	1.8311E-150	9.9950E-01	4.9609E-04
19	3.9863E-134	9.9930E-01	6.9541E-04
20	2.9121E-300	9.9989E-01	1.0839E-04
21	6.3415E-131	9.9925E-01	7.4762E-04
22	3.2262E-165	9.9962E-01	3.8079E-04
23	1.4312E-262	9.9987E-01	1.3102E-04
24	1.2921E-114	9.9888E-01	1.1208E-03
25	7.8069E-124	9.9912E-01	8.8466E-04

t	γ_{t1}	γ_{t2}	γ_{t3}
26	9.4728E-121	0.0000E+00	1.0000E+00
27	1.0530E-184	9.9972E-01	2.8234E-04
28	1.3799E-42	0.0000E+00	1.0000E+00
29	2.2434E-137	0.0000E+00	1.0000E+00
30	1.0000E+00	0.0000E+00	1.5835E-08
31	1.0000E+00	0.0000E+00	5.9453E-11
32	1.0000E+00	0.0000E+00	2.6398E-07
33	2.5792E-206	9.9979E-01	2.1451E-04
34	7.5289E-138	9.9936E-01	6.4097E-04
35	5.7231E-142	0.0000E+00	1.0000E+00
36	0.0000E+00	9.9990E-01	9.6148E-05
37	0.0000E+00	9.9990E-01	9.6643E-05
38	1.0884E-09	0.0000E+00	1.0000E+00
39	9.9995E-01	0.0000E+00	4.7155E-05
40	7.5289E-138	9.9936E-01	6.4097E-04
41	5.7231E-142	0.0000E+00	1.0000E+00
42	0.0000E+00	9.9990E-01	9.6148E-05
43	0.0000E+00	9.9990E-01	9.6643E-05
44	7.7536E-205	9.9978E-01	2.1821E-04
45	1.1470E-49	0.0000E+00	1.0000E+00
46	6.7542E-202	9.9977E-01	2.2590E-04
47	7.1087E-153	9.9953E-01	4.7394E-04
48	1.9515E-276	0.0000E+00	1.0000E+00
49	9.9994E-01	0.0000E+00	6.3295E-05

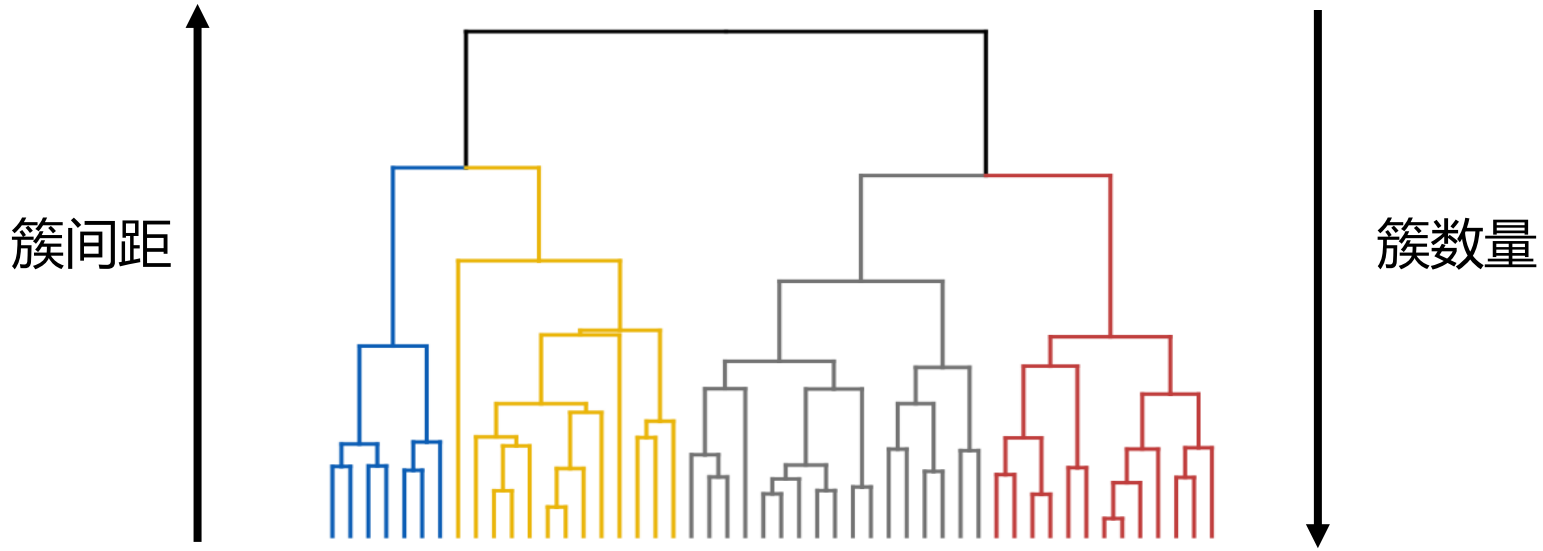
4 层次聚类算法

- 算法基本思想
- 算法过程

算法基本思想

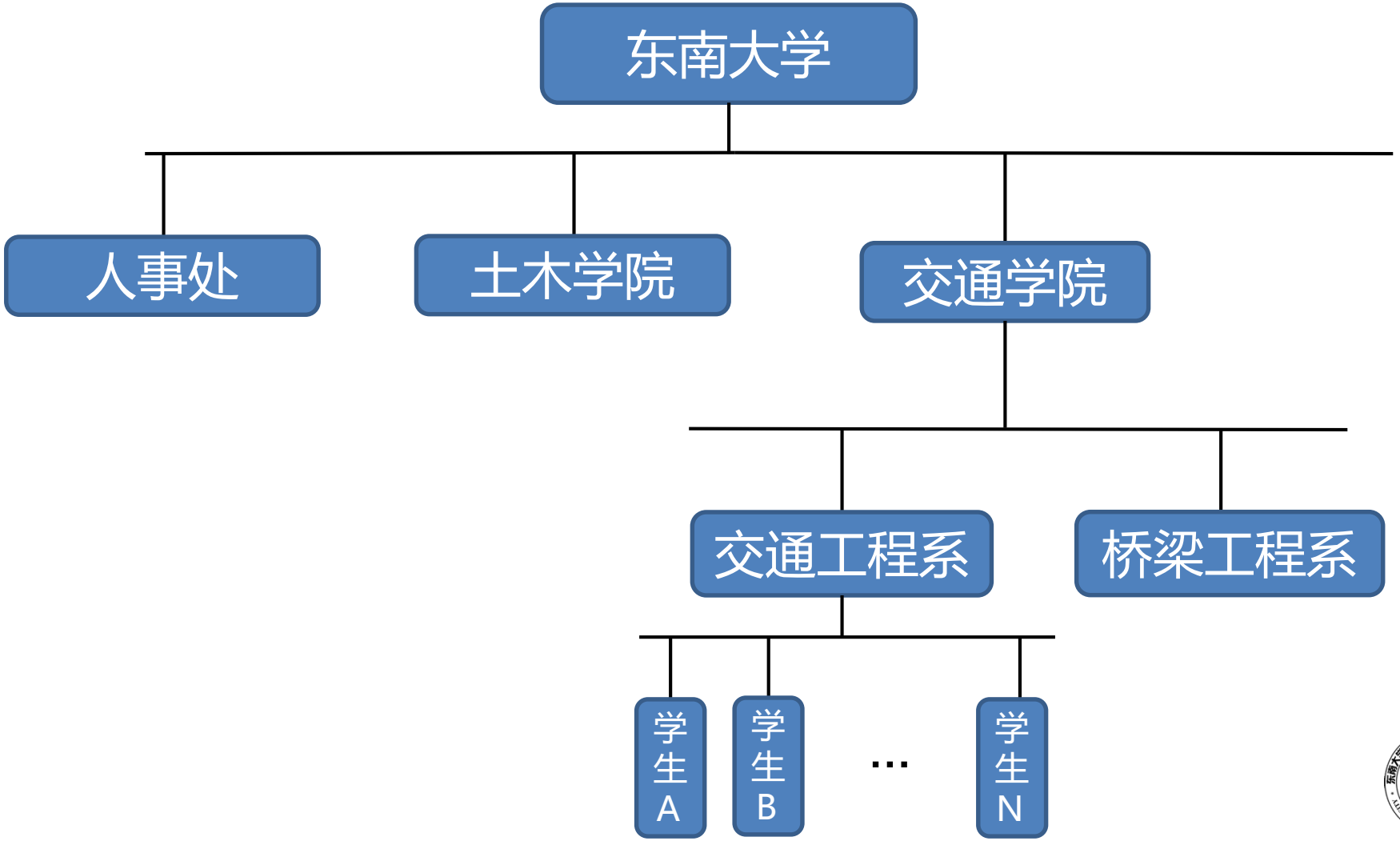
层次聚类（又称为系统聚类或系谱聚类）是一种常用聚类算法，该算法**通过合成或分割簇，生成嵌套的集群**。层次聚类算法的策略可以分为“**自底向上聚合**”和“**自顶向下分拆**”两种。

本课介绍的算法策略为自底向上聚合地层次聚类算法，其算法层次结构可以表示为一棵树：**树的根是一个唯一的簇，包含所有的样本，而树的叶子节点是单独的一个样本。通过树的叶子节点的相互合并，最终合称为树的根节点**。



算法基本思想

例子：学校人员的“层次聚类”



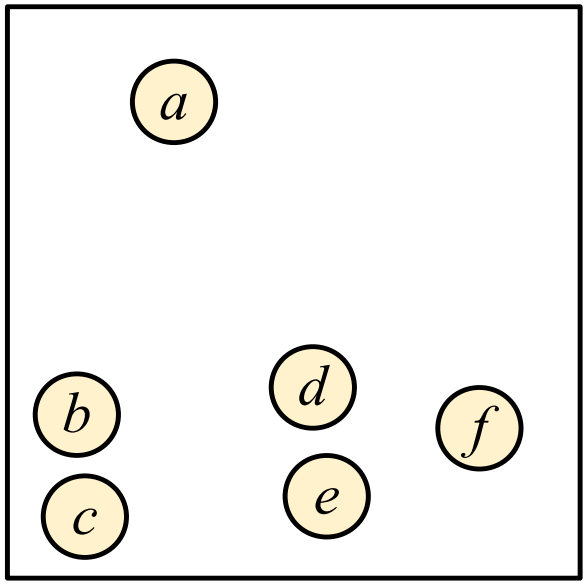
算法基本思想

在定义簇间距离计算方法的基础上，层次聚类算法的**基本思想**是：先将各个样本看作单独的一簇，选择距离最小的一对簇合并成为一个新的簇。然后，重新计算簇间的距离，再将距离最近的两簇合并，循环计算，直至合成为一个唯一的簇。

对于如右图所示的样本分布，**层次聚类过程如下**：

（1）首先计算各簇距离，将距离最近的b、c合并为一簇，此时得到五个簇：$\{a\},\{b,c\},\{d\},\{e\},\{f\}$

（2）计算各簇之间的距离，将距离最近的d，e合并为一簇，此时有四个簇： $\{a\},\{b,c\},\{d,e\},\{f\}$



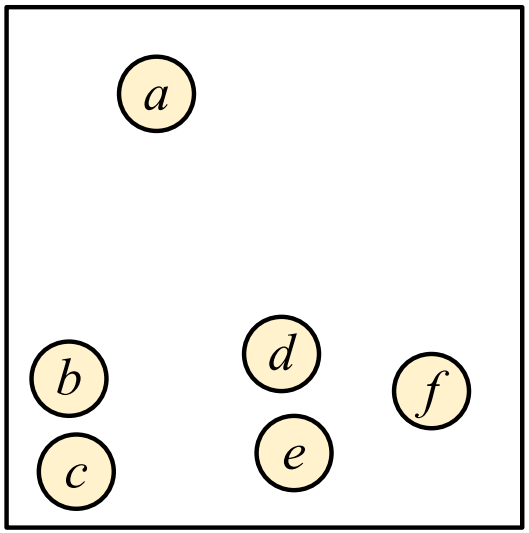
算法基本思想

对于如下图所示的样本分布，**层次聚类过程如下**：

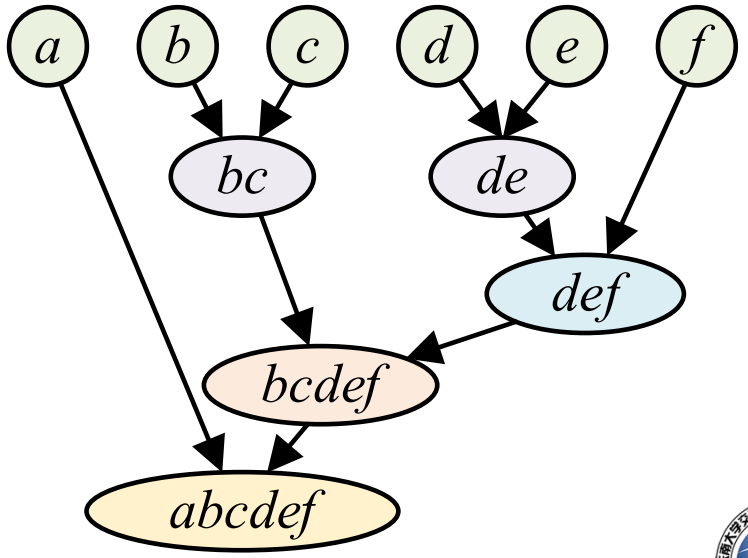
（3）计算各簇之间的距离，将距离最近的$\{d,e\}$和$\{f\}$合并为一簇，此时有三个簇：$\{a\},\{b,c\},\{d,e,f\}$

（4）计算各簇之间的距离，将距离最近的$\{b,c\}$和$\{d,e,f\}$合并为一簇，此时有两个簇：$\{a\},\{b,c,d,e,f\}$

（5）将剩余两簇$\{a\}$和$\{b,c,d,e,f\}$合并



聚类结果



算法基本思想

在实际应用中，可以根据具体问题选取**簇间距离计算方式**：

（1）完全连接聚类（complete-linkage agglomerative clustering）：将簇间距离定义为两个簇中**距离最远的**两个点之间的距离。这种计算方式容易受到极端值的影响，两个距离较近的簇可能因为极端值无法组合在一起

$$d(E_l, E_p) = \max(dist(\boldsymbol{x}_s, \boldsymbol{x}_t)), \boldsymbol{x}_s \in E_l, \boldsymbol{x}_t \in E_p$$

（2）单一连接聚类（single-linkage agglomerative clustering）：将簇间距离定义为两个簇中**距离最近的**两个点之间的距离。这种计算方式容易受到极端值的影响，两个距离较远的簇可能因为极端值被组合在一起

$$d(E_l, E_p) = \min(dist(\boldsymbol{x}_s, \boldsymbol{x}_t)), \boldsymbol{x}_s \in E_l, \boldsymbol{x}_t \in E_p$$

算法基本思想

在实际应用中，可以根据具体问题选取簇间距离计算方式：

（3）平均连接聚类（average agglomerative clustering）：**计算两个簇中每个样本到另一个簇中每一个样本距离的均值**。这种距离计算方式计算量较大，但更为合理，不容易受到异常值影响

$$d(E_l, E_p) = \frac{1}{|E_l||E_p|} \sum_{\boldsymbol{x}_s \in E_l} \sum_{\boldsymbol{x}_t \in E_p} dist(\boldsymbol{x}_s, \boldsymbol{x}_t)$$

算法基本思想

（4）离差平方和（Ward's Method） （阅读内容）

离差平方和法来自于方差分析。对于包含k个簇的数据集，其离差平方和被定义为：

$$S = \sum_{l=1,\ldots,k} \sum_{\boldsymbol{x}_i \in E_l} \left(\boldsymbol{x}_i - \boldsymbol{\mu}_l \right)^2$$

其中， $\boldsymbol{\mu}_l$ 表示簇E_l的聚类中心。

基于离差平方和进行层次聚类的基本思想是：**簇进行合并会增加离差平方和，每轮迭代选取使离差平方和增加最小的簇进行合并**。初始情况下，每个样本为一簇（此时离差平方和为0），后续每轮迭代将**使离差平方和增加最少**的两簇进行合并。

算法过程

考虑使用层次聚类算法将数据集的n个样本划分为k个不相交的簇，**算法步骤如下**：

（1）初始化，选择样本间距离和簇间距离的计算方法，将每个样本点各自设为独立的一簇，记为$E_1, \dots, E_n$

（2）计算任意两个簇间的距离$d(E_i, E_j)$，将距离最近的两个簇E_i 与 E_j 合并为$\{E_i, E_j\}$，将新的簇记为： $E_1, \dots, E_{n-p}$ (p为该步骤参与计算的次数)

（3）如果已经聚为k簇则停止，否则重复步骤（2），继续合并簇

层次聚类算法的优点在于灵活的距离计算公式使得它有很广的适用性，并且能够通过建立树的过程发现类的层次关系。但注意层次聚类算法的计算复杂度很高，所以**处理大规模数据聚类问题时，建议不要选择此算法**。

试一试

利用层次聚类算法，对无真实标签下的网格拥堵状态进行判别。选取“平均速度”和“平均停车次数”作为聚类特征，在50个样本的小数据集上进行聚类分析，来判断网格的拥堵状态。**查阅官方文档，调整模型参数，观察聚类结果变化。**

表 数据示例

编号	平均速度	平均停车次数	编号	平均速度	平均停车次数
1	4.99	3.26	26	20.83	0
2	7.11	0.66	27	21.01	1
3	3.64	0.72	28	25.61	0
4	10.17	3.32	29	12.18	0.29
5	8.23	0.48	30	22.74	2.5

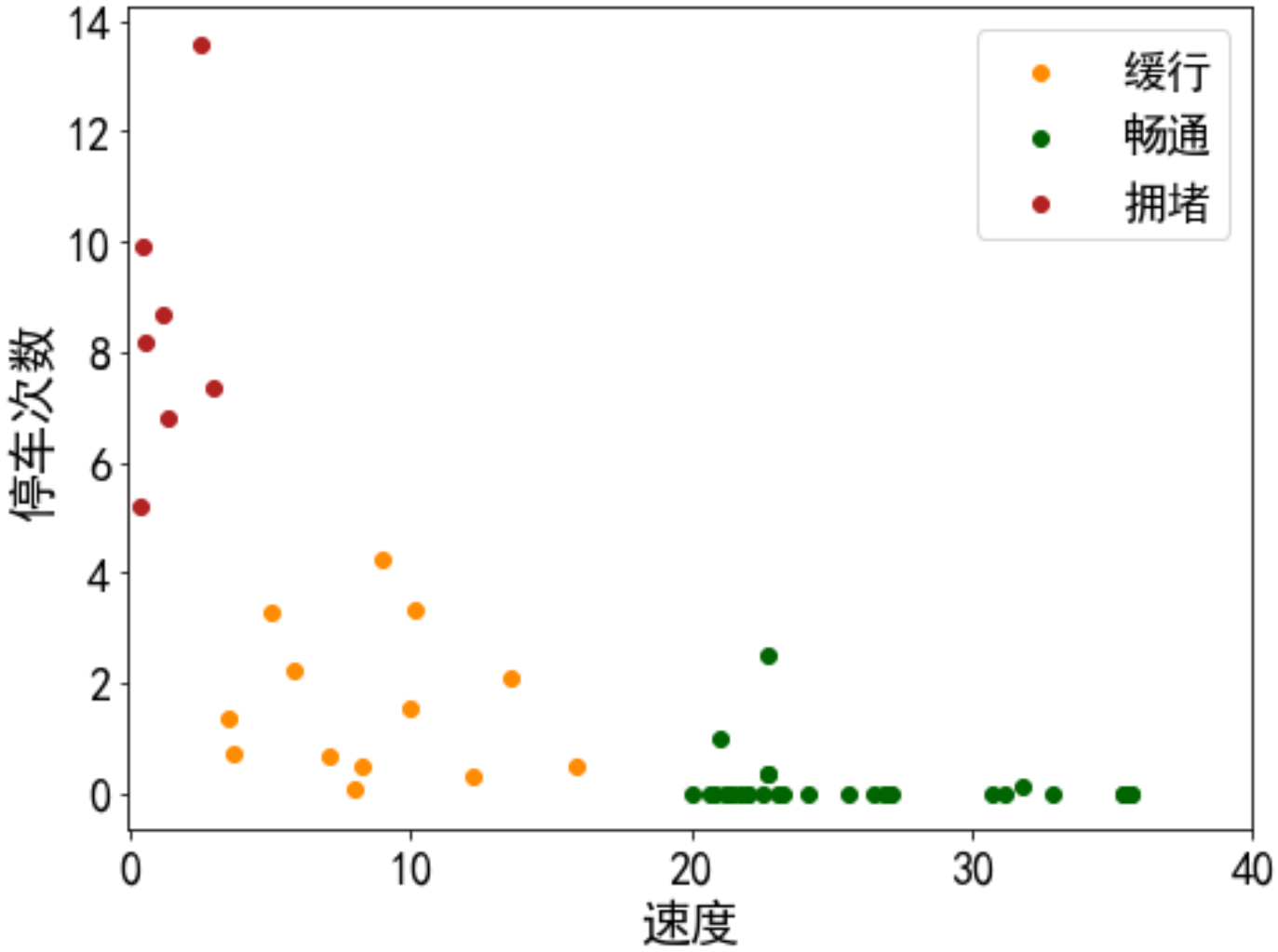
试一试

Agglomerative Clustering对象用于执行层次聚类，查阅官网文档（https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html）调整类内距离和连接规则的表示方法，观察聚类结果

```
# 读取数据
data_ori = pd.read_csv('聚类数据集.csv')
# 选择特征
feature = ['stopNum', 'aveSpeed']
# 数据标准化
scaler = StandardScaler()
scaler.fit(data_ori[feature])
data_ori_nor = scaler.transform(data_ori[feature])
# 层次聚类
n = 3
labels = AgglomerativeClustering(n_clusters=n).fit(data_ori_nor).labels_
# 输出数据集
output_data = pd.concat((data_ori,
                         pd.DataFrame(labels, columns = ['labels'])),
                         axis=1)
output_data.to_csv('层次聚类结果.csv', index=False)
```

试一试

结果可视化



5 基于密度的DBSCAN聚类方法

- 基本概念
- 算法过程
- 算法变体：ST-DBSCAN

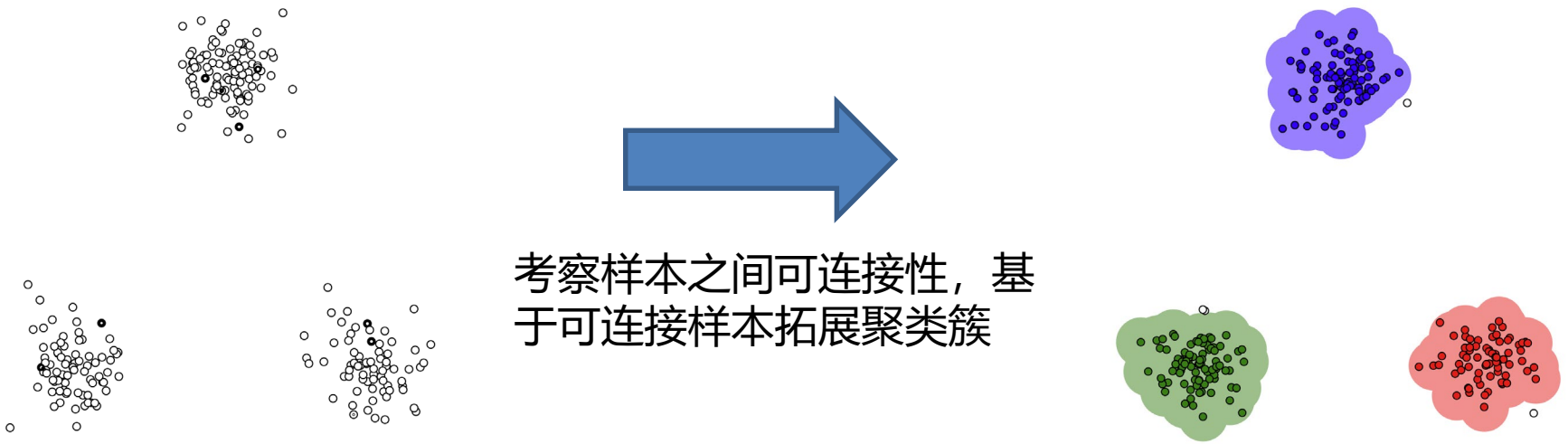
在前面几章的学习中，我们学习了几类经典聚类算法：K-means聚类，高斯混合聚类和层次聚类算法。前面介绍的几类算法有各自的局限性。

算法	局限性
K-means聚类	（1）需要事先确定K值； （2）初始聚类中心对最终结果影响大 （3）受离群点影响大
高斯混合聚类	（1）假设样本服从高斯分布，分布类型偏离真实情况时会导致聚类结果不准
层次聚类	（1）需要计算并存储所有对象之间的距离，这在处理大规模数据集时会产生较大的计算和存储开销。

为了解决这些问题，**基于密度的聚类算法**应运而生。该算法通过识别具有不同密度的数据区域，将数据集划分为不同的聚类，**具有更高的鲁棒性和适应性**。基于密度的聚类算法**不需要事先指定聚类簇数，**能够自适应地发现任意形状的聚类结构，**同时也不需要计算所有对象之间的距离，从而具有更高的计算效率**。因此，基于密度的聚类算法在处理大规模、高维度数据集时具有优越性，成为现代聚类分析中的重要方法之一。

基本概念

基于密度的聚类：根据样本的密度分布来进行聚类。通常情况下，密度聚类**从样本密度的角度出来，来考查样本之间的可连接性**，并**基于可连接样本不断扩展聚类簇**，以获得最终的聚类结果。DBSCAN（Density-Based Spatial Clustering of Applications with Noise）是最具代表性的一种密度聚类方法，DBSCAN可以在包含噪声的数据中发现任意形状的聚类

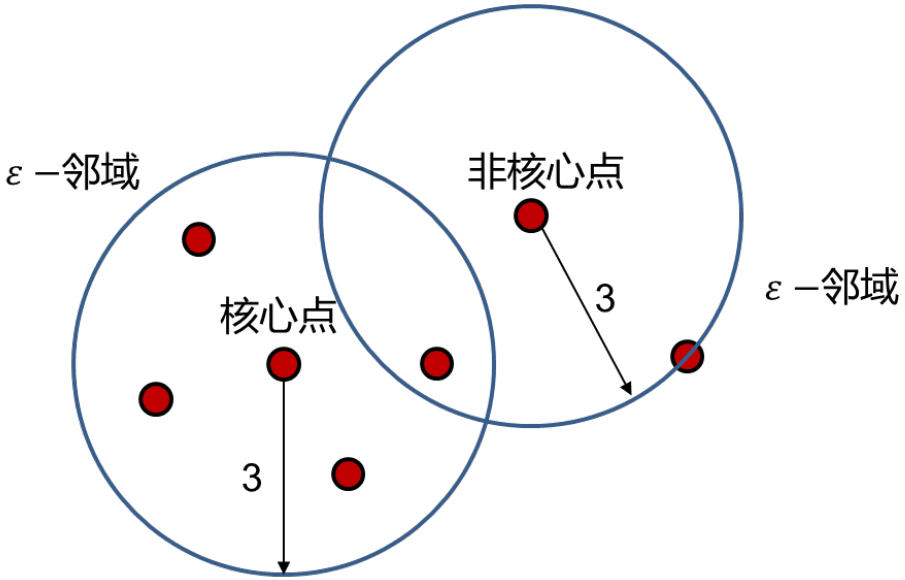


基本概念

给定**数据集**$D=\{\boldsymbol{x}_1,\dots\boldsymbol{x}_n\}$与**邻域参数**：$\varepsilon$（邻域半径），$MinPts$（邻域内最小包含点数），DBSCAN算法首先进行以下定义，

（1）ε −邻域：对于任意数据点$\boldsymbol{x}_j\in D$，其ε −邻域为数据集D中与$\boldsymbol{x}_j$的距离不大于ε的样本子集，即：$N_\varepsilon(\boldsymbol{x}_j)=\{\boldsymbol{x}_i\in D|dist(\boldsymbol{x}_i,\boldsymbol{x}_j)<=\varepsilon\}$

（2）核心点：若某样本点的ε −邻域包含不少于$MinPts$个样本，则称该点为核心点



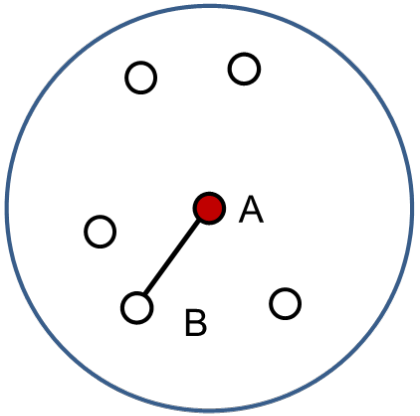
$\varepsilon=3,MinPts=4$ 时的ε −邻域与核心点

基本概念

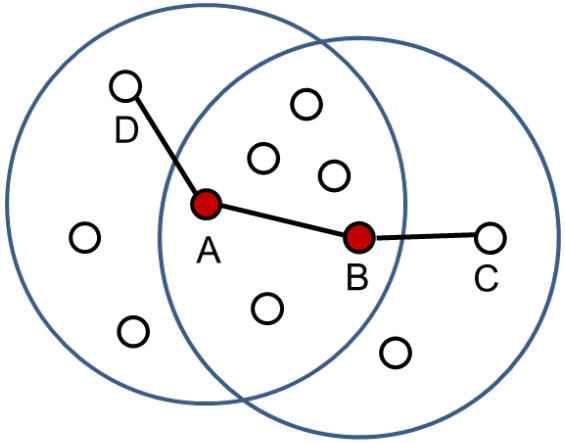
（3）密度直达：若$\boldsymbol{x}_j$在$\boldsymbol{x}_i$的ε −邻域中，且$\boldsymbol{x}_i$是核心点，则称$\boldsymbol{x}_j$ 由 $\boldsymbol{x}_i$密度直达（不具有对称性，即反之不一定成立）

（4）密度可达：对于$\boldsymbol{x}_j$ 与 $\boldsymbol{x}_i$，若存在样本序列$p_1, \ldots p_n$，其中$p_1 = \boldsymbol{x}_i$，$p_n = \boldsymbol{x}_j$，且p_{i+1} 由 p_i密度直达，则称$\boldsymbol{x}_j$ 由 $\boldsymbol{x}_i$密度可达（同样不具有对称性）

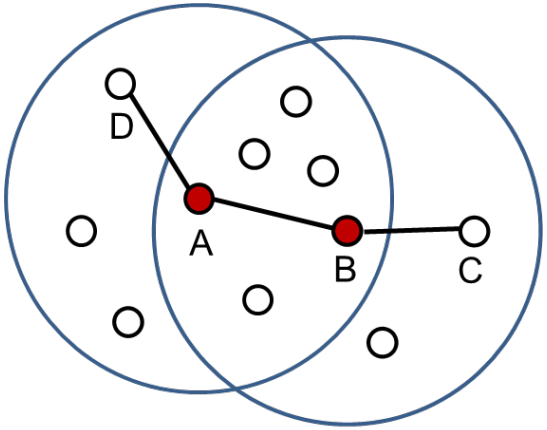
（5）密度相连：对于$\boldsymbol{x}_j$ 与 $\boldsymbol{x}_i$，若存在$\boldsymbol{x}_k$使$\boldsymbol{x}_j$ 与 $\boldsymbol{x}_i$均由$\boldsymbol{x}_k$密度可达，则称$\boldsymbol{x}_j$ 与 $\boldsymbol{x}_i$密度相连（具有对称性）



B由A密度直达



C由A密度可达
D由B密度可达



C与D密度相连

算法过程

给定数据集$D = \{\boldsymbol{x}_1, \ldots \boldsymbol{x}_n\}$，与邻域参数$\varepsilon$, $MinPts$, DBSCAN算法流程如下：

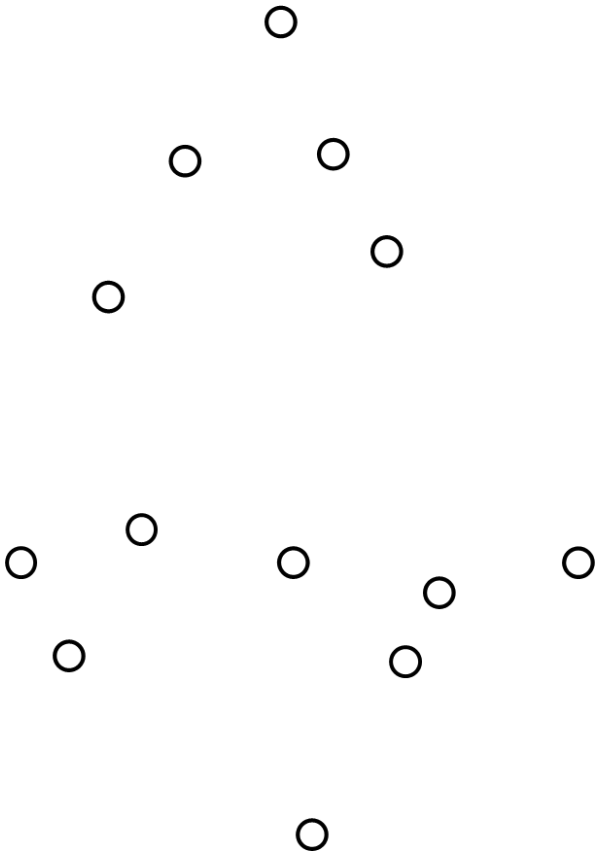
（1）遍历数据点，确定核心点

对于每个数据点，计算该点ε邻域内的样本点数，若点数低于$Minpts$，则称该点为低密度点，可能是边缘点或噪声点，而点数大于或等于$Minpts$的称为高密度点，即核心点。

（2）遍历无类别核心点，进行聚类

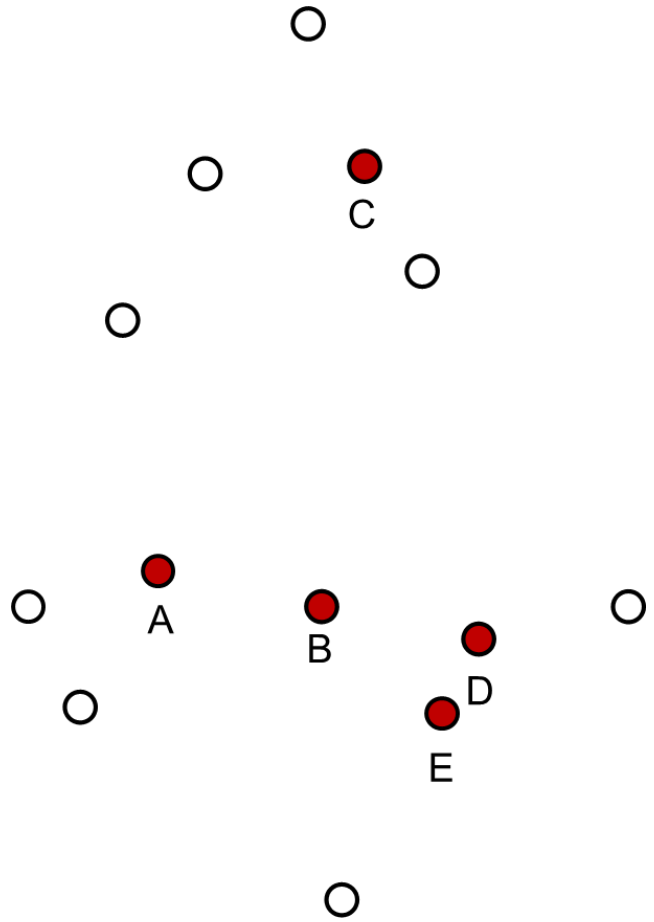
任意选择一个没有类别的核心点作为种子，**找到所有这个核心点能够密度可达的样本集合，即为一个聚类簇**。接着继续选择另一个没有类别的核心点去寻找密度可达的样本集合，这样就得到另一个聚类簇 。一直到所有核心点都有类别为止。**不在任何高密度点的圈内的低密度点就是噪声点**。

算法过程



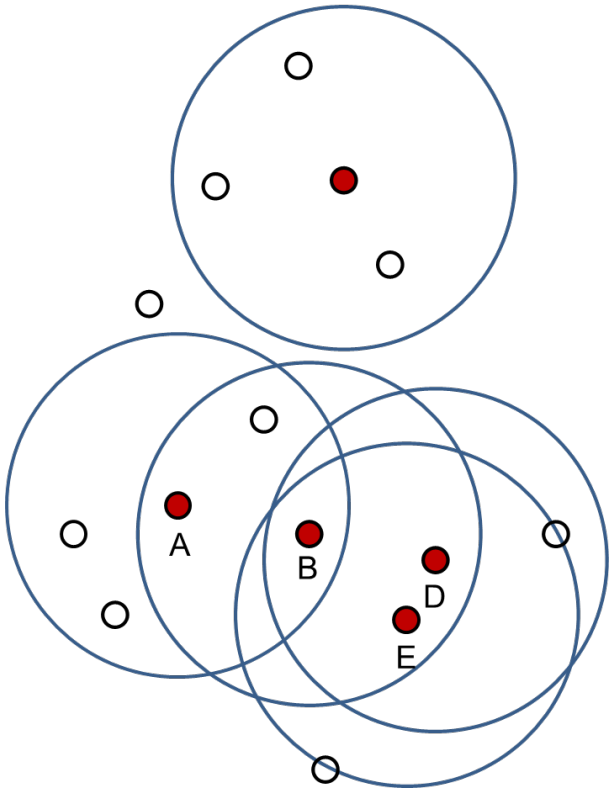
初始样本点

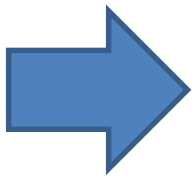
遍历样本点
找到核心点



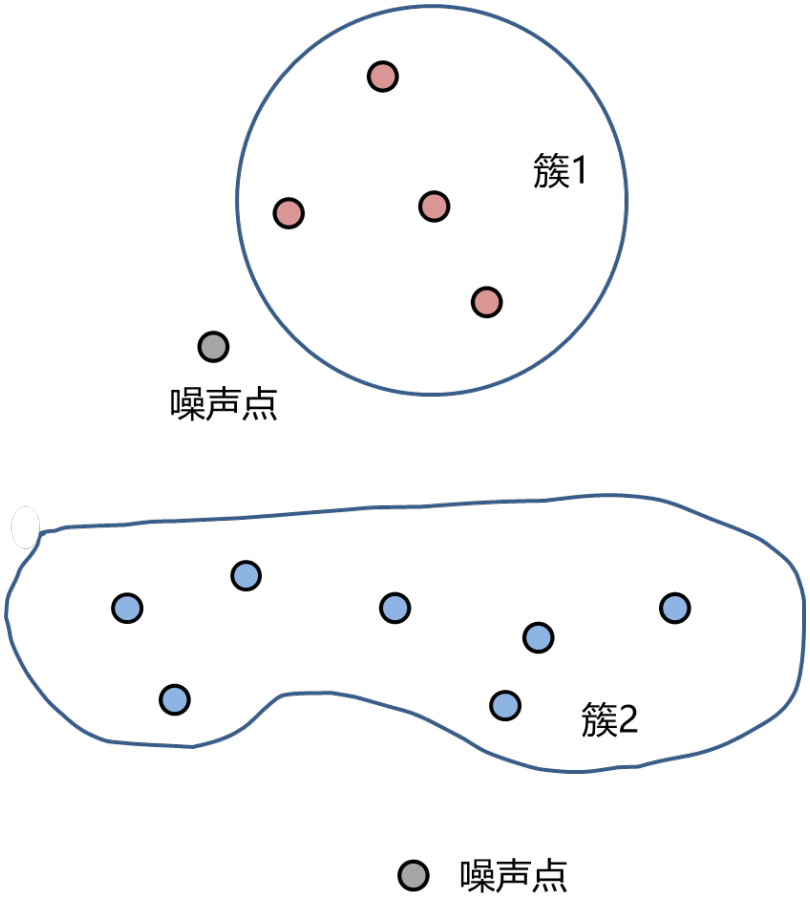
核心点与边缘点

算法过程





得到聚类结果

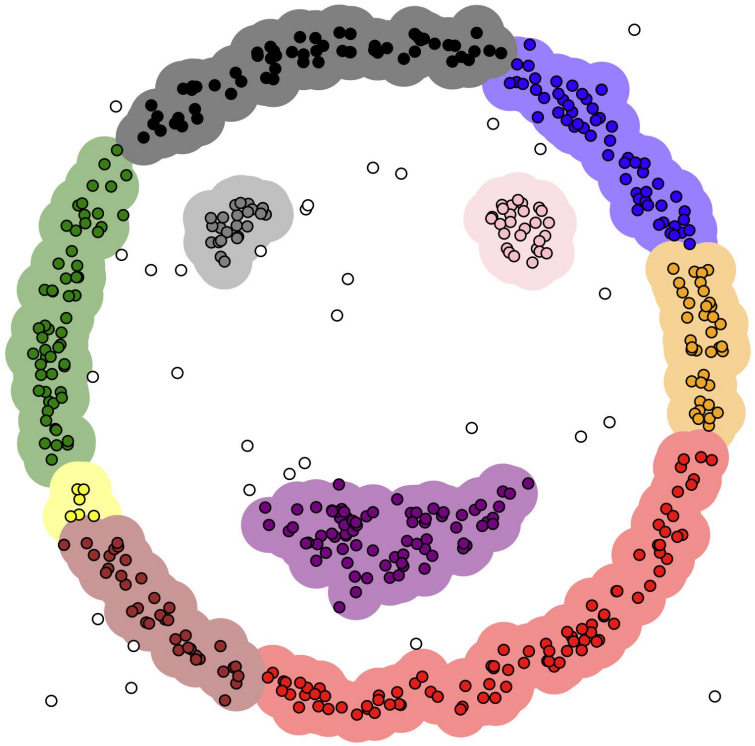


遍历核心点找到核心点的密度可达点

聚类结果

试一试

登录以下网站（https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/），自行选择数据集，设置参数，观察DBSCAN的聚类流程，找到核心点、噪声点和最终聚类得到的簇。



试一试

利用DBSCAN算法，对无真实标签下的网格拥堵状态进行判别。选取“平均速度”和“平均停车次数”作为聚类特征，在50个样本的小数据集上进行聚类分析，来判断网格的拥堵状态。**自行调整邻域参数，观察聚类结果变化情况**

表 数据示例

编号	平均速度	平均停车次数	编号	平均速度	平均停车次数
1	4.99	3.26	26	20.83	0
2	7.11	0.66	27	21.01	1
3	3.64	0.72	28	25.61	0
4	10.17	3.32	29	12.18	0.29
5	8.23	0.48	30	22.74	2.5

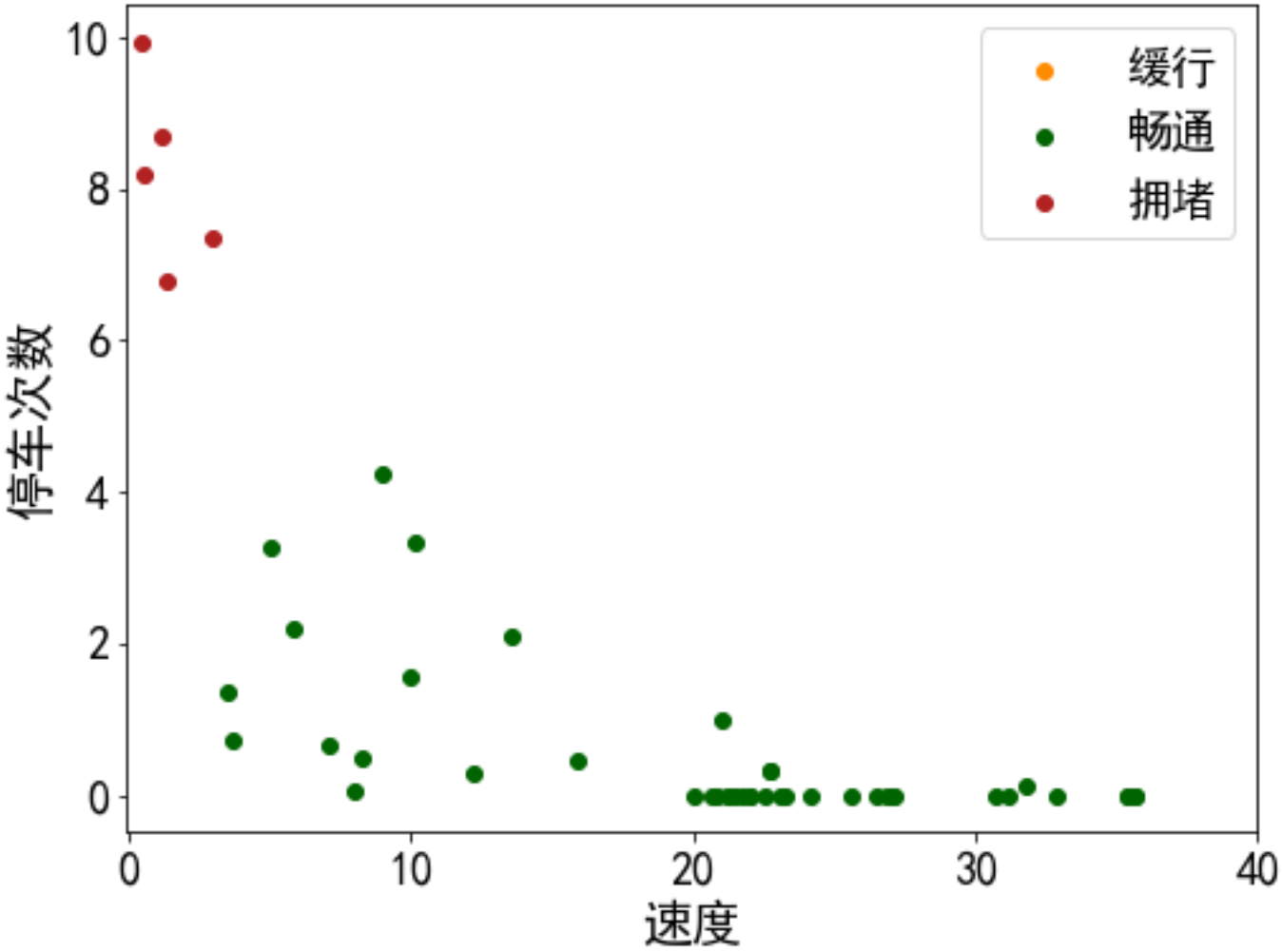
试一试

数据预处理与模型训练代码如下，**自行调整邻域参数，观察聚类结果变化情况**

```
# 读取数据
data_ori = pd.read_csv('聚类数据集.csv')
# 选择特征
feature = ['stopNum', 'aveSpeed']
# 数据标准化
scaler = StandardScaler()
scaler.fit(data_ori[feature])
data_ori_nor = scaler.transform(data_ori[feature])
# DBSCAN
eps = 0.5
min_samples = 3
labels = DBSCAN(eps=eps, min_samples=min_samples).fit(data_ori_nor).labels_
# 输出数据集
output_data = pd.concat((data_ori,
                         pd.DataFrame(labels, columns = ['labels'])),
                         axis=1)
output_data.to_csv('DBSCAN聚类结果.csv', index=False)
```

试一试

结果可视化



算法变体：ST-DBSCAN

DBSCAN算法也有很多变体，其中以Birant等首次提出的**ST-DBSCAN**最为著名，这一算法考虑了时空数据的聚类。时空数据，就是样本的特征还对应了时间和空间两个维度。空间信息通常由经度和纬度表示，时间信息通常由时间单元（time unit）表示。

执行ST-DBSCAN算法需要定义四个参数：ε_1，ε_2，$MinPts$，$\Delta\varepsilon$，

- ε_1用于搜索空间维度上的近邻点；
- ε_2用于搜索非空间维度上的近邻点；
- $MinPts$为核心点邻域内最小包含点数；
- $\Delta\varepsilon$是寻找密度相连样本时用到的阈值；

算法变体：ST-DBSCAN

ST-DBSCAN相比于DBSCAN进行了两方面的改进：

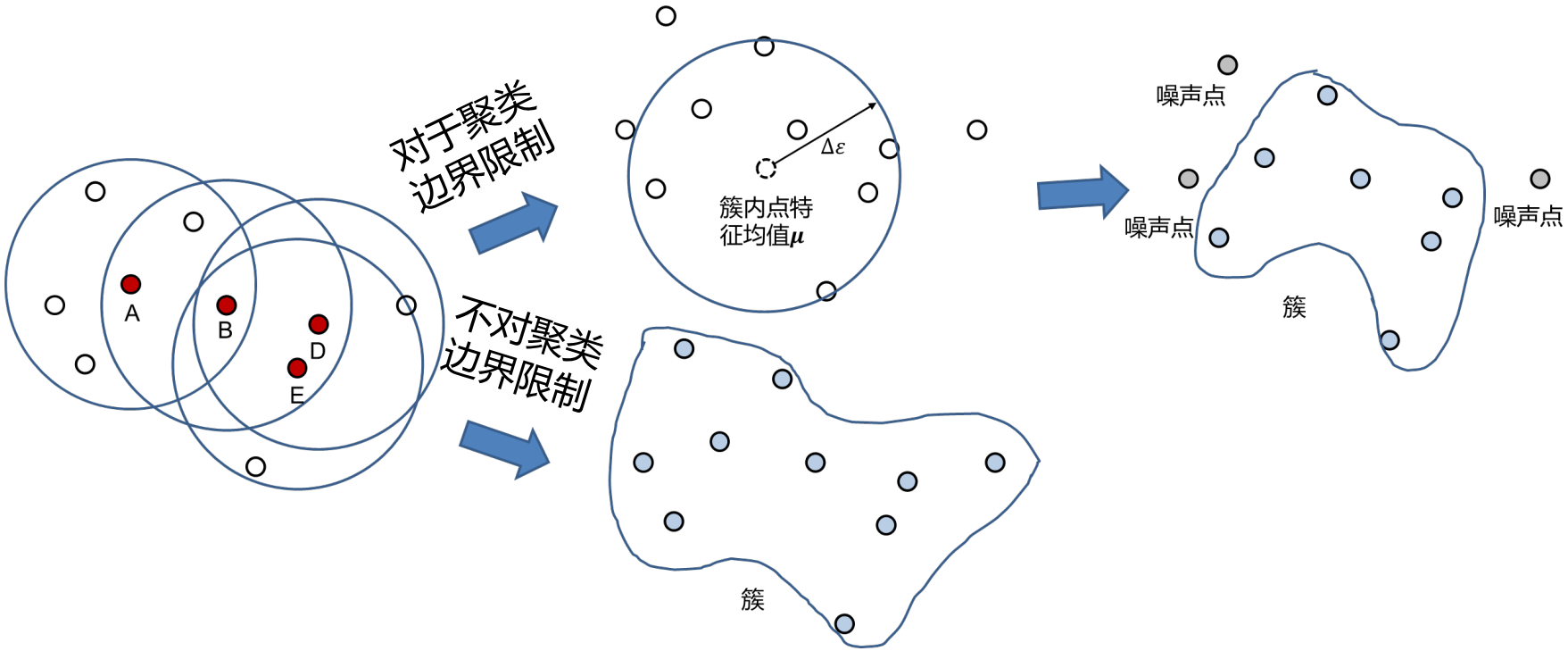
（1）在ST-DBSCAN中，点 x_j 由 x_i **密度直达**需要满足以下条件：

- x_i是核心点
- x_j 与 x_i空间距离小于ε_1
- $\mathbf{x}_\mathrm{j}$ 与 $\mathbf{x}_\mathrm{i}$非空间距离小于ε_2
- 两个样本在相邻的时间单元中被观察到（如一年内的相邻两天）

算法变体：ST-DBSCAN

（2）ST-DBSCAN利用$\Delta\varepsilon$**对聚类边界进行限制**

为了避免簇的边界对于相邻点的位置过于敏感，对于某核心点的密度可达点$\boldsymbol{x}_i$,仅当$dist(\boldsymbol{\mu}, \boldsymbol{x}_i) < \Delta\varepsilon$时， $\boldsymbol{x}_i$才归入该核心点所在的簇。这里$\boldsymbol{\mu}$为该簇所有点特征的均值



在实际应用中，ST-DBSCAN对于空间和时间特征的处理方式也可以用于其他数据特征