

一、理论部分

聚类总结

最重要的是，必须注意如何报告聚类分析的结果。

- 这些结果不应被视为数据集的绝对真理
- 相反，它们应该成为科学假设发展和进一步研究的起点，最好是在独立数据的基础上（按照各自的领域知识）
 - k-means：地理数据
 - 层次聚类：自底向上，获得更多信息 ## 聚类 ### 聚类的目的

给定一个多元数据集，寻找有意义的对象组，这样一个组中的对象彼此相似（或相关），并且与其他组中的对象不同。

聚类的应用

- 理解数据
 - 价格波动相似的集团股票
 - 分组研究具有相似功能的基因和蛋白质
- 总结数据
 - 减少大型数据集的大小 ### 评价聚类结果

对于无监督学习的聚类分析，需要建立数值评价标准。

- WSSE：在簇平方和。越小越好，k数量增多WSSE减少
- BSSE：簇间平方和。越小表示越集中，随着数量k增加而增加
- 轮廓系数 ### 聚类的分类
- 层次聚类：自底向上
- 划分聚类：k-means。将数据对象划分为不重叠的子集（集群），这样每个数据对象就恰好在一个子集中 ### k-means聚类
- 1、设置k个中心点
- 2、样本点距离k->分类
- 3、k-mean适合地理位置

比如：3-means聚类

- 1、随机设3个中心点。初始值的选择会带来影响
- 2、按照距离分成三类聚类结果
- 3、重新计算k中心的位置（质点）
- 4、重新计算距离 **一些困难**
- 不同大小和密度的簇和非球形的簇的困难：倾向于把大的和稀疏的簇打散 ### 层次聚类
- 如何定义集群间的差异性

方法如下：

- MIN(single-linkage)：容易受噪音值影响
- MAX(complete-linkage)：和k-means有点像，倾向于打散大的簇

- 为了选择集群，我们在树状图上划线（断点）

我们可以根据我们画的断点形成任意数量的集群，从而可以分成几类

- 缺点：
 - 无优化标准
 - 贪婪算法
 - 测量集群间距离的不同方法会产生非常不同的解决方案

二、实验部分

引入库

- 画图的：matplotlib.pyplot
- 聚类的：sklearn.cluster
- 数据集：sklearn.datasets

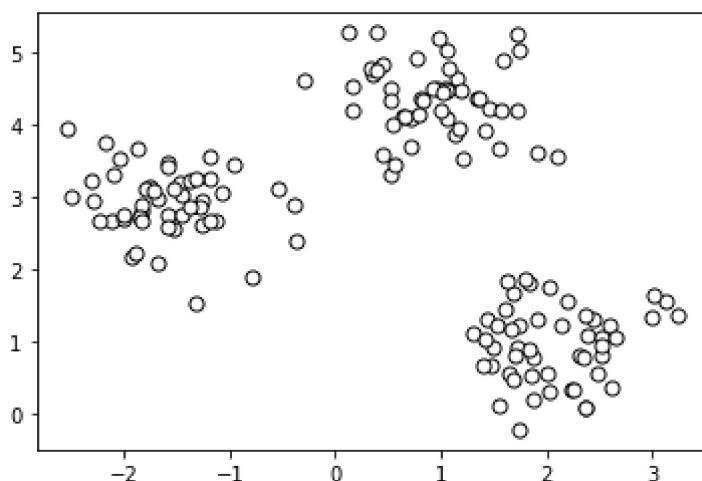
In []:

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
```

In []:

```
# make_blobs: 生成聚类的数据集
# n_samples: 生成的样本点个数, n_features: 样本特征数, centers: 样本中心数
# cluster_std: 聚类标准差, shuffle: 是否打乱数据, random_state: 随机种子
X, y = make_blobs(n_samples=150, n_features=2, centers=3, cluster_std=0.5, shuffle=True)

# 散点图
# c: 点的颜色, marker: 点的形状, edgecolor: 点边缘的形状, s: 点的大小
plt.scatter(X[:, 0], X[:, 1], c='white', marker='o', edgecolor='black', s=50)
plt.show()
```



In []:

```
# 定义模型
# n_clusters: 要形成的簇数, 即k均值的k, init: 初始化方式, tol: Frobenius 范数收敛的阈值
model = KMeans(n_clusters=3, init='random', n_init=10, max_iter=300, tol=1e-04, random_state=42)
# 训练加预测
y_pred = model.fit_predict(X)
y_pred
```

```
Out[ ]: array([1, 0, 0, 0, 1, 0, 0, 1, 2, 0, 1, 2, 2, 0, 0, 2, 2, 1, 2, 1, 0, 1,
   ..., 0, 0, 2, 1, 1, 0, 2, 1, 2, 2, 2, 2, 0, 1, 1, 1, 0, 0, 2, 2, 0, 1,
   ..., 1, 1, 2, 0, 2, 0, 1, 0, 0, 1, 1, 2, 0, 1, 2, 0, 2, 2, 2, 2, 0, 2,
   ..., 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 2, 2, 0, 1, 1, 0, 0, 1, 1, 1, 2,
   ..., 2, 1, 1, 0, 1, 0, 2, 2, 1, 1, 1, 2, 1, 1, 0, 2, 0, 0, 0,
   ..., 2, 0, 1, 2, 0, 0, 2, 2, 0, 1, 0, 0, 1, 1, 2, 1, 2, 2, 2, 2,
   ..., 1, 2, 2, 2, 0, 2, 1, 2, 0, 0, 1, 1, 2, 2, 2, 2, 1, 1])
```

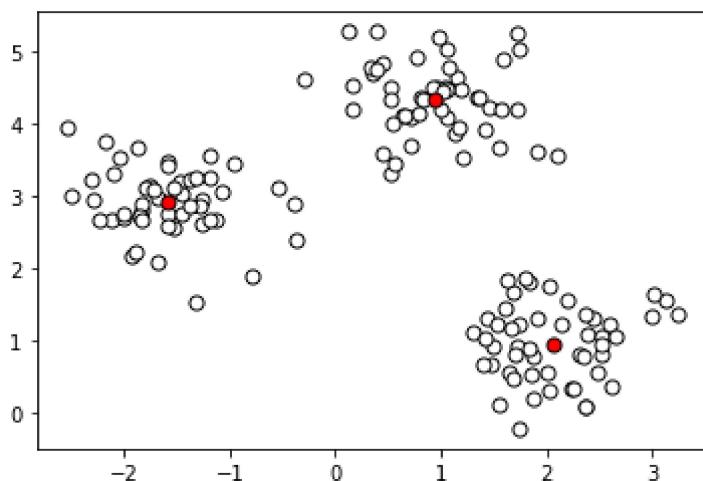
```
In [ ]: #获取聚类标签
#实际上得到的跟y_pred一致的
label = model.labels_
label
```

```
Out[ ]: array([1, 0, 0, 0, 1, 0, 0, 1, 2, 0, 1, 2, 2, 0, 0, 2, 2, 1, 2, 1, 0, 1,
   ..., 0, 0, 2, 1, 1, 0, 2, 1, 2, 2, 2, 2, 0, 1, 1, 1, 0, 0, 2, 2, 0, 1,
   ..., 1, 1, 2, 0, 2, 0, 1, 0, 0, 1, 1, 2, 0, 1, 2, 0, 2, 2, 2, 2, 0, 2,
   ..., 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 2, 2, 0, 1, 1, 0, 0, 1, 1, 1, 2,
   ..., 2, 1, 1, 0, 1, 0, 2, 2, 1, 1, 1, 2, 1, 1, 0, 2, 0, 0, 0,
   ..., 2, 0, 1, 2, 0, 0, 2, 2, 0, 1, 0, 0, 1, 1, 2, 1, 2, 2, 2, 2,
   ..., 1, 2, 2, 2, 0, 2, 1, 2, 0, 0, 1, 1, 2, 2, 2, 2, 1, 1])
```

```
In [ ]: #中心点
center = model.cluster_centers_
center
```

```
Out[ ]: array([[ 0.9329651 ,  4.35420712],
   ..., [ 2.06521743,  0.96137409],
   ..., [-1.5947298 ,  2.92236966]])
```

```
In [ ]: #显示中心点
plt.scatter(X[:, 0], X[:, 1], c='white', marker='o', edgecolor='black', s=50)
plt.scatter(center[:, 0], center[:, 1], c='red', marker='o', edgecolor='black', s=50)
plt.show()
```



```
In [ ]: # 画出预测的三个簇类
plt.scatter(
    X[y_pred == 0, 0], X[y_pred == 0, 1],
    s=50, c='lightgreen',
    marker='s', edgecolor='black',
    label='cluster 1'
)

plt.scatter(
    X[y_pred == 1, 0], X[y_pred == 1, 1],
    s=50, c='orange',
```

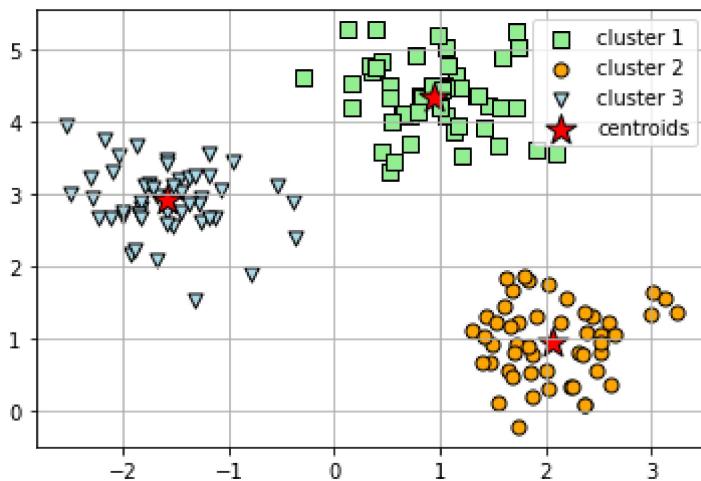
```

        marker='o', edgecolor='black',
        label='cluster 2'
    )

    plt.scatter(
        X[y_pred == 2, 0], X[y_pred == 2, 1],
        s=50, c='lightblue',
        marker='v', edgecolor='black',
        label='cluster 3'
    )

# 画出聚类中心
plt.scatter(
    center[:, 0], center[:, 1],
    s=250, marker='*',
    c='red', edgecolor='black',
    label='centroids'
)
#设置图例，图例中标记点的个数=1
plt.legend(scatterpoints=1)
plt.grid()
plt.show()

```

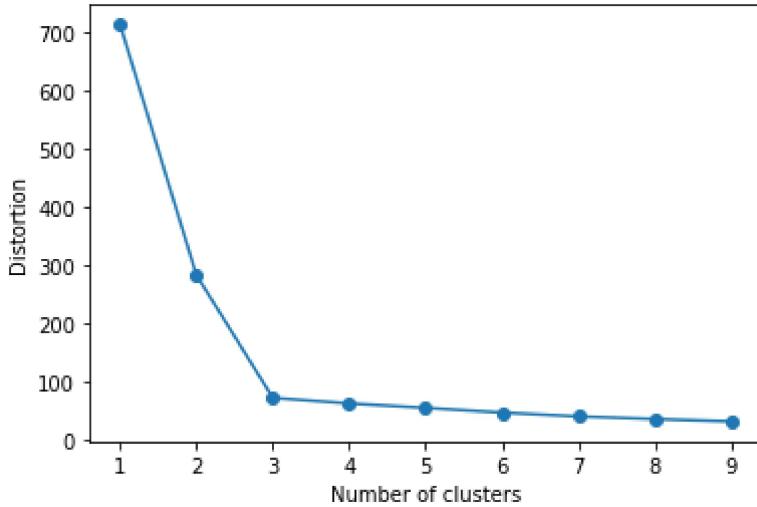


```

In [ ]: #检查最佳的聚类（聚成几类）
# 计算inertia随着k变化的情况
distortions = []
for i in range(1, 10):
    model = KMeans(
        n_clusters=i, init='random',
        n_init=10, max_iter=300,
        tol=1e-04, random_state=0
    )
    model.fit(X)
    distortions.append(model.inertia_)

# 画图可以看出k越大inertia越小，追求k越大对应用无益处，应该找到一个折点
plt.plot(range(1, 10), distortions, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.show()

```



有效性评价

因为是无监督学习，需要对模型进行有效性评价。常用的有Rand指数、轮廓系数.....

使用轮廓系数进行有效性评价

轮廓系数出于-1至1之间，**越大表示簇间相似度高而不同簇相似度低，也就是聚类效果好。**

```
In [ ]: from sklearn.metrics import silhouette_samples
lkxs = silhouette_samples(X, y_pred)
lkxs
```

```
Out[ ]: array([0.75956181, 0.43233677, 0.7747553 , 0.72286402, 0.76729229,
   0.72280273, 0.69484791, 0.76101499, 0.77303671, 0.67320272,
   0.62375802, 0.40348569, 0.80092777, 0.73225118, 0.59884632,
   0.78954087, 0.75955598, 0.81572606, 0.79897477, 0.7936958 ,
   0.72221435, 0.76384238, 0.72671597, 0.73219594, 0.74183186,
   0.81463826, 0.76045682, 0.62065362, 0.57912988, 0.76929277,
   0.707709 , 0.8045906 , 0.7116979 , 0.78561661, 0.66997314,
   0.65285806, 0.68519649, 0.77015391, 0.72638012, 0.77905009,
   0.56028722, 0.70806176, 0.66201873, 0.77669657, 0.79206045,
   0.79499104, 0.67874476, 0.43073904, 0.72878986, 0.75911278,
   0.61479508, 0.68506268, 0.76602547, 0.80521304, 0.79571018,
   0.77872478, 0.63708395, 0.62515462, 0.58878683, 0.7767243 ,
   0.72889549, 0.40154174, 0.80287718, 0.78189854, 0.76687374,
   0.35466307, 0.79135592, 0.56967556, 0.37334246, 0.54630918,
   0.61918571, 0.70584253, 0.79635805, 0.61369253, 0.74071059,
   0.77509834, 0.7351302 , 0.72821908, 0.74135332, 0.78854416,
   0.80090725, 0.78961228, 0.51299076, 0.78719254, 0.71872541,
   0.6397918 , 0.78855017, 0.78917529, 0.7061704 , 0.76230562,
   0.79536861, 0.77859235, 0.70994165, 0.73209753, 0.80237062,
   0.50658208, 0.79045011, 0.74253797, 0.75363719, 0.65437536,
   0.77241946, 0.79373523, 0.64697003, 0.8139295 , 0.76899613,
   0.79172653, 0.79274343, 0.69700878, 0.78652188, 0.67972242,
   0.71886096, 0.75294696, 0.81081904, 0.74715119, 0.70247941,
   0.71866965, 0.79175454, 0.71721843, 0.75476664, 0.80440263,
   0.67305151, 0.77173621, 0.71490094, 0.56114424, 0.59401827,
   0.77385159, 0.77492826, 0.7884554 , 0.75189627, 0.78474349,
   0.77665128, 0.67446502, 0.7425489 , 0.78239431, 0.72242176,
   0.5257976 , 0.49846727, 0.800814 , 0.70372273, 0.79779972,
   0.77516089, 0.74387209, 0.70953106, 0.81307092, 0.53081359,
   0.75791717, 0.66709398, 0.73611334, 0.79529978, 0.73534283])
```

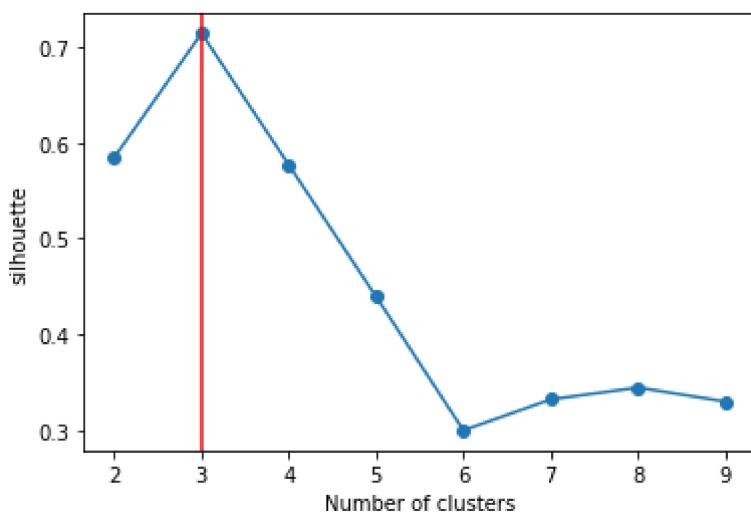
```
In [ ]: #求均值
import numpy as np
means = np.mean(lkxs)
means
```

```
Out[ ]: 0.7143417887288687
```

可以看到均值是0.71，可以写一个循环计算聚类数从2到n-1的轮廓系数进行评价，能看出最佳聚类数

```
In [ ]: def cluster_judge(n):
    model = KMeans(
        n_clusters=n, init='random',
        n_init=10, max_iter=300,
        tol=1e-04, random_state=0
    )
    model.fit(X)
    label = model.labels_
    lkxs = silhouette_samples(X, label)
    means = np.mean(lkxs)
    return means
```

```
In [ ]: y1 = []
for n in range(2, 10):
    means = cluster_judge(n)
    y1.append(means)
# 画图
plt.plot(range(2, 10), y1, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('silhouette')
max_n = np.max(y1)
plt.axvline(x=3, ymin=0, ymax=1, c='red')
plt.show()
```



结论

最高的是聚类为3的点