菜菜的scikit-learn课堂06

sklearn中的聚类算法K-Means

小伙伴们晚上好~o(￣▽￣)ブ

我是菜菜，这里是我的sklearn课堂第六期，今晚的直播内容是聚类算法K-Means~

我的开发环境是**Jupyter lab**，所用的库和版本大家参考：

Python 3.7.1（你的版本至少要3.4以上

Scikit-learn 0.20.1 （你的版本至少要0.20

Numpy 1.15.4, **Pandas** 0.23.4, **Matplotlib** 3.0.2, **SciPy** 1.1.0

请扫码进群领取课件和代码源文件，**扫描二维码后回复"K"就可以进群哦~**

菜菜的scikit-learn课堂06

sklearn中的聚类算法K-Means

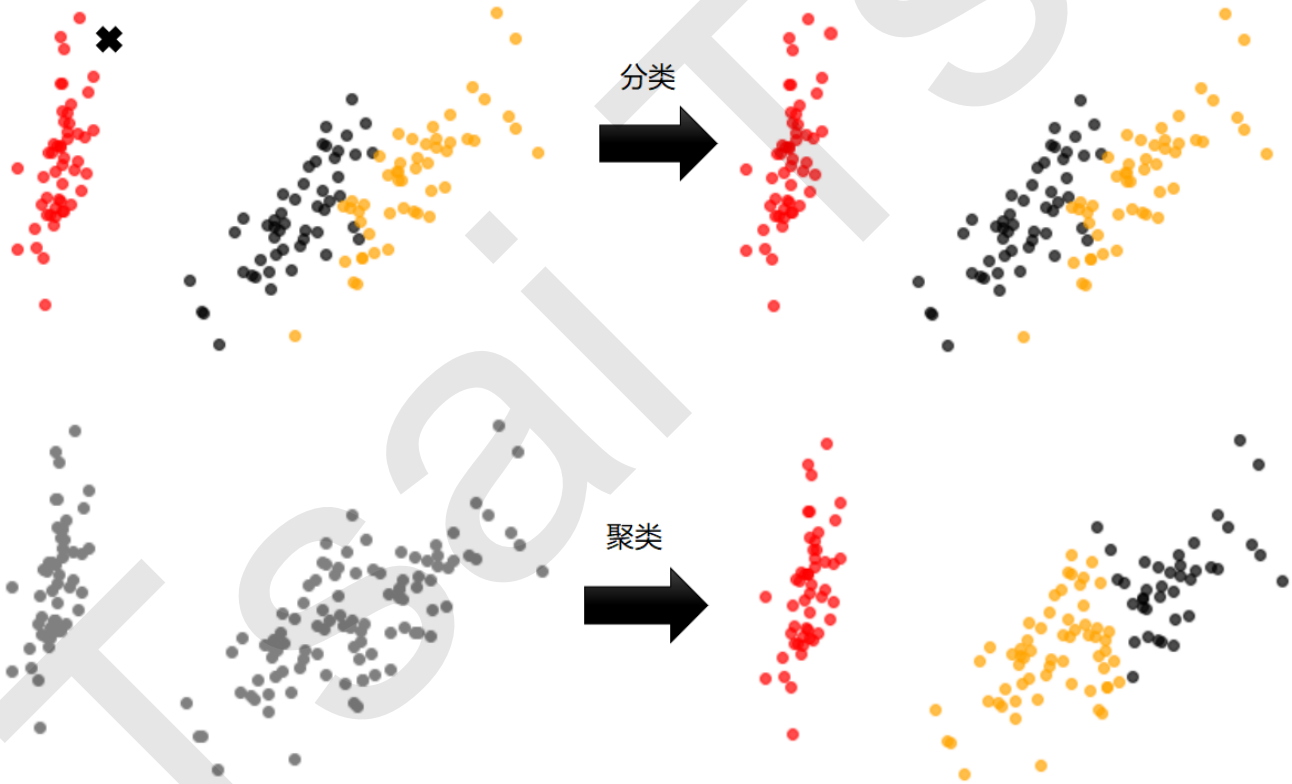
1 概述

1.1 无监督学习与聚类算法

在过去的五周之内，我们学习了决策树，随机森林，PCA和逻辑回归，他们虽然有着不同的功能，但却都属于“有监督学习”的一部分，即是说，模型在训练的时候，即需要特征矩阵X，也需要真实标签y。机器学习当中，还有相当一部分算法属于“无监督学习”，无监督的算法在训练的时候只需要特征矩阵X，不需要标签。而聚类算法，就是无监督学习的代表算法。

聚类算法又叫做“无监督分类”，其目的是将数据划分成有意义或有用的组（或簇）。这种划分可以基于我们的业务需求或建模需求来完成，也可以单纯地帮助我们探索数据的自然结构和分布。比如在商业中，如果我们手头有大量的当前和潜在客户的信息，我们可以使用聚类将客户划分为若干组，以便进一步分析和开展营销活动，最有名的客户价值判断模型RFM，就常常和聚类分析共同使用。再比如，聚类可以用于降维和矢量量化（vector quantization），可以将高维特征压缩到一列当中，常常用于图像，声音，视频等非结构化数据，可以大幅度压缩数据量。

- **聚类vs分类**



	聚类	**分类**
核心	将数据分成多个组 探索每个组的数据是否有联系	从已经分组的数据中去学习 把新数据放到已经分好的组中去
学习类型	无监督，无需标签进行训练	有监督，需要标签进行训练
典型算法	K-Means，DBSCAN，层次聚类，光谱聚类	决策树，贝叶斯，逻辑回归
算法输出	聚类结果是不确定的 不一定总是能够反映数据的真实分类 同样的聚类，根据不同的业务需求 可能是一个好结果，也可能是一个坏结果	分类结果是确定的 分类的优劣是客观的 不是根据业务或算法需求决定

1.2 sklearn中的聚类算法

聚类算法在sklearn中有两种表现形式，一种是类（和我们目前为止学过的分类算法以及数据预处理方法们都一样），需要实例化，训练并使用接口和属性来调用结果。另一种是函数（function），只需要输入特征矩阵和超参数，即可返回聚类的结果和各种指标。

类	含义	输入 [ ]内代表可以选择输入, [ ]外代表必须输入
cluster.AffinityPropagation	执行亲和传播数据聚类	[damping, ···]
cluster.AgglomerativeClustering	凝聚聚类	[···]
cluster.Birch	实现Birch聚类算法	[threshold, branching_factor, ···]
cluster.DBSCAN	从矢量数组或距离矩阵执行DBSCAN聚类	[eps, min_samples, metric, ···]
cluster.FeatureAgglomeration	凝聚特征	[n_clusters, ···]
cluster.KMeans	K均值聚类	[n_clusters, init, n_init, ···]
cluster.MiniBatchKMeans	小批量K均值聚类	[n_clusters, init, ···]
cluster.MeanShift	使用平坦核函数的平均移位聚类	[bandwidth, seeds, ···]
cluster.SpectralClustering	光谱聚类，将聚类应用于规范化拉普拉斯的投影	[n_clusters, ···]

函数	含义	输入
cluster.affinity_propagation	执行亲和传播数据聚类	S[, ···]
cluster.dbscan	从矢量数组或距离矩阵执行DBSCAN聚类	X[, eps, min_samples, ···]
cluster.estimate_bandwidth	估计要使用均值平移算法的带宽	X[, quantile, ···]
cluster.k_means	K均值聚类	X, n_clusters[, ···]
cluster.mean_shift	使用平坦核函数的平均移位聚类	X[, bandwidth, seeds, ···]
cluster.spectral_clustering	将聚类应用于规范化拉普拉斯的投影	affinity[, ···]
cluster.ward_tree	光谱聚类，将聚类应用于规范化拉普拉斯的投影	X[, connectivity, ···]

- **输入数据**

需要注意的一件重要事情是，该模块中实现的算法可以采用不同类型的矩阵作为输入。 所有方法都接受形状[n_samples，n_features]的标准特征矩阵，这些可以从sklearn.feature_extraction模块中的类中获得。对于亲和力传播，光谱聚类和DBSCAN，还可以输入形状[n_samples，n_samples]的相似性矩阵，我们可以使用sklearn.metrics.pairwise模块中的函数来获取相似性矩阵。

2 KMeans

2.1 KMeans是如何工作的

作为聚类算法的典型代表，KMeans可以说是最简单的聚类算法没有之一，那它是怎么完成聚类的呢？

关键概念：簇与质心

KMeans算法将一组N个样本的特征矩阵X划分为K个无交集的簇，直观上来看是簇是一组一组聚集在一起的数据，在一个簇中的数据就认为是同一类。簇就是聚类的结果表现。

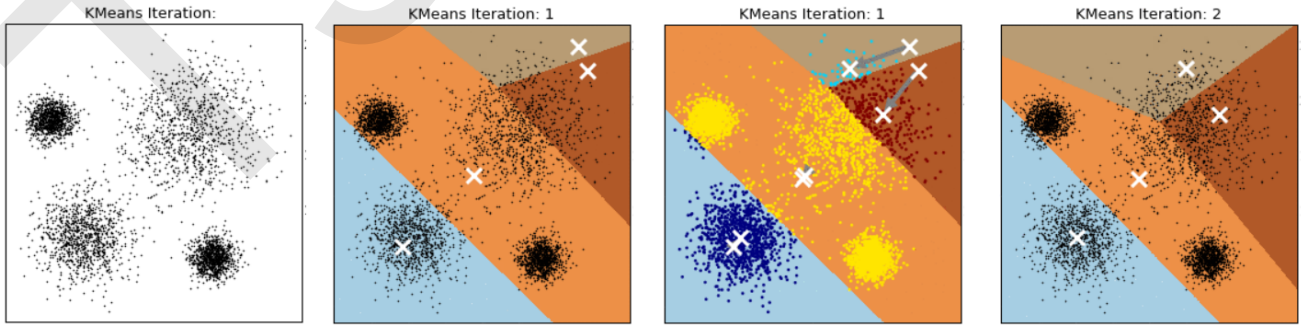
簇中所有数据的均值μ_j通常被称为这个簇的"质心"（centroids）。在一个二维平面中，一簇数据点的质心的横坐标就是这一簇数据点的横坐标的均值，质心的纵坐标就是这一簇数据点的纵坐标的均值。同理可推广至高维空间。

在KMeans算法中，簇的个数K是一个超参数，需要我们人为输入来确定。KMeans的核心任务就是根据我们设定好的K，找出K个最优的质心，并将离这些质心最近的数据分别分配到这些质心代表的簇中去。具体过程可以总结如下：

顺序	过程
1	随机抽取K个样本作为最初的质心
2	开始循环：
2.1	将每个样本点分配到离他们最近的质心，生成K个簇
2.2	对于每个簇，计算所有被分到该簇的样本点的平均值作为新的质心
3	当质心的位置不再发生变化，迭代停止，聚类完成

那什么情况下，质心的位置会不再变化呢？当我们找到一个质心，在每次迭代中被分配到这个质心上的样本都是一致的，即每次新生成的簇都是一致的，所有的样本点都不会再从一个簇转移到另一个簇，质心就不会变化了。

这个过程在可以由下图来显示，我们规定，将数据分为4簇（K=4），其中白色X代表质心的位置：



在数据集下多次迭代(iteration)，模型就会收敛。第六次迭代之后，基本上质心的位置就不再改变了，生成的簇也变得稳定。此时我们的聚类就完成了，我们可以明显看出，KMeans按照数据的分布，将数据聚集成了我们规定的4类，接下来我们就可以按照我们的业务需求或者算法需求，对这四类数据进行不同的处理。

2.2 簇内误差平方和的定义和解惑

聚类算法聚出的类有什么含义呢？这些类有什么样的性质？我们认为，**被分在同一个簇中的数据是有相似性的，而不同簇中的数据是不同的**，当聚类完毕之后，我们就要分别去研究每个簇中的样本都有什么样的性质，从而根据业务需求制定不同的商业或者科技策略。这个听上去和我们在上周的评分卡案例中讲解的"分箱"概念有些类似，即我们分箱的目的是希望，一个箱内的人有着相似的信用风险，而不同箱的人的信用风险差异巨大，以此来区别不同信用度的人，因此我们追求"组内差异小，组间差异大"。聚类算法也是同样的目的，我们追求"簇内差异小，簇外差异大"。而这个"差异"，由**样本点到其所在簇的质心的距离**来衡量。

对于一个簇来说，所有样本点到质心的距离之和越小，我们就认为这个簇中的样本越相似，簇内差异就越小。而距离的衡量方法有多种，令x表示簇中的一个样本点，μ表示该簇中的质心，n表示每个样本点中的特征数目，i表示组成点x的每个特征，则该样本点到质心的距离可以由以下距离来度量：

$$\text{欧几里得距离：}\ d(x,\mu)=\sqrt{\sum_{i=1}^{n}(x_i-\mu_i)^2}$$

$$\text{曼哈顿距离：}\ d(x,\mu)=\sum_{i=1}^{n}(|x_i-\mu|)$$

$$\text{余弦距离：}\ cos\theta=\frac{\sum_1^n(x_i*\mu)}{\sqrt{\sum_1^n(x_i)^2}*\sqrt{\sum_1^n(\mu)^2}}$$

如我们采用欧几里得距离，则一个簇中所有样本点到质心的距离的平方和为：

$$Cluster\ Sum\ of\ Square（CSS）=\sum_{j=0}^{m}\sum_{i=1}^{n}(x_i-\mu_i)^2$$

$$Total\ Cluster\ Sum\ of\ Square=\sum_{l=1}^{k}CSS_l$$

其中，m为一个簇中样本的个数，j是每个样本的编号。这个公式被称为**簇内平方和**（cluster Sum of Square），又叫做Inertia。而将一个数据集中的所有簇的簇内平方和相加，就得到了整体平方和（Total Cluster Sum of Square），又叫做total inertia。Total Inertia越小，代表着每个簇内样本越相似，聚类的效果就越好。**因此KMeans追求的是，求解能够让Inertia最小化的质心。**实际上，在质心不断变化不断迭代的过程中，总体平方和是越来越小的。我们可以使用数学来证明，当整体平方和最小的时候，质心就不再发生变化了。如此，K-Means的求解过程，就变成了一个最优化问题。

这是我们在这个课程中第二次遇见最优化问题，即需要将某个指标最小化来求解模型中的一部分信息。记得我们在逻辑回归中式怎么做的吗？我们在一个固定的方程$y(x)=\frac{1}{1+e^{\theta^T x}}$中最小化损失函数来求解模型的参数向量$\theta$，并且基于参数向量$\theta$的存在去使用模型。而在KMeans中，我们在一个固定的簇数K下，最小化总体平方和来求解最佳质心，并基于质心的存在去进行聚类。两个过程十分相似，并且，整体距离平方和的最小值其实可以使用梯度下降来求解。因此，有许多博客和教材都这样写道：簇内平方和/整体平方和是KMeans的损失函数。

解惑：Kmeans有损失函数吗？

记得我们在逻辑回归中曾有这样的结论：损失函数本质是用来衡量模型的拟合效果的，只有有着求解参数需求的算法，才会有损失函数。Kmeans不求解什么参数，它的模型本质也没有在拟合数据，而是在对数据进行一种探索。所以如果你去问大多数数据挖掘工程师，甚至是算法工程师，他们可能会告诉你说，K-Means不存在什么损失函数，Inertia更像是Kmeans的模型评估指标，而非损失函数。

但我们类比过了Kmeans中的Inertia和逻辑回归中的损失函数的功能，我们发现它们确实非常相似。所以，从"求解模型中的某种信息，用于后续模型的使用"这样的功能来看，我们可以认为Inertia是Kmeans中的损失函数，虽然这种说法并不严谨。

对比来看，在决策树中，我们有衡量分类效果的指标准确度accuracy，我们不能通过最小化accuracy来求解某个模型中需要的信息。因此决策树，KNN等算法，是绝对没有损失函数的。

大家可以发现，我们的Inertia是基于欧几里得距离的计算公式得来的。实际上，我们也可以使用其他距离，每个距离都有自己对应的Inertia。在过去的经验中，我们总结出不同距离所对应的质心选择方法和Inertia，在Kmeans中，只要使用了正确的质心和距离组合，无论使用什么样的距离，都可以达到不错的聚类效果：

距离度量	质心	Inertia
欧几里得距离	均值	最小化每个样本点到质心的欧式距离之和
曼哈顿距离	中位数	最小化每个样本点到质心的曼哈顿距离之和
余弦距离	均值	最小化每个样本点到质心的余弦距离之和

而这些组合，都可以由严格的数学证明来推导。在sklearn当中，我们无法选择使用的距离，只能使用欧式距离。因此，我们也无需去担忧这些距离所搭配的质心选择是如何得来的了。

3 sklearn.cluster.KMeans

class `sklearn.cluster.KMeans`(*n_clusters=8*, *init='k-means++'*, *n_init=10*, *max_iter=300*, *tol=0.0001*, *precompute_distances='auto'*, *verbose=0*, *random_state=None*, *copy_x=True*, *n_jobs=None*, *algorithm='auto'*)

3.1 重要参数n_clusters

n_clusters是KMeans中的k，表示着我们告诉模型我们要分几类。这是KMeans当中唯一一个必填的参数，默认为8类，但通常我们的聚类结果会是一个小于8的结果。通常，在开始聚类之前，我们并不知道n_clusters究竟是多少，因此我们要对它进行探索。

3.1.1 先进行一次聚类看看吧

当我们拿到一个数据集，如果可能的话，我们希望能够通过绘图先观察一下这个数据集的数据分布，以此来为我们聚类时输入的n_clusters做一个参考。

首先，我们来自己创建一个数据集。这样的数据集是我们自己创建，所以是有标签的。

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

X, y = make_blobs(n_samples=500,n_features=2,centers=4,random_state=1)

fig, ax1 = plt.subplots(1)
ax1.scatter(X[:, 0], X[:, 1]
            ,marker='o'
            ,s=8
           )
plt.show()

color = ["red","pink","orange","gray"]
fig, ax1 = plt.subplots(1)

for i in range(4):
    ax1.scatter(X[y==i, 0], X[y==i, 1]
            ,marker='o'
            ,s=8
            ,c=color[i]
           )
plt.show()
```

基于这个分布，我们来使用Kmeans进行聚类。首先，我们要猜测一下，这个数据中有几簇？

```
from sklearn.cluster import KMeans

n_clusters = 3

cluster = KMeans(n_clusters=n_clusters, random_state=0).fit(X)

y_pred = cluster.labels_
y_pred
```

```
pre = cluster.fit_predict(X)
pre == y_pred

centroid = cluster.cluster_centers_
centroid

centroid.shape

inertia = cluster.inertia_
inertia

color = ["red","pink","orange","gray"]
fig, ax1 = plt.subplots(1)

for i in range(guessK):
    ax1.scatter(X[y_pred==i, 0], X[y_pred==i, 1]
            ,marker='o'
            ,s=8
            ,c=color[i]
           )
ax1.scatter(centroid[:,0],centroid[:,1]
           ,marker="x"
           ,s=15
           ,c="black")
plt.show()

n_clusters = 4

cluster_ = KMeans(n_clusters=n_clusters, random_state=0).fit(X)
inertia_ = cluster_.inertia_
inertia_
```

3.1.2 聚类算法的模型评估指标

不同于分类模型和回归，聚类算法的模型评估不是一件简单的事。在分类中，有直接结果（标签）的输出，并且分类的结果有正误之分，所以我们使用预测的准确度，混淆矩阵，ROC曲线等等指标来进行评估，但无论如何评估，都是在"模型找到正确答案"的能力。而回归中，由于要拟合数据，我们有SSE均方误差，有损失函数来衡量模型的拟合程度。但这些衡量指标都不能够使用于聚类。

面试高危问题：如何衡量聚类算法的效果？
聚类模型的结果不是某种标签输出，并且聚类的结果是不确定的，其优劣由业务需求或者算法需求来决定，并且没有永远的正确答案。那我们如何衡量聚类的效果呢？

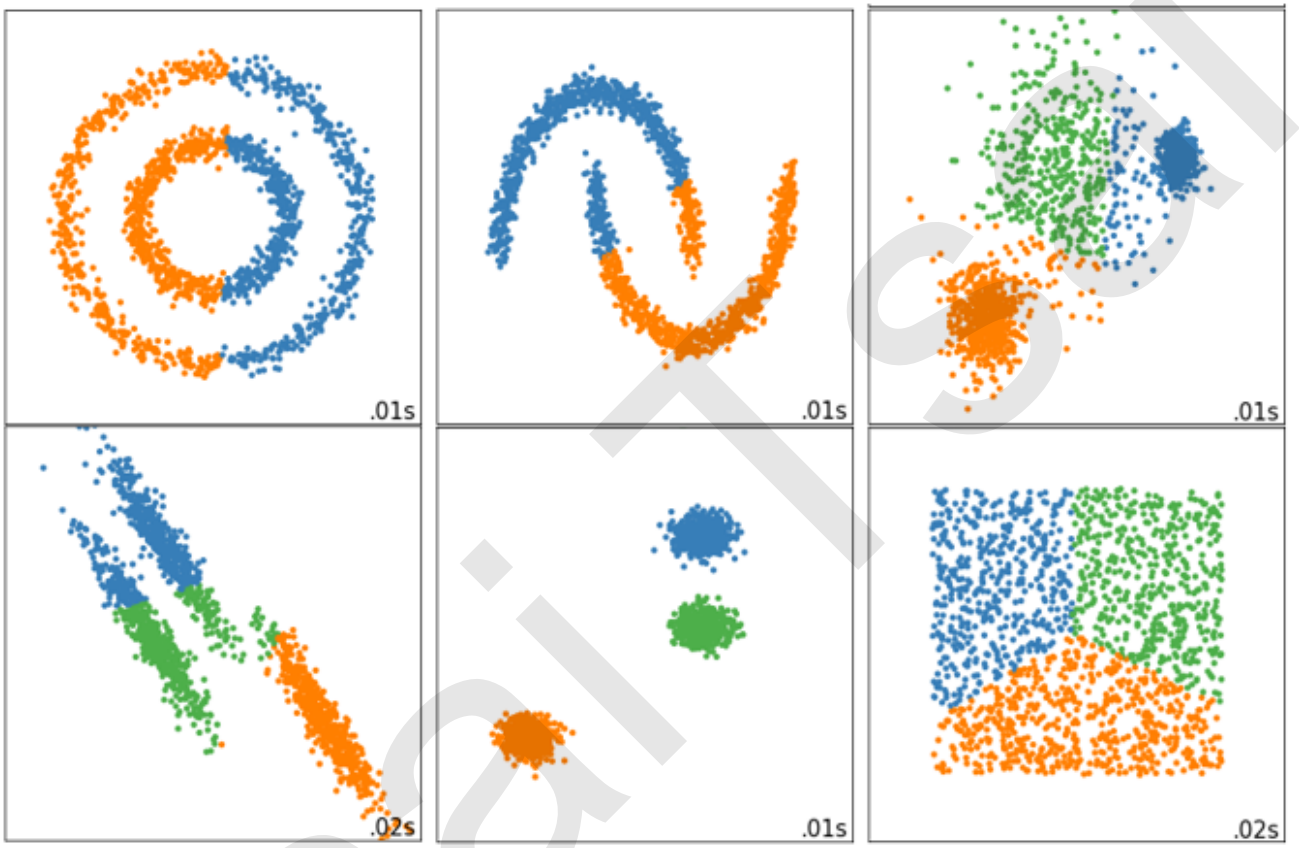
记得我们说过，KMeans的目标是确保"簇内差异小，簇外差异大"，我们就可以通过**衡量簇内差异来衡量聚类的效果**。我们刚才说过，Inertia是用距离来衡量簇内差异的指标，因此，我们是否可以使用Inertia来作为聚类的衡量指标呢？Inertia越小模型越好嘛。

可以，但是这个指标的缺点和极限太大。

首先，它不是有界的。我们只知道，Inertia是越小越好，是0最好，但我们不知道，一个较小的Inertia究竟有没有达到模型的极限，能否继续提高。

第二，它的计算太容易受到特征数目的影响，数据维度很大的时候，Inertia的计算量会陷入维度诅咒之中，计算量会爆炸，不适合用来一次次评估模型。

第三，Inertia对数据的分布有假设，它假设数据满足凸分布（即数据在二维平面图像上看起来是一个凸函数的样子），并且它假设数据是各向同性的（isotropic），即是说数据的属性在不同方向上代表着相同的含义。但是现实中的数据往往不是这样。所以使用Inertia作为评估指标，会让聚类算法在一些细长簇，环形簇，或者不规则形状的流形时表现不佳：



那我们可以使用什么指标呢？来使用轮廓系数。

在99%的情况下，我们是对没有真实标签的数据进行探索，也就是对不知道真正答案的数据进行聚类。这样的聚类，是完全依赖于评价簇内的稠密程度（簇内差异小）和簇间的离散程度（簇外差异大）来评估聚类的效果。其中轮廓系数是最常用的聚类算法的评价指标。它是对每个样本来定义的，它能够同时衡量：

1）样本与其自身所在的簇中的其他样本的相似度**a**，等于样本与同一簇中所有其他点之间的平均距离

2）样本与其他簇中的样本的相似度**b**，等于样本与下一个最近的簇中得所有点之间的平均距离

根据聚类的要求"簇内差异小，簇外差异大"，我们希望b永远大于a，并且大得越多越好。

单个样本的轮廓系数计算为：

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

这个公式可以被解析为：

$$s = \begin{cases} 1 - a/b, & if\ a < b \\ 0, & if\ a = b \\ b/a - 1, & if\ a > b \end{cases}$$

很容易理解轮廓系数范围是(-1,1)，其中值越接近1表示样本与自己所在的簇中的样本很相似，并且与其他簇中的样本不相似，当样本点与簇外的样本更相似的时候，轮廓系数就为负。当轮廓系数为0时，则代表两个簇中的样本相似度一致，两个簇本应该是一个簇。

如果一个簇中的大多数样本具有比较高的轮廓系数，则簇会有较高的总轮廓系数，则整个数据集的平均轮廓系数越高，则聚类是合适的。如果许多样本点具有低轮廓系数甚至负值，则聚类是不合适的，聚类的超参数K可能设定得太大或者太小。

在sklearn中，我们使用模块metrics中的类silhouette_score来计算轮廓系数，它返回的是一个数据集中，所有样本的轮廓系数的均值。但我们还有同在metrics模块中的silhouette_sample，它的参数与轮廓系数一致，但返回的是数据集中每个样本自己的轮廓系数。

我们来看看轮廓系数在我们自建的数据集上表现如何：

```
from sklearn.metrics import silhouette_score
from sklearn.metrics import silhouette_samples

X
y_pred

silhouette_score(X,y_pred)

silhouette_score(X,cluster_.labels_)

silhouette_samples(X,y_pred)
```

轮廓系数有很多优点，它在有限空间中取值，使得我们对模型的聚类效果有一个“参考”。并且，轮廓系数对数据的分布没有假设，因此在很多数据集上都表现良好。但它在每个簇的分割比较清洗时表现最好。但轮廓系数也有缺陷，它在凸型的类上表现会虚高，比如基于密度进行的聚类，或通过DBSCAN获得的聚类结果，如果使用轮廓系数来衡量，则会表现出比真实聚类效果更高的分数。

3.1.3 案例：基于轮廓系数来选择n_clusters

我们通常会绘制轮廓系数分布图和聚类后的数据分布图来选择我们的最佳n_clusters。

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np

n_clusters = 4
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.set_size_inches(18, 7)
ax1.set_xlim([-0.1, 1])
```

```
ax1.set_ylim([0, X.shape[0] + (n_clusters + 1) * 10])
clusterer = KMeans(n_clusters=n_clusters, random_state=10).fit(X)
cluster_labels = clusterer.labels_

silhouette_avg = silhouette_score(X, cluster_labels)
print("For n_clusters =", n_clusters,
      "The average silhouette_score is :", silhouette_avg)

sample_silhouette_values = silhouette_samples(X, cluster_labels)

y_lower = 10

for i in range(n_clusters):
    ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels == i]
    ith_cluster_silhouette_values.sort()
    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i
    color = cm.nipy_spectral(float(i)/n_clusters)
    ax1.fill_betweenx(np.arange(y_lower, y_upper)
                      ,ith_cluster_silhouette_values
                      ,facecolor=color
                      ,alpha=0.7
                     )
    ax1.text(-0.05
             , y_lower + 0.5 * size_cluster_i
             , str(i))
    y_lower = y_upper + 10

ax1.set_title("The silhouette plot for the various clusters.")
ax1.set_xlabel("The silhouette coefficient values")
ax1.set_ylabel("Cluster label")
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")
ax1.set_yticks([])
ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])
colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
ax2.scatter(X[:, 0], X[:, 1]
            ,marker='o' #点的形状
            ,s=8 #点的大小
            ,c=colors
           )
centers = clusterer.cluster_centers_
ax2.scatter(centers[:, 0], centers[:, 1], marker='x',
            c="red", alpha=1, s=200)
ax2.set_title("The visualization of the clustered data.")
ax2.set_xlabel("Feature space for the 1st feature")
ax2.set_ylabel("Feature space for the 2nd feature")
plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
              "with n_clusters = %d" % n_clusters),
             fontsize=14, fontweight='bold')
plt.show()
```

将上述过程包装成一个循环，可以得到：

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np

for n_clusters in [2,3,4,5,6,7]:
    n_clusters = n_clusters
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)
    ax1.set_xlim([-0.1, 1])

    ax1.set_ylim([0, X.shape[0] + (n_clusters + 1) * 10])
    clusterer = KMeans(n_clusters=n_clusters, random_state=10).fit(X)
    cluster_labels = clusterer.labels_

    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)

    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10

    for i in range(n_clusters):
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels == i]
        ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i
        color = cm.nipy_spectral(float(i)/n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper)
                          ,ith_cluster_silhouette_values
                          ,facecolor=color
                          ,alpha=0.7
                         )
        ax1.text(-0.05
                 , y_lower + 0.5 * size_cluster_i
                 , str(i))
        y_lower = y_upper + 10

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")
    ax1.set_yticks([])
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(X[:, 0], X[:, 1]
                ,marker='o' #点的形状
                ,s=8 #点的大小
                ,c=colors
               )
```

```
    centers = clusterer.cluster_centers_
    ax2.scatter(centers[:, 0], centers[:, 1], marker='x',
                c="red", alpha=1, s=200)
    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel("Feature space for the 1st feature")
    ax2.set_ylabel("Feature space for the 2nd feature")
    plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
                  "with n_clusters = %d" % n_clusters),
                 fontsize=14, fontweight='bold')
    plt.show()
```