Kmeans算法的K值最优解

非凡wang咖 2019-04-02

画

赵雷 - 赵小雷

上篇文章为大家介绍了我们常用的聚类算法Kmenas算法，也为大家整理了一点小案例，今天为大家继续分享我们Kmenas算法，对Kmenas算法来说，如何确定簇数K值是一个至关重要的问题，为了解决这个问题，通常会选用探索法，即给定不同的k值下，对比某些评估指标的变动情况，进而选择一个比较合理的k值，在这我们上篇文章给大家推荐了三种方法（簇内离差平方和拐点法，轮廓系数法和间隔统计量法），接下来我们分别看看这三种方法是如何实现的：

Kmenas算法基础公式：

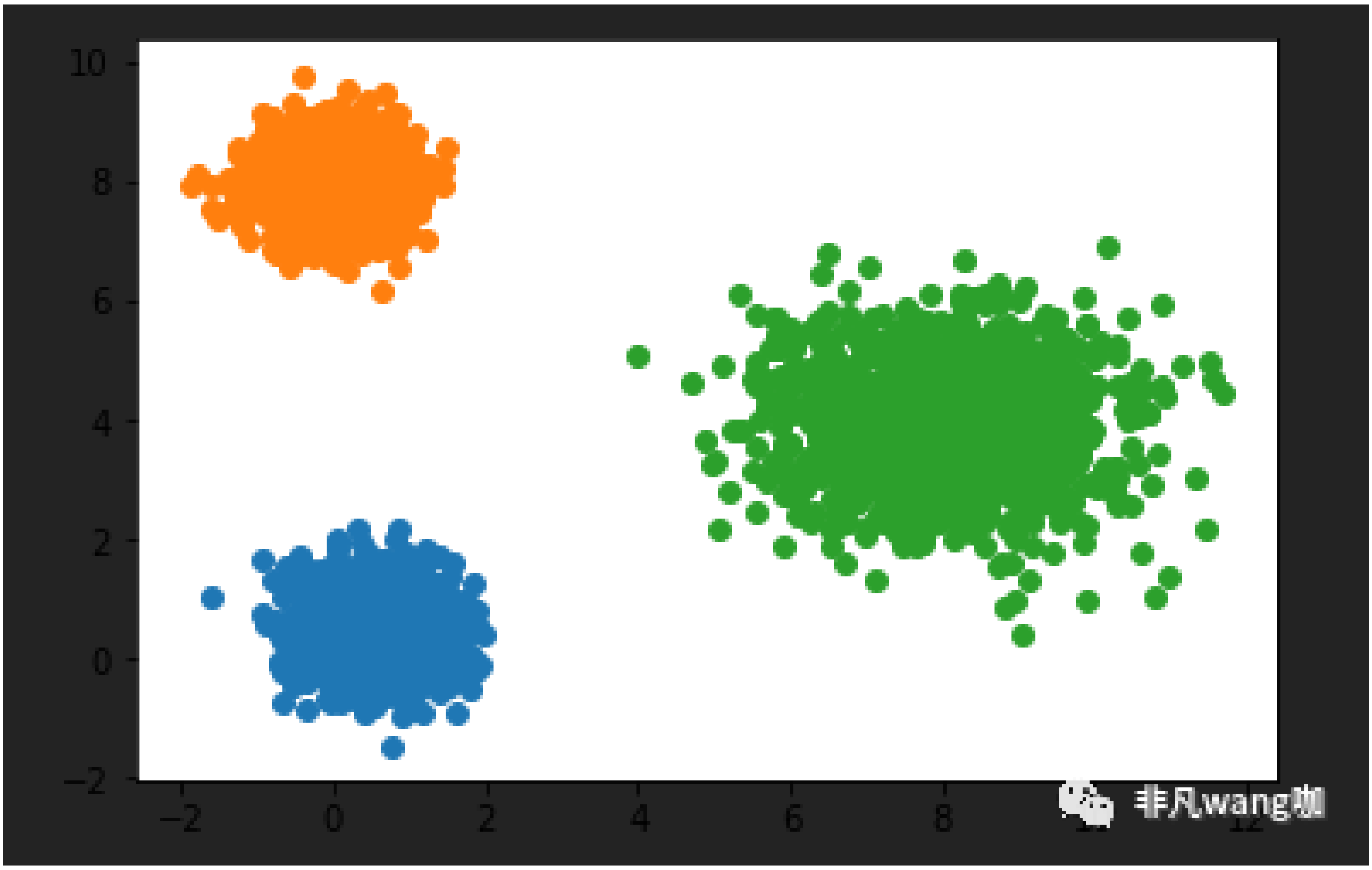
$$J=\sum_{i=1}^{k}\sum_{j\in c_k}(x^{(j)}-\mu_i)^2$$

拐点法

簇内离差平方和拐点法的思想很简单，就是在不同的k值下计算簇内离差平方和，然后通过可视化的方法找到“拐点”所对应的k值，J为Kmeans算法的目标函数，随着簇数量的增加，簇中的样本量会越来越少，进而导致目标函数J的值也会越来越小，通过可视化方法，重点关注的是斜率的变化，当斜率由大突然变小时，并且之后的斜率变化缓慢，则认为突然变化的点就是寻找的目标点，因为继续随着簇数K的增加，聚类效果不再有大的变化。

接下来我们就验证这个方法，随机生成三组二元正态分布数据，首先基于该数据绘制散点图，如下代码：

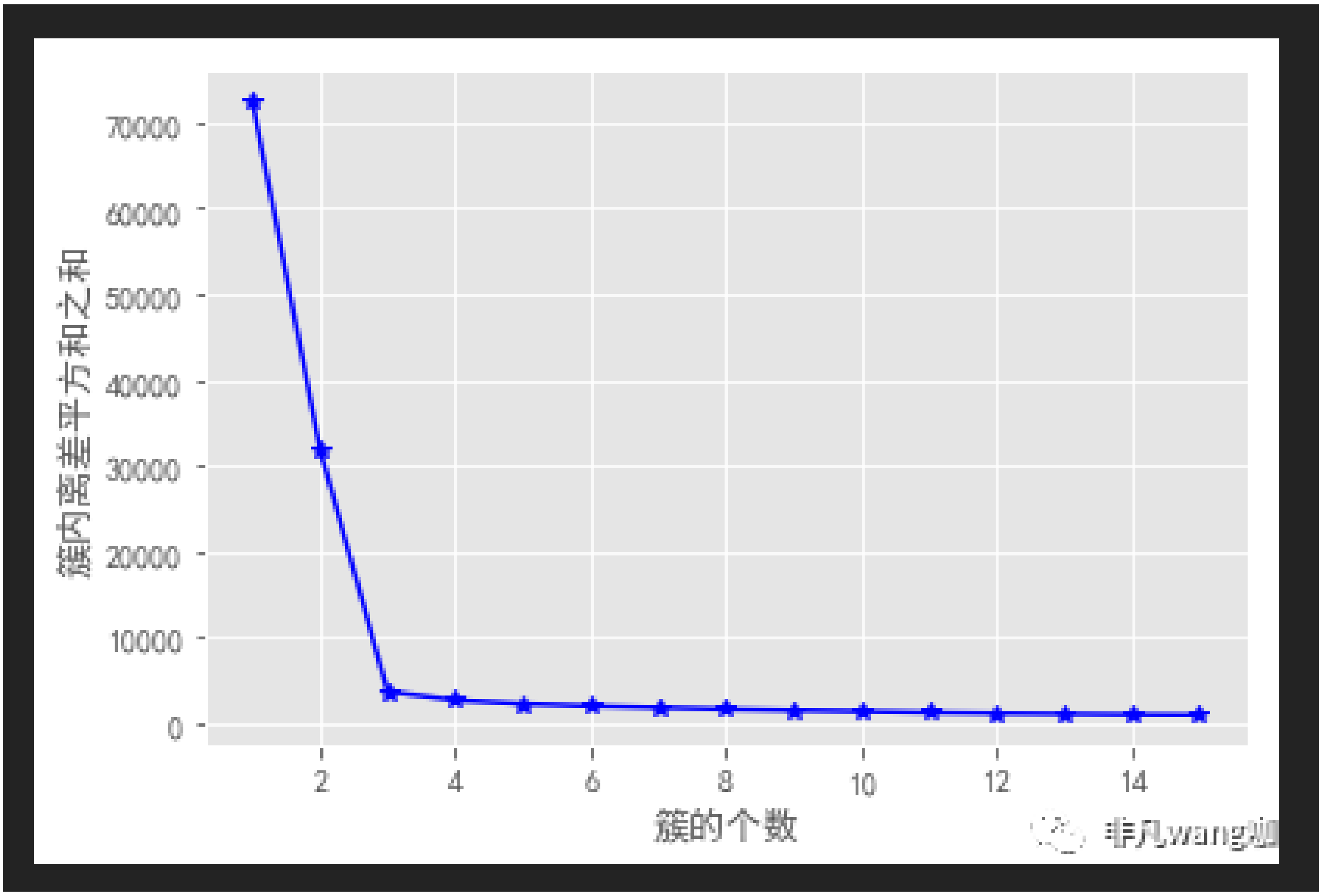
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

#随机生成三组二元正态分布随机数
np.random.seed(1234)
mean1 = [0.5,0.5]
cov1 = [[0.3,0],[0,0.3]]
x1,y1 = np.random.multivariate_normal(mean1,cov1,1000).T

mean2 = [0,8]
cov2 = [[0.3,0],[0,0.3]]
x2,y2 = np.random.multivariate_normal(mean2,cov2,1000).T

mean3 = [8,4]
cov3 = [[1.5,0],[0,1]]
x3,y3 = np.random.multivariate_normal(mean3,cov3,1000).T

#绘制三组数据的散点图
plt.scatter(x1,y1)
plt.scatter(x2,y2)
plt.scatter(x3,y3)
plt.show()
```



如上图，虚拟出来的数据呈现出三个簇，接下来基于这个虚拟数据，使用拐点法绘制簇的个数与总的簇内离差平方和之间的折线图，确定最终的k值，代码如下：

```
#构造自定义函数，用于绘制不同的k值和对应总的簇内离差平方和的折线图
def k_SSE(X,clusters):
    #选择连续的K种不同的值
    K= range(1,clusters+1)
    #构建空列表用于存储总的簇内离差平方和
    TSSE= []
    for k in K:
        #用于存储各个簇内离差平方和
        SSE = []
        kmeans = KMeans(n_clusters=k)
        kmeans.fit(X)
        #返回簇标签
        labels = kmeans.labels_
        #返回簇中心
        centers = kmeans.cluster_centers_
        #计算各簇样本的离差平方和，并保存到列表中
        for label in set(labels):
            SSE.append(np.sum((X.loc[labels==label,]-centers[label,:])**2))
        #计算总的簇内离差平方和
```

```
        TSSE.append(np.sum(SSE))
    #中文和负号正常显示
    plt.rcParams['font.sans-serif'] = 'SimHei'
    plt.rcParams['axes.unicode_minus']  =False
    #设置绘画风格
    plt.style.use('ggplot')
    #绘制K的个数与TSSE的关系
    plt.plot(K,TSSE,'b*-')
    plt.xlabel('簇的个数')
    plt.ylabel('簇内离差平方和之和')
    #
    plt.show()
#将三组数据集汇总到数据框中
X = pd.DataFrame(np.concatenate([np.array([x1,y1]),np.array([x2,y2]),np.array(
k_SSE(X,15)
```



如上图，当簇的个数为3时，形成了一个明显的“拐点”,因为K值从1到3时，折线的斜率都比较大，但是k值为4时斜率突然就降低了很多，并且之后的簇对应的斜率都变动很小，所以，合理的k值应该为3，与虚拟数据集的三个簇相吻合。

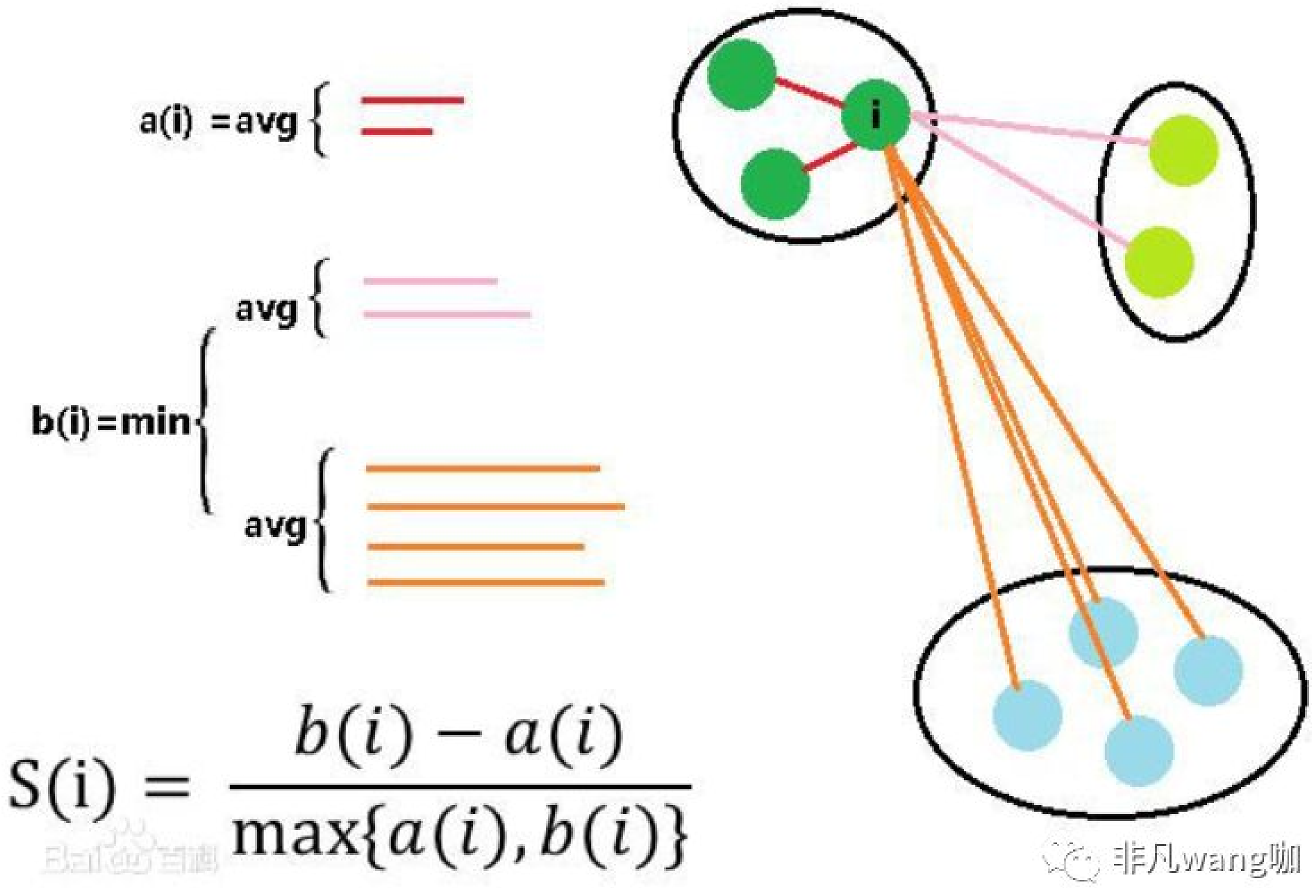
轮廓系数法

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

$$s(i)=\begin{cases}1-\frac{a(i)}{b(i)}, & a(i)<b(i)\\ 0, & a(i)=b(i)\\ \frac{b(i)}{a(i)}-1, & a(i)>b(i)\end{cases}$$

啥都不说先上公式，该方法综合考虑了簇的密集性和分散性俩个信息，如果数据集被分割为理想的K个簇，那么对应的簇内样本会很密集，而簇间样本会狠分散。上述公式中a(i)体现了簇内的秘籍性，代表样本i与同簇内其他样本点距离的平均值；b(i)反映了簇间的分散性，他的计算过程是样本i与其他非同簇样本点距离的平均值，然后从平均值中挑选出最小值。

通过公式可知当S(i)接近于-1时，说明样本i分配的不合理，需要将其分配到其他簇中；当S(i)近似为0时，说明样本i落在了模糊地带，即簇的边界处，当S(i)近似为1时，说明样本i的分配是合理的。

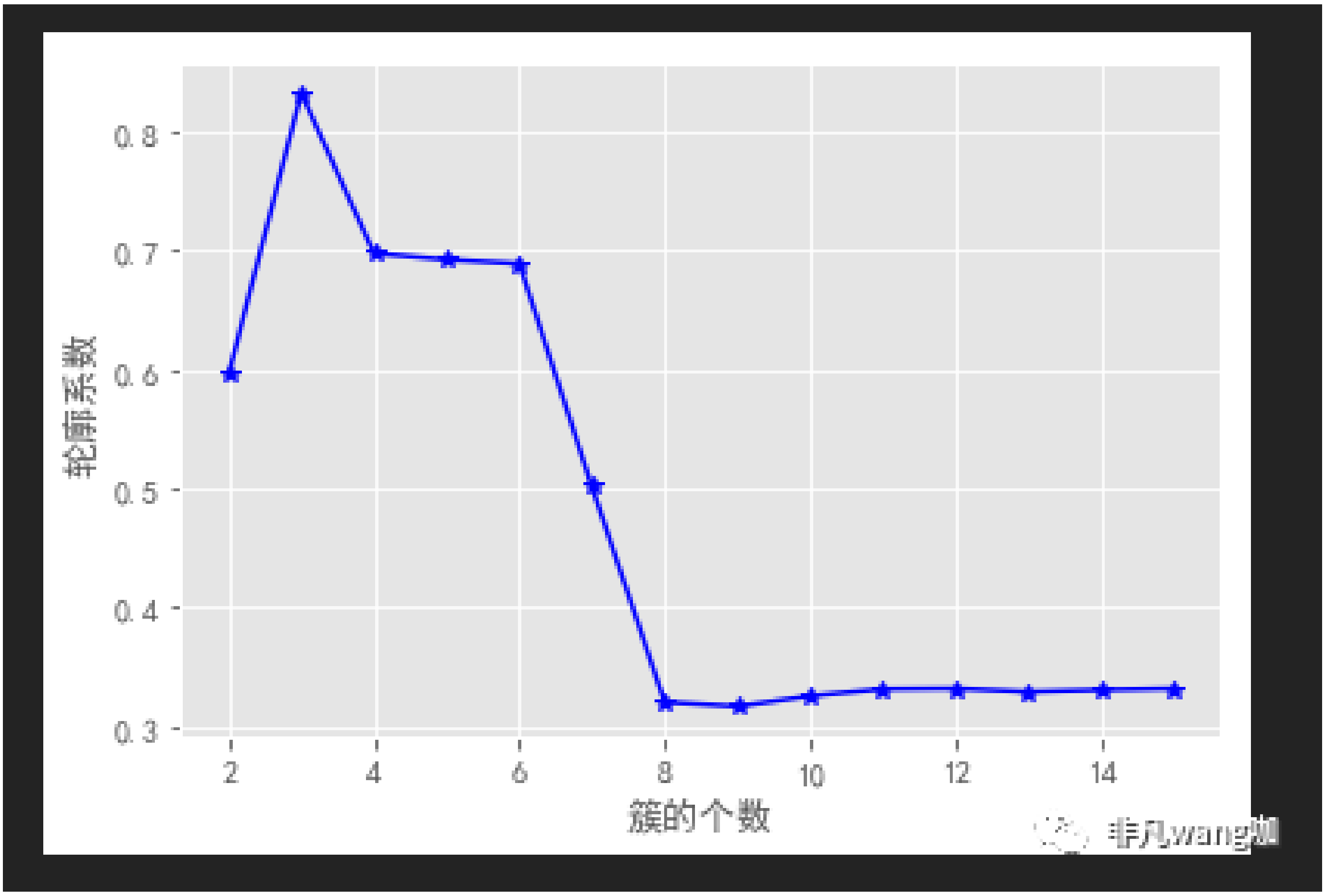


上图是百度百科对该方法的解释：https://baike.baidu.com/item/轮廓系数/17361607?fr=aladdin

接下来我们就看看如何用轮廓系数解决我们的k取值问题，由于轮廓系数计算较复杂，所以我们直接使用sklearn中的metrics中的silhouette_score方法，需要注意的是该方法需要接受的聚类簇数必须大于等于2.代码如下：

```
from sklearn import metrics
#构造自定义函数
def k_silhouette(X,clusters):
    K = range(2,clusters+1)
    #构建空列表，用于存储不同簇数下的轮廓系数
    S = []
    for k in K:
        kmeans = KMeans(n_clusters=k)
        kmeans.fit(X)
        labels = kmeans.labels_
        #调用子模块metrics中的silhouette_score函数，计算轮廓系数
        S.append(metrics.silhouette_score(X,labels,metric='euclidean'))
    #设置绘图风格
    plt.rcParams['font.sans-serif'] = 'SimHei'
```

```
    plt.rcParams['axes.unicode_minus']  =False
    #设置绘画风格
    plt.style.use('ggplot')
    #绘制K的个数与轮廓系数的关系
    plt.plot(K,S,'b*-')
    plt.xlabel('簇的个数')
    plt.ylabel('轮廓系数')
    plt.show()
k_silhouette(X,15)
```



如上图，利用之前构造的虚拟数据，绘制了不同K值下对应的轮廓系数图，当k取值为3时轮廓系数最大，且比较接近于1，说明应该把虚拟数据聚为3类比较合理。

间隔统计法

2000年Hastie等人提出了间隔统计量法（Gap Statistic方法），该方法可以适用与任何聚类算法，公式如下：

最简单的方法是使用类内样本点之间的欧式距离来表示，记为D_k，D_K越小聚类的紧支性越好。Ref

$$D_k = \sum_{x_i \in C_k} \sum_{x_j \in C_k} ||x_i - x_j||^2 = 2n_k \sum_{x_i \in C_k} ||x_i - \mu_k||^2$$

标准化后:

$$W_k = \sum_{k=1}^{K} \frac{1}{2n_k} D_k$$

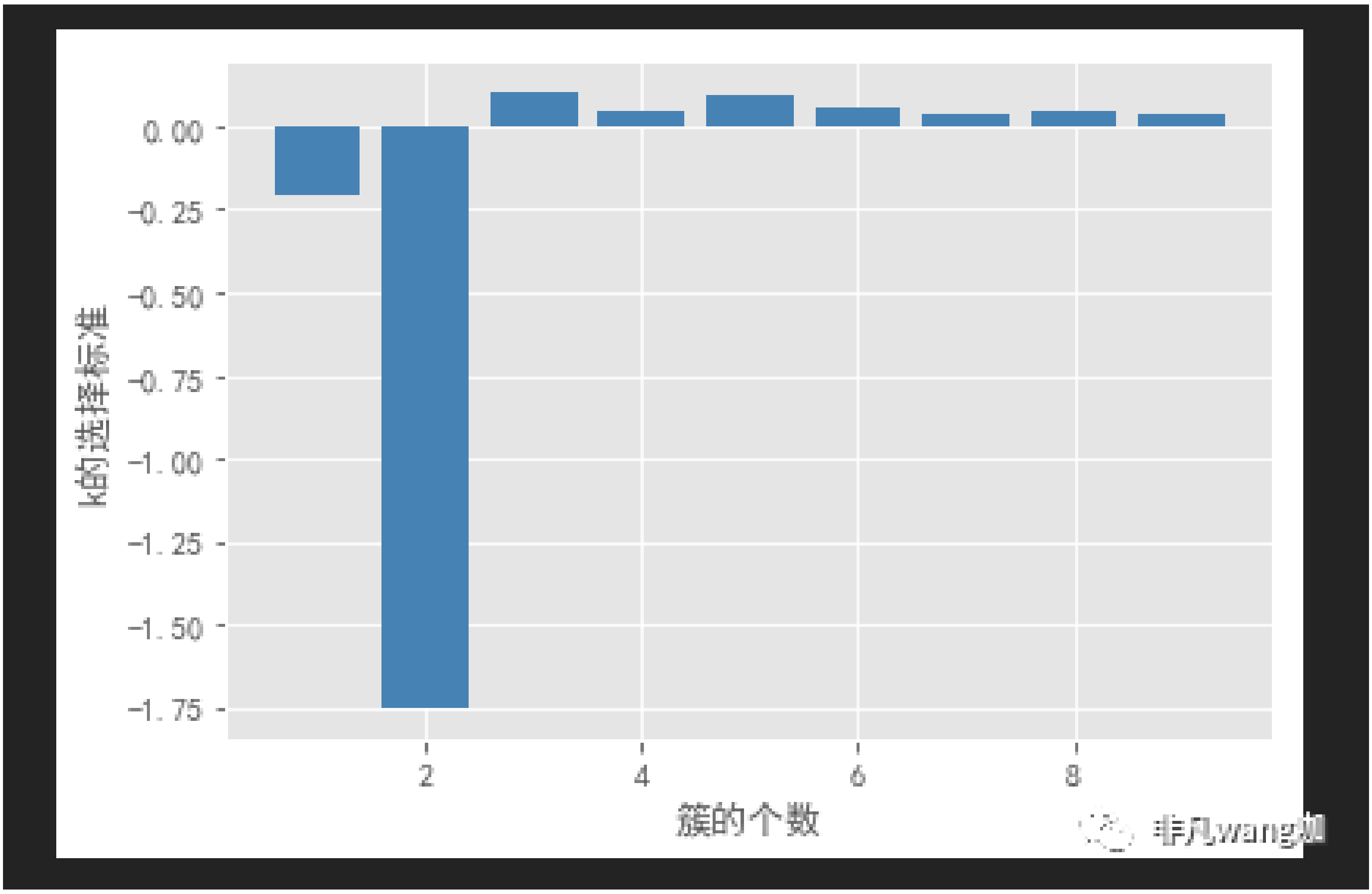
详情参考地址: https://blog.csdn.net/baidu_17640849/article/details/70769555

接下来我们构造自定义函数，绘制不同K值对应的间隙统计量折线图:

```
#自定义函数，计算簇内任意俩样本之间的欧式距离Dk
def short_pair_wise_D(each_cluster):
    mu = each_cluster.mean(axis=0)
    Dk = sum(sum((each_cluster-mu)**2*each_cluster.shape[0]))
    return Dk
#自定义函数，计算簇内的Wk值
def compute_Wk(data, classfication_result):
    Wk = 0
    label_set = set(classfication_result)
    for label in label_set:
        each_cluster = data[classfication_result == label, :]
        Wk = Wk + short_pair_wise_D(each_cluster)/(2.0*each_cluster.shape[0])
    return Wk

# 计算GAP统计量
def gap_statistic(X, B=10, K=range(1,11), N_init = 10):
    # 将输入数据集转换为数组
    X = np.array(X)
    # 生成B组参照数据
    shape = X.shape
    tops = X.max(axis=0)
    bots = X.min(axis=0)
    dists = np.matrix(np.diag(tops-bots))
    rands = np.random.random_sample(size=(B,shape[0],shape[1]))
```

```
    for i in range(B):
        rands[i,:,:] = rands[i,:,:]*dists+bots

    # 自定义0元素的数组，用于存储gaps、Wks和Wkbs
    gaps = np.zeros(len(K))
    Wks = np.zeros(len(K))
    Wkbs = np.zeros((len(K),B))
    # 循环不同的k值，
    for idxk, k in enumerate(K):
        k_means =  KMeans(n_clusters=k)
        k_means.fit(X)
        classfication_result = k_means.labels_
        # 将所有簇内的Wk存储起来
        Wks[idxk] = compute_Wk(X,classfication_result)

        # 通过循环，计算每一个参照数据集下的各簇Wk值
        for i in range(B):
            Xb = rands[i,:,:]
            k_means.fit(Xb)
            classfication_result_b = k_means.labels_
            Wkbs[idxk,i] = compute_Wk(Xb,classfication_result_b)

    # 计算gaps、sd_ks、sk和gapDiff
    gaps = (np.log(Wkbs)).mean(axis = 1) - np.log(Wks)
    sd_ks = np.std(np.log(Wkbs), axis=1)
    sk = sd_ks*np.sqrt(1+1.0/B)
    # 用于判别最佳k的标准，当gapDiff首次为正时，对应的k即为目标值
    gapDiff = gaps[:-1] - gaps[1:] + sk[1:]

    #设置绘图风格
    plt.rcParams['font.sans-serif'] = 'SimHei'
    plt.rcParams['axes.unicode_minus']  =False
    #设置绘画风格
    plt.style.use('ggplot')
    # 绘制gapDiff的条形图
    plt.bar(np.arange(len(gapDiff))+1, gapDiff, color = 'steelblue')
    plt.xlabel('簇的个数')
    plt.ylabel('k的选择标准')
    plt.show()
```

```
# 自定义函数的调用
gap_statistic(X)
```



如上图，x轴代表了不同的簇数k，y轴代表k值选择的判断指标gapDiff，gapDiff首次出现正值时对应的k为3，所以对于虚拟的数据集来说，将其划分为三个簇是比较合理的。

以上就是Kmeans算法中中簇数k的确定方法，嗯，是有点小小的难度😁