KMeans评估（如何选择最佳的k）

原本准备自己总结一篇关于KMeans评估的文章，发现网上这篇已经总结的很全面了，所以收藏下。如果有其他方法也欢迎留言交流。原文来自微信公众号：非凡wang咖

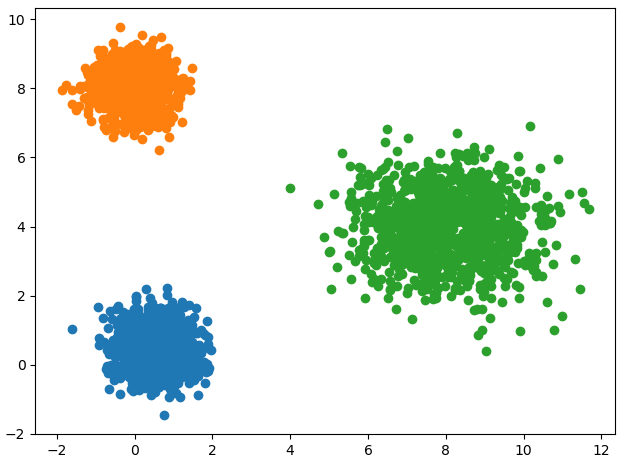
在Kmenas算法中，如何确定簇数K值是一个至关重要的问题，为了解决这个问题，通常会选用探索法，即给定不同的k值下，对比某些评估指标的变动情况，进而选择一个比较合理的k值。本篇文章总结三种常用的k值选择方法：簇内离差平方和拐点法，轮廓系数法和间隔统计量法）

# 拐点法

簇内离差平方和拐点法的思想是：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。随着簇数量的增加，簇中的样本量会越来越少，通过可视化方法，重点关注曲线斜率的变化，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拐点，也就是最佳的k值，因为继续随着簇数K的增加，聚类效果不再有大的变化。

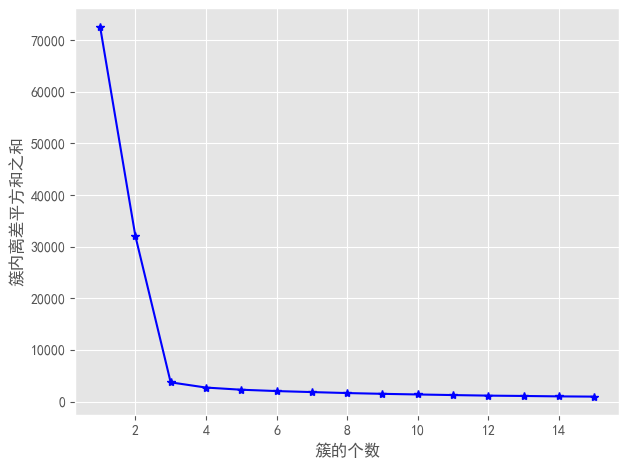
接下来我们验证这个方法，随机生成三组二元正态分布数据，首先基于该数据绘制散点图，如下代码：

|  |
| --- |
| # 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：

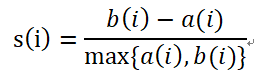
|  |
| --- |
| def k\_SSE(X, clusters):  """  拐点法  绘制不同的k值和对应总的簇内离差平方和的折线图  """  # 选择连续的K种不同的值  K = range(1, clusters+1)  # 构建空列表用于存储总的簇内离差平方和  TSSE = []  for k in K:  # 用于存储各个簇内离差平方和  SSE = []  kmeans = KMeans(n\_clusters=k)  kmeans.fit(X)  # 返回簇标签  labels = kmeans.labels\_  # 返回簇中心  centers = kmeans.cluster\_centers\_  # 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([x3, y3])], axis=1).T)  k\_SSE(X, 15) |

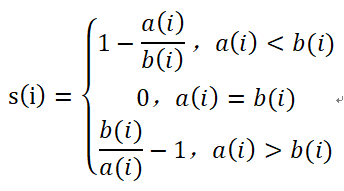


如上图，当簇的个数为3时，形成了一个明显的“拐点”，因为K值从1到3时，折线的斜率都比较大，但是k值为4时斜率突然就降低了很多，并且之后的簇对应的斜率都变动很小，所以，合理的k值应该为3，与虚拟数据集的三个簇相吻合。

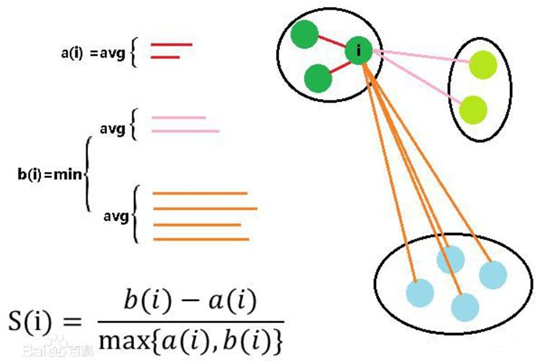
# 轮廓系数法

轮廓系数法综合考虑了簇的密集性和分散性，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，而簇间样本会很分散。其公式如下：





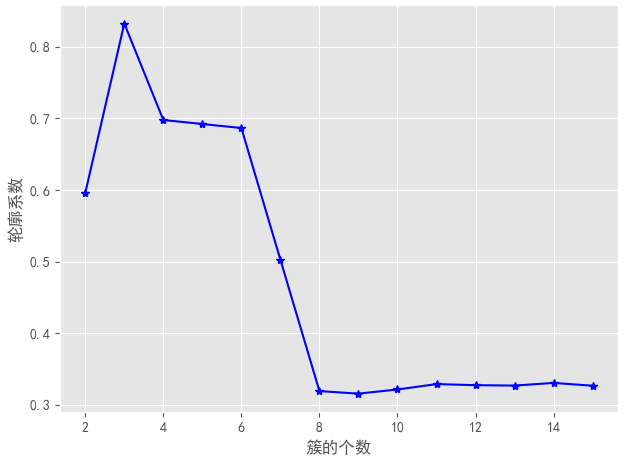
其中a(i) 是样本i与同簇内其他样本点距离的平均值，体现了簇内的密集性；b(i) 是样本i与其他非同簇样本点距离的平均值，然后从平均值中挑选出最小值，反映了簇间的分散性。



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。

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：

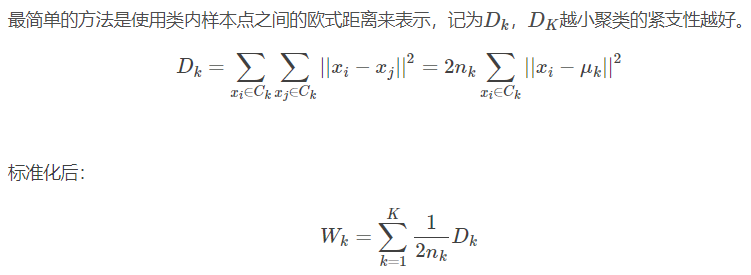
|  |
| --- |
| def k\_silhouette(X, clusters):  """  轮廓系数法  """  K = range(2, clusters+1)  #构建空列表，用于存储不同簇数下的轮廓系数  S = []  for k in K:  kmeans = KMeans(n\_clusters=k)  kmeans.fit(X)  labels = kmeans.labels\_  #调用子模块metrics中的silhouette\_score函数，计算轮廓系数  S.append(metrics.silhouette\_score(X, labels, metric='euclidean'))  #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。

# 间隔统计量法

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：

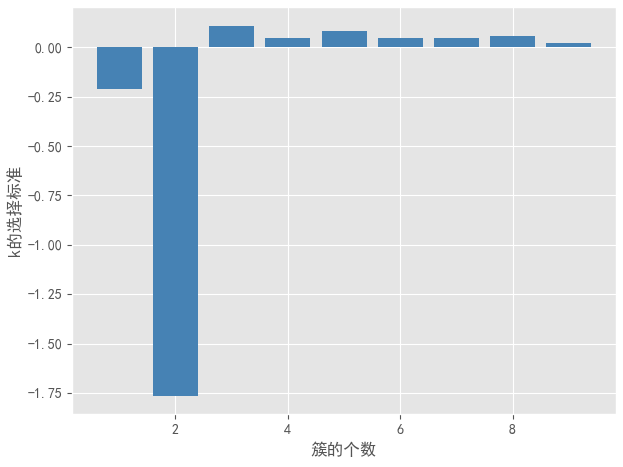


详情参考地址：

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：

|  |
| --- |
| def short\_pair\_wise\_D(each\_cluster):  """  计算簇内任意俩样本之间的欧式距离Dk  """  mu = each\_cluster.mean(axis=0)  Dk = sum(sum((each\_cluster - mu) \*\* 2 \* each\_cluster.shape[0]))  return Dk  def compute\_Wk(data, classfication\_result):  """  计算簇内的Wk值  """  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  def gap\_statistic(X, B=10, K=range(1, 11), N\_init=10):  """  间隔统计法  计算GAP统计量  """  # 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()  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。

完整代码链接：

<https://github.com/jpegbert/MachineLearning/tree/master/kmeans/kmeans_evaluation>

# 参考

<https://mp.weixin.qq.com/s/87YfGDWuhmwB7k4hbluIbA>

<https://mp.weixin.qq.com/s/SGtXTzQNwuKPp15Y1dL0Dg>

<https://mp.weixin.qq.com/s/d7M2FV3rjLdCxSZk-bC3Fg>

<https://blog.csdn.net/baidu_17640849/article/details/70769555>