# 《数据挖掘与统计决策课程》第一次作业（聚类）

Python软件实现K-Means聚类

姓名：邓睿 学号：202130320342 班别：21统计

## 一、作业题1

### 1.1数据处理与分析

首先，我们针对所收集的数据进行了详尽的预处理工作。在查阅相关资料后，我们了解到经营现金净流量这一变量无法通过表中其他数据直接计算得出。鉴于该变量的数据量过少，不足以支撑我们的建模及后续分析工作，因此我们决定将这一变量从数据集中剔除，建模和后续的所有分析过程均不再考虑这一因素。这一决策旨在确保我们分析的准确性和有效性，避免由于数据不足而导致的偏差。

接下来，我们对数据集中剩余的变量进行了全面审查。经过分析，我们发现表格中部分变量实际上是其他变量的计算结果。如果我们在建模和分析中将这些变量全部考虑进去，那么很可能导致信息冗余，即一些变量的信息实际上是重复的，没有为分析提供新的视角或价值。因此，我们决定对于这类可以由其他变量计算得出的变量，将不再纳入我们的考虑范围。这样做可以精简我们的数据集，去除冗余信息，使得我们的模型和分析更加聚焦于那些真正独立的、具有实际意义的变量。

此外，我们确保数据集中其他变量的完整性，不存在任何缺失情况。这意味着我们的数据集是完整的，没有因为数据缺失而影响分析的准确性。同时，我们也对数据进行了一致化和无量纲化处理，以确保不同变量之间的可比性，使得我们的分析更加客观和准确。

### 1.2数学建模过程论述

K-means数学模型：K-means聚类算法是一种常用的无监督学习方法，用于将数据集中的观测点划分为不同的群组或簇。其核心思想是通过迭代的方式，将数据点分配到距离其最近的簇中心，并更新簇中心的位置，直到达到某种收敛条件或预定的迭代次数。具体来说，K-means聚类算法的工作原理如下：

**Step 1 选择簇的数量（K）：**用户需要指定要将数据分成多少个簇。这通常是在算法应用之前由用户提供的一个参数。

**Step 2 初始化聚类中心：**随机选择K个数据点作为初始聚类中心（centroid）。这些中心点代表了每个簇的中心。

**Step 3 分配数据点到最近的聚类中心：**对于每个数据点，计算其与每个聚类中心的距离，并将其分配给距离最近的聚类中心所在的簇。

**Step 4 更新聚类中心：**对于每个簇，计算该簇中所有数据点的平均值，将这个平均值作为新的聚类中心。

**Step 5 重复步骤3和步骤4：**重复这个过程，直到聚类中心不再发生显著变化，或者达到预定的迭代次数。

K-means算法的目标是最小化簇内的平方误差，即最小化每个数据点到其簇内聚类中心的距离的平方和。因此，通常会多次运行算法，并选择效果最好的一组聚类结果。

### 1.3软件辅助过程

首先是数据准备，将excel数据用pandas导入python，并选定所要年份的数据，此处假定选择的年份为2000年。

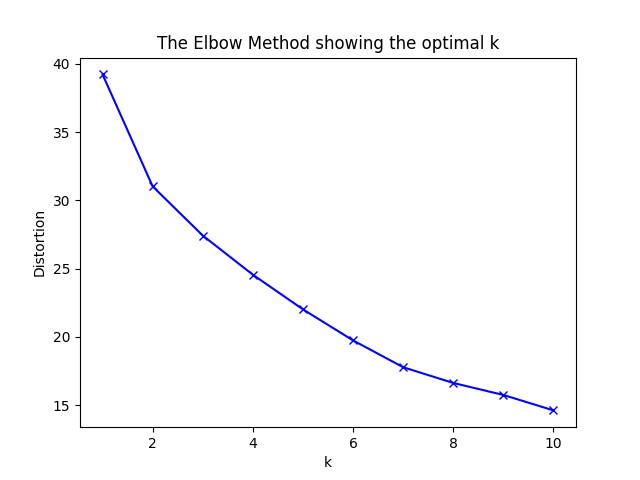
接着用手肘法确定最佳的k值。手肘法是一种在K-means聚类分析中确定最佳聚类数K的常用方法。它基于的原理是，随着K值的增大，每个簇的聚合程度会逐渐提高，导致误差平方和（SSE）逐渐减小。然而，当K值达到真实聚类数时，继续增加K值所带来的聚合程度提升会迅速减小，反映在曲线上就是一个明显的“肘部”。通过绘制SSE与K值的关系曲线，并观察曲线的形状，可以找到这个“肘部”点，从而确定数据的真实聚类数，即最佳的K值。所得到的SSE与K值关系曲线如图1所示。

图 1 SSE与K值关系曲线图

在利用手肘法确定K-means聚类分析的最佳K值时，通过观察SSE（误差平方和）与K值的关系曲线，我们可以发现曲线在K值逐渐增大的过程中存在转折点。在给出的图中，较明显的转折点出现在K=2和K=7时。这意味着当K值从1增加到2时，SSE的下降幅度较大，表示增加一个簇能显著提高聚类的效果；同样，当K值从某个较小值增加到7时，SSE也出现了一个显著的下降。为了进一步确定最佳K值，需要内部评价指标以及其他背景知识。一旦确定了最佳的K值，我们就可以将其带入sklearn库中的KMeans类进行聚类分析。KMeans类会根据指定的K值将数据划分为相应的簇，并输出聚类结果。

聚类结果如表1和表2所示。

表 1 K=2时的分类结果

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 类1 | 类2 | | | | | | | | |
| 1 | 4 | 22 | 38 | 52 | 67 | 82 | 96 | 110 | 124 |
| 2 | 7 | 23 | 39 | 53 | 68 | 83 | 97 | 111 | 125 |
| 3 | 10 | 25 | 40 | 54 | 69 | 84 | 98 | 112 | 126 |
| 5 | 11 | 26 | 41 | 55 | 71 | 85 | 99 | 113 | 128 |
| 6 | 12 | 27 | 42 | 56 | 72 | 86 | 100 | 114 | 129 |
| 8 | 13 | 28 | 43 | 57 | 73 | 87 | 101 | 115 | 130 |
| 9 | 14 | 29 | 44 | 58 | 74 | 88 | 102 | 116 | 131 |
| 24 | 15 | 30 | 45 | 59 | 75 | 89 | 103 | 117 | 132 |
| 34 | 16 | 31 | 46 | 60 | 76 | 90 | 104 | 118 | 133 |
| 64 | 17 | 32 | 47 | 61 | 77 | 91 | 105 | 119 |  |
| 68 | 18 | 33 | 48 | 62 | 78 | 92 | 106 | 120 |  |
| 70 | 19 | 35 | 49 | 63 | 79 | 93 | 107 | 121 |  |
| 80 | 20 | 36 | 50 | 65 | 81 | 94 | 108 | 122 |  |
| 127 | 21 | 37 | 51 | 66 | 82 | 95 | 109 | 123 |  |

注：表中数字为股票号

表 2 K=7时的分类结果

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 类1 | 类2 | 类3 | 类4 | 类5 | | 类6 | | | 类7 | | | | |
| 1 | 80 | 49 | 2 | 19 | 66 | 10 | 53 | 115 | 7 | 36 | 82 | 103 | 119 |
|  |  | 75 | 3 | 22 | 69 | 13 | 54 | 116 | 11 | 39 | 83 | 104 | 121 |
|  |  | 100 | 4 | 24 | 74 | 14 | 58 | 125 | 12 | 40 | 84 | 105 | 124 |
|  |  |  | 5 | 26 | 81 | 16 | 62 | 128 | 15 | 45 | 86 | 106 | 126 |
|  |  |  | 6 | 28 | 85 | 20 | 65 | 131 | 17 | 48 | 88 | 107 | 129 |
|  |  |  | 8 | 32 | 87 | 25 | 67 | 133 | 18 | 52 | 89 | 108 | 130 |
|  |  |  | 9 | 41 | 92 | 31 | 72 |  | 21 | 59 | 90 | 109 | 132 |
|  |  |  | 34 | 43 | 93 | 37 | 76 |  | 23 | 60 | 91 | 110 |  |
|  |  |  | 64 | 47 | 111 | 38 | 77 |  | 27 | 61 | 95 | 112 |  |
|  |  |  | 68 | 50 | 120 | 42 | 79 |  | 29 | 63 | 96 | 113 |  |
|  |  |  | 70 | 55 | 123 | 44 | 94 |  | 30 | 71 | 97 | 114 |  |
|  |  |  | 127 | 56 |  | 46 | 98 |  | 33 | 73 | 101 | 117 |  |
|  |  |  |  | 57 |  | 51 | 99 |  | 35 | 78 | 102 | 118 |  |

注：表中数字为股票号

### 1.4软件辅助结果分析

用轮廓系数、误差平方和、Calinski-Harabasz指数、Davies-Bouldin指数等对聚类结果进行评价。

首先K=2时，轮廓系数为0.4684，这个值虽然位于轮廓系数可能取值范围的中等偏下位置，但并不是特别高。轮廓系数是用来衡量聚类效果的指标，其值越高，表示聚类效果越好。因此，轮廓系数0.4684可能意味着聚类结果中存在一定的重叠或误分现象。具体来说，有些样本点可能被错误地划分到了其他簇中，或者某些簇的边界可能不够清晰，导致聚类效果不是非常理想。为了进一步提升聚类效果，可能需要调整聚类算法的参数或尝试其他聚类算法。SSE（误差平方和）为31.0196，这个值反映了样本点到其所属簇中心的总体距离。SSE越小，说明样本点与其所在簇的中心越近，聚类效果越好。然而，仅凭SSE的绝对值很难判断聚类效果的好坏，因为SSE的大小与数据集的特点、样本数量以及聚类数等因素都有关。因此，我们需要将当前的SSE值与不同聚类数下的SSE值进行比较，以判断当前聚类数下的SSE是否处于较低水平，从而评估聚类效果的好坏。Calinski-Harabasz指数为10.8443，这是一个相对较高的值。Calinski-Harabasz指数是通过计算簇内样本点的聚合度与簇间样本点的分离度来评估聚类效果的。指数值越高，说明簇内的样本点聚合得越好，且簇与簇之间分得越清晰。因此，Calinski-Harabasz指数10.8443可能表明当前的聚类效果相对较好，簇内的样本点较为紧密，且不同簇之间的分离度也较高。然而，Davies-Bouldin指数为1.8804，这个值相对较高。Davies-Bouldin指数是通过计算簇内样本点的平均距离与簇间样本点的平均距离之比来评估聚类效果的。指数值越低，说明聚类效果越好。因此，Davies-Bouldin指数1.8804可能意味着存在某些簇之间的分离度不够高，或者某些簇内部的样本点分布较为松散，从而影响了整体的聚类效果。为了提升聚类效果，可能需要进一步优化聚类算法或调整相关参数。

K=7时，首先，平均轮廓系数（Silhouette Score）为0.1590，这个值相对较低，表明聚类效果并不理想。轮廓系数衡量的是样本点与其所在簇内其他样本点的相似度与其邻近簇样本点的相似度的比较。值越低，意味着聚类结果中样本点的归属可能较为模糊，簇的边界不清晰，存在大量的重叠或误分现象。其次，SSE（误差平方和）值为17.7708，这个值反映了样本点到其所属簇中心的总体距离。然而，与之前的SSE值相比，这个值是否更优取决于数据集的特性和所使用的聚类算法。在没有比较不同聚类数下的SSE值的情况下，我们无法直接判断这个SSE值的好坏。再来看Calinski-Harabasz指数，其值为2.9346。与之前的值相比，这个值较低，说明簇内的聚合度和簇间的分离度可能并不理想。这个指数通常希望其值越高越好，因为它代表了簇的紧凑性和不同簇之间的区分度。最后，Davies-Bouldin指数为15.5028，这是一个非常高的值。Davies-Bouldin指数越小，表示聚类效果越好。如此高的指数值表明簇间的分离度很差，很可能存在多个簇之间的重叠，严重影响了聚类效果。综合以上分析，我们可以得出结论：当前的聚类效果非常不理想。轮廓系数低，表明簇的边界不清晰；Calinski-Harabasz指数低，说明簇内聚合度和簇间分离度不佳；而Davies-Bouldin指数高，进一步证明了簇间分离度的问题。

对比K=2和K=7的情形，可得聚类模型K=2时更好，但效果仍不是很佳。因此为得到更优的聚类结果，应该进一步对原始数据进行处理，可以通过降维的方式降低变量个数。

作业题1代码如下：

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.cluster import KMeans  from sklearn.metrics import silhouette\_score  import matplotlib.pyplot as plt  # 1. 数据准备  df = pd.read\_excel("C:\College\数学\大三下学习资料\数据挖掘\数据挖掘作业（第1次作业聚类分析）\数据挖掘作业（第1次作业聚类分析）\去除因子数据.xlsx")  # 假设df是包含所有股票数据的DataFrame，并且已经按年度筛选  # 假设我们只关注某一年的数据，例如2000年  df\_2000 = df[df['年份'] == 2000].drop('年份', axis=1)  # X是特征矩阵，即除去股票号之外的所有指标数据  X = df\_2000.drop('股票号', axis=1)  # 2. 确定合适的聚类数目K  # 初始化一个用于存储不同K值对应的SSE的列表  distortions = []  # 设定要测试的K值范围  K = range(1, 7) # 例如，测试从1到10个聚类  # 对于每个K值，计算SSE并存储  for k in K:  kmeans = KMeans(n\_clusters=k, random\_state=42)  kmeans.fit(X)  distortions.append(kmeans.inertia\_)  # 绘制手肘图  plt.plot(K, distortions, '-')  plt.xlabel('k')  plt.ylabel('Distortion')  plt.title('The Elbow Method showing the optimal k')  plt.show()  # 观察图形并选择SSE开始平缓下降的点作为K值  optimal\_k = 7 # 根据图形选择合适的K值  # 3. K-均值聚类  # 使用最佳的K值进行K-均值聚类  kmeans = KMeans(n\_clusters=optimal\_k, random\_state=42)  kmeans.fit(X)  # 获取每个股票的聚类标签  labels = kmeans.labels\_  # 将聚类标签添加到原始DataFrame中  df\_2000['Cluster'] = labels  print(df\_2000.columns)  # 4. 划分股票板块  # 打印每个聚类的股票列表  for i in range(optimal\_k):  cluster\_stocks = df\_2000[df\_2000['Cluster'] == i]['股票号'].tolist()  print(f"Cluster {i+1}: {cluster\_stocks}")  # 5. 评价各类的优劣性  # 计算轮廓系数  silhouette\_avg = silhouette\_score(X, labels)  print("The average silhouette\_score is :", silhouette\_avg)  # SSE即聚类目标函数值  sse\_final = kmeans.inertia\_  print("SSE (Clustering Objective Function):", sse\_final)  from sklearn.metrics import calinski\_harabasz\_score, davies\_bouldin\_score  from sklearn.metrics import calinski\_harabasz\_score  # 首先，我们需要将特征列和聚类标签分开  X\_features = df\_2000.drop('Cluster', axis=1) # 特征数据矩阵  clusters = df\_2000['Cluster'] # 聚类标签数组  # 计算Calinski-Harabasz Index  calinski\_harabasz = calinski\_harabasz\_score(X\_features, clusters)  print("Calinski-Harabasz Index: ", calinski\_harabasz)  # 计算Davies-Bouldin Index  davies\_bouldin = davies\_bouldin\_score(X\_features, clusters)  print("Davies-Bouldin Index:", davies\_bouldin) |

## 二、作业题2

### 2.1数据处理与分析

首先，我们使用Python的PIL库（Python Imaging Library，也被称为Pillow）来打开一幅RGB图像。RGB图像是一种常见的彩色图像格式，它使用红、绿、蓝三种颜色的不同组合来表示图像中的每个像素点。通过PIL库，我们可以方便地读取图像文件，并将其加载到程序中。图像加载后，我们将其转换为NumPy数组。NumPy是Python中用于处理大型多维数组和矩阵的库，它提供了大量的数学函数来操作这些数组。将图像转换为NumPy数组后，我们可以方便地对图像数据进行数值计算，如矩阵运算、统计分析等。

接下来，为了应用K-Means算法进行图像压缩，我们需要将三维的图像数组重塑为二维数组。原始的三维图像数组通常具有形状（高度，宽度，颜色通道），其中颜色通道对应RGB三个分量。通过重塑操作，我们将每个像素的三个颜色通道值组合成一行，从而得到一个二维数组，其中每一行代表一个像素，包含三个颜色通道的值。通过重塑图像数据为二维数组，我们可以将每个像素的颜色作为输入特征提供给K-Means算法。这样，算法就能够根据颜色相似性将像素划分为不同的聚类，并为每个聚类选择一个代表性的中心颜色。在后续的压缩过程中，我们将使用这些中心颜色来替换原始图像中的像素颜色，从而实现图像的压缩。

### 2.2软件辅助过程

由题目给出的指示，我们决定设置聚类的数量为16，这意味着我们希望将图像中的丰富颜色信息简化为16个主要的颜色类别。为了实现这一目标，我们选择了sklearn库中功能强大的KMeans类来执行K-Means聚类算法。KMeans类是sklearn提供的一个用于执行K-Means聚类的工具，它能够高效地处理大规模数据并给出准确的聚类结果。我们初始化KMeans对象，并设置n\_clusters参数为16，以指定我们想要的聚类数量。然后，我们调用fit方法将像素数据传递给KMeans对象进行聚类计算。KMeans算法会根据像素之间的颜色相似度将它们划分为16个不同的聚类，并为每个聚类计算出一个代表性的中心颜色。聚类完成后，我们获得了两个重要的输出：每个像素的聚类标签以及对应的聚类中心颜色。聚类标签是一个与像素数据形状相同的数组，它告诉我们每个像素属于哪个聚类。而聚类中心颜色则是一个形状为(16, 3)的数组，其中每一行代表一个聚类的中心颜色值，即该聚类中所有像素的平均颜色。

有了这些信息，我们就可以开始压缩图像了。我们遍历原始图像的每个像素，根据其聚类标签查找对应的聚类中心颜色，并用这个中心颜色替换像素的原始颜色。这个过程实际上是将图像中的每个像素映射到最近的聚类中心，从而实现颜色的量化和压缩。完成压缩后，为了评估压缩效果，我们使用matplotlib库来展示原始图像和压缩后的图像。matplotlib是一个强大的绘图库，它可以帮助我们方便地绘制和比较图像。通过imshow函数，我们可以将原始图像和压缩后的图像加载到matplotlib中，并以可视化的方式展示出来。这样，我们就可以直观地看到压缩对图像质量的影响，比如颜色数量的减少以及可能出现的失真情况。

最后将压缩后的图片以jpg的格式储存。为了初步观察图片压缩的效果，将原图片和压缩后的图片同时输出，如图2所示。从图中可以看出，压缩后的图片虽然颜色与原图相比颜色边界没有那么分明，且颜色没有原来那么丰富，但大体上看与原图较接近。

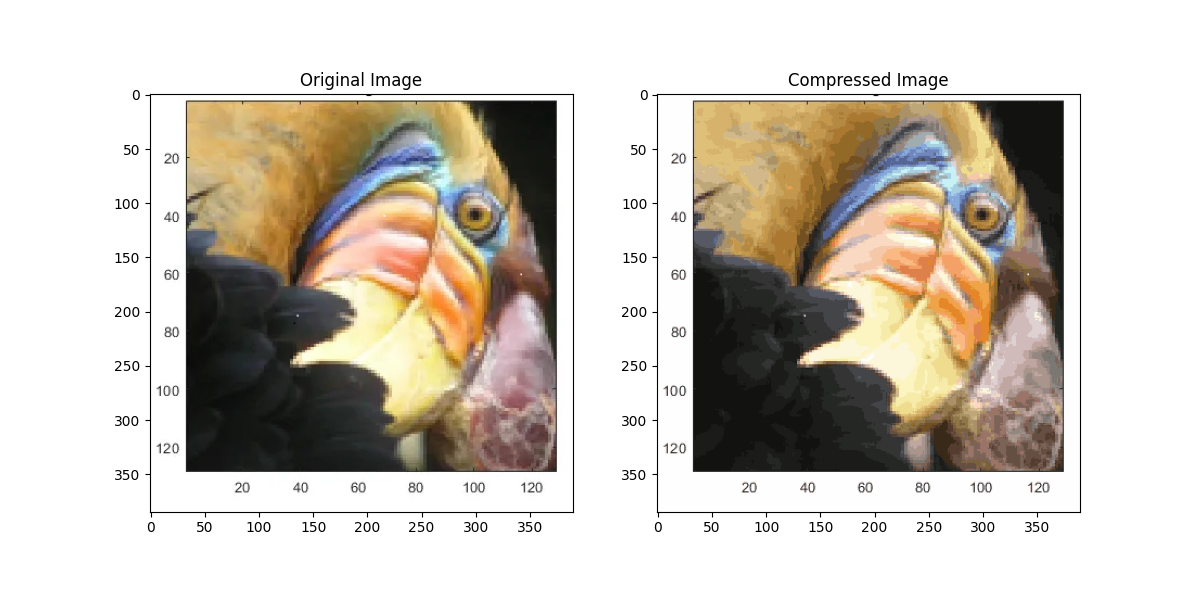
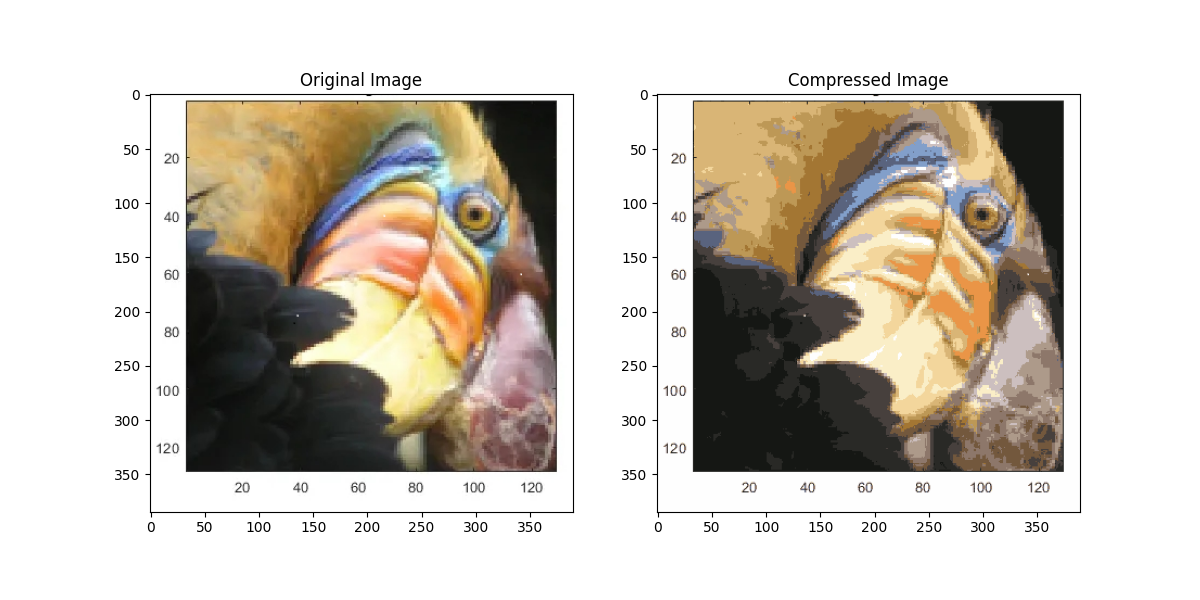
为了看K值对压缩图片的影响，取K=48同上述过程进行图片压缩，得到对比图如图3。

图 3 K=48时压缩图片对比图

图 2 K=16时压缩图片对比图

对比图2和图3可以看到K值越大压缩后的图片与原图片越接近。

### 2.3软件辅助结果分析

为进一步评价图片压缩的效果，需要引入一些指标。运用压缩比率、均方误差、结构相似性指数、峰值信噪比来评价图片压缩的优劣。用python求出各个指数并进行以下分析。

K=16时，压缩比率为7.26意味着原始图像的大小是压缩后图像大小的7.26倍，这通常意味着较好的压缩效果。然而，压缩效果的好坏还需要结合其他指标来综合评估。均方误差（MSE）为48.48，这个值越低，说明压缩后的图像与原始图像之间的差异越小。尽管MSE的值本身并不能直接反映图像质量的优劣，但它是衡量图像差异的一个常用指标。结构相似性指数（SSIM）为0.8916，这个值接近1，说明压缩后的图像与原始图像在结构上保持了较高的相似性。SSIM是一个较为全面的图像质量评估指标，它考虑了图像的亮度、对比度和结构信息，因此具有较高的参考价值。峰值信噪比（PSNR）为31.28 dB，这个值通常越高，说明压缩后的图像质量越好。一般来说，PSNR值在30 dB以上通常被认为是较好的图像质量。综合考虑这些指标，可以认为该图像的压缩效果相对较好。压缩比率较高，同时MSE较低，SSIM接近1，PSNR值也较高，这些都表明压缩后的图像在保持较高质量的同时，有效地减小了文件大小。

K=48时，压缩比率为7.40，这意味着原始图像的大小是压缩后图像大小的7.4倍，这是一个相对较高的压缩比率，表明压缩算法在减小文件大小方面表现得很好。均方误差（MSE）为31.99，与之前的48.48相比有所降低。MSE值越低，说明压缩后的图像与原始图像之间的差异越小，因此这是一个积极的迹象。结构相似性指数（SSIM）为0.9376，这个值比之前的0.8916要高。SSIM值越接近1，表示压缩后的图像在结构上与原始图像越相似。0.9376是一个相当高的值，说明压缩后的图像在结构上保持了很好的相似性。峰值信噪比（PSNR）为33.08 dB，比之前的31.28 dB有所提高。PSNR值越高，通常意味着图像质量越好。33.08 dB的PSNR值表明压缩后的图像在质量上表现良好。综合这些指标，我们可以得出结论，这次压缩的效果相当不错。高压缩比率有效地减小了文件大小，同时MSE的降低、SSIM的提高和PSNR的增加都表明压缩后的图像在质量上与原始图像非常接近。

虽然K=48各个指标上都优于K=16，但K=48时程序运行时间远大于K=16，故在实际应用中应该平衡图片压缩的效果和程序运行的时间。

K—Means在用于图片压缩的优缺点如下：

优点：

1、颜色量化与压缩：使用K-Means算法进行图像压缩，实质上是对图像中的颜色进行量化处理。通过将颜色空间划分为固定数量的聚类，算法能够有效减少图像中颜色的数量，从而实现图像数据的压缩。这种量化方法通常能够保留图像的主要视觉特征，同时大幅度减少存储需求。

2、计算效率：K-Means算法是一种迭代求解的聚类方法，其计算效率相对较高。尤其在处理大规模数据集时，通过优化算法实现和选择合适的聚类数量，可以在保证压缩效果的同时保持较高的处理速度。

3、灵活性：K-Means算法允许用户通过调整聚类数量来控制压缩程度。聚类数量越多，压缩后的图像保留的原始信息就越多，但压缩率也相应降低；反之，聚类数量越少，压缩率越高，但可能损失更多的图像细节。这种灵活性使得算法能够适应不同的应用场景和需求。

4、易于实现与集成：K-Means算法是机器学习领域中的经典算法，已经有成熟的实现库可供使用（如sklearn）。这使得算法易于实现和集成到现有的图像处理流程中。

局限性：

1、颜色失真：由于K-Means算法通过聚类中心颜色替换原始像素颜色来实现压缩，这可能导致图像中出现颜色失真现象。特别是当聚类数量较少时，这种失真可能更加明显，影响图像的视觉质量。

2、对噪声敏感：K-Means算法对图像中的噪声较为敏感。如果图像中存在大量的噪声点，这些点可能会被错误地划分到某些聚类中，从而影响聚类中心颜色的选择，导致压缩后的图像质量下降。

3、局部最优问题：K-Means算法是一种迭代求解的方法，其结果可能受到初始聚类中心选择的影响。如果初始聚类中心选择不当，可能导致算法陷入局部最优解，从而影响压缩效果。

4、计算资源需求：虽然K-Means算法相对高效，但在处理大规模图像或高分辨率图像时，仍然需要较大的计算资源和存储空间。这可能会限制算法在一些资源受限场景下的应用。

作业2代码如下：

|  |
| --- |
| import numpy as np  from sklearn.cluster import KMeans  from PIL import Image  import os  from skimage.metrics import structural\_similarity as ssim  # 已经有了一个RGB图片，将其转换为numpy数组  img = Image.open(r'C:\College\数学\大三下学习资料\数据挖掘\数据挖掘作业（第1次作业聚类分析）\数据挖掘作业（第1次作业聚类分析）\图片1.png').convert('RGB')  img\_array = np.array(img)  # 获取图片的高度和宽度  m, n, \_ = img\_array.shape # 这里的\_用于接收通道数，因为我们知道它是3（RGB）  # 第一步：将三维数据转化为二维数据  pixels = img\_array.reshape(-1, 3)  # 第二步：聚类  num\_clusters = 48 # 设置聚类数为16  kmeans = KMeans(n\_clusters=num\_clusters, random\_state=0).fit(pixels)  labels = kmeans.labels\_ # 每个像素点的聚类下标  cluster\_centers = kmeans.cluster\_centers\_ # 聚类中心颜色  # 第三步：我们已经有了特征值和聚类下标，不需要额外操作  # 第四步：使用for循环遍历每个像素，并将其颜色替换为对应的聚类中心颜色  compressed\_img\_array = np.zeros\_like(img\_array) # 创建一个与原图同样大小的零数组用于存放压缩后的颜色  for i in range(m):  for j in range(n):  index = i \* n + j # 计算当前像素在二维数组中的索引  label = labels[index] # 获取当前像素的聚类标签  compressed\_img\_array[i, j] = cluster\_centers[label] # 设置压缩后的颜色  # 显示原始图片和压缩后的图片  from matplotlib import pyplot as plt  fig, axs = plt.subplots(1, 2, figsize=(12, 6))  axs[0].imshow(img\_array)  axs[0].set\_title('Original Image')  axs[1].imshow(compressed\_img\_array.astype(np.uint8))  axs[1].set\_title('Compressed Image')  plt.show()  # 保存压缩后的图片  compressed\_img = Image.fromarray(compressed\_img\_array.astype(np.uint8), 'RGB')  compressed\_img.save('compressed\_image.jpg')  # 原始图像和压缩后的图像文件路径分别为  original\_image\_path = "C:\College\数学\大三下学习资料\数据挖掘\数据挖掘作业（第1次作业聚类分析）\数据挖掘作业（第1次作业聚类分析）\图片1.png"  compressed\_image\_path = "C:\College\数学\大三下学习资料\数据挖掘\数据挖掘作业（第1次作业聚类分析）\数据挖掘作业（第1次作业聚类分析）\compressed\_image.jpg"  # 获取文件大小  original\_size = os.path.getsize(original\_image\_path)  compressed\_size = os.path.getsize(compressed\_image\_path)  # 计算压缩比  compression\_ratio = original\_size / compressed\_size  print(f"Compression Ratio: {compression\_ratio:.2f}")  # 计算MSE  # 读取图像  with Image.open(original\_image\_path) as original\_img:  original\_image = np.array(original\_img)  with Image.open(compressed\_image\_path) as compressed\_img:  compressed\_image = np.array(compressed\_img)  # 确保图像具有相同的维度  if original\_image.shape != compressed\_image.shape:  raise ValueError("Images must have the same shape")  # 为了计算MSE，我们需要将它们转换为一维  if len(original\_image.shape) == 3:  original\_image = original\_image.reshape(-1)  compressed\_image = compressed\_image.reshape(-1)  # 计算MSE  mse = np.mean((original\_image - compressed\_image) \*\* 2)  print(f"Mean Squared Error (MSE): {mse:.2f}")  # 计算SSIM  ssim\_value = ssim(original\_image, compressed\_image, multichannel=True)  print(f"Structural Similarity Index (SSIM): {ssim\_value:.4f}")  def calculate\_psnr(img1, img2):  mse = np.mean((img1 - img2) \*\* 2)  if mse == 0:  return 100  PIXEL\_MAX = 255.0  return 20 \* np.log10(PIXEL\_MAX / np.sqrt(mse))  # 计算PSNR  psnr = calculate\_psnr(original\_image, compressed\_image)  print(f"Peak Signal-to-Noise Ratio (PSNR): {psnr:.2f} dB") |