**《机器学习基础》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2023级计算机科学与技术卓越01班** | | | **姓名** | **Xx** |
| **实验题目** | **聚类算法实践** | | | | | |
| **实验时间** | **2025年5月16日** | | **实验地点** | **DS3304** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 ☑设计性 □综合性** | | |
| 教师评价：  **□**算法/实验过程正确； **□**源程序/实验内容提交 **□**程序结构/实验步骤合理；  **□**实验结果正确； **□**语法、语义正确； **□**报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的  掌握聚类算法原理。 | | | | | | |
| 二、实验项目内容  1.理解并**描述**各一种原型聚类算法、密度聚类算法的原理。  2.**编程**实践，将k均值算法应用于合适的数据集（西瓜数据集、鸢尾花数据集或其它合适数据集），设置三组以上不同的k值，分别使用三组不同初始中心点，对比实验结果，分析聚类结果的优劣。 | | | | | | |
| 三、实验过程或算法（源程序）  原型聚类算法：  原型聚类算法假设聚类结构能通过一组原型刻画。而“原型”则指的是样本空间中具有代表性的点。通常情形下, 算法先对原型进行初始化，然后对原型进行迭代更新求解。  原型聚类算法中的代表有k-mean算法，其优化目标如下：    直观来说就是优化簇内的紧密程度。并且想要获得这个优化的精确解是一个NP难的问题，于是实际的k-mean算法使用了贪心策略进行简化：首先随机选取k个簇中心向量，然后对每一个数据点计算其与这k个向量之间的距离，然后选择最小的中心向量，并合并入簇。当所有数据点都经过一轮合并之后，重新计算簇中心向量，然后再次重复上面的步骤，直到簇中心向量不再变动或者达到最大的迭代轮次。  密度聚类算法：  密度聚类亦称“基于密度的聚类"(density-based clustering),此类算法假设聚类结构能通过样本分布的紧密程度确定。通常情形下，密度聚类算法从样本密度的角度来考察样本之间的可连接性，并基于可连接样本不断扩展聚类簇以获得最终的聚类结果。  DBSCAN是一种著名的密度聚类算法，它使用一组“领域”参数来刻画样本之间的紧密程度。 该算法将“簇”定义为：由密度可达关系导出的最大密度相连的样本集合。不属于任何簇的样本点会被判定为噪声或异常数据点。  具体的算法流程如下：  首先，算法需要预先设定两个关键参数：邻域半径（eps）和最小样本数（MinPts）。对于数据集中的每一个样本点，算法会计算其eps邻域内的样本数量，若该数量大于或等于MinPts，则该点被标记为核心点，否则暂时标记为噪声点。接着，从任意一个未访问的核心点出发，通过密度可达性搜索所有与其密度相连的样本点，形成一个簇。这一过程通过递归或迭代的方式扩展，直到所有密度可达的点都被包含到当前簇中。对于非核心点，若其位于某个核心点的eps邻域内，则被划入该核心点对应的簇，否则仍视为噪声。算法重复上述步骤，直到所有核心点均被访问且归类完毕，最终输出一组密度簇及噪声点。  k-mean算法实现：  import numpy as np  class KMeans:      def \_\_init\_\_(self, n\_clusters=3, max\_iters=300):          """          初始化KMeans类            参数:              n\_clusters: 聚类数量              max\_iters: 最大迭代次数          """          self.n\_clusters = n\_clusters          self.max\_iters = max\_iters          self.centroids = None          self.labels = None        def fit(self, X):          """          训练KMeans模型            参数:              X: 训练数据, shape=(n\_samples, n\_features)          """          # 从数据点中随机选择k个作为初始聚类中心          n\_samples, n\_features = X.shape          idx = np.random.choice(n\_samples, self.n\_clusters, replace=False)          self.centroids = X[idx]            for \_ in range(self.max\_iters):              # 保存旧的聚类中心              old\_centroids = self.centroids.copy()                # 计算每个样本到各个聚类中心的距离              distances = np.sqrt(((X - self.centroids[:, np.newaxis])\*\*2).sum(axis=2))                # 将每个样本分配给最近的聚类中心              self.labels = np.argmin(distances, axis=0)                # 更新聚类中心              for i in range(self.n\_clusters):                  cluster\_points = X[self.labels == i]                  if len(cluster\_points) > 0:                      self.centroids[i] = cluster\_points.mean(axis=0)                # 检查是否收敛              if np.all(old\_centroids == self.centroids):                  break            return self        def predict(self, X):          """          对新数据进行预测            参数:              X: 需要预测的数据, shape=(n\_samples, n\_features)          返回:              预测的聚类标签          """          distances = np.sqrt(((X - self.centroids[:, np.newaxis])\*\*2).sum(axis=2))          return np.argmin(distances, axis=0)  在鸢尾花数据集上的聚类结果：  import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import silhouette\_score  from k\_mean import KMeans  import matplotlib.pyplot as plt  # 读取鸢尾花数据集  f = pd.read\_csv(r'exp4/data/iris.csv')  X = f.iloc[:,1:5].values                             # 属性值,numpy数组（跳过Unnamed: 0列）  y = f.iloc[:,5].values                               # 标签值,numpy数组（Species列）  feature\_names = f.columns.tolist()[1:5]              # 特征的列表（跳过Unnamed: 0列）  def run\_kmeans(X, k, random\_seed):      """运行KMeans算法并返回结果"""      np.random.seed(random\_seed)      kmeans = KMeans(n\_clusters=k)      kmeans.fit(X)      labels = kmeans.predict(X)      score = silhouette\_score(X, labels)      return kmeans, labels, score  def analyze\_cluster\_sizes(labels, k):      """分析每个簇的大小"""      sizes = []      for i in range(k):          size = np.sum(labels == i)          sizes.append(size)      return sizes  # 设置不同的k值和随机种子  k\_values = [2, 3, 4]  random\_seeds = [42, 123, 256]  # 存储所有结果  results = []  # 运行不同参数组合的实验  print("\n=== 聚类实验结果分析 ===")  for k in k\_values:      print(f"\nk = {k} 的实验结果：")      print("-" \* 40)      for seed in random\_seeds:          kmeans, labels, score = run\_kmeans(X, k, seed)          cluster\_sizes = analyze\_cluster\_sizes(labels, k)          results.append({              'k': k,              'seed': seed,              'kmeans': kmeans,              'labels': labels,              'score': score,              'cluster\_sizes': cluster\_sizes          })          print(f"随机种子 = {seed}:")          print(f"- 轮廓系数: {score:.4f}")          print(f"- 各簇大小: {cluster\_sizes}")  # 找出每个k值的最佳结果  print("\n=== 最佳结果分析 ===")  best\_results = {}  for k in k\_values:      k\_results = [r for r in results if r['k'] == k]      best\_k\_result = max(k\_results, key=lambda x: x['score'])      best\_results[k] = best\_k\_result      print(f"\nk = {k} 的最佳结果：")      print("-" \* 40)      print(f"- 随机种子: {best\_k\_result['seed']}")      print(f"- 轮廓系数: {best\_k\_result['score']:.4f}")      print(f"- 各簇大小: {best\_k\_result['cluster\_sizes']}")  # 让图像显示中文  plt.rcParams['font.sans-serif'] = ['SimHei']  plt.rcParams['axes.unicode\_minus'] = False  # 选择两个最具代表性的特征  feature\_pair = (0, 1)  # sepal length 和 sepal width  # 创建一个大图，展示所有k值的结果  plt.figure(figsize=(15, 5))  for idx, k in enumerate(k\_values, 1):      plt.subplot(1, 3, idx)        result = best\_results[k]        # 绘制数据点      scatter = plt.scatter(X[:, feature\_pair[0]], X[:, feature\_pair[1]],                           c=result['labels'], cmap='viridis', alpha=0.6)        # 绘制聚类中心      plt.scatter(result['kmeans'].centroids[:, feature\_pair[0]],                  result['kmeans'].centroids[:, feature\_pair[1]],                  marker='\*', s=100, c='red', label='聚类中心')        plt.xlabel(feature\_names[feature\_pair[0]])      plt.ylabel(feature\_names[feature\_pair[1]])      plt.title(f'k={k}, 轮廓系数={result["score"]:.4f}\n簇大小={", ".join(str(size) for size in result["cluster\_sizes"])}')      plt.legend()        # 添加颜色条      plt.colorbar(scatter)  plt.suptitle('不同k值的聚类结果比较', fontsize=12)  plt.tight\_layout()  plt.show()  # 绘制轮廓系数随机种子的变化  plt.figure(figsize=(10, 6))  for k in k\_values:      k\_results = [r for r in results if r['k'] == k]      scores = [r['score'] for r in k\_results]      plt.plot(random\_seeds, scores, 'o-', label=f'k={k}')  plt.xlabel('随机种子')  plt.ylabel('轮廓系数')  plt.title('不同k值和随机种子的轮廓系数比较')  plt.legend()  plt.grid(True)  plt.show() | | | | | | |
| 四、实验结果及分析  聚类结果可视化：    随机种子对聚类结果的影响：    基于实验结果得出如下分析：  1. k=2的聚类结果：  - 最高轮廓系数：0.6810  - 簇的大小分布：[97, 53]  - 所有随机种子得到相同的结果，说明这是一个非常稳定的聚类  - 簇的大小分布显示数据大致按照2:1的比例被分为两类  2. k=3的聚类结果：  - 轮廓系数：0.5528  - 簇的大小分布：[62, 50, 38]  - 同样具有很好的稳定性，不同随机种子得到相同结果  - 三个簇的大小相对均衡，接近实际的鸢尾花三个品种的分布  3. k=4的聚类结果：  - 最好的轮廓系数：0.4954（种子42和123）  - 最好结果的簇大小分布：[47, 50, 23, 30]  - 随机种子256得到了不同的结果，簇大小分布为[22, 38, 62, 28]，轮廓系数降至0.4152  - 显示出明显的不稳定性，对初始中心点的选择比较敏感  综合分析：  1. 聚类效果随k值增加而下降，说明数据本身可能更适合较少的簇数  2. k=2时虽然轮廓系数最高，但k=3的结果可能更有实际意义（考虑到鸢尾花实际有三个品种）  3. k=4的结果显示出过度分类的特征，簇的大小分布不均匀且结果不稳定  两个关键发现：  1. 模型稳定性：k值越小，聚类结果越稳定，不受初始中心点影响  2. 簇的大小分布：k=3时的分布最接近实际情况，虽然轮廓系数不是最高  这个实验很好地展示了在选择最佳k值时需要平衡多个因素：  - 数学指标（轮廓系数）  - 结果稳定性  - 实际业务意义 | | | | | | |