机器学习实验报告——K-means

221900180 田永铭

2024 年 5 月 7 日

目录

一、实验内容

自选数据集（例如 lris 数据集）实现 K-means 聚类算法，并进行验证集评估，不能使用现成的算法库。

数据集：Iris

采用模型：手搓 K-means

二、实验环境

- 环境工具：Anaconda
- python 环境：python 3.10
- pytorch 版本：CPU 版本
- 环境：本实验采用 CPU 跑
- 代码运行平台:jupyter notebook

三、实验原理

以下是本次实验用到的实验原理：

K 均值聚类（K-means clustering）是一种常用的聚类算法，用于将数据点划分成具有相似特征的几个组或簇。该算法的原理相对简单，适用于大型数据集，并且容易理解和实现。

算法的步骤如下：

1. 初始化：首先，选择要分成的簇的数量 K。然后，随机选择 K 个数据点作为初始的聚类中心（centroids）。

2. 分配数据点到最近的聚类中心：对于每个数据点，计算它与每个聚类中心的距离，并将其分配到距离最近的聚类中心所对应的簇中。

3. 更新聚类中心：对于每个簇，计算其所有成员数据点的平均值，将该平均值作为新的聚类中心。

4. 重复步骤 2 和 3：重复以上步骤，直到聚类中心不再发生变化，或者达到预定义的迭代次数。

5. 收敛：当聚类中心不再发生变化或者达到迭代上限时，算法收敛，输出最终的聚类结果。

K 均值聚类的目标是最小化每个数据点与其所属簇中心之间的距离的总和，这种距离通常是欧式距离。

K 均值聚类的优点是简单易实现，并且对于大型数据集具有较高的效率。然而，它也有一些缺点，如对初始聚类中心的选择敏感，可能会收敛到局部最优解，而不是全局最优解。此外，对于非凸形状的簇，效果可能不佳。

四、根据要求依次汇报

汇报主要分为以下几个方面：

1. 数据的分析与处理
2. K-means 的设计原理和核心代码
3. 实验结果评估 (如总内聚度)
4. 聚类结果的可视化结果
5. 对实验结果的分析
6. 加分项：比较不同超参数对实验结果的影响
7. 加分项：对于有标签验证集，如何进行聚类结果的评估

4.1 数据的分析与处理

我使用的是 Iris 数据集。鸢尾花数据集（Iris dataset）是一个经典的机器学习数据集，通常用于分类和聚类算法的演示和测试。该数据集由英国统计学家罗纳德 • 费舍尔于 1936 年收集，并首次用于分类算法的研究。

该数据集包含了来自三个不同品种（或类别）的鸢尾花的样本。对每个样本，测量了四个特征：花萼（Sepal）长度和宽度，以及花瓣（Petal）长度和宽度。每个类别有 50 个样本，总共包含 150 个样本。

4.2 K-means 的设计原理和核心代码

原理已经在第一部分介绍，下面展示核心代码。

导入所需的包：

```
from sklearn.datasets import load_iris
import pandas as pd
import numpy as np
```

加载数据集:

```
# 加载数据集
iris = load_iris()
data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
target = iris.target
data['target'] = target
labels_true = target
```

K-means 实现核心:

```
# 自定义K—means算法使用现成的K—means库
class KMeans:
def __init__(self, n_clusters, max_iter=300):
        self.n_clusters = n_clusters
        self.max_iter = max_iter

def fit(self, X):
        n_samples, n_features = X.shape

# 随机初始化质心
self.centroids = X[np.random.choice(
        n_samples, self.n_clusters, replace=False)]

# 迭代优化质心
for _ in range(self.max_iter):
# 分配样本到最近的质心
        labels = self._assign_clusters(X)

# 更新质心为每个簇的平均值
new_centroids = np.array([X[labels == i].mean(axis=0)
         for i in range(self.n_clusters)])

# 如果质心不再变化, 则停止迭代
if np.allclose(new_centroids, self.centroids):
        break

        self.centroids = new_centroids

        return labels,self.centroids

def _assign_clusters(self, X):
# 计算每个样本到每个质心的距离
distances = np.linalg.norm(
        X[:, np.newaxis] - self.centroids, axis=2)
```

```
# 找到距离最近的质心的索引
return np.argmin(distances, axis=1)
```

使用 K-means 算法进行聚类:

```
kmeans = KMeans(n_clusters=3)
labels = kmeans.fit(data.values[:, :-1])
```

注意：在实现 Kmeans 的时候没有调用库函数，而是根据步骤一步一步计算中心、分配到簇、更新来从底层实现 Kmeans。而求均值或者求欧氏距离可以简单地用 numpy 中的方法实现，当然也可以用 for 循环自己计算（上学期已经实现过）。

4.3 实验结果评估 (如总内聚度)

利用以下代码计算一下总内聚度：

```
def total_intra_cluster_distance(X, centroids, labels):
total_distance = 0
for i in range(len(centroids)):
cluster_points = X[labels == i]
centroid = centroids[i]
total_distance += np.sum(np.linalg.norm(
cluster_points - centroid, axis=1)) # 计算欧氏距离(2-norms)
return total_distance

total_distance = total_intra_cluster_distance(data.values[:, :-1], kmeans.centroi
print("Total intra-cluster distance:", total_distance)
```

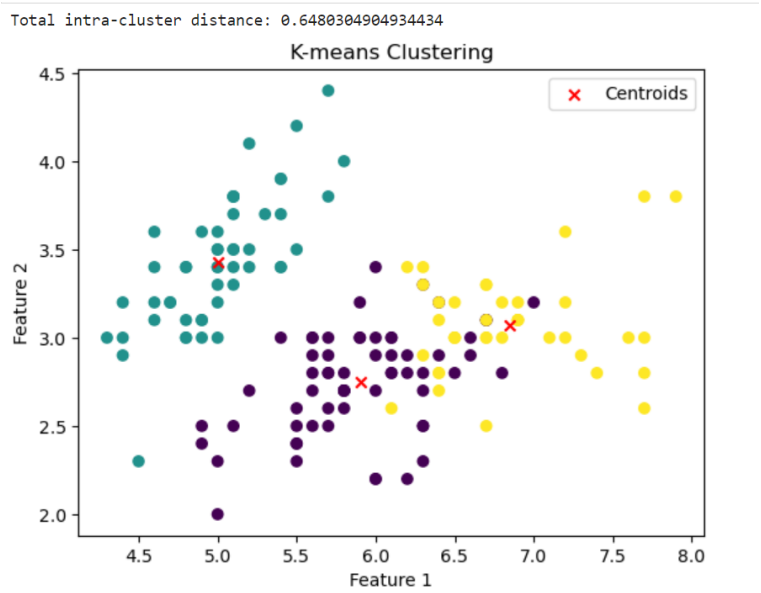
得到，平均总内聚度为 0.6480304904934434。仅看这个数字可能没有什么好的概念，这个数字将在后面进行对比参考，后续将说明这个数字说明实验结果比较好。
另外，还可以使用（Calinski-Harabasz 指数），它考虑了样本方差和簇间方差，来衡量聚类的可分离性。
而为了再考虑上簇间的分离度，还需要其他指标，这将在 “加分项：对于有标签验证集，如何进行聚类结果的评估” 部分展示。

4.4 聚类结果的可视化结果

为了方便对实验结果的分析，将聚类结果进行可视化。

```
import matplotlib.pyplot as plt
# 可视化聚类结果
plt.scatter(data.values[:, 0], data.values[:, 1],
            c=labels, cmap='viridis')
plt.scatter(kmeans.centroids[:, 0], kmeans.centroids[:, 1],
            marker='x', c='red', label='Centroids')
plt.title('K-means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

得到分类结果图如下：



可以看到，图中共有三个样本中心点，样本点根据临近关系分为了三种颜色，代表三个不同的类，这符合我们的预期。

4.5 对实验结果的分析

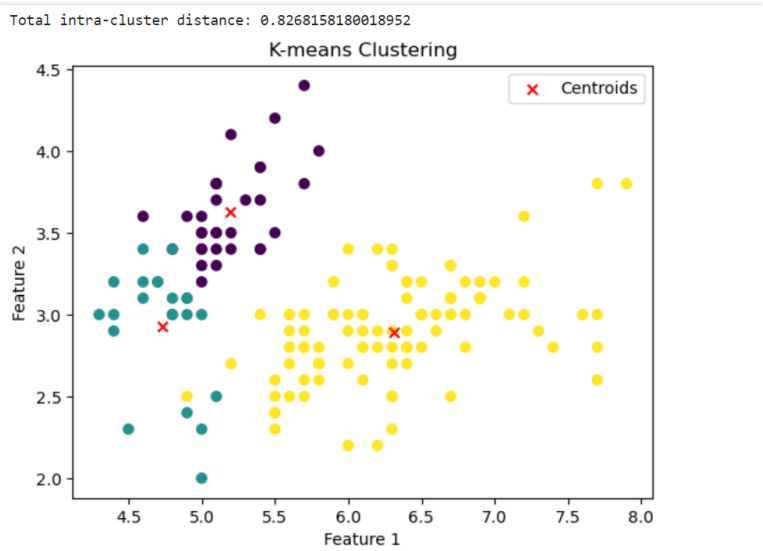
由于我们对 Iris 数据集的了解，我们知道它一共就是 3 类，所以设置 K 为 3 是非常合理的，而结果图也直观地反映了分类的合理性。同时，我们能够直接获取这个数据集的真实样本，经过对比，发现该方法每类差不多是 50 个样本，分类的正确率已经很高，对比网上

已有的分类结果图，发现和我的比较相似，这进一步验证了实验结果的正确性。

4.6 加分项：比较不同超参数对实验结果的影响

4.6.1 初始化的簇中心点的不同带来的影响

由于我在代码中采用的是随机化初始的簇中心点，所以只需要多次跑同样的代码就能看到影响。比如，我能跑出一个比原始结果差好一些的结果:



该结果不仅聚合度变大（反映簇内不紧密），而且结果也不对头。因为黄色的样本点数量远远大于绿色的，而实际上我们知道每一类应该是 50 个，所以出入还是比较大的。这就是初始化中心点初始的不好，比如说三个中心点全初始化到一个很相近的范围内，这样效果将会很差。

4.6.2 K 值不同带来的影响

以下分别是 K=2，10，100 的实验结果图：

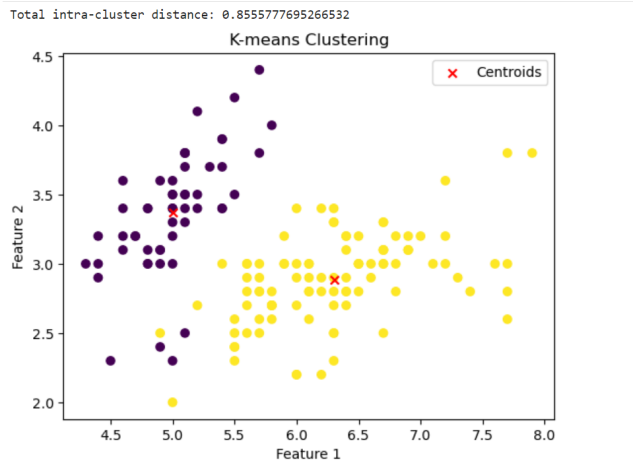

Total intra-cluster distance: 0.8555777695266532
K-means Clustering
Centroids
Feature 2
Feature 1
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

图 1: K=2

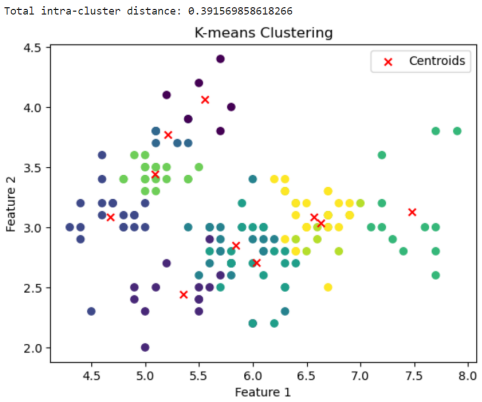

Total intra-cluster distance: 0.391569858618266
K-means Clustering
Centroids
Feature 2
Feature 1
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

图 2: K=10

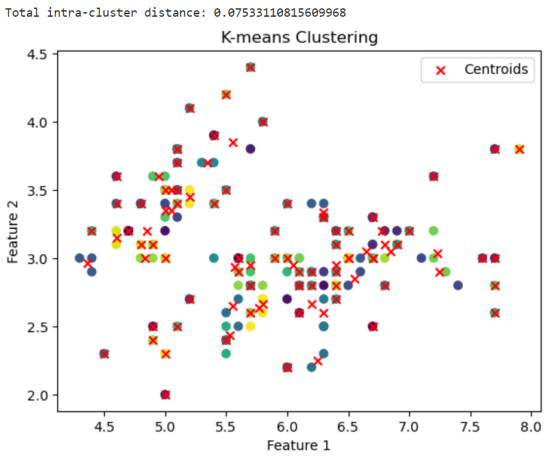


图 3: K=100

由此可见，不同的 K 值也会带来完全不一样的结果，对于 Iris 数据集，显然 K=3 是最合适的，但是 K=2 和 K=10 也不错，但是 K=100 却显然没有用，尽管它的总聚合度很小，但是它簇间的可分离度小，而且分类的很多类别中只有 1 到 2 个样本，这完全是一种过拟合。因此，K 值对实验结果影响也很大，同时我们还知道，只用聚合度评价结果不合适，所以要引入下一部分的其他评价方法。

4.7 加分项：对于有标签验证集，如何进行聚类结果的评估

首先一定要注意：需要划分训练集和测试机，才能合理地验证聚类结果。 我采用三种方法进行评估：

1. **兰德指数（Rand Index）**：度量两个分配的相似程度，即它们在一致性和不一致性方面的比率。
2. **互信息（Mutual Information）**：度量聚类结果和真实标签之间的信息交集。
3. **Fowlkes-Mallows 指数**：度量两个聚类之间的相似性，是准确率和召回率的几何平均值

这些指标越接近 1，表示聚类结果越好。

```
# 划分数据集为训练集和测试集（150=120+30）
X_train, X_test, y_train, y_test = train_test_split(
        data.values[:, :-1], data['target'],
        test_size=0.2, random_state=42)

# 使用K-means算法进行聚类
```

```
# 训练集
kmeans = KMeans(n_clusters=3)
train_labels,centroids = kmeans.fit(X_train)

# 测试集，根据样本点到训练集已经训练好的中心
        （存储在centroids变量中）的距离来直接给标签
def test_assign_clusters(X):
distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)
# 找到距离最近的质心的索引
return np.argmin(distances, axis=1)

test_labels = test_assign_clusters(X_test)



# Evaluating performance metrics on the testing set
rand_index_test = adjusted_rand_score(y_test, test_labels)
mutual_info_test = adjusted_mutual_info_score(y_test, test_labels)
fowlkes_mallows_test = fowlkes_mallows_score(y_test, test_labels)

print("Adjusted Rand Index (testing set):", rand_index_test)
print("Adjusted Mutual Information (testing set):", mutual_info_test)
print("Fowlkes-Mallows Index (testing set):", fowlkes_mallows_test)
```

由此得到结果图如下：首先说明一下，测试集的三个样本中心点是不做更新的，它是直接根据训练集的中心来划分测试集样本。两图中的 × 的位置是完全一样的，但是要注意两图坐标的中心点不同，需要仔细看看。再发现，训练集和测试机样本的确是互补的，而且它们的分类结果符合直觉，大体上的分布也是对应的。

指标方面，图片显示，三个评价指标都很接近 1，从更好的维度上综合说明了分类结果的优良性质。

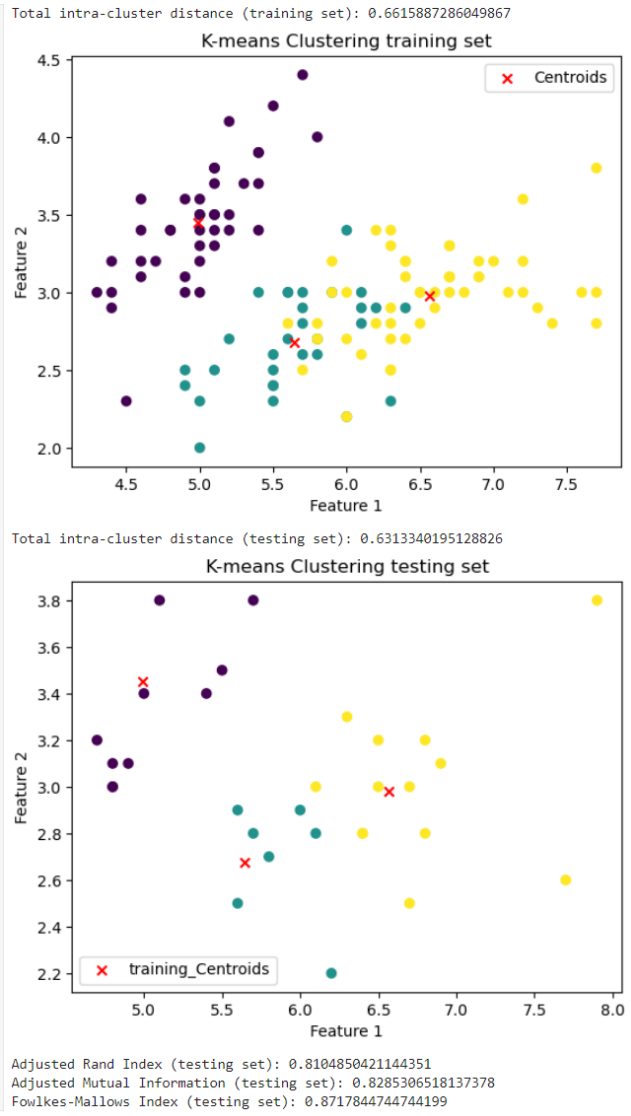
而若取 K=50 这样不好的参数，相应指标分别为

Adjusted Rand Index:0.08338375050194084

Adjusted Mutual Information: 0.3257944062533498

Fowlkes-Mallows Index: 0.2493441737975899

与此形成对比，还是 K=3 分类非常好。当然，对于新的数据集，K 基本上是接近真实类别数目的话，结果比较好。



4.8 加分项：训练中遇到的任何异常情况和调试过程

4.8.1 KeyError: "None of [Index([55, 67, 148], dtype='int32')] are in the [columns]"

这个错误表明我试图从 DataFrame 中提取名为'target' 的列，但实际上并没有该列。通过调试我发现，应该将 fit 方法中的数据改为 data.values[:, :-1]，这样就排除了最后一列目标变量，否则将会出问题。

4.8.2 分割测试集和训练集出现的问题

一开始，我按照如下方式分割训练集和测试集：

```
X_train, X_test, y_train, y_test = train_test_split
        (data.values[:, :-1], data, test_size=0.2, random_state=42)
```

报错 ValueError: Found input variables with inconsistent numbers of samples: [150, 30]。经过检验发现是我分割的不对。train_test_split 函数的参数 data 被传递了两次：一次作为特征数据的输入，一次作为目标标签的输入。因此，y_test 会包含整个 data DataFrame 中的所有列，而不仅仅是目标标签列。

为了解决这个问题，只需将 data DataFrame 中的目标标签列作为输入，而不是整个 DataFrame。改为 data['target'] 即解决了问题。

4.8.3 以小见大，自主寻找合适的 K 值

Iris 数据集太过于简单，只有 3 类，K 取 3 即可，我还尝试了 Mnist 等数据集，取到了合适的 K，也得到不错的分类结果。然而数据集可多了，于是我想以小见大，自主寻找合适的 K 值，相应代码如下：

```
best_k = 0
best_score = 0
for k in range(1, 100):  # 尝试不同的K值
        kmeans = KMeans(n_clusters=k)
        kmeans.fit(X)
        ch_score = calinski_harabaz_score(X, kmeans.labels_)
        if ch_score > best_score:
        best_k = k
        best_score = ch_score
print(f"Best␣K:␣{best_k},
␣␣␣␣␣␣␣␣Best␣Calinski-Harabasz␣Score:␣{best_score}")
```

应用这个代码到 Iris 上，确实是 K=3 最好。

五、总结

通过这次实验，我成功使用 K-means 算法完成了对 Iris 等数据集的分类。在此过程中，我勤加思考，从多维角度评价分类结果，并且考虑 K 值和初始化簇内中心点位置的影响，并且不局限于 Iris 数据集，探索适应多个数据集的灵活查找合适的 K 的方法。当然，我还学

习了一些前沿的 K-means 方法的变形，例如 K-means++ 等，这都使得我对 K-means 的理解更进一步。另外，本次实验中我也综合解决了一些问题，实现了各种功能，能力得到了提升，也对机器学习更加感兴趣了。