南開大學
Nankai University

南　开　大　学

计 算 机 学 院

机器学习实验报告

实验四 层次聚类

姓名：王泳鑫

学号：1911479

年级：2019级

专业：计算机科学与技术

指导教师：卫金茂

2021 年 11 月 17 日

摘要

关键字：层次聚类，**Machine Learning ， Deep Learning**

目录

一、 实验描述与要求

数据集：生成2000个样例，每个样例的前3列表示特征，第四列表示标签
基本要求：绘制聚类前后样本分布情况

1. 实现single-linkage层次聚类算法；
2. 实现complete-linkage层次聚类算法。

中级要求：实现average-linkage层次聚类算法，绘制样本分布图。
提高要求：对比上述三种算法，给出结论。

二、 程序设计与代码实现

首先，我们还是要使用lab3中生成数据集的函数来生成本次实验的数据集，只不过我们需要进行一些修改，每个样例由三个特征，和一个label，所以是四维的样例。但是我们可以借助一些sklearn中的函数，make_blobs方法就是一个专门为聚类产生数据集的方法，可以产生一个数据相应的标签，我们要对产生时的标签进行保存，以保证后面验证正确率的逻辑正确性。

我们实现聚合聚类的方法就是通过计算欧式距离，代码如下：

```
def cal_node_distance(node1, node2):
    return np.sqrt(np.sum(np.square(node1 - node2)))  # 计算欧式距离

def cal_set_distance(set1, set2, method):
    dists = []  #初始化距离矩阵
    #遍历两个类的每一个样例，并且计算两个样例之间的欧氏距离，存入中dists
    for i in range(set1.shape[0]):
        cur1 = set1[i]
        for j in range(set2.shape[0]):
            cur2 = set2[j]
            dist = cal_node_distance(cur1, cur2)
            dists.append(dist)
    #不同的方法对两个类中欧氏距离的要求不同
    if method is 'single':
        return np.min(dists)  #最短距离
    elif method is 'complete':
        return np.max(dists)  #最长距离
    elif method is 'average':
        return np.mean(dists) #平均距离
    else:
        raise IOError('Illegal method')
```

然后就是我们的聚合聚类操作方法，这里以average-linkage层次聚类算法为例，single-linkage和complete-linkage算法都与其相似，代码如下：

```
def average_linkage_clustering(sample, k):
    clusters = []   #初始化聚类
    for i in range(sample.shape[0]):
        node = []
        node.append(i)
        clusters.append(node)
    dists_matrix = []
    for i in range(len(clusters)):
```

```
        dists = []
        cur_set1 = sample[clusters[i]]
        for j in range(len(clusters)):
            cur_set2 = sample[clusters[j]]
            dist = cal_set_distance(cur_set1, cur_set2, method='average')
            dists.append(dist)
        dists_matrix.append(dists)
    dists_matrix = np.array(dists_matrix)
    while len(clusters) != k:  #当簇没有到我们要求的个时，停止聚合聚类。k
        base_idx = np.argmin(np.min(dists_matrix, 1))
        base_dist = dists_matrix[base_idx]
        target_idx = np.argsort(base_dist)[1]
        for each in clusters[target_idx]:
            clusters[base_idx].append(each)
        del clusters[target_idx]
        dists_matrix = np.delete(dists_matrix, target_idx, axis=0)
        dists_matrix = np.delete(dists_matrix, target_idx, axis=1)
        if base_idx > target_idx:
            base_idx -= 1
        for i in range(len(clusters)):
            dists_matrix[base_idx, i] = cal_set_distance(sample[clusters[base_idx]],
                                                         sample[clusters[i]], method='
                                                         average')
            dists_matrix[i, base_idx] = dists_matrix[base_idx, i]
    return clusters, dists_matrix
```

然后对聚类聚合计算的结果进行正确率的计算，通过使用一开始生成数据集的正确标签和最后聚合聚类的簇进行比对，代码如下：

```
def cal_acc(pred, true):
    correct = 0
    for i in range(pred.shape[0]):
        if pred[i] == true[i]:
            correct += 1
    return correct/pred.shape[0]
```

最后,我们通过结果分析函数对最后预测的结果，把0、1、2、3四种标签的排列情况依次遍历，然后再选出正确率最高的标签作为预测标签，然后返回正确率和预测标签，返回预测标签方便通过散点的形式表示出来，代码如下：

```
def result_analysis(sample, label, k, method='average'):
    '''结果分析接口'''
    if method == 'single':
        clusters, dists_matrix = single_linkage_clustering(sample, k)
    elif method == 'average':
        clusters, dists_matrix = average_linkage_clustering(sample, k)
    elif method == 'complete':
        clusters, dists_matrix = complete_linkage_clustering(sample, k)
    accs = []
    pred_labels = []
    cases = [x for x in range(k)]
    cases = permutations(cases) # 、、、四种标签的排列情况0123
    for case in cases:
        pred_label = np.zeros(label.shape)#首先创建一个零矩阵
        for i in range(case.__len__()):#依次遍历
            pred_label[clusters[i]] = case[i]
```

```
        pred_labels.append(pred_label)
        acc = cal_acc(pred_label, label)
        accs.append(acc)
    acc = np.max(accs) # 将排列中准确度最高的情况作为预测标签
    pred_label = pred_labels[np.argmax(accs)]
    return acc, pred_label
```

三、 结果演示与分析

首先，我们可以通过plot来展示我们随机生成的数据集的散点图，如图6所示

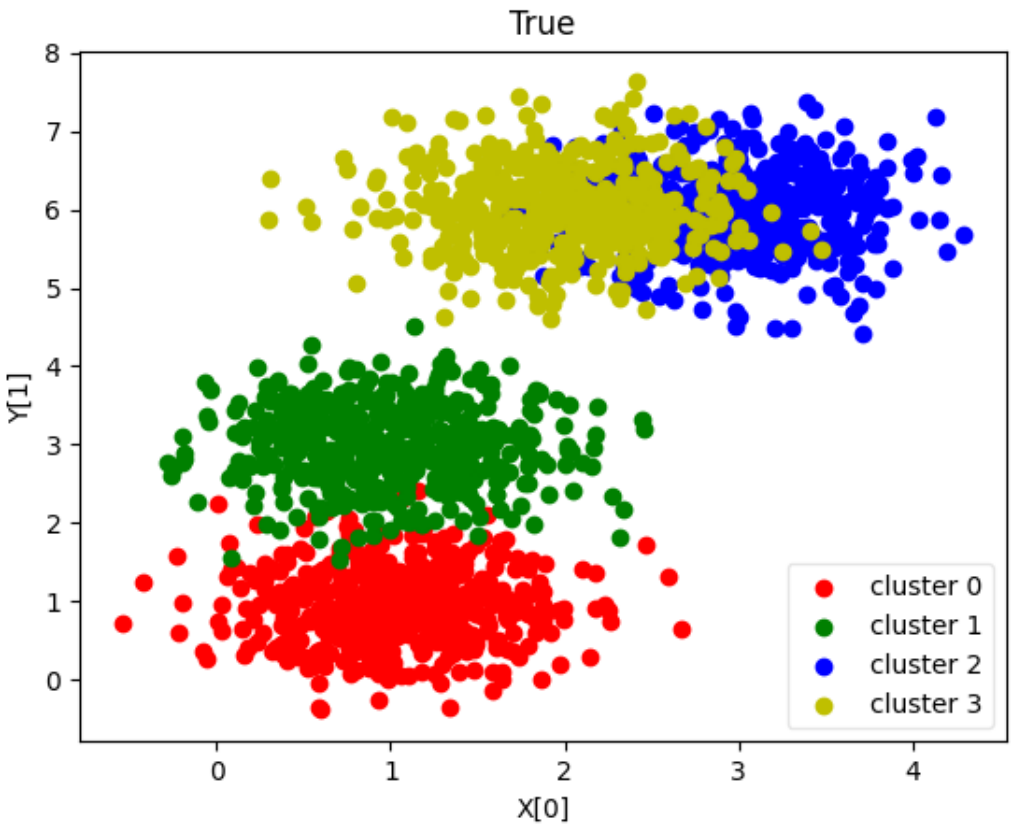

图 1: 数据集

如图6所示,我们可以看到single-linkage的聚合聚类的结果。

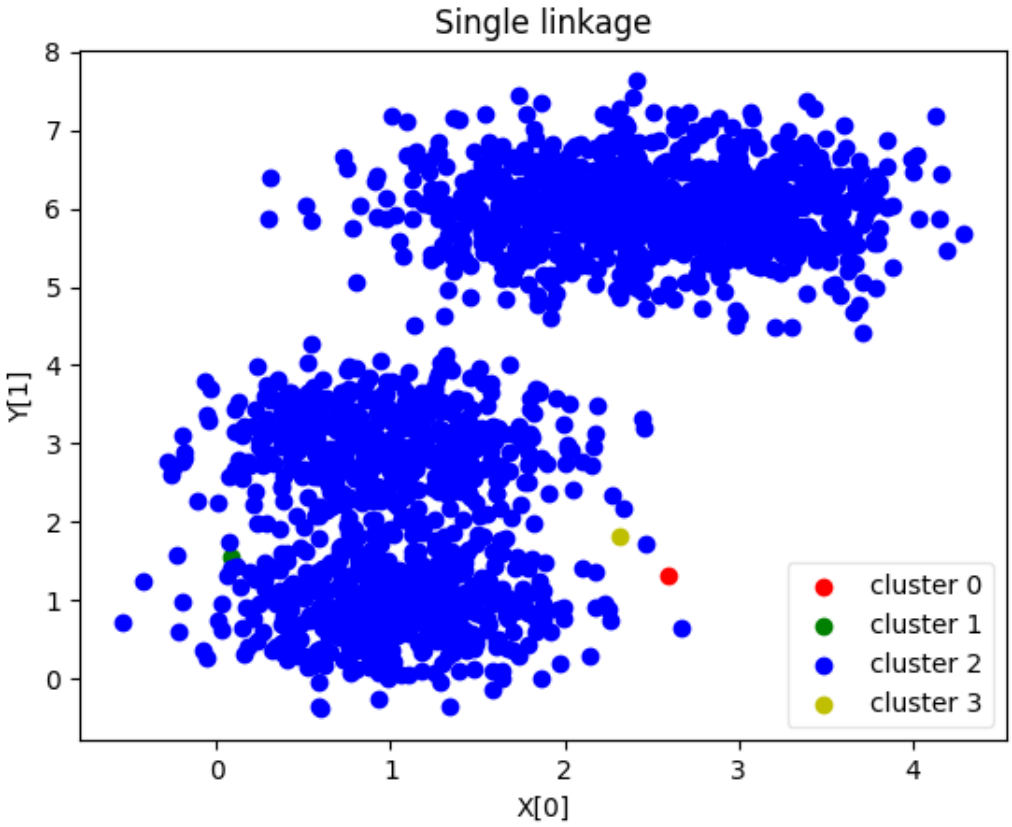

图 2: single-linkage

如图6所示,我们可以看到average-linkage的聚合聚类的结果。

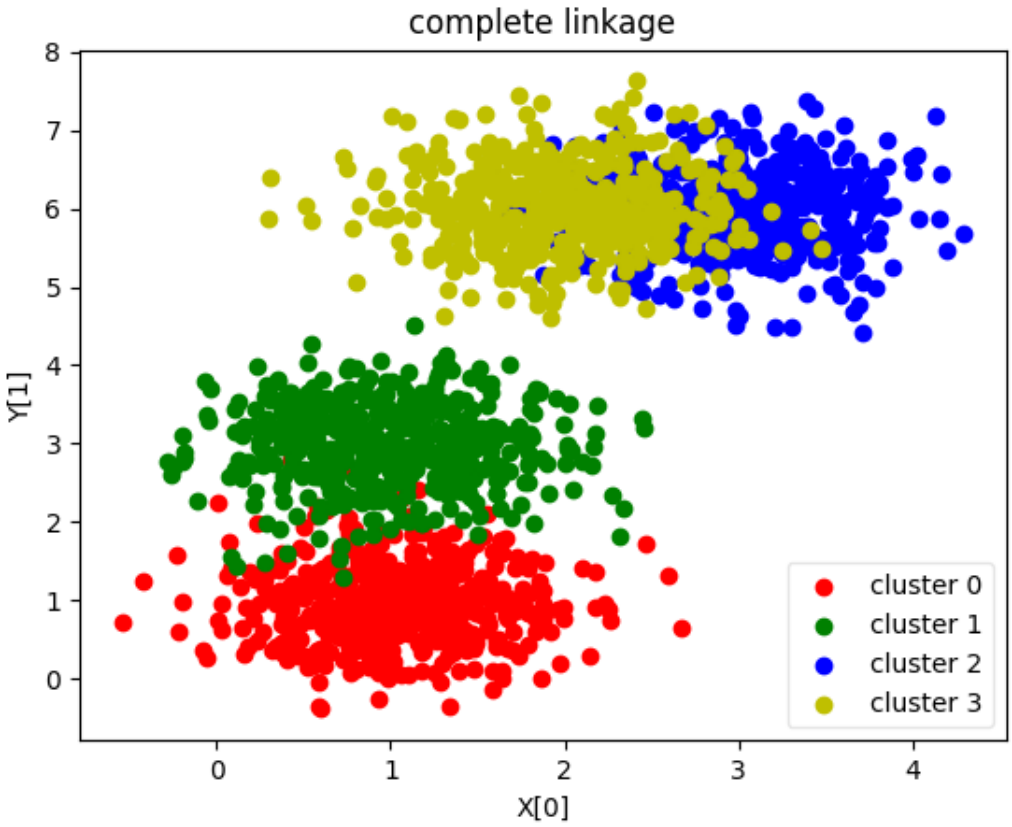

图 3: average-linkage

如图6所示,我们可以看到complete-linkage的聚合聚类的结果。

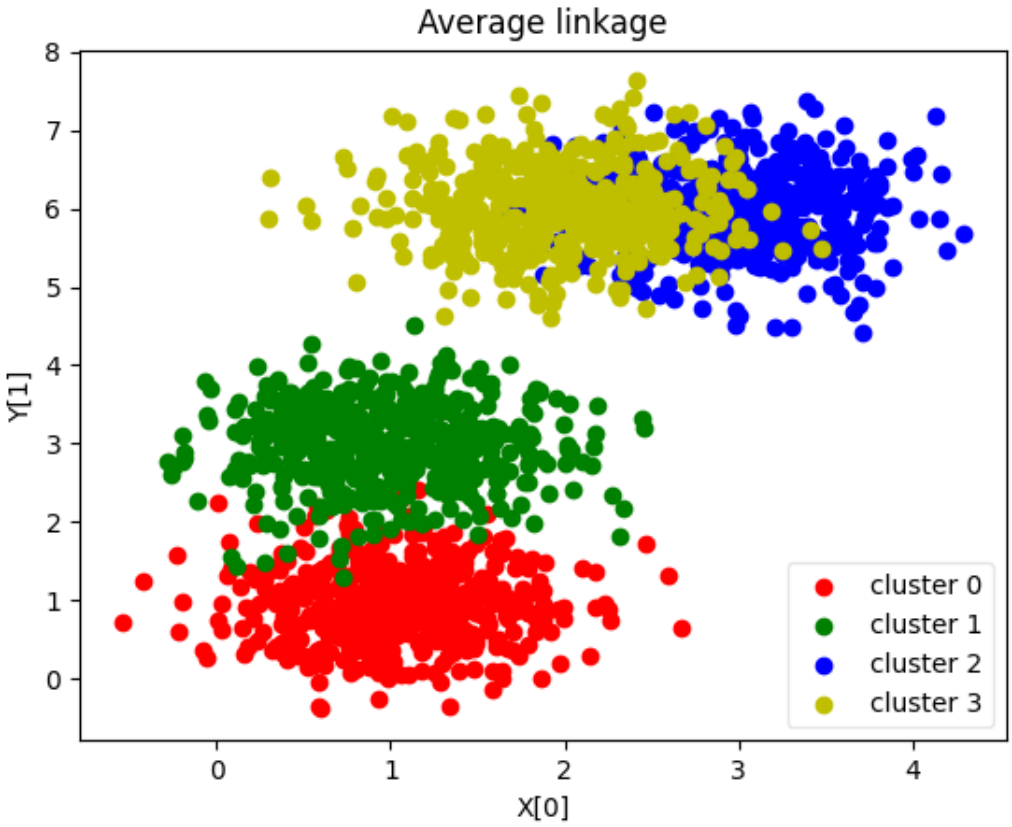

图 4: complete-linkage

根据上面三幅图展示的结果，我们可以看到average-linkage和complete-linkage这两种聚合聚类算法最后预测的结果都是不错的，相较于single-linkage聚合聚类算法正确率更高，具体的正确率如图6所示：

```
acc using single linkage: 0.251
acc using complete linkage: 0.995
acc using average linkage: 0.996
```

图 5: 正确率

为了对比上述三种算法的效果，我改变了k的大小来探究不同算法在k值为多少时最优，以及随k值变化算法效果变化的浮动，结果如图6所示：

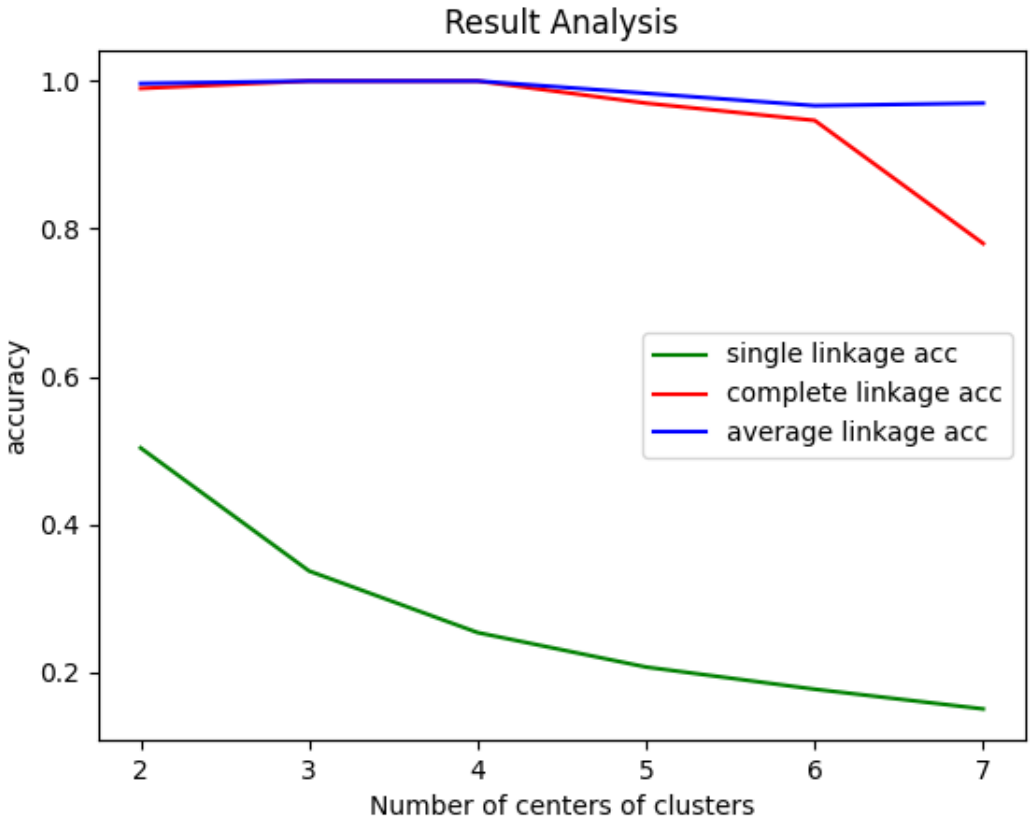

图 6: 准确率变化

我们可以发现随着k值的变化，single-linkage的准确率变化在慢慢减小，最后甚至低于0.2的准确率，complete-linkage算法一开始保持一个比较高的准确率，但是随着k值的增加准确率开始下降，当k=7时，准确率降到了0.8以下。相比之下， average-linkage算法的准确率一直都保持在一个比较高的状态。