实验五：层次聚类

杨扬　计算机科学与技术　1611132

摘　要

本次实验是层次聚类算法的学习及实现，所使用的数据集基于sklearn 中的 make_blobs 方法生成。本文档针对于此问题，分析了数据本身的特点，介绍了层次聚类算法的原理及各种簇间距离度量方法的特性，结合于实验数据集本身设计了算法，并基于源程序对实验的设计进行了解析，最后对实验的结果进行了分析。简单来说，本文档在对实验所有要求均做出实现的同时，也成功地进行了多方面的拓展，以极高的质量完成了此次实验。

关键词：层次聚类、簇、距离度量

目录

1 问题描述

聚类分析（Cluster analysis）亦称为群集分析，是对于统计数据分析的一门技术，在许多领域受到广泛应用，包括机器学习，数据挖掘，模式识别，图像分析以及生物信息。聚类是把相似的对象通过静态分类的方法分成不同的组别或者更多的子集（subset），这样让在同一个子集中的成员对象都有相似的一些属性，常见的包括在坐标系中更加短的空间距离等。

在机器学习领域中，聚类属于无监督学习（Unsupervised learning），即数据样本在学习过程中没有标签。简单来说，聚类算法在实际运行时，其根据样本间特征的异同来将样本分为若干类，使得同一类中的样本具有更强的共性，如更短的欧氏距离等。

数据聚类算法可以分为结构性或者分散性。结构性算法利用以前成功使用过的聚类器进行分类，而分散型算法则是一次确定所有分类。结构性算法可以从上至下或者从下至上双向进行计算。从下至上算法从每个对象作为单独分类开始，不断融合其中相近的对象。而从上至下算法则是把所有对象作为一个整体分类，然后逐渐分小。

本次实验的数据集文件并没有特别给出，而是通过 sklearn 中的 make_blobs 方法生成。简单来说，用户通过自定义若干个簇心的坐标值，再调整一些如聚类点个数的参数，即可生成一个可用于聚类分析研究的数据集。具体来说，在本实验中，生成的数据集一共 4 列，其中前 3 列为数据点的特征，最后 1 列为其标签，标签并不参与实际的学习过程，仅用于我们对聚类算法的结果分析。我们通过调用 matplotlib 中的方法，可将 1000 个生成的数据点可视化如下图：

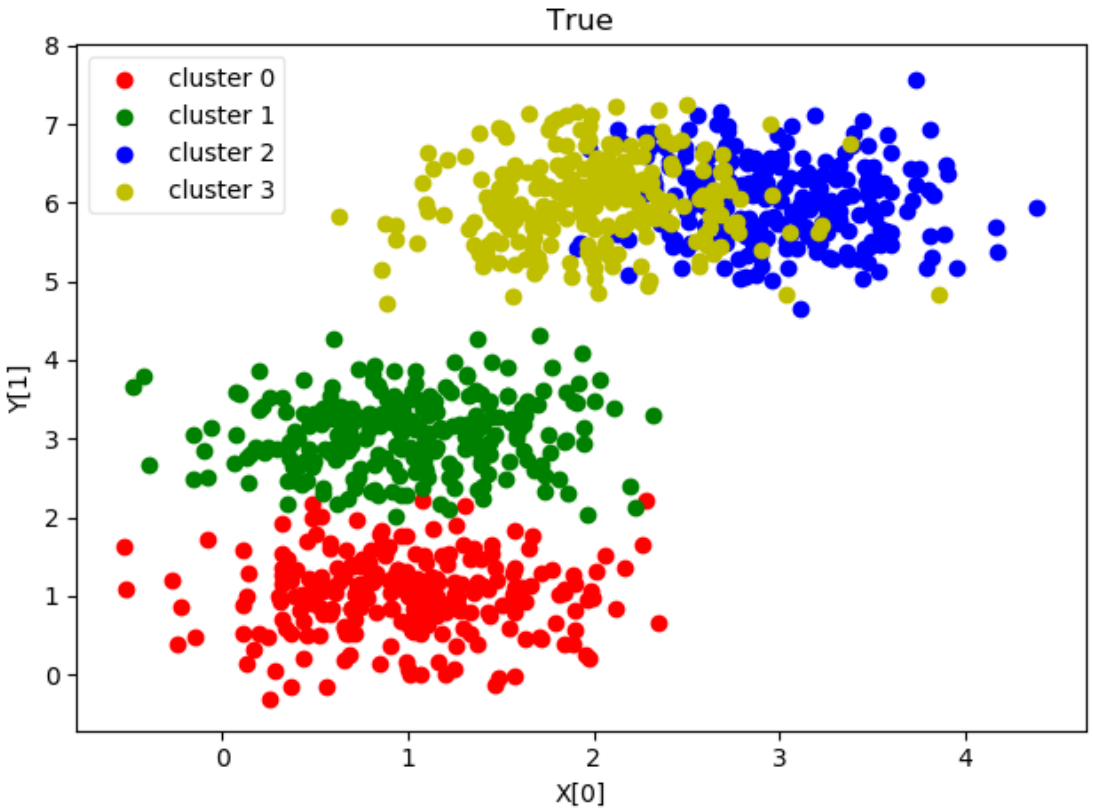

图 1: 数据点可视化

2 层次聚类

顾名思义，层次聚类算法 (Hierarchical clustering) 将数据集划分为一层一层的簇（Cluster），后面一层生成的 cluster 基于前面一层的结果。如前文所提到的普通聚类算法一样，层次聚类算法一般也分为两类：

1. Agglomerative 层次聚类：又称自底向上（bottom-up）的层次聚类，每一个对象最开始都是一个 cluster，每次按一定的准则将最相近的两个 cluster 合并生成一个新的 cluster，如此往复，直至最终所有的对象都属于一个 cluster。本文主要关注此类算法。
2. Divisive 层次聚类：又称自顶向下（top-down）的层次聚类，最开始所有的对象均属于一个 cluster，每次按一定的准则将某个 cluster 划分为多个 cluster，如此往复，直至每个对象均是一个 cluster。

不难看出，层次聚类算法是一种贪心思想的体现。本次实验中，我们采用自底向上的方法来实现层次聚类。

具体来说，我所实现的层次聚类算法步骤如下：

1. 初始时每个样本为一个 cluster，计算距离矩阵 D，其中元素 D_{ij} 为样本点 x_i 和 x_j 之间的距离；
2. 遍历距离矩阵 D，找出其中的最小距离（对角线上的除外），并由此得到拥有最小距离的两个 cluster 的编号，将这两个 cluster 合并为一个新的 cluster，然后更新距离矩阵 D（以其中一个 cluster 为基簇，将另一 cluster 合并至其中，删除被合并的 cluster，并重新计算基簇在 D 中所对应的行和列）。
3. 重复 2 的过程，直至最终只剩下 k 个 cluster 。

不难看出，上述算法的核心内容即在于 cluster 间的距离度量，而这也正是本次实验的出题基础。

具体来说，cluster 间距离度量有以下几种方式：

- Single linkage，定义两个 cluster 之间的距离为两个 cluster 之间距离最近的两个点间的距离：

$$D\left(C_i, C_j\right) = \min_{x \in C_i, y \in C_j} d(x, y)$$

- Complete linkage，定义两个 cluster 之间的距离为两个 cluster 之间距离最远的两个点间的距离：

$$D\left(C_i, C_j\right) = \max_{x \in C_i, y \in C_j} d(x, y)$$

- Average linkage，定义两个 cluster 之间的距离为两个 cluster 中点与点距离的平均值：

$$D\left(C_i, C_j\right) = \frac{1}{\left|C_i\right| \left|C_j\right|} \sum_{x \in C_i, y \in C_j} d(x, y)$$

在本次实验中，我们定义两个数据点间的距离为欧氏距离，即上式中的 $d(x, y)$ 为：

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

值得注意的是，从算法步骤上，我们不难看出层次聚类算法是一个复杂度在 $O(n^3)$ 级别的算法，因此，该算法并不太适合大规模数据点的聚类，这一点在我们后续实验及今后的研究或工作中尤为重要。

至此，我们完成了本次实验的理论分析部分。

3 实验及源程序分析

3.1 实验环境

本次实验在 Ubuntu 18.04 下进行，实验所使用的编程语言为 Python 3.6，IDE 为 Pycharm 2018.3.2，主要使用了 numpy、sklearn（仅用于生成数据点）、itertools、matplotlib 等 Python 包。

3.2 数据生成源程序分析

由于此次实验的内容较少，实验数据规模也不算太大，所以我们在一个 py 文件里即可完成本次实验，具体来说，程序的步骤如下所示：

首先我们在 Python 中导入所需要用到的 Python 包：

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_blobs
from itertools import permutations
```

本次实验提供了事先编写好的数据生成代码，其是通过调用 sklearn 中的 make_blobs 方法进行实现的，具体的接口调用代码如下：

```
def create_data(centers,num=100,std=0.7):
    '''
    生成用于聚类的数据集
    :param centers: 聚类的中心点组成的数组。如果中心点是二维的，则产生
        的每个样本都是二维的。
    :param num: 样本数
    :param std: 每个簇中样本的标准差
    :return: 用于聚类的数据集。是一个元组，第一个元素为样本集，第二个元
        素为样本集的真实簇分类标记
    '''
    X, labels_true = make_blobs(n_samples=num, centers=centers,
        cluster_std=std)
```

```
    return  X, labels_true
```

程序返回一个大小为 $num \times 3$ 的样本 X 和大小为 $num \times 1$ 的标签 $labels\_true$，即是我们本次实验所需的数据集。

由于聚类属于无监督学习，所以本次实验我们无需划分训练集和测试集，以整个数据集测试即可。接下来我们便直接进入到层次聚类的核心内容。

首先我们定义点与点间的欧氏距离计算函数：

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

一个 cluster 中并不只含一个数据点，因此我们需定义一个簇间距离计算函数，通过遍历数据点的方式来分别计算出 single linkage、complete linkage 或 average linkage 距离：

```
def cal_set_distance(set1, set2, method='average'):
    '''计算两个cluster间的距离，将其当作数据点的集合，遍历地进行处理'''
    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)
    if method is 'single':
        return np.min(dists)
    elif method is 'complete':
        return np.max(dists)
    elif method is 'average':
        return np.mean(dists)
    else:
        raise IOError('Illegal method')
```

3.3 基于 single linkage 的层次聚类

根据前文描述的层次聚类算法执行步骤，在程序中，我们先计算出距离矩阵，每一次迭代只合并两个 cluster，然后更新矩阵即可：

```
def single_linkage_clustering(sample, k):
    clusters = []
    for i in range(sample.shape[0]):
        node = []
        node.append(i)
        clusters.append(node)
    '''计算距离矩阵'''
    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='single'
                )
            dists.append(dist)
        dists_matrix.append(dists)
    dists_matrix = np.array(dists_matrix)
    '''直至该层次下只剩我们想要的k个簇'''
    while len(clusters) != k:
        base_idx = np.argmin(np.min(dists_matrix, 1))
        base_dist = dists_matrix[base_idx]
        target_idx = np.argsort(base_dist)[1] # 选取除本身外最小距离的
            cluster进行聚合
        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 # 如果被聚合cluster索引在基cluster之前, 删除
                之后需将基cluster索引减1
        for i in range(len(clusters)):
            dists_matrix[base_idx, i] = cal_set_distance(sample[
                clusters[base_idx]], sample[clusters[i]], method='
                single')
            dists_matrix[i, base_idx] = dists_matrix[base_idx, i] # 更
```

```
                新距离矩阵
        # if len(clusters) % 10 == 0:
        print('single linkage clustering number of clusters:', len(
            clusters))
    return clusters, dists_matrix
```

3.4 基于 complete linkage 的层次聚类

与 single linkage 的思想一致，不同的是需将 cluster 间距离计算方式做相应改变:

```
def complete_linkage_clustering(sample, k):
    '''思路与single linkage一样, 将距离计算参数修改即可'''
    clusters = []
    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='
                complete')
            dists.append(dist)
        dists_matrix.append(dists)
    dists_matrix = np.array(dists_matrix)
    while len(clusters) != 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='
                    complete')
                dists_matrix[i, base_idx] = dists_matrix[base_idx, i]
            # if len(clusters) % 10 == 0:
            print('complete linkage clustering number of clusters:', len(
                clusters))
        return clusters, dists_matrix
```

3.5 基于 average linkage 的层次聚类

同样的，基于前文的距离计算函数，我们也可以写出 average linkage 的层次聚类函数

```
def average_linkage_clustering(sample, k):
    '''与前两种算法思路一致，修改距离计算方式即可'''
    clusters = []
    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:
```

```
        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]
        # if len(clusters) % 10 == 0:
        print('average linkage clustering number of clusters:', len(
            clusters))
    return clusters, dists_matrix
```

最后，为方便后续我们对模型进行调用和结果分析，我们定义一个函数名为 result_analysis 的接口函数，这里需注意的是，由于聚类是一种无监督算法，本实验中我们只是将 4 类数据点简单地分开，但并不对应一个实际的标签值，而在 labels_true 中实际上是有如 1、2、3、4 的标签的，为与这种事先设定的 4 种实际标签相比较，我通过遍历 4 种标签的排列情况，将准确率最高的一种情况作为分类的实际情况，并将此准确率作为算法的实际准确率：

```
def result_analysis(sample, label, k, method='average'):
    '''结果分析接口'''
    if method is 'single':
        clusters, dists_matrix = single_linkage_clustering(sample, k)
    elif method is 'average':
        clusters, dists_matrix = average_linkage_clustering(sample, k)
    elif method is 'complete':
        clusters, dists_matrix = complete_linkage_clustering(sample, k)
    accs = []
    pred_labels = []
```

```
    cases = [x for x in range(k)]
    cases = permutations(cases) # 0、1、2、3四种标签的排列情况
    for case in cases:
        pred_label = np.zeros(label.shape)
        for i in range(case.__len__()):
            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
```

接口函数中用到的用于计算准确率的 cal_acc 函数为:

```
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]
```

3.6 数据可视化源程序分析

本次实验为我们提供了一个基于 matplotlib 中方法实现的数据可视化函数，其函数体如下:

```
def plot_data(*data, method='single linkage'):
    '''
    绘制用于聚类的数据集
    :param data: 可变参数。它是一个元组。元组元素依次为: 第一个元素为样
        本集, 第二个元素为样本集的真实簇分类标记
    :return: None
    '''
    X, labels_true=data
    labels=np.unique(labels_true)
    fig=plt.figure()
    ax=fig.add_subplot(1,1,1)
```

```
    colors='rgbyckm' # 每个簇的样本标记不同的颜色
    for i,label in enumerate(labels):
        position=labels_true==label
        ax.scatter(X[position,0],X[position,1],label="cluster %d"%label
            ,
        color=colors[i%len(colors)])

    ax.legend(loc="best",framealpha=0.5)
    ax.set_xlabel("X[0]")
    ax.set_ylabel("Y[1]")
    ax.set_title(method)
    plt.show()
```

此外，为测试簇心数量对几种算法性能的影响，我定义了一个测试函数如下：

```
def perform_test():
    '''测试簇心数量对模型性能的影响'''
    accs_single = []
    accs_complete = []
    accs_average = []
    samples = []
    centers = [[1, 1, 1], [1, 3, 3], [3, 6, 5], [2, 6, 8], [3, 2, 2],
               [1, 5, 2], [3, 1, 4], [4, 2, 1], [2, 3, 1], [3, 3, 5]]
    centers = np.array(centers)
    labels_true = []
    labels_single = []
    labels_complete = []
    labels_average = []
    for i in range(2, 8):
        center = centers[[x for x in range(i)]]
        X, label_true = create_data(center, 300, 0.5)  # 产生用于聚类的
            数据集，聚类中心点的个数代表类别数
        acc_single, label_single = result_analysis(X, label_true, i,
            method='single')
        acc_complete, label_complete = result_analysis(X, label_true, i
            , method='complete')
        acc_average, label_average = result_analysis(X, label_true, i,
            method='average')
```

```
        accs_single.append(acc_single)
        accs_complete.append(acc_complete)
        accs_average.append(acc_average)
        samples.append(X)
        labels_true.append(label_true)
        labels_single.append(label_single)
        labels_complete.append(label_complete)
        labels_average.append(label_average)
    return samples, labels_true, labels_single, labels_complete,
        labels_average, \
            accs_single, accs_complete, accs_average
```

最后，本次实验的主函数体为：

```
if __name__=='__main__':
    centers = [[1,1,1], [1,3,3], [3,6,5], [2,6,8]] # 用于产生聚类的中心
        点，聚类中心的维度代表产生样本的维度
    X,labels_true = create_data(centers, 1000, 0.5) # 产生用于聚类的数
        据集，聚类中心点的个数代表类别数
    print(X.shape)
    plot_data(X, labels_true, method='True')
    acc_single, labels_single = result_analysis(X, labels_true, 4,
        method='single')
    acc_complete, labels_complete = result_analysis(X, labels_true, 4,
        method='complete')
    acc_average, labels_average = result_analysis(X, labels_true, 4,
        method='average')
    print('acc using single linkage:', acc_single)
    print('acc using complete linkage:', acc_complete)
    print('acc using average linkage:', acc_average)
    plot_data(X, labels_single, method='Single linkage')
    plot_data(X, labels_complete, method='complete linkage')
    plot_data(X, labels_average, method='Average linkage')

    test_samples, test_labels_true, test_labels_single,
        test_labels_complete, \
    test_labels_average, test_accs_single, test_accs_complete,
        test_accs_average = perform_test()
```

```
    k = [x for x in range(2, 8)]

    plt.title('Result Analysis')
    plt.plot(k, test_accs_single, color='green', label='single linkage acc')
    plt.plot(k, test_accs_complete, color='red', label='complete linkage acc')
    plt.plot(k, test_accs_average, color='blue', label='average linkage acc')

    plt.legend()  # 显示图例
    plt.xlabel('Number of centers of clusters')
    plt.ylabel('accuracy')
    plt.show()
```

4 实验结果分析

实验中我设置数据点数量为 1000，测试 3 种聚类算法准确率如下：

	single linkage	complete linkage	average linkage
Accuracy	0.253	0.999	0.999

表 1: 3 种聚类算法的准确率

3 种聚类算法在 1000 个数据点情况下聚类可视化结果如下：

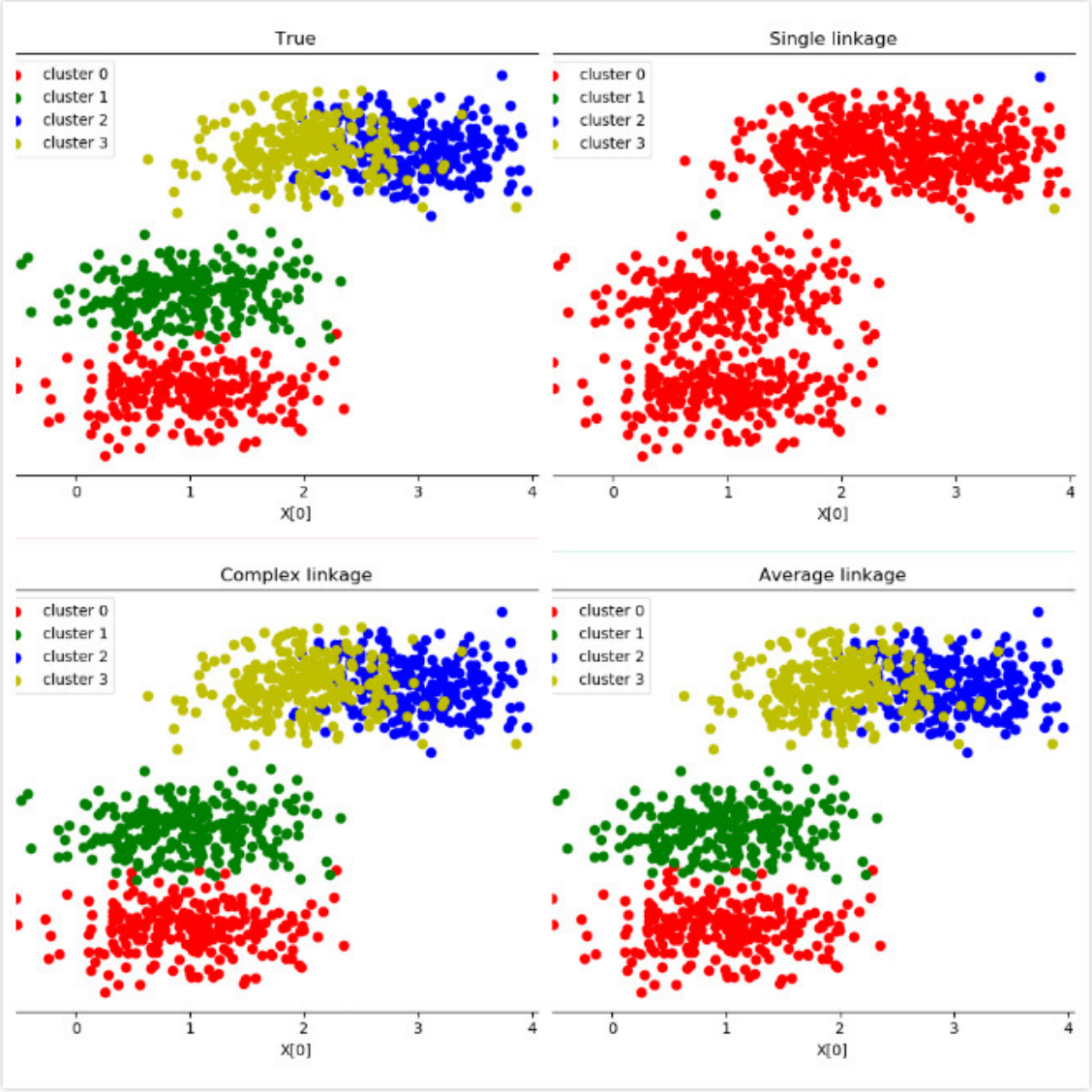

图 2: 实际标签及 3 种聚类算法预测标签可视化结果

不得不说，这种结果大大出乎了我的意料，基于 single linkage 的层次聚类的准确率出奇的低，准确来讲，算法将所有的点都认为为 1 类，仅有个别离群点得以幸免，从而得到了 0.25 的准确度。而后，通过对 single linkage 距离度量方法的简单研究，我发现这正是 single linkage 最明显的缺点的体现，即链式效应（Chaining）。

具体来说，single linkage 这种以最小值为距离度量标准的计算方式导致两个 cluster 明明从“大局”上离得比较远，但是由于其中个别的点距离比较近就被合并了，并且这样合并之后 Chaining 效应会进一步扩大，最后会得到比较松散的 cluster。因此，single linkage 的极差表现也就变得可以理解了。

此外，为对层次聚类算法做更深入的研究，我还尝试了随机增加簇心，改变

cluster 的数量来观察 3 种聚类算法的表现，得到结果如下：

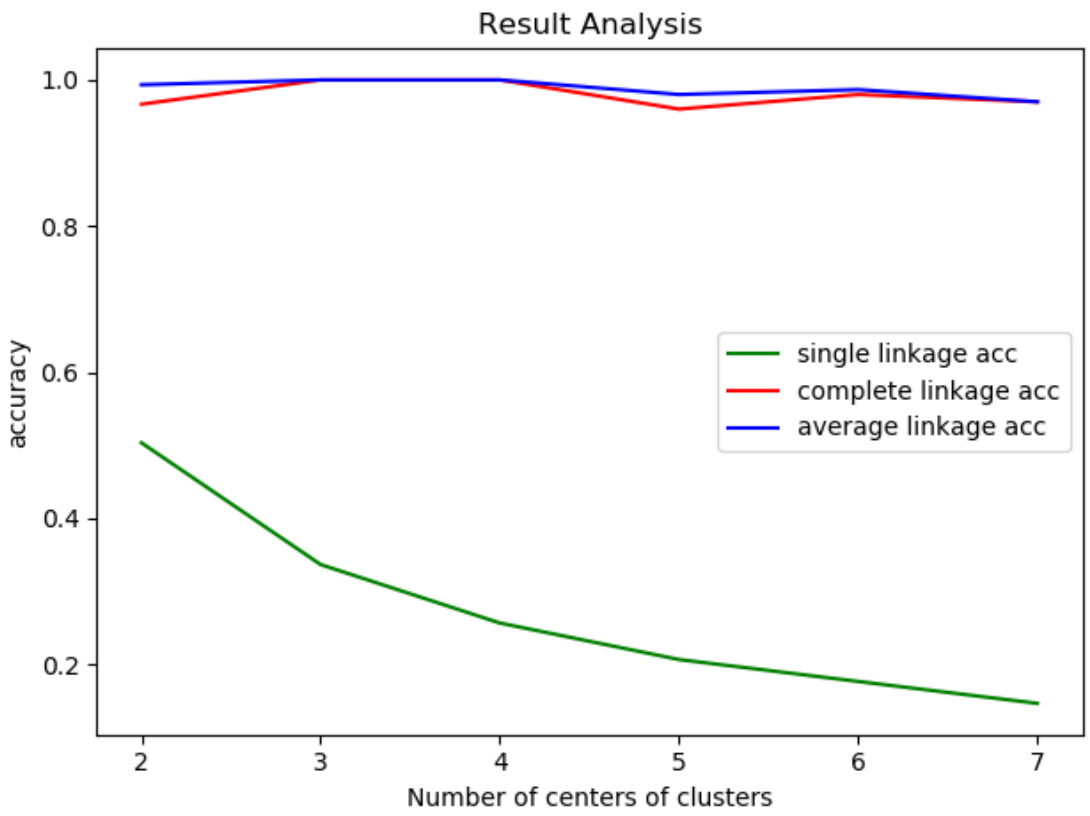

图 3: 3 种算法在不同簇心数情况下的表现

从图上看，complete linkage 和 average linkage 的效果远好于 single linkage，这也验证了我们前面所提到的，single linkage 在多个簇心较为接近时很容易出现 Chaining 的现象，导致模型将所有数据点归为 1 类，在改变簇心数时也就出现了 $1/2, 1/3, 1/4 \ldots$ 这样的变化。

而在 complete linkage 和 average linkage 两种算法之间，average linkage 又要更胜一筹。其实这也很好理解，complete linkage 可以理解为与 single linkage 背道而驰的另一个极端，它虽然解决了 single linkage 中的 Chaining 问题，但又出现了另一个问题：对离群点（outliers）的处理能力不足，从而可能产生不理想的聚类结果。

5 总结

聚类是我们所接触的第一种无监督学习算法，其思路与前几次实验有很大的不同，但同样也带来了更多的乐趣。总的来说此次实验我收获颇丰，对机器学习也有了更深层次的理解。

参考文献

[1] Thomas M. Mitchell. 1997. Machine Learning (1 ed.). McGraw-Hill, Inc., New York, NY, USA.

[2] https://zh.wikipedia.org/wiki/Wikipedia