



Assignment 3: Hierarchical Clustering Algorithm

2051498 储岱泽

算法设计

聚类算法分为“自顶向下”的分裂式和“自下向上”的凝聚式。在我的实现中，我采用的是**自下向上**的“**凝聚式**”聚类算法。下面是我写的一个名为 `HierarchicalClustering` 的类。

该类的成员变量包括：

- `n_clusters` : 期望分成的簇的个数。
- `linkage` : 链接方式选择，即在合并簇时采用的策略，可以是single、complete或average。
- `distance_method` : 距离计算方法，支持欧氏距离、曼哈顿距离和切比雪夫距离。
- `cluster_points` : 存储每个簇中的数据点索引。
- `distances_dict` : 存储样本间的距离字典。
- `link` : 存储层次聚类的链接矩阵。
- `time` : 记录算法的执行时间。

该类的方法包括：

- `fit(X)`: 实现了层次聚类算法的主要逻辑。在该方法中, 首先计算样本间的距离矩阵, 然后根据指定的链接方式合并簇, 直到达到期望的簇的个数。最后, 根据簇的合并情况得到最终的标签, 并计算算法的执行时间。
- `calculate_distance(x1, x2)`: 计算两个样本之间的距离, 根据指定的距离计算方法进行计算。
- `get_labels(distances_dict)`: 根据样本间的距离字典得到最终的标签。
- `plot_clusters(X, x_label, y_label, title)`: 可视化聚类结果, 将数据点按簇进行着色并绘制散点图。

聚类过程的实现

对于聚类的整个主要逻辑, 我主要是在这个类的fit函数中实现的, 下面我将对我的fit函数进行具体的分析。

首先, 我们先对距离矩阵 `distances` 进行初始化, 同时也对保存聚类结果的 `cluster_points` 数组进行初始化。

在最开始的时候, 由于我采用的是“自下而上”的聚类方式, 所以一开始`cluster_points`里面有 `n_samples` 个簇, 每一个簇里面有一个点。

```
1 n_samples, _ = X.shape
2 self.labels_ = np.zeros(n_samples)
3 distances = np.zeros((n_samples, n_samples))
4 self.cluster_points = [[i] for i in range(n_samples)] # 每个点为一个簇
```

计算距离矩阵

接下来计算最初的距离矩阵, 遍历输入的矩阵X中的每两个点, 计算任意两点间的距离:

```
1 #计算距离矩阵字典
2 for i in range(n_samples):
3     for j in range(n_samples):
4         distances[i][j] = self.calculate_distance(X[i], X[j])
```

其中, 在计算距离的函数中, 我采用了三种不同的距离计算公式供使用者自己选择:

```
1 # 运用距离公式计算距离
2 def calculate_distance(self, x1, x2):
3     if self.distance_method == 'Manhattan':
4         return np.linalg.norm(x1 - x2, ord=1) # 曼哈顿距离
```

```

5     elif self.distance_method == 'Chebyshev':
6         return np.linalg.norm(x1 - x2, ord=np.inf) # 切比雪夫距离
7     elif self.distance_method == 'Euclidean':
8         return np.linalg.norm(x1 - x2) # 欧氏距离

```

1. **曼哈顿距离 (Manhattan Distance)**：也称为城市街区距离或 L1 范数，它衡量两个点在标准坐标系上沿着矩形网格的边缘移动的距离。

$$d_{\text{Manhattan}}(x_1, x_2) = \sum_{i=1}^n |x_{1i} - x_{2i}|$$

2. **切比雪夫距离 (Chebyshev Distance)**：切比雪夫距离是在几何空间中点的最大距离。

$$d_{\text{Chebyshev}}(x_1, x_2) = \max_{i=1}^n |x_{1i} - x_{2i}|$$

3. **欧氏距离 (Euclidean Distance)**：欧氏距离是最常见的距离度量方法，它是在标准的欧几里得空间中计算的两个点之间的直线距离。

$$d_{\text{Euclidean}}(x_1, x_2) = \sqrt{\sum_{i=1}^n (x_{1i} - x_{2i})^2}$$

聚类

在做完前面的准备之后就可以开始正式聚类了。首先，先遍历距离矩阵，找到距离矩阵中距离最近的两个簇，并在 `cluster_points` 中进行簇的合并。

```

1  #先寻找距离最短的两个簇
2  min_distance = np.inf
3  for b in range(n_samples):
4      for c in range(b+1, n_samples):
5          if distances[b][c] < min_distance:
6              min_distance = distances[b][c]
7              min_i = b #将距离最短的这两个簇给记下来
8              min_j = c
9              # 先对存储簇的ID的数组给处理了
10 self.cluster_points[min_i].extend(self.cluster_points[min_j]) # 将第min_j行合并
    到上面的min_i
11 del self.cluster_points[min_j] # 然后将min_j行删掉

```

更新距离矩阵

在进行聚类之后，我们需要将我们的距离矩阵更新到最新的状态。在我的代码中，首先通过迭代样本数量的方式遍历距离矩阵字典的每一行，对于每一行中的每个元素，根据聚类的链接方式更新距离值。根据不同的链接方式（`self.linkage`），分别采用**单链接 (single)**、**完全链接 (complete)** 和**平均链接 (average)** 方法来更新距离值。

- **对于单链接（single linkage）**：将簇间最短距离作为簇间距离。因此，更新距离字典时，使用两个簇中距离最短的点之间的距离。
- **对于完全链接（complete linkage）**：将簇间最远距离作为簇间距离。因此，更新距离字典时，使用两个簇中距离最远的点之间的距离。
- **对于平均链接（average linkage）**：将簇间所有点对之间的平均距离作为簇间距离。因此，更新距离字典时，使用两个簇中所有点对之间距离的平均值。

在更新完距离字典后，将被合并的簇的行和列从距离矩阵中删除，因为它们已经合并成一个新的簇，不再需要计算它们之间的距离。

```
1 # 更新距离字典
2 for i in range(n_samples):
3     if i != min_i and i != min_j:
4         if self.linkage == 'single':
5             distances[min_i, i] = min(distances[min_i, i], distances[min_j, i])
6         elif self.linkage == 'complete':
7             distances[min_i, i] = max(distances[min_i, i], distances[min_j, i])
8         elif self.linkage == 'average':
9             distances[min_i, i] = (distances[min_i, i] + distances[min_j, i]) / 2
10
11 distances = np.delete(distances, min_j, axis=0)
12 distances = np.delete(distances, min_j, axis=1)
```

然后接下来一直不断重复前面的两个步骤：聚类以及更新距离矩阵，重复 `n_samples - self.n_clusters` 次，达到我们要分的类的个数就可以停止了。

时间和空间复杂度分析

- **时间复杂度**：算法的主要时间开销集中在计算样本间的距离矩阵和合并簇的过程中。假设有 n 个样本，计算距离矩阵的时间复杂度为 $O(n^2)$ ，合并簇的时间复杂度为 $O(n^3)$ 。因此，整体时间复杂度为 $O(n^3)$ 。
- **空间复杂度**：算法的空间开销主要来自距离矩阵和簇的存储。距离矩阵需要 $O(n^2)$ 的空间，簇的存储需要 $O(n)$ 的空间。因此，整体空间复杂度为 $O(n^2)$ 。

数据集选择

我选择的是经典的机器学习数据集——鸢尾花数据集

（<https://archive.ics.uci.edu/dataset/53/iris>）。该数据集包含150个样本，涵盖了3种不同品种的鸢尾花，每种品种50个样本。

每个样本包含4个特征，分别是花萼长度（sepal length）、花萼宽度（sepal width）、花瓣长度（petal length）和花瓣宽度（petal width）。鸢尾花数据集是一个经典的分类问题示例，旨在根据花萼和花瓣的尺寸将鸢尾花分为三个类别：山鸢尾（Iris-setosa）、变色鸢尾（Iris-versicolor）和维吉尼亚鸢尾（Iris-virginica）。

在我们的项目中，我们仅仅对其前面的四个特征进行聚类，然后将聚类的结果与其一开始标明的标签进行对比，计算出聚类的准确性。

数据集上的实验

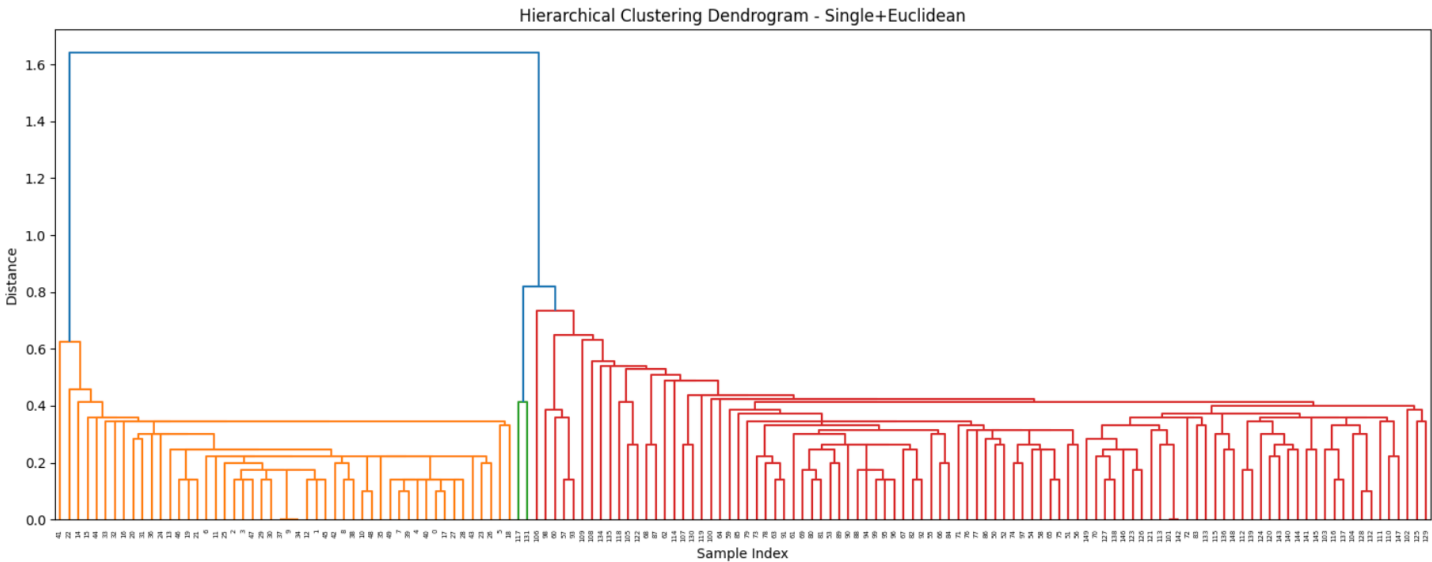
接下来我们对数据集进行了一组实验，来研究**三种聚类链接策略（single, complete, average）与三种距离公式（Manhattan, Euclidean）的9种不同的聚类方案**在鸢尾花数据集上的表现情况。

首先我们先重新开一个jupyter notebook进行我们的实验，在jupyter notebook中引入我刚刚写的类。然后，首先先对四维的数据进行一个聚类。

对四维数据进行聚类

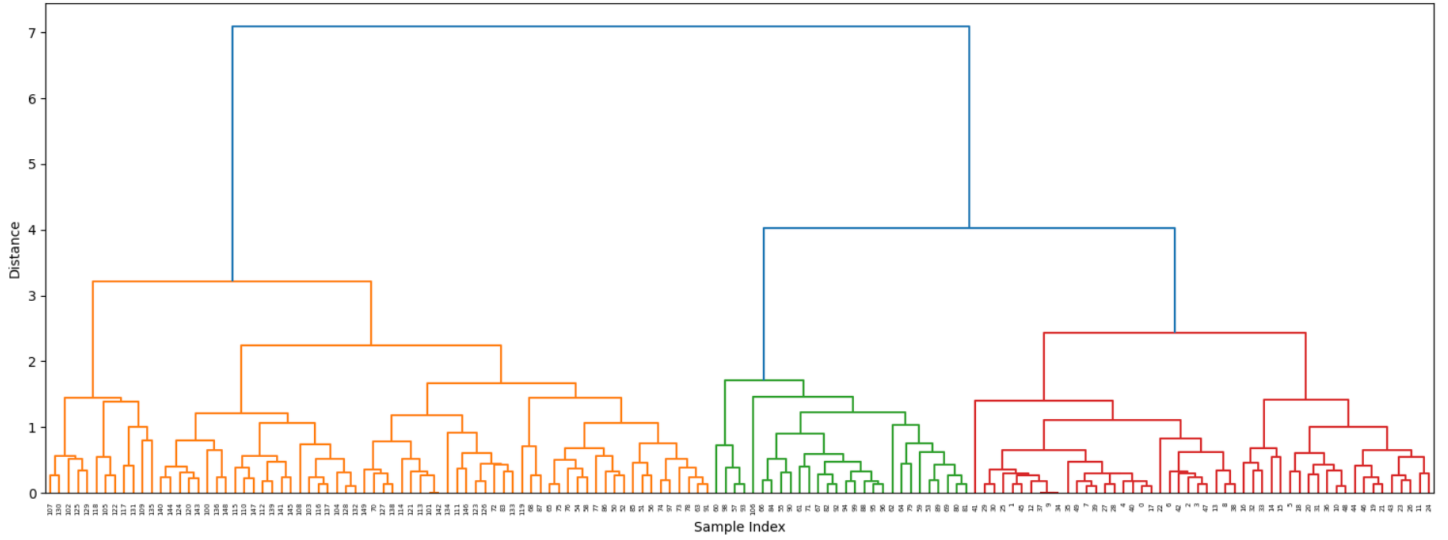
Euclidean距离公式

Single-linkage + Euclidean



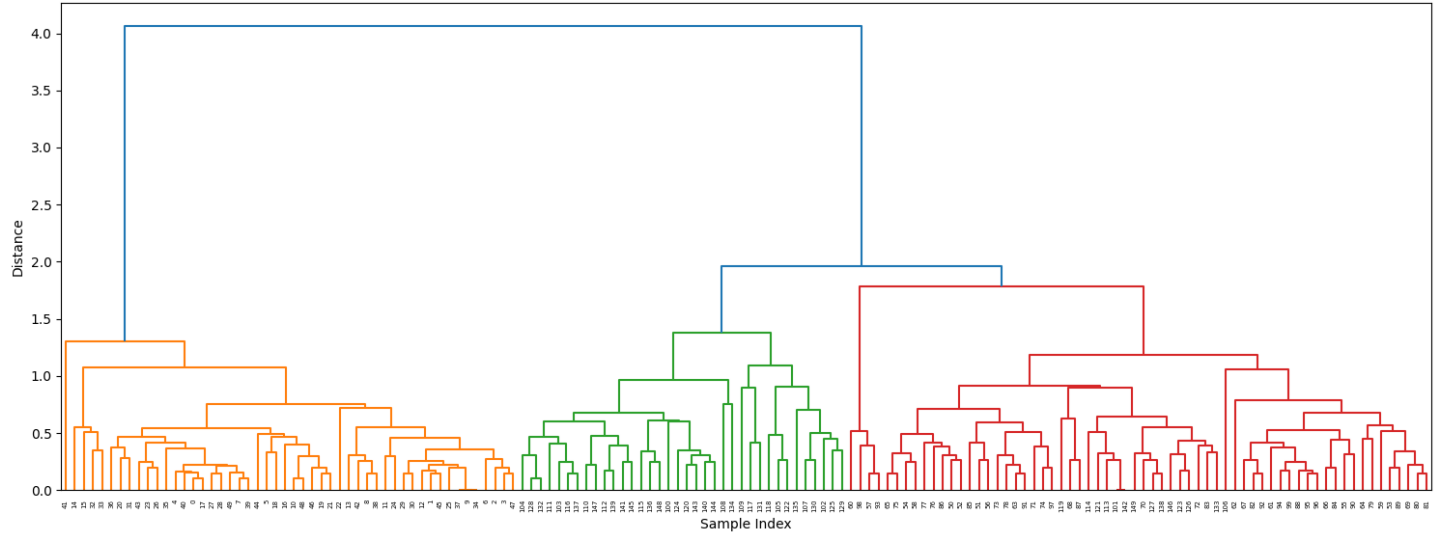
Complete-linkage + Euclidean

Hierarchical Clustering Dendrogram - Complete+Euclidean



Average-linkage + Euclidean

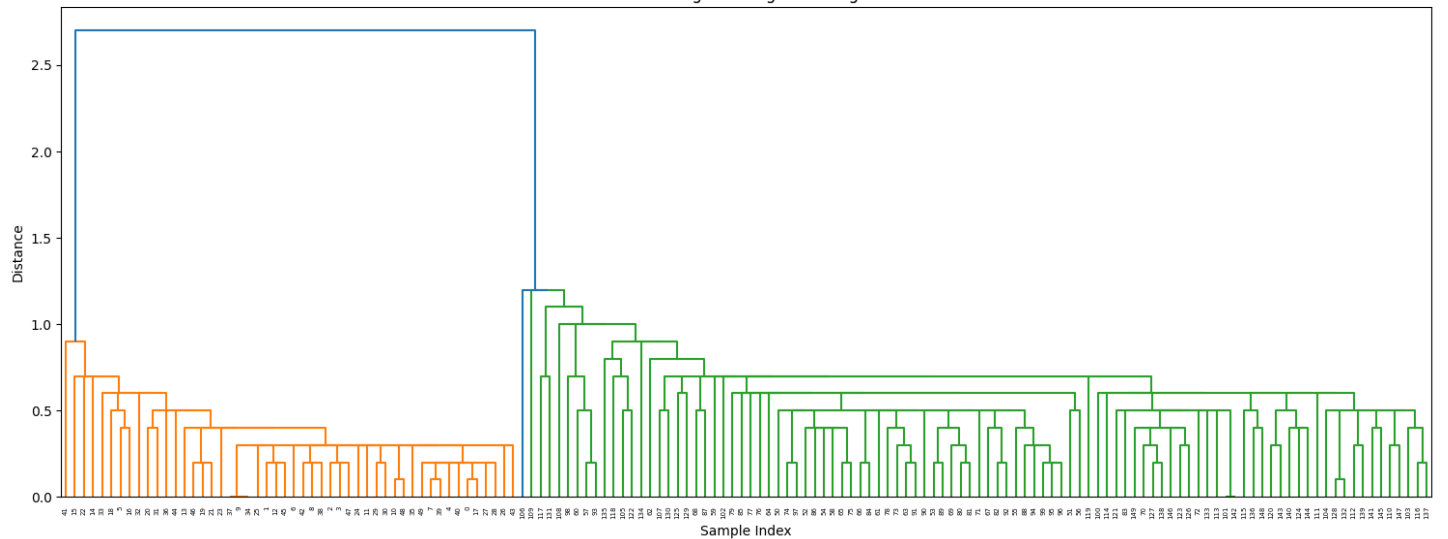
Hierarchical Clustering Dendrogram - Average+Euclidean



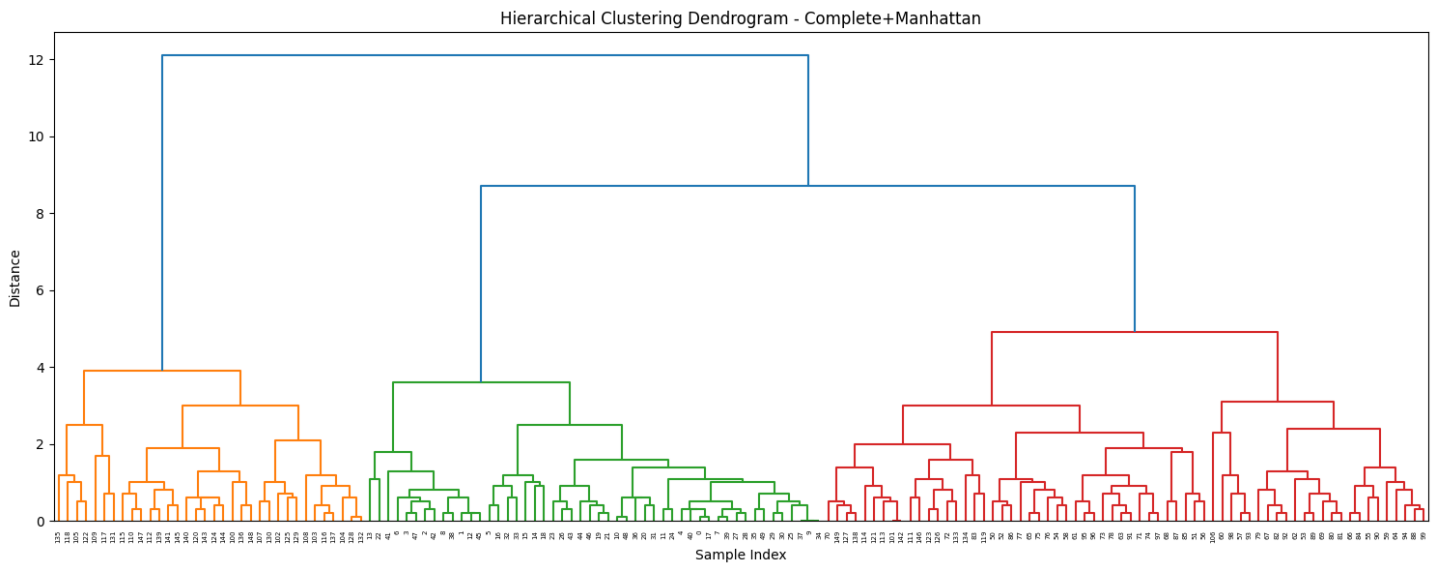
Manhattan距离公式

Single-linkage + Manhattan

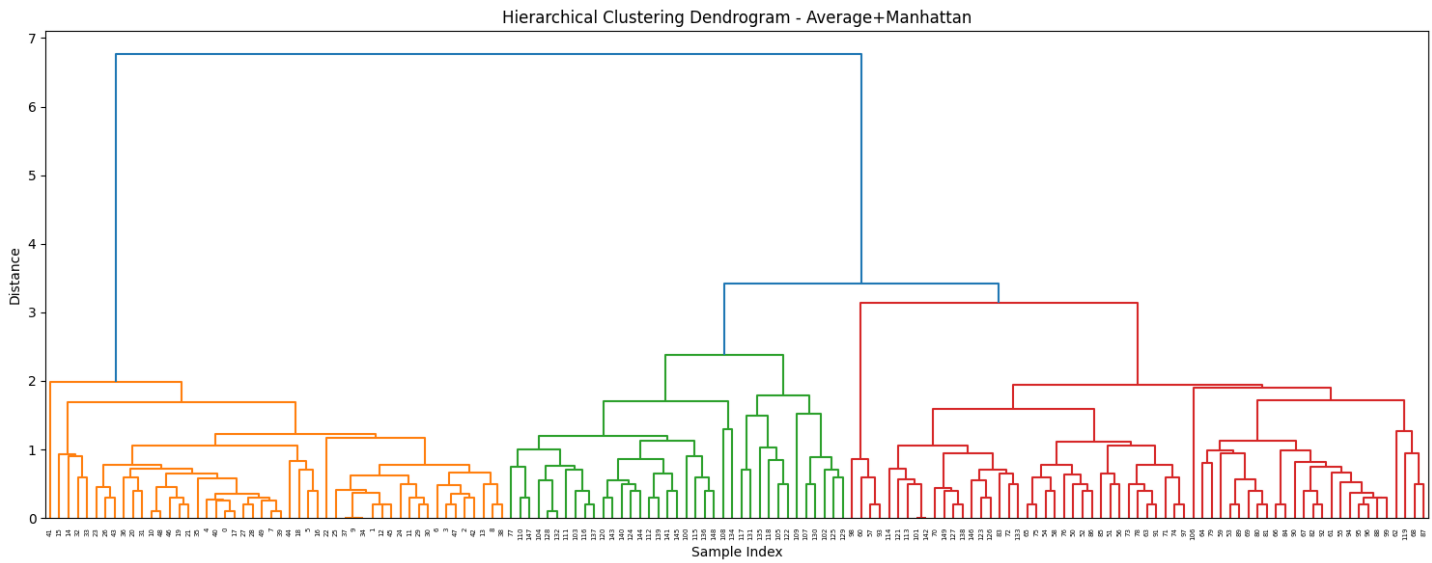
Hierarchical Clustering Dendrogram - Single+Manhattan



Complete-linkage + Manhattan



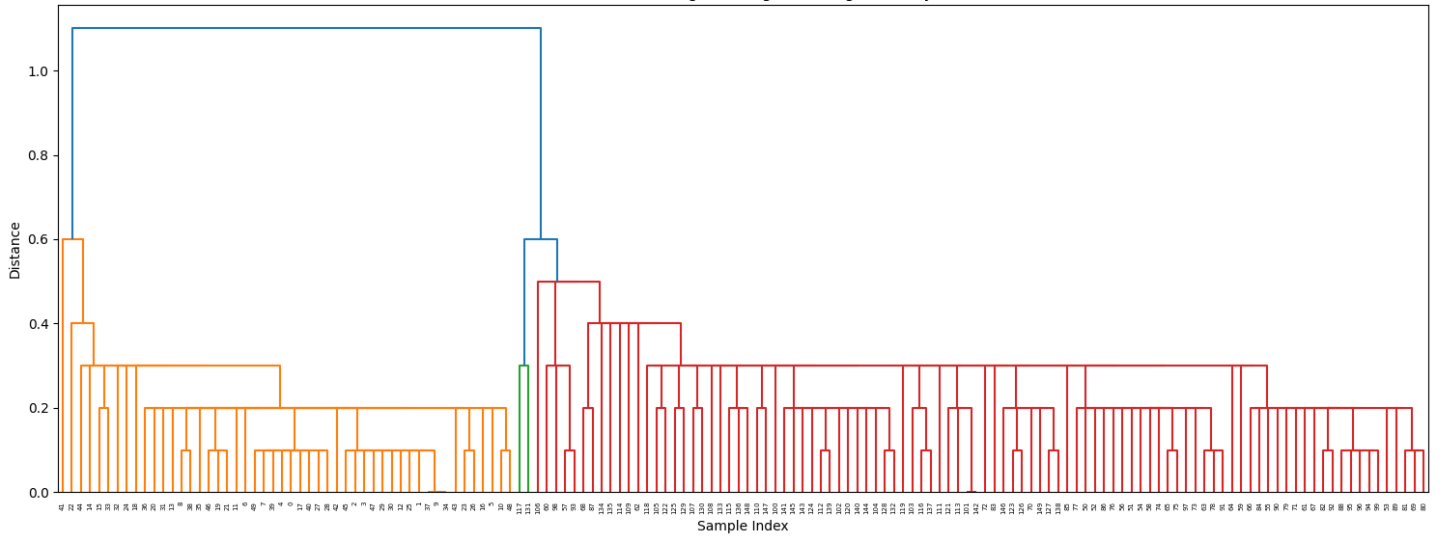
Average-linkage + Manhattan



Chebyshev距离公式

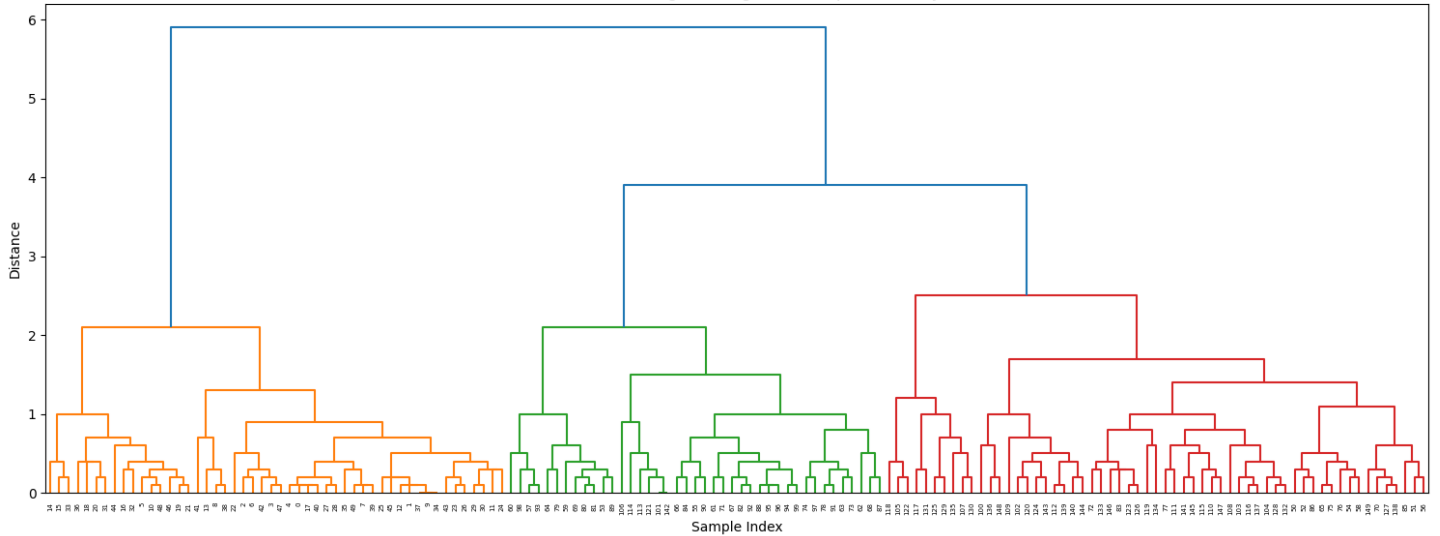
Single-linkage + Chebyshev

Hierarchical Clustering Dendrogram - Single+Chebyshev



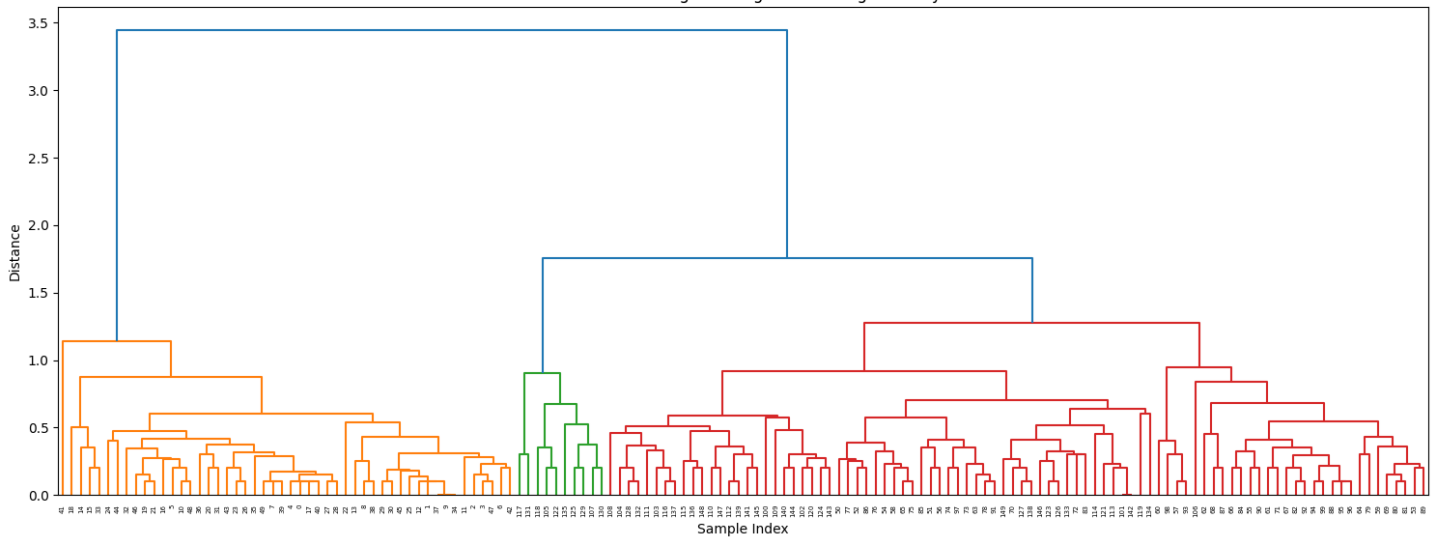
Complete-linkage + Chebyshev

Hierarchical Clustering Dendrogram - Complete+Chebyshev



Average-linkage + Chebyshev

Hierarchical Clustering Dendrogram - Average+Chebyshev

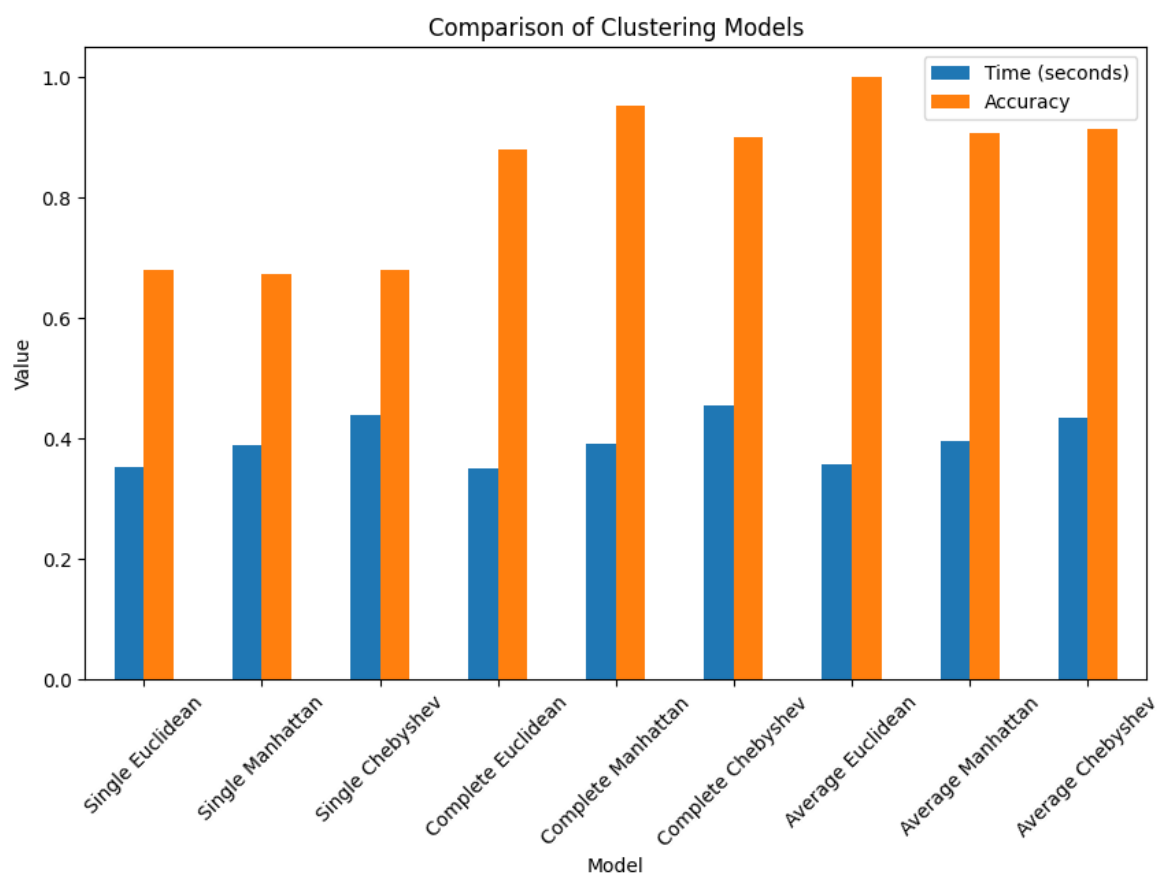


对上述九个方法的总结

以上便是9种不同的聚类策略聚类结果的树形图，接下来我们对其进行准确性，以及所用时间长短的分析，得到以下结果：

	Model	Time (seconds)	Accuracy
6	Average Euclidean	0.357053	1.000000
4	Complete Manhattan	0.391142	0.953333
8	Average Chebyshev	0.433251	0.913333
7	Average Manhattan	0.395126	0.906667
5	Complete Chebyshev	0.455184	0.900000
3	Complete Euclidean	0.350054	0.880000
0	Single Euclidean	0.351405	0.680000
2	Single Chebyshev	0.439568	0.680000
1	Single Manhattan	0.387728	0.673333

同时，为了观察的更加清楚，我用了条形图对其可视化：



于是，通过观察我们发现：

- 时间效率：** "Average Euclidean"模型的运行时间最短，为0.357秒，而"Complete Chebyshev"模型的运行时间最长，为0.455秒。因此，我们可以推断在处理鸢尾花数据集这个数据下，"Average Euclidean"模型是最快的，而"Complete Chebyshev"模型是最慢的。
- 准确性：** "Average Euclidean"模型的准确率为1.0，而"Single Euclidean"和"Single Chebyshev"模型的准确率都为0.68。因此，我们可以看出，在鸢尾花数据集这个数据下，"Average Euclidean"模型具有最高的准确率。

3. **聚类策略和距离度量的影响：** 因此，我们根据上面的结果可以得出，聚类策略和距离度量的选择会对最后聚类结果的准确性和运行时间造成一定的影响。**比如说在这个问题中综合看来，用Average-linkage+Euclidean的策略可以获得最好的效果，而使用Single-linkage+Euclidean的方案将会获得最差的结果。**

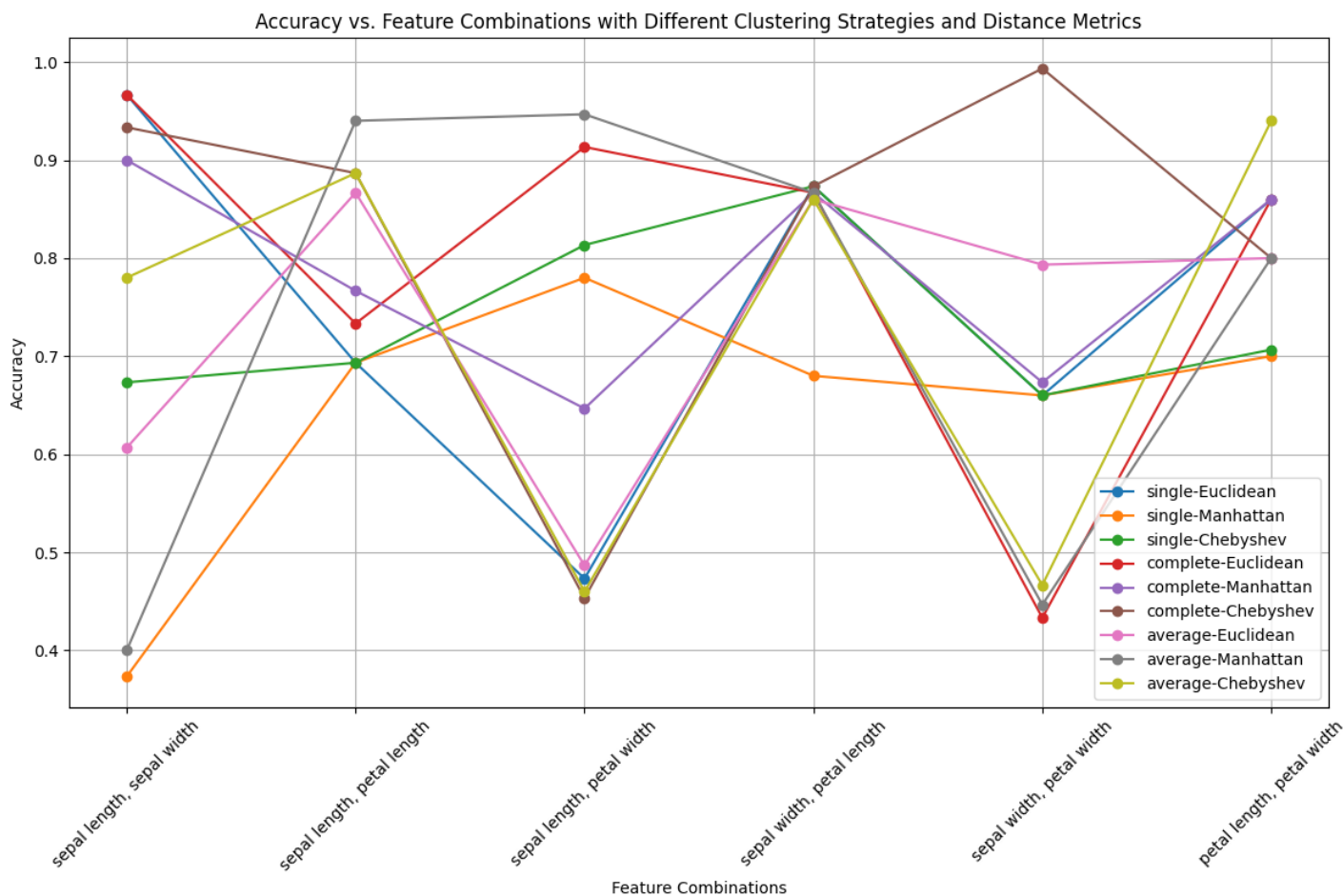
对二维数据进行聚类

接下来，为了进一步试验我写的聚类算法，我对数据集中的四个属性进行了拆分，然后两两组合进行聚类，以此来探究在不同的数据特征的数据上可能的更好的聚类策略的选择。

于是我们分别使用[sepal length,sepal width]，[sepal length,petal length]，[sepal length,petal width]，[sepal width,petal length]，[sepal width,petal width]，[petal length,petal width]，这6组数据分别采用9种不同的聚类策略来进行聚类准确性以及效率的分析，于是我们一共运行出了54种结果如下所示：

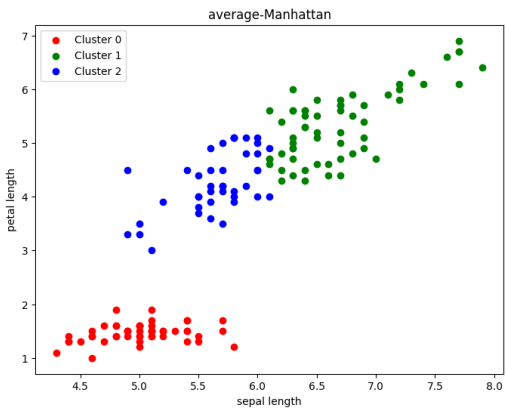
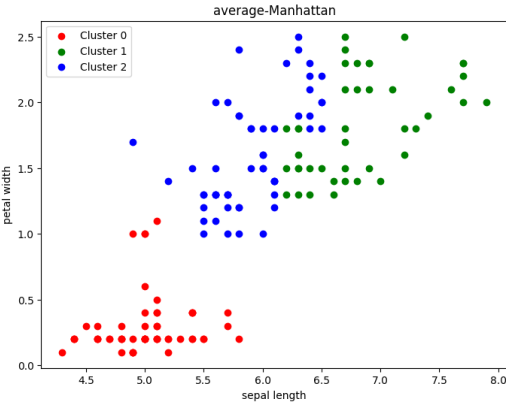
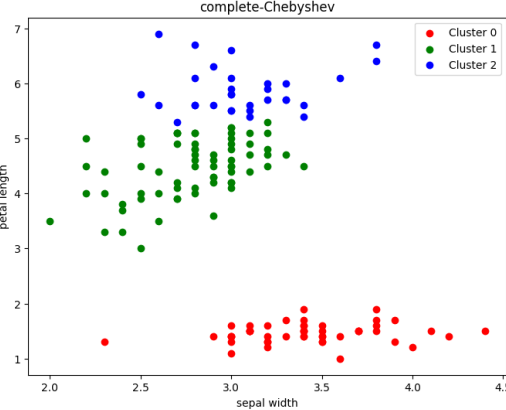
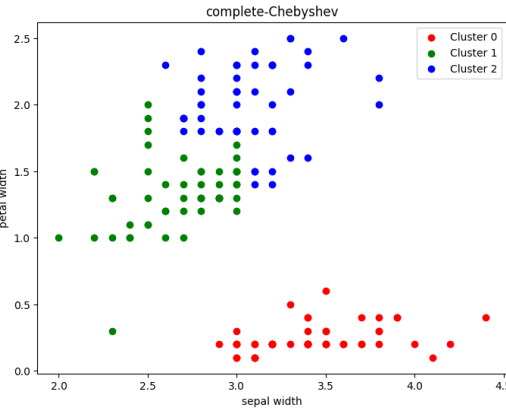
Features	Clustering Strategy	Distance Metric	Accuracy	Time
['sepal length', 'sepal width']	single	Euclidean	0.966666667	0.335850954
['sepal length', 'sepal width']	complete	Euclidean	0.966666667	0.383500099
['sepal length', 'sepal width']	complete	Chebyshev	0.933333333	0.430824995
['sepal length', 'sepal width']	complete	Manhattan	0.9	0.392833948
['sepal length', 'sepal width']	average	Chebyshev	0.78	0.431054115
['sepal length', 'sepal width']	single	Chebyshev	0.673333333	0.423910856
['sepal length', 'sepal width']	average	Euclidean	0.606666667	0.342473984
['sepal length', 'sepal width']	average	Manhattan	0.4	0.392094851
['sepal length', 'sepal width']	single	Manhattan	0.373333333	0.454981804
['sepal length', 'petal length']	average	Manhattan	0.94	0.386075974
['sepal length', 'petal length']	complete	Chebyshev	0.886666667	0.427604914
['sepal length', 'petal length']	average	Chebyshev	0.886666667	0.416636944
['sepal length', 'petal length']	average	Euclidean	0.866666667	0.374577999
['sepal length', 'petal length']	complete	Manhattan	0.766666667	0.389145136
['sepal length', 'petal length']	complete	Euclidean	0.733333333	0.411011934
['sepal length', 'petal length']	single	Euclidean	0.693333333	0.360995054
['sepal length', 'petal length']	single	Manhattan	0.693333333	0.380162001
['sepal length', 'petal length']	single	Chebyshev	0.693333333	0.413996935
['sepal length', 'petal width']	average	Manhattan	0.946666667	0.391015768
['sepal length', 'petal width']	complete	Euclidean	0.913333333	0.334345102
['sepal length', 'petal width']	single	Chebyshev	0.813333333	0.428013086
['sepal length', 'petal width']	single	Manhattan	0.78	0.387809992
['sepal length', 'petal width']	complete	Manhattan	0.646666667	0.383897781
['sepal length', 'petal width']	average	Euclidean	0.486666667	0.36809206
['sepal length', 'petal width']	single	Euclidean	0.473333333	0.357034922
['sepal length', 'petal width']	average	Chebyshev	0.46	0.429793119
['sepal length', 'petal width']	complete	Chebyshev	0.453333333	0.471099138
['sepal width', 'petal length']	single	Euclidean	0.873333333	0.370063066
['sepal width', 'petal length']	single	Chebyshev	0.873333333	0.43294096
['sepal width', 'petal length']	complete	Chebyshev	0.873333333	0.437329292
['sepal width', 'petal length']	complete	Euclidean	0.866666667	0.359770298
['sepal width', 'petal length']	complete	Manhattan	0.866666667	0.39088273
['sepal width', 'petal length']	average	Manhattan	0.866666667	0.387460947
['sepal width', 'petal length']	average	Euclidean	0.86	0.403429031
['sepal width', 'petal length']	average	Chebyshev	0.86	0.429925919
['sepal width', 'petal length']	single	Manhattan	0.68	0.389766216
['sepal width', 'petal width']	complete	Chebyshev	0.993333333	0.437646866
['sepal width', 'petal width']	average	Euclidean	0.793333333	0.397050858
['sepal width', 'petal width']	single	Euclidean	0.687142857	0.45027084
['sepal width', 'petal width']	complete	Manhattan	0.673333333	0.393126011
['sepal width', 'petal width']	single	Euclidean	0.66	0.379636049
['sepal width', 'petal width']	single	Manhattan	0.66	0.380847216
['sepal width', 'petal width']	single	Chebyshev	0.66	0.442054033
['sepal width', 'petal width']	average	Manhattan	0.446666667	0.478139162
['sepal width', 'petal width']	complete	Euclidean	0.433333333	0.35108304
['petal length', 'petal width']	average	Chebyshev	0.94	0.516466856
['petal length', 'petal width']	single	Euclidean	0.86	0.372261047
['petal length', 'petal width']	complete	Euclidean	0.86	0.365941763
['petal length', 'petal width']	complete	Manhattan	0.86	0.392055035
['petal length', 'petal width']	complete	Chebyshev	0.8	0.434802294
['petal length', 'petal width']	average	Euclidean	0.8	0.357517719
['petal length', 'petal width']	average	Manhattan	0.8	0.388402939
['petal length', 'petal width']	single	Chebyshev	0.706666667	0.422791958
['petal length', 'petal width']	single	Manhattan	0.7	0.385932684

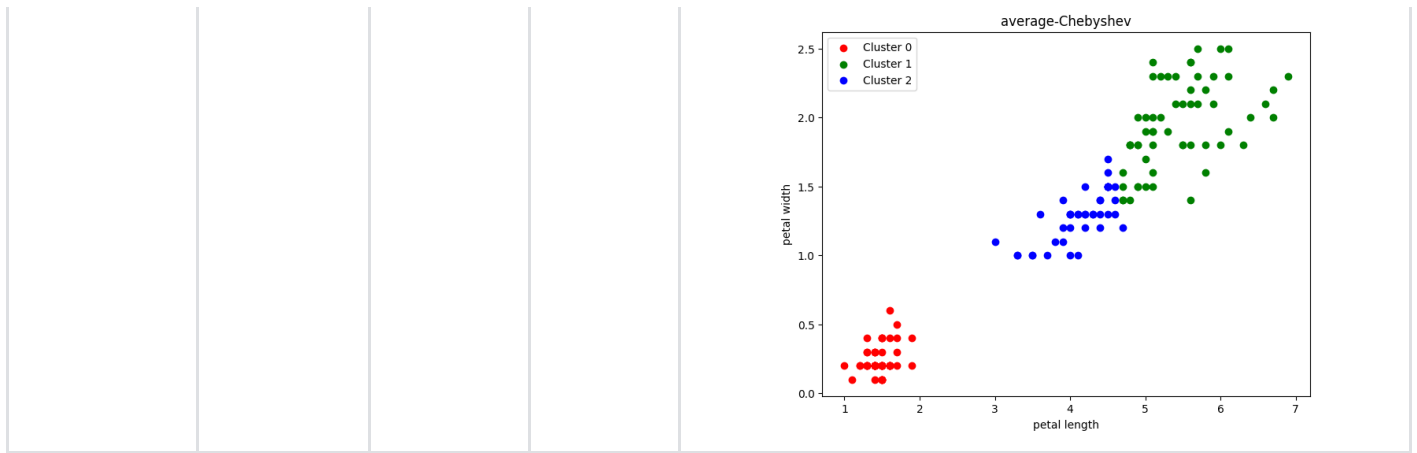
为了更加直观的看见对于同一组数据不同算法组合的结果，我们将结果可视化在一个折线图上，如下所示：



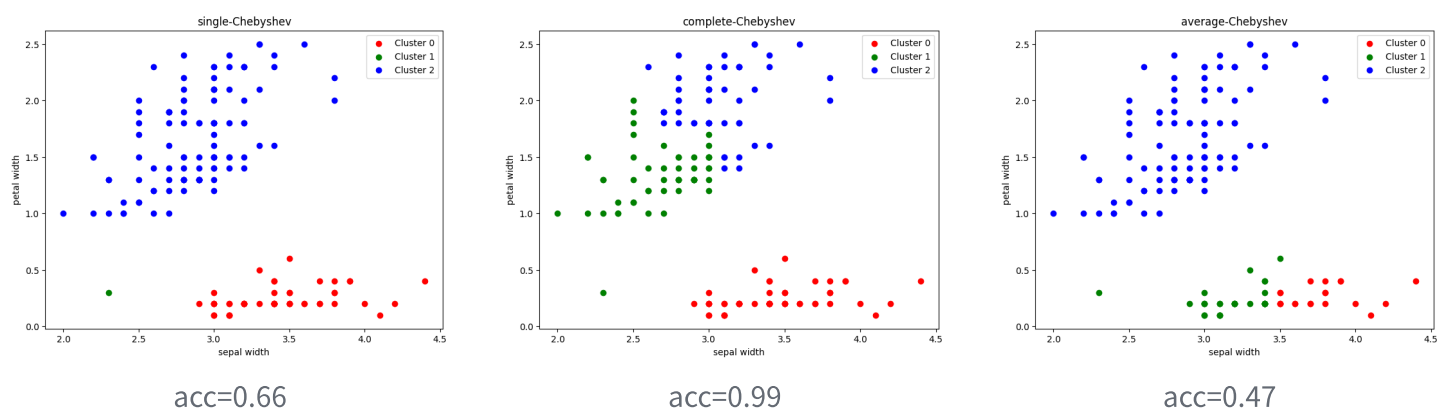
于是我们可以总结出对于每一组数据进行聚类最适合采用的聚类策略：

数据组合	linkage	distance Metric	accuracy	散点图
sepal length, sepal width	complete	Euclidean	0.97	
sepal length, petal length	average	Manhattan	0.94	

				
sepal length, petal width	average	Manhattan	0.95	
sepal width, petal length	complete	Chebyshev	0.87	
sepal width, petal width	complete	Chebyshev	0.99	
petal length, petal width	average	Chebyshev	0.94	



同时，我们以同样是采用[sepal width,petal width]这两个数据以及Chebyshev距离公式进行聚类为例，对比采用single、complete与average三种聚类策略的聚类结果：



结论

通过对上面的图标进行分析，我们可以看见在层次聚类中，不同的链接（linkage）方法适合处理不同类型的数据。以下是对三种常见的链接方法的适用情况的概述：

- 1. 单链接（Single Linkage）：**更适合于处理具有明显的“链状”结构的数据，其中相邻数据点之间的距离较小，而簇内的数据点之间的距离较大。单链接方法对异常值敏感，可能会产生“链状”的聚类结构。单链接方法常用于处理分散在空间中的长条形状或链状分布的数据。
- 2. 完全链接（Complete Linkage）：**完全链接方法将两个簇之间的最大距离定义为这两个簇的距离。因此，更适合于处理密集簇之间有明显边界的数据，其中簇内的数据点之间的距离较小，而不同簇之间的距离较大。完全链接方法在处理具有明显边界的数据时效果较好，但对于具有噪声或密集簇内部变化较大的数据可能不太适用。完全链接方法常用于处理密集簇之间有明显边界的数据，例如球状簇、圆形簇等。
- 3. 平均链接（Average Linkage）：**平均链接方法将两个簇之间所有数据点之间的平均距离定义为这两个簇的距离。因此，它更适合于处理簇内分布较为均匀、簇间距离相对较小的数据。平均链接方法对于各个簇之间的距离差异不敏感，能够较好地处理不同大小和密度的簇。平均链接方法常用于处理簇内分布均匀的数据，例如球形簇、椭圆形簇等，以及处理具有不同密度和大小的簇的数据。