HW3

Experiment settings

1. 目标：对iris数据集进行聚类分析。
2. 数据集：iris数据集，来自uci公开数据集，大小5kb，特征数量共4个，分别是sepal length，sepal width，petal length，petal width。
3. 参数设置：聚类数量参考iris数据集target的数目，设置为3。层次聚类方法选择自下而上，距离度量采用欧式距离，链接方法包括single、complete、average和ward。
4. 聚类效果评价指标：采用轮廓系数评价。
5. 实验环境：Python3.10
6. 需要安装的包：sklearn
7. 实验过程：先进行数据预处理，然后对4种链接方法分别比较用时、聚类效果，画出图像进行可视化分析。

数据预处理

Code:

```
# 加载数据集
def load_iris_dataset(filename):
    with open(filename, 'r') as csvfile:
        csvreader = csv.reader(csvfile)
        next(csvreader)  # Skip header if present
        dataset = [list(map(float, row[:-1])) for row in csvreader if row]  #
Exclude the label
    return dataset
```

下载uci iris数据集，读取csv内容。

链接方式

一共采用4种链接方式：single、complete、average、ward。

Code:

```
# 距离计算方法
def single_linkage(cluster1, cluster2, distance_matrix):
    return np.min([distance_matrix[i][j] for i in cluster1 for j in cluster2])

def complete_linkage(cluster1, cluster2, distance_matrix):
    return np.max([distance_matrix[i][j] for i in cluster1 for j in cluster2])

def average_linkage(cluster1, cluster2, distance_matrix):
    return np.mean([distance_matrix[i][j] for i in cluster1 for j in cluster2])

def ward_linkage(cluster1, cluster2, data):
    # Calculate the centroids of the clusters
    centroid1 = np.mean(data[cluster1], axis=0)
    centroid2 = np.mean(data[cluster2], axis=0)
    merged_cluster = np.vstack((data[cluster1], data[cluster2]))
    merged_centroid = np.mean(merged_cluster, axis=0)
```

```
    # Compute the sum of squares within each cluster
    ssw_cluster1 = np.sum((data[cluster1] - centroid1) ** 2)
    ssw_cluster2 = np.sum((data[cluster2] - centroid2) ** 2)
    ssw_merged_cluster = np.sum((merged_cluster - merged_centroid) ** 2)

    # The increase in the sum of squares is the criterion for Ward's method
    return ssw_merged_cluster - (ssw_cluster1 + ssw_cluster2)
```

层次聚类算法实现

Code:

```
def hierarchical_clustering(data, linkage, num_clusters):
    clusters = {i: [i] for i in range(len(data))}
    cluster_labels = [-1] * len(data)
    distance_matrix = squareform(pdist(data, 'euclidean'))

    # Create a priority queue with initial distances
    pq = []
    for i in clusters.keys():
        for j in clusters.keys():
            if i < j:
                d = linkage(clusters[i], clusters[j], distance_matrix)
                heapq.heappush(pq, (d, (i, j)))

    while len(clusters) > num_clusters:
        min_distance, (ci, cj) = heapq.heappop(pq)
        if ci not in clusters or cj not in clusters:
            # One or both clusters have been merged already
            continue

        # Merge the two clusters
        merged_cluster = clusters[ci] + clusters[cj]
        del clusters[cj]
        clusters[ci] = merged_cluster

        # Update the distances in the priority queue
        for ck in clusters.keys():
            if ck != ci:
                d = linkage(clusters[ci], clusters[ck], distance_matrix)
                heapq.heappush(pq, (d, (ci, ck)))

    # Assign cluster labels
    for label, cluster in enumerate(clusters.values()):
        for index in cluster:
            cluster_labels[index] = label

    return cluster_labels
```

这段代码是一个函数 `hierarchical_clustering` 的实现，它通过层次聚类算法对数据进行聚类。层次聚类是一种聚类算法，它将数据集中的每个数据点视为单个聚类，并逐步合并最近的聚类直到达到所需的聚类数目。这里是实现的思路：

1. 初始化聚类字典 `clusters`，其中每个数据点是自己的聚类。
2. 初始化一个数组 `cluster_labels` 用于存储每个数据点的聚类标签。

3. 计算数据点之间的距离矩阵 `distance_matrix` 使用欧几里得距离。
4. 创建一个优先队列 `pq`，并以初始距离填充。
5. 继续合并聚类，直到聚类的数量减少到 `num_clusters` 指定的数目。
6. 从优先队列中弹出最小距离的聚类对，检查它们是否已经被合并，如果没有，则合并它们。
7. 更新优先队列中的距离，以反映最新的聚类合并。
8. 为每个数据点分配聚类标签。

指标分析

Code:

```
# Calculate silhouette scores
silhouette_single = calculate_silhouette(iris_dataset, labels_single)
silhouette_complete = calculate_silhouette(iris_dataset, labels_complete)
silhouette_average = calculate_silhouette(iris_dataset, labels_average)
silhouette_ward = calculate_silhouette(iris_dataset, labels_ward)

# Print silhouette scores
print(f"Silhouette Coefficient for single-linkage: {silhouette_single}")
print(f"Silhouette Coefficient for complete-linkage: {silhouette_complete}")
print(f"Silhouette Coefficient for average-linkage: {silhouette_average}")
print(f"Silhouette Coefficient for Ward's method: {silhouette_ward}")

# Find the best linkage method based on silhouette score
best_method = max(
    [
        ('single', silhouette_single),
        ('complete', silhouette_complete),
        ('average', silhouette_average),
        ('ward', silhouette_ward)
    ],
    key=lambda x: x[1]
)

print(f"The best linkage method is: {best_method[0]} with a silhouette score of
{best_method[1]}")
```

计算每个链接方法对应的轮廓系数与执行时间，找到轮廓系数最大的链接方法，进行分析。

可视化分析

Code:

```
# Function to plot dendrogram
def plot_dendrogram(data, method):
    Z = linkage(data, method=method)
    plt.figure(figsize=(25, 10))
    plt.title(f'Dendrogram for {method} linkage')
    dendrogram(Z)
    plt.show()

# Function to plot pairplot
def plot_pairplot(data, labels):
    # Convert the data to a DataFrame for seaborn compatibility
    df = pd.DataFrame(data, columns=[f'feature{i}' for i in
range(data.shape[1])])
```

```
    df['label'] = labels
    sns.pairplot(df, hue='label', palette='bright')
    plt.show()

# Calculate labels for all linkage methods
labels_dict = {
    'single': hierarchical_clustering(iris_dataset, single_linkage,
num_clusters),
    'complete': hierarchical_clustering(iris_dataset, complete_linkage,
num_clusters),
    'average': hierarchical_clustering(iris_dataset, average_linkage,
num_clusters),
    'ward': hierarchical_clustering(iris_dataset, ward_linkage, num_clusters)
}

# Plot dendrogram and pairplot for all linkage methods
for method, labels in labels_dict.items():
    print(f"Plotting {method} linkage dendrogram and pairplot.")
    plot_dendrogram(iris_dataset, method)
    plot_pairplot(iris_dataset, labels)
```

画出了在4种link下的树状图与成对关系图。

Result

时间复杂度和空间复杂度分析

- 时间复杂度：在最坏的情况下，每次迭代合并两个聚类都需要更新优先队列，这可能需要 `O(n^2 log n)` 的时间，其中 `n` 是数据点的数量。这是因为每次合并后都可能需要更新所有聚类之间的距离，这需要 `O(n)` 操作，而在优先队列中进行 `push` 和 `pop` 操作的时间复杂度是 `O(log n)`。因为这种更新在每次迭代中都会发生，所以总的时间复杂度是 `O(n^2 log n)`。
- 空间复杂度：数据点之间距离矩阵 `distance_matrix` 的大小是 `O(n^2)`。优先队列 `pq` 的大小在最坏情况下也可以是 `O(n^2)`，因为它包含所有可能的聚类对的距离。因此，空间复杂度是 `O(n^2)`。

运行时间分析

```
Single-linkage clustering took 0.19545412063598633 seconds
Complete-linkage clustering took 0.19653964042663574 seconds
Average-linkage clustering took 0.26613306999206543 seconds
Ward's method clustering took 2.2349936962127686 seconds
```

可以看出single与complete方法用时较少，average居中，而ward方法时间最长，且明显长于其他三个。（大一个数量级）

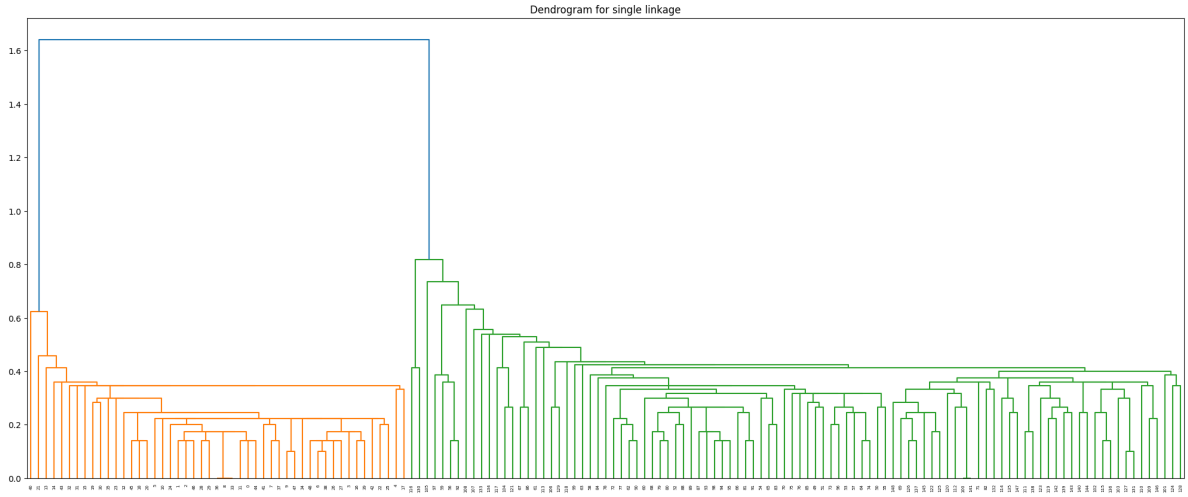
运行结果比较分析

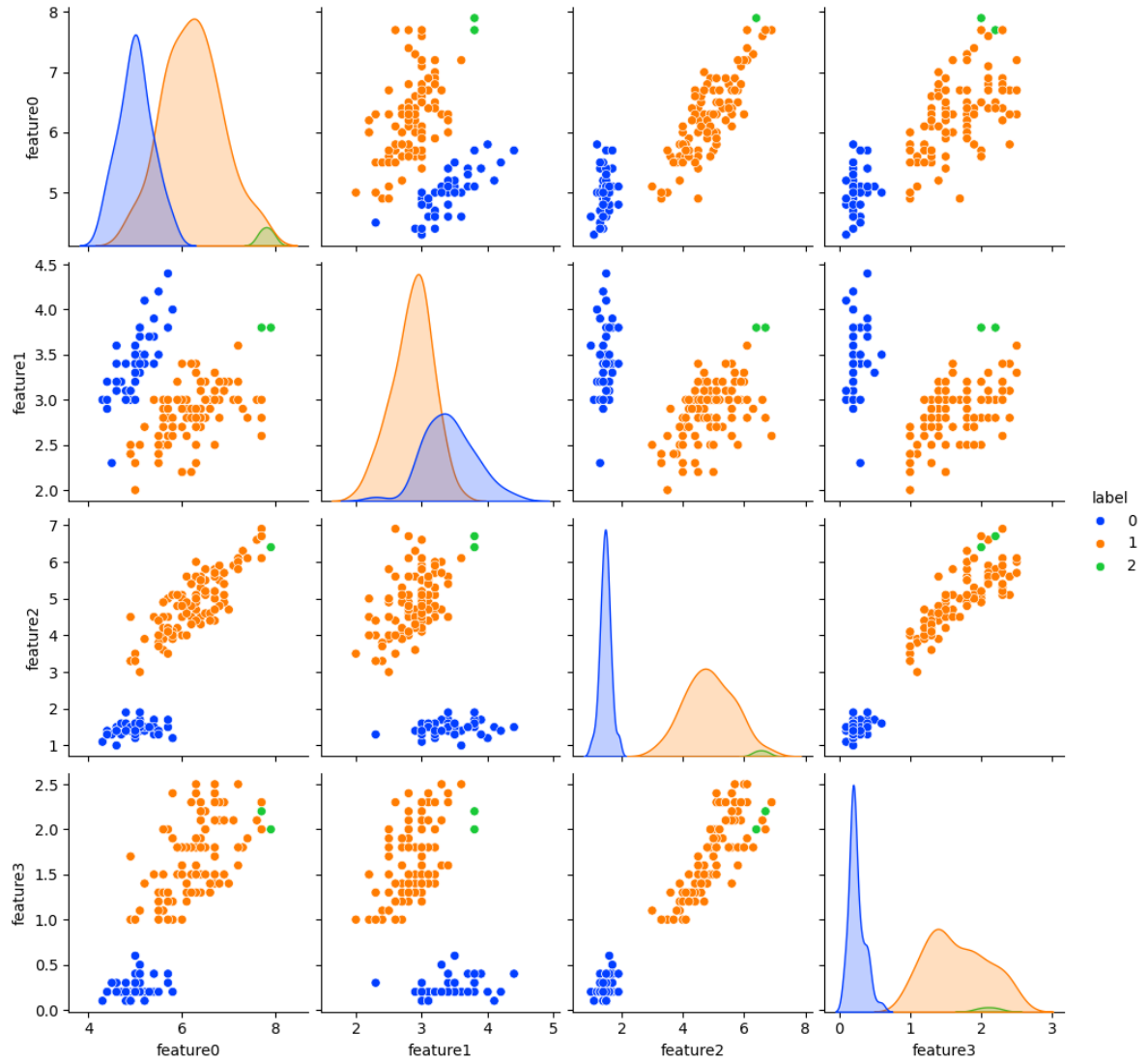
```
Silhouette Coefficient for single-linkage: 0.5090689670612821
Silhouette Coefficient for complete-linkage: 0.48326881977704095
Silhouette Coefficient for average-linkage: 0.551545299987358
Silhouette Coefficient for Ward's method: 0.5093580572587506
The best linkage method is: average with a silhouette score of 0.551545299987358
```

比较轮廓系数可以发现，在iris数据集上，average方法对应的轮廓系数最大，所以average方法为最佳的选择。

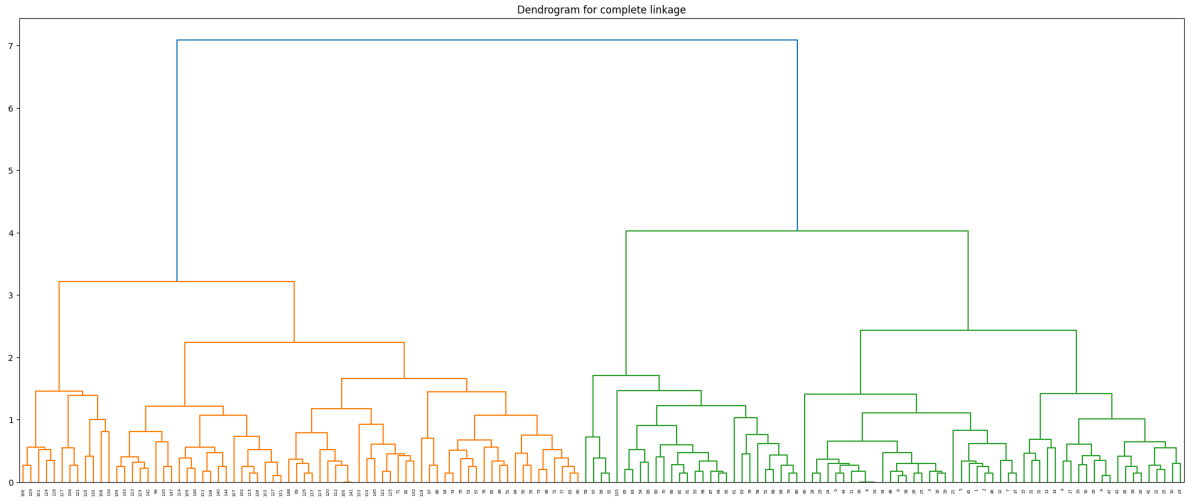
可视化分析

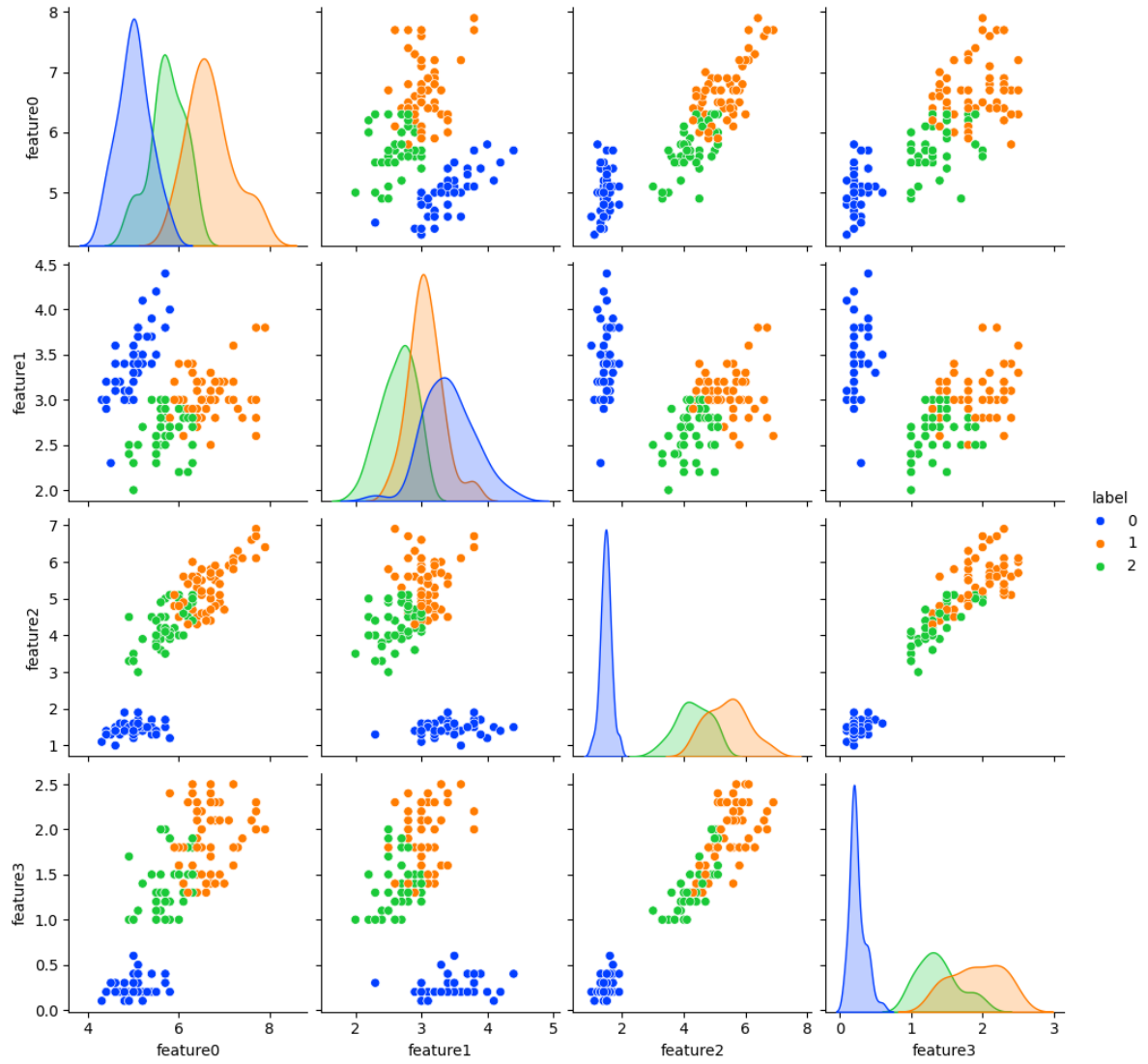
single:



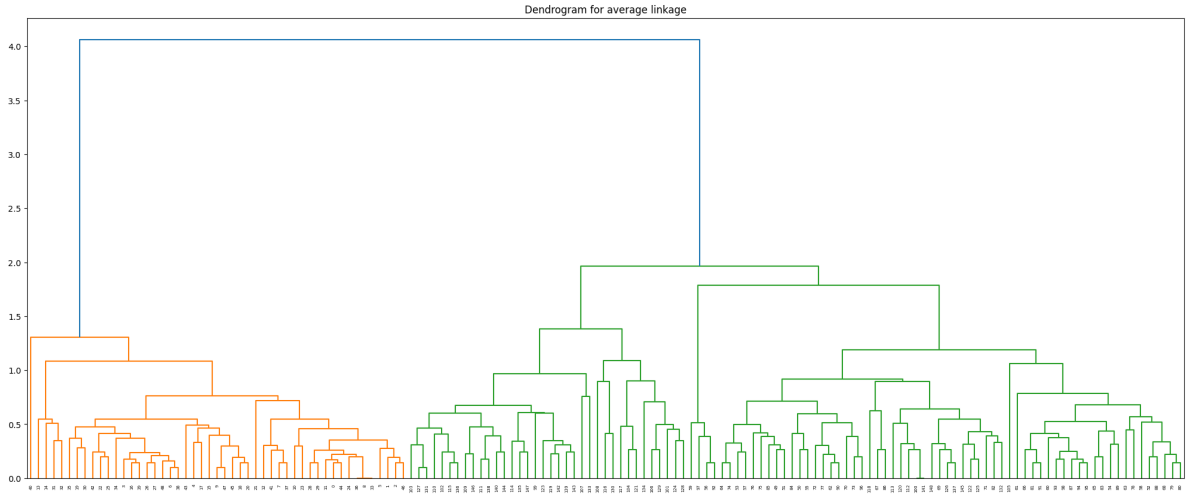


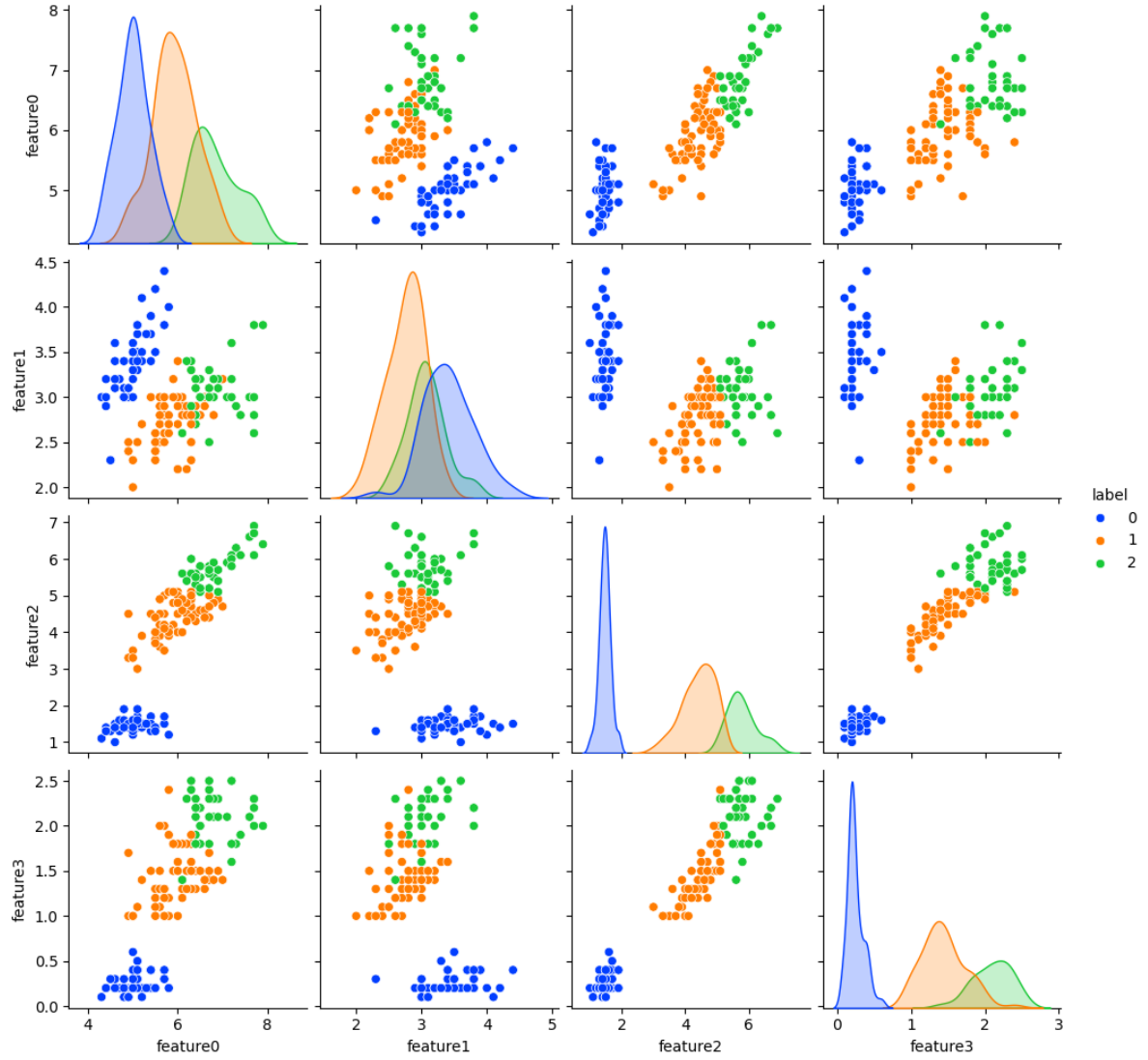
complete:


Dendrogram for complete linkage

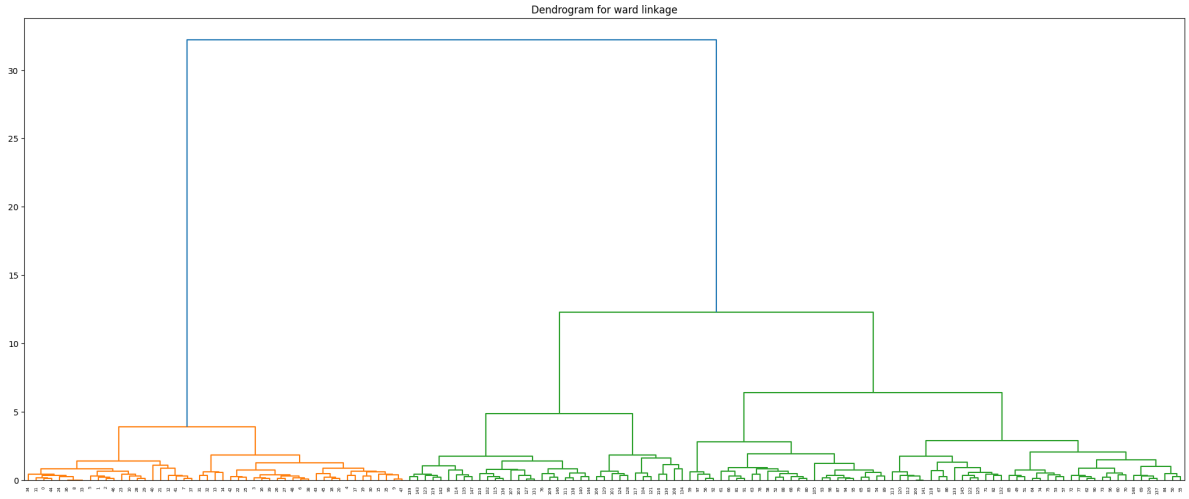

feature0
feature1
feature2
feature3
label
0
1
2

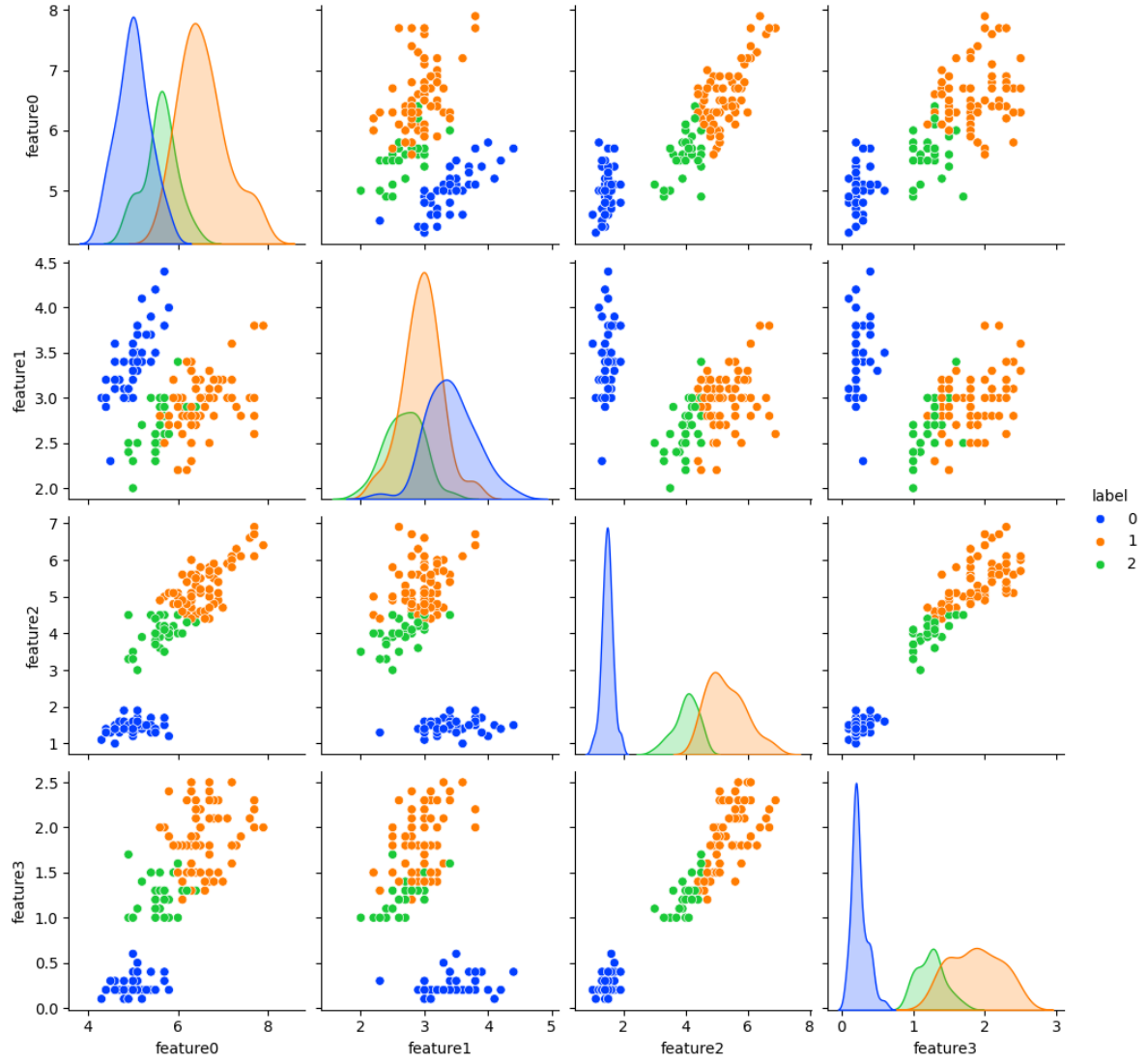
average:


Dendrogram for average linkage


feature0
feature1
feature2
feature3
label
0
1
2

ward:


Dendrogram for ward linkage


feature0
feature1
feature2
feature3
label
0
1
2