Hierarchical Clustering Experiment

Experiment Description

聚类就是对大量未知标注的数据集，按数据的内在相似性将数据集划分为多个类别，使类别内的数据相似度较大而类别间的数据相似度较小。

1. 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。

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

Experiment Preparition

Data Preparition

Firstly, set the randon seed = 42 to make sure the random numbers are the same in every experiment, so that the result can be reproduced. Then use mean1, mean2, mean3 to define three mean value of three three-dimentional normal distribution: [1, 1, 1] , [4, 4, 4]and [8, 1, 1] respectively. Finally difine a 3*3 covariance matrix conv1[[1, 0, 0], [0, 1, 0], [0, 0, 1]].

```
mean1, mean2, mean3 = [1, 1, 1], [4, 4, 4], [8, 1, 1]
cov1 = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]

data1 = np.random.multivariate_normal(mean1, cov1, 667)
data2 = np.random.multivariate_normal(mean2, cov1, 667)
data3 = np.random.multivariate_normal(mean3, cov1, 666)
X = np.vstack((data1, data2, data3))
labels = np.concatenate((np.full(667, 1), np.full(667, 2), np.full(666, 3)))

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=labels, cmap='rainbow', s=5)
plt.show()
```

Define Class Linkage

```
class Linkage:
    def __init__(self, n_clusters=2, linkage_name= 'single_linkage'):
        self.n_clusters = n_clusters
        self.labels_ = None
        self.linkage_matrix_ = None
        self.linkage_name_ = linkage_name
```

Implement single-linkage, complete-linkage, average-linkage hierarchical clustering.

- Single Linkage: calculate the minimum distance between samples in cluster A and samples in cluster B.

$$D_{pq} = min\{d_{ij} \mid x_i \in G_p, x_j \in G_q\}$$

```
    def calculate_single_linkage_min_distance(c1, c2, clusters, distances):
        min_dist = np.inf
        to_merge = (None, None)
        for p1 in clusters[c1]:
            for p2 in clusters[c2]:
                dist = distances[p1, p2]
                if dist < min_dist:
                    min_dist = dist
                    to_merge = (c1, c2)
        return min_dist, to_merge
```

- Complete Linkage: calculate the maximum distance between samples in cluster A and samples in cluster B.

$$D_{pq} = max\{d_{ij} \mid x_i \in G_p, x_j \in G_q\}$$

```
def calculate_complete_linkage_max_distance(c1, c2, clusters, distances):
        max_dist = -np.inf
        to_merge = (None, None)
        for p1 in clusters[c1]:
            for p2 in clusters[c2]:
                dist = distances[p1, p2]
                if dist > max_dist:
                    max_dist = dist
                    to_merge = (c1, c2)
        return max_dist, to_merge
```

- Average Linkage: calculate the average distance of arbitrary samples in cluster A and samples in cluster B.

$$D_{pq} = \frac{1}{n_p n_q} \sum_{x_i \in G_p} \sum_{x_j \in G_q} d_{ij}$$

```
def calculate_complete_linkage_max_distance(c1, c2, clusters, distances):
        max_dist = -np.inf
        to_merge = (None, None)
        for p1 in clusters[c1]:
            for p2 in clusters[c2]:
                dist = distances[p1, p2]
                if dist > max_dist:
                    max_dist = dist
                    to_merge = (c1, c2)
        return max_dist, to_merge
```

Clustering

1. Calculate the distance matrix of all samples: Calculating the Euclidean distance of every couple of samples to calculate the distance matrix of all samples.
2. Find the most close clusters: traverse to find the most close clusters, use branch structure to call single-linkage, complete-linkage and average-linkage.
3. Merge the nearest two clusters: merge the second cluster into the first cluster and delete the second cluster.
4. Assign labels to samples: assign labels to facilitate visualization

```
def fit(self, X):
        n_samples = X.shape[0]

        # Calculate the distance matrix of all samples
        distances = np.sqrt(np.sum((X[:, np.newaxis, :] - X[np.newaxis, :, :]) ** 2,
axis=2))
        clusters = {i: [i] for i in range(n_samples)}
        linkage_matrix = []

        while len(clusters) > self.n_clusters:
            min_dist = np.inf
            to_merge = (None, None)

            # Find the most close clusters
            for i in range(len(clusters)):
                for j in range(i + 1, len(clusters)):
                    c1, c2 = list(clusters.keys())[i], list(clusters.keys())[j]
                    if self.linkage_name_ == 'single_linkage':
                        dist, merge_pair = self.calculate_single_linkage_min_distance(c1,
c2, clusters, distances)
                    elif self.linkage_name_ == 'complete_linkage':
                        dist, merge_pair =
self.calculate_complete_linkage_max_distance(c1, c2, clusters, distances)
                    else:
                        dist, merge_pair = self.calculate_average_linkage_distance(c1, c2,
clusters, distances)

                    if dist < min_dist:
```

```
                        min_dist = dist
                        to_merge = merge_pair

            # Merge the nearest two clusters
            c1, c2 = to_merge
            clusters[c1].extend(clusters[c2])
            del clusters[c2]

            # print merge process
            linkage_matrix.append([c1, c2, min_dist, len(clusters[c1])])
            print(linkage_matrix[-1])

        self.linkage_matrix_ = np.array(linkage_matrix)

        # Assign labels to samples
        self.labels_ = np.zeros(n_samples)
        for cluster_id, points in enumerate(clusters.values()):
            for point in points:
                self.labels_[point] = cluster_id

        return self
```

Implement classes and visualize results

Implement single-linkage, complete-linkage and average-linkage respectively. Plot the results as scatter plots.

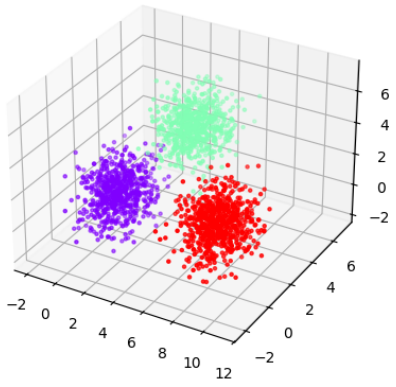
```
linkage_models = {
    'single_linkage': Linkage(n_clusters=3, linkage_name='single_linkage'),
    'complete_linkage': Linkage(n_clusters=3, linkage_name='complete_linkage'),
    'averaged_linkage': Linkage(n_clusters=3, linkage_name='averaged_linkage')
}

for linkage_name, model in linkage_models.items():
    model.fit(X)
    labels = model.labels_
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=labels, cmap='rainbow', s=5)
    plt.title(f"{linkage_name.capitalize()} Clustering")
    plt.show()
```

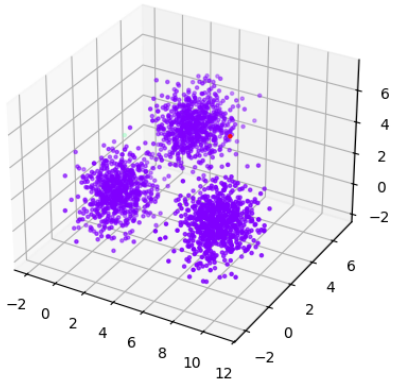
Experiment results and analysis

The experiment results are plotted below:

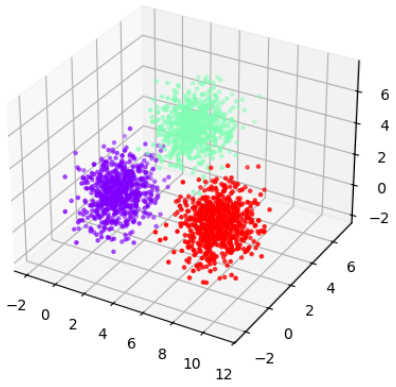
1. Original clusters and single-linkage clusters

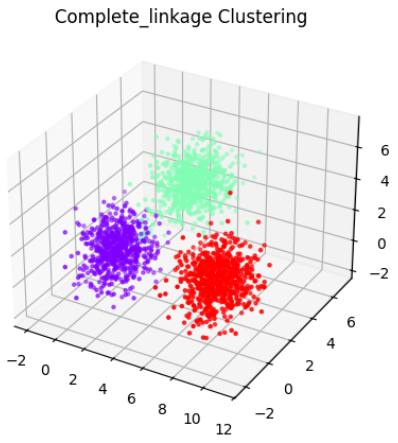


Single_linkage Clustering

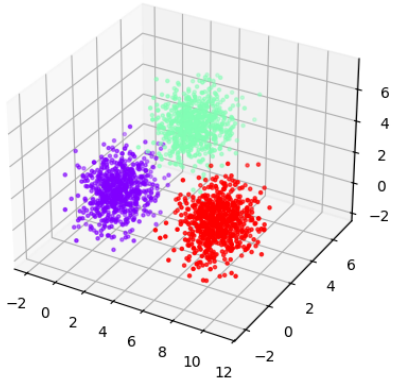


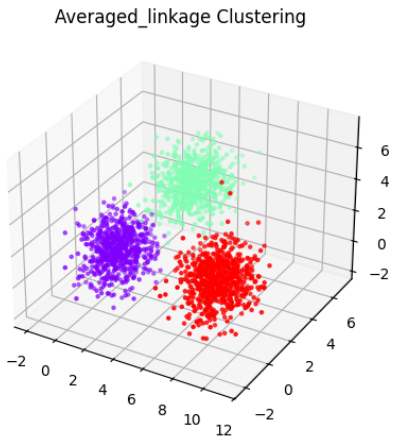
2. Original clusters and complete-linkage clusters





3. Original clusters and average-linkage clusters





As can be seen from the experimental results, the accuracy of single-linkage clustering is low -- about 2/3 samples are incorrectly classified into the red cluster. The accuracy of complete-linkage and average-linkage is higher. In complete-linkage clustering, some green samples are falsely classified into purple cluster, and a few red samples classified into green cluster. The mistakenly classified samples are few in average-linkage clustering. By analyzing the three clustering algorithm, it can be concluded that:

- Single-linkage: This method calculates the distance of the most close samples in two clusters. In other words, the distance of 2 clusters is determined by the most close samples. The method can easily lead to clusters to 'absorb' each other and have a bad classification result.

- Complete-linkage: This method tend to generate compact spherical clusters. However, during the process, two clusters can be difficult to combine because the long distance of the extreme values. Thus, the peripheral samples of the two different clusters can be poorly classified.
- Average-linkage: This method calculates the average distance of all samples in two clusters and has a balanced result.