110612117 張仲瑜 HW3 report

I. Introduction

1. Goal

In this task, we aim to try five 5 types of clustering methods and observe the internal and external evaluation under different settings.

2. Core Idea

I chose K-means, DBSCAN, Agglomerative Clustering, Spectral Clustering and Gaussian Mixture Clustering and observed the Silhouette Score and Adjusted Rand Index under different settings.

3. Dataset

- ✧ Abalone dataset (3 classes with 8 features, #4177)
- ✧ Using other 7 features to predict rings of abalone (either as a continuous value or as a classification problem.)

II. Method

1. Load data and preprocessing

Split the ring into 5 categories such that the number of samples in each category is roughly the same.

2. K-means [1][6]

Observe the effect of cluster number and try different initialization methods.

3. DBSCAN [2][7]

Observe the effect of eps that determines the maximum distance between two points to be considered as neighbors, and the min samples required to form a dense region.

4. Agglomerative Clustering [3]

Observe the effect of different linkage methods and different number of clusters.

5. Mean-shift Clustering [4]

   Observe the effect of different bandwidth.

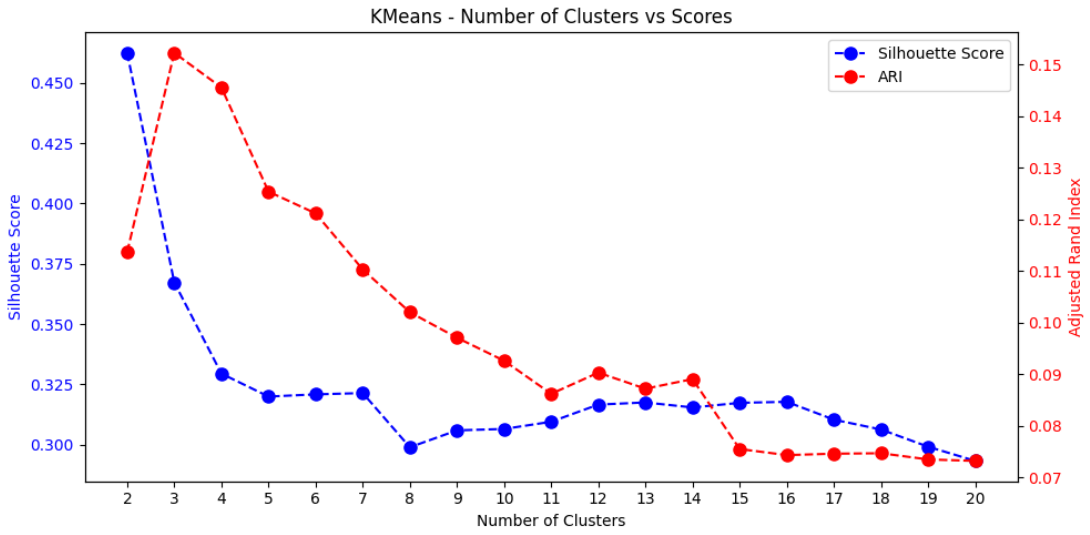
6. Gaussian Mixture Model [5]

   Observe the effect of different number of components and different covariance types.

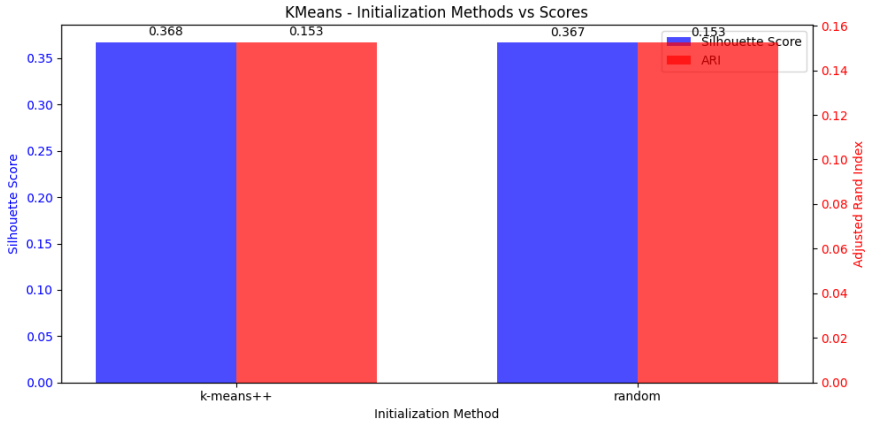
III. Experiment Result and Observation

Note: In all of the settings, the external evaluation shows that the result is only slightly better than random guessing, but I still want to focus on the difference between different settings though the correlation is week from the beginning.

1. K-means
   - ✧ Cluster Number Experiment:

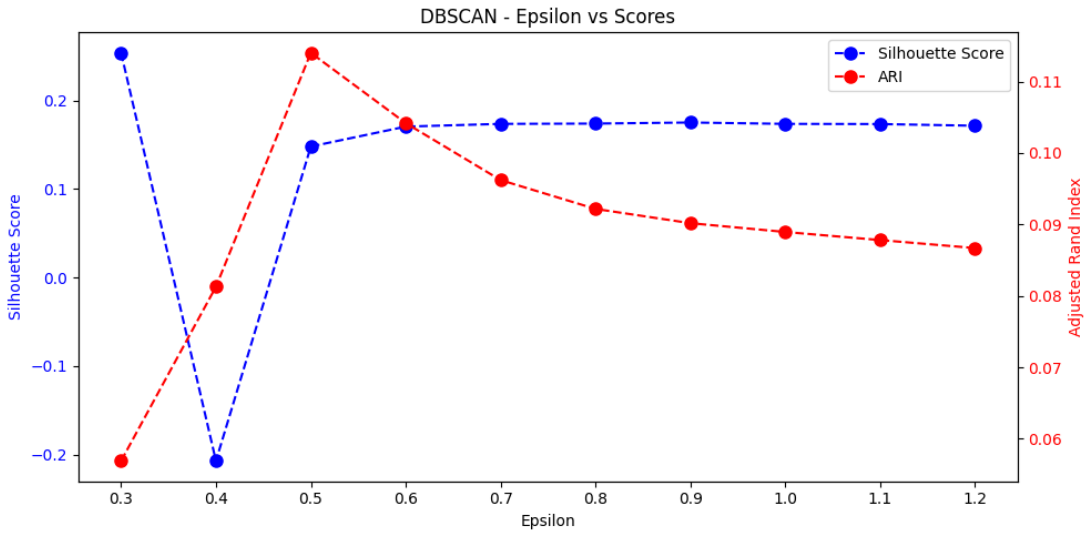


   - ✧ Initialization Methods Experiment:

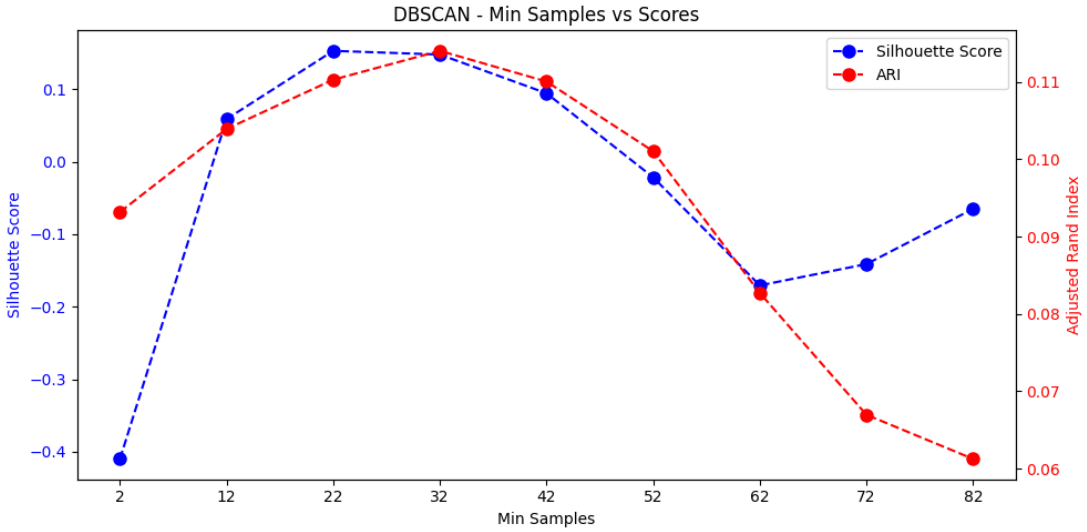


- ✧ Observation:
  - ➢ The silhouette score indicates that k=2 exhibits better intra-cluster cohesion and inter-cluster separation, while k=3 maximizes the Adjusted Rand Index (ARI), suggesting optimal alignment with the true age-based labels.
  - ➢ Although K-means is theoretically sensitive to initialization points, in this experiment, both k-means++ and random initialization methods produce nearly identical results, with both achieving a silhouette score of approximately 0.368 and an ARI of 0.15 at k=3.
  - ➢ The ARI evaluation (maximum of 0.15) indicates that the clustering results are only slightly better than random assignments, suggesting a weak correlation between the physical features (length, diameter, weight, etc.) and the age-based grouping of abalones.

2. DBSCAN
   - ✧ Radius Value Experiment (fixed lower bound = 32):



✧ Lower-bound Samples Experiment (eps = 0.5):

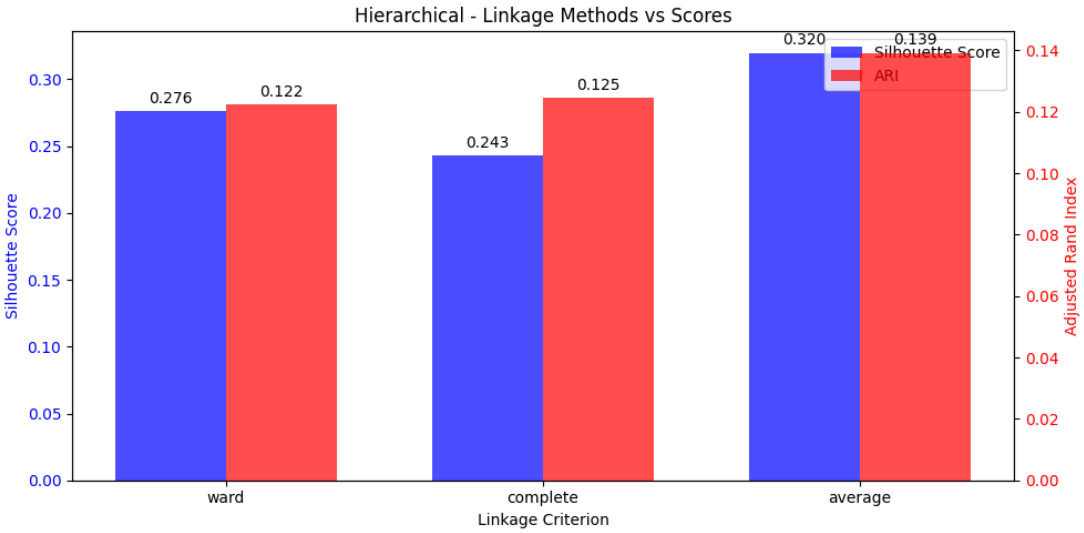


✧ Observation:

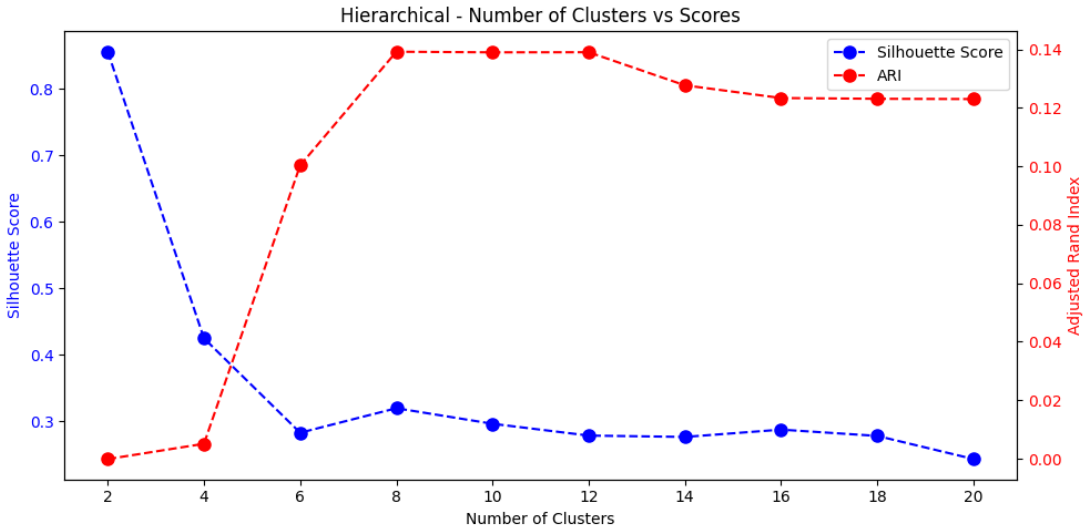
- The Lower-bound Samples parameter analysis shows that when Lower-bound Samples is set between 22-32, it achieves optimal performance with both the highest Silhouette score (approximately 0.15) and ARI score (approximately 0.11), indicating that this range provides the best balance between intra-cluster cohesion and alignment with true age-based labels.
- The Epsilon (radius) parameter experiments reveal that while epsilon=0.3 yields the highest Silhouette score (around 0.25), it results in a lower ARI score (about 0.055); conversely, epsilon=0.5 maximizes the ARI (approximately 0.115), demonstrating a clear trade-off between cluster quality and label alignment when selecting this parameter.
- Both parameter experiments indicate relatively modest performance of DBSCAN on this dataset (maximum ARI of about 0.115 and highest Silhouette score of about 0.25), suggesting that the physical features of abalones may not exhibit clear density-based patterns in

relation to age groupings, making it challenging for density-based clustering to discover ideal cluster structures.

3. Agglomerative Clustering
   - ✧ Linkage Methods Experiment (Fixed cluster num = 8):



   - ✧ Cluster Number Experiment (Fixed linkage = average):

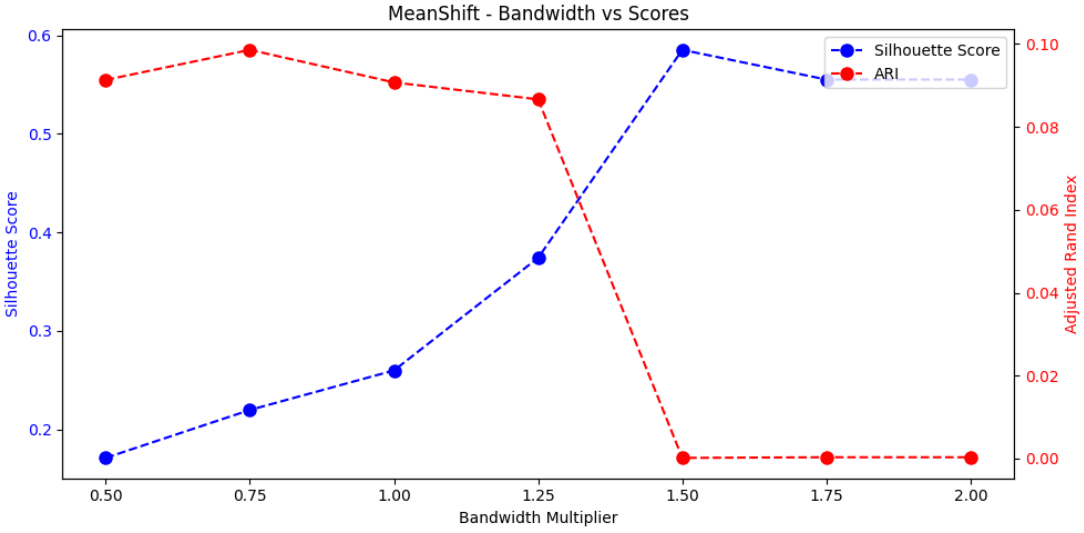


   - ✧ Observation:
     - ➢ Among different linkage methods, average linkage demonstrates the best overall performance (Silhouette Score = 0.320, ARI = 0.139). Given that average linkage represents a middle ground between ward's variance

minimization and complete's maximum distance approach, it appears to be most suitable for the abalone dataset's natural distribution patterns.

- The number of clusters (k) experiment shows a striking contrast: k=2 achieves the highest Silhouette Score (around 0.85) but with a very low ARI (near 0), while k=8 reaches the peak ARI (about 0.14) but with a reduced Silhouette Score (about 0.32). This significant trade-off suggests:
  - From a data structure perspective, the abalone dataset might naturally separate into two main groups.
  - From a biological age classification standpoint, a finer granularity (k=8) is needed to better match the actual age distribution

4. MeanShift Clustering
   - Bandwidth Experiment:

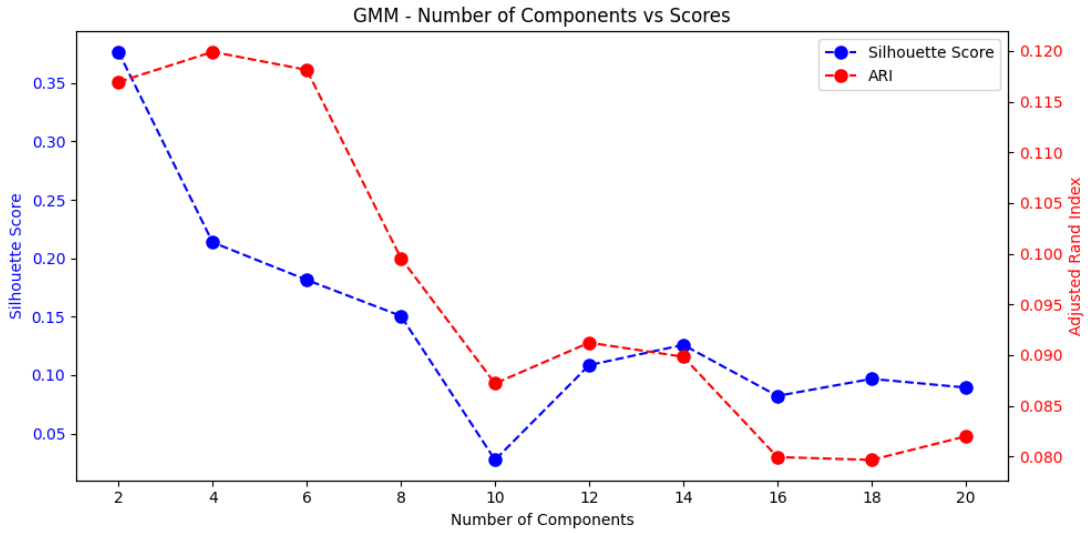


   - Observation:
     - The Silhouette shows a consistent increasing trend as the bandwidth multiplier increases (from 0.17 at 0.5x to 0.58 at 1.5x), suggesting that larger bandwidths create more cohesive and well-separated clusters,

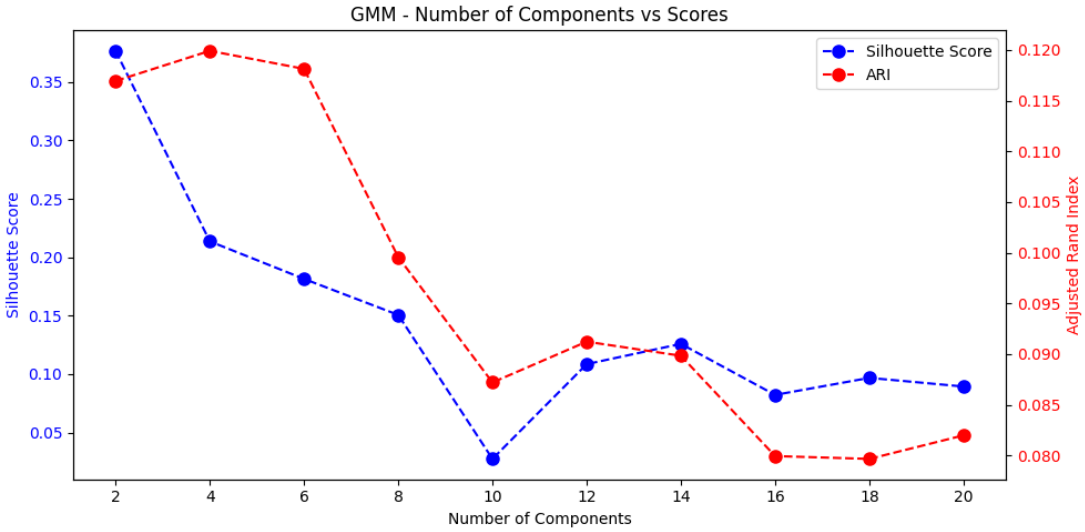
likely due to the algorithm's tendency to merge nearby data points into larger clusters.

- The ARI score exhibits an inverse relationship with bandwidth, peaking at 0.095 with a 0.75x multiplier and dropping significantly after 1.25x multiplier. This indicates that while larger bandwidths may create more distinct clusters, they fail to capture the natural age-based groupings in the abalone dataset.
- There's a clear trade-off point around the 1.25-1.5x bandwidth multiplier range, where the Silhouette Score continues to improve but the ARI drops dramatically (near 0). This suggests that excessive bandwidth values oversimplify the clustering structure, merging distinct age groups into too few clusters despite creating mathematically "better" cluster separation.

5. Gaussian Mixture Model
   - ✧ Component Number Experiment:



✧ Covariance Type Experiment:

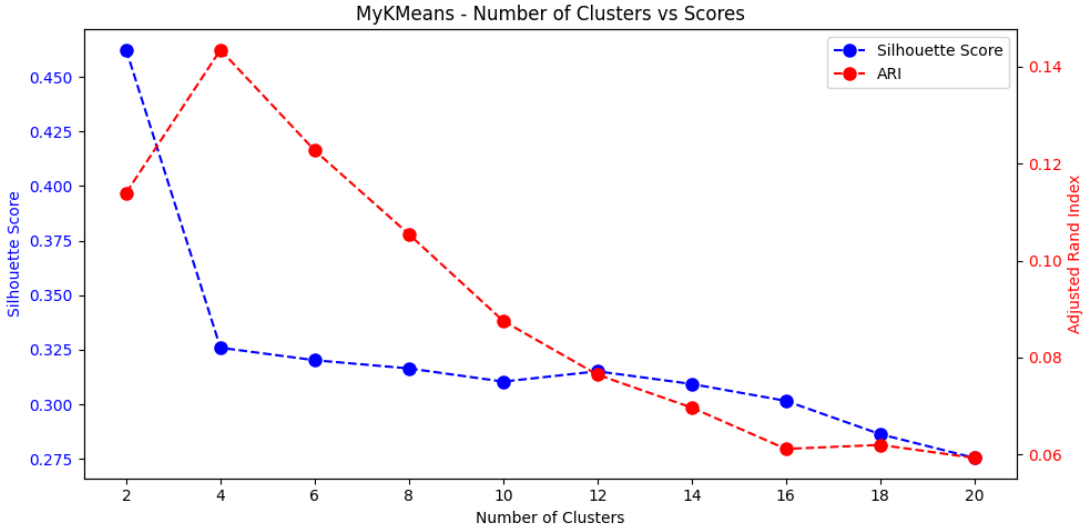


✧ Observation:

- ➢ The number of components experiment shows that simpler models (2-4 components) achieve better performance, with both metrics peaking at low component numbers (Silhouette Score ≈ 0.37 at k=2, ARI ≈ 0.12 at k=4). This suggests that the abalone dataset's underlying distribution might be better represented by fewer, well-defined Gaussian distributions.
- ➢ The covariance type comparison reveals that spherical covariance outperforms full covariance in both metrics (Silhouette: 0.327 vs 0.214, ARI: 0.145 vs 0.120). This indicates that the clusters in the abalone dataset might have similar variances in all directions, and the simpler spherical model is sufficient to capture the data's structure.
- ➢ As the number of components increases beyond 8, both metrics show significant deterioration and instability, particularly in the Silhouette Score which drops to around 0.1. This degradation suggests that

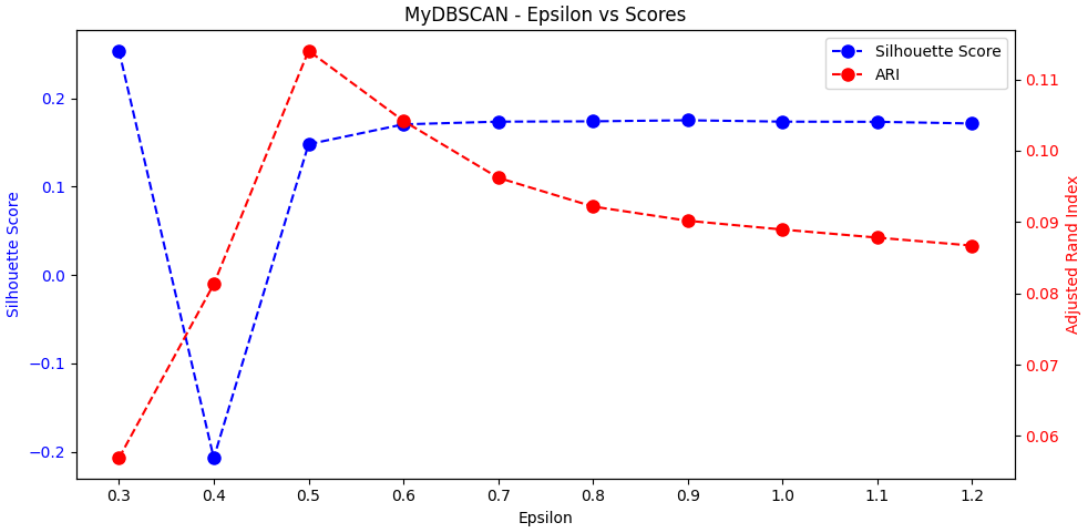
using too many components leads to overfitting and creates artificial divisions in the data that don't reflect natural groupings.

6. My own implementation of K-means
   - ✧ Cluster Number Experiment:



   - ✧ Difference between my Implementation and Sklearn:
     - ➢ Initialization: Our implementation only uses random initialization, while sklearn provides advanced methods like k-means++ initialization which leads to better initial centroids placement
     - ➢ Multiple Runs: Sklearn performs multiple runs (n_init=10 by default) and selects the best result, while our implementation only runs once, potentially missing better solutions.

7. My own implementation of DBSCAN
   - ✧ Experiment: Radius Value Experiment (fixed lower bound = 32)



- ✧ Difference between my Implementation and Sklearn:
  - ➢ Neighbor Search Efficiency:
    - ✧ My implementation uses brute-force distance calculation for finding neighbors, computing distances between each point and all other points
    - ✧ Sklearn uses spatial indexing structures (like KD-trees or Ball-trees) to efficiently find neighbors, significantly reducing computation time for large datasets

IV. Appendix

1. Implements K-means with sklearn package to conduct experiment

```python
def experiment_kmeans(X, true_labels):
    """Experiment with KMeans parameters"""
    print("\nKMeans Experiments:")
    print("-" * 50)

    # Experiment 1: Different number of clusters
    n_clusters_range = range(2, 21)
    silhouette_scores = []
    ari_scores = []

    for n_clusters in n_clusters_range:
        kmeans = KMeans(n_clusters=n_clusters, random_state=42)
        labels = kmeans.fit_predict(X)
        silhouette = silhouette_score(X, labels)
        ari = adjusted_rand_score(true_labels, labels)
        silhouette_scores.append(silhouette)
        ari_scores.append(ari)
        print(f"n_clusters={n_clusters}:")
        print(f"Silhouette Score: {silhouette:.3f}")
        print(f"Adjusted Rand Index: {ari:.3f}")
        print("-" * 30)

    plot_scores(list(n_clusters_range), silhouette_scores, ari_scores,
                'Number of Clusters', 'KMeans - Number of Clusters vs Scores')

# Experiment 2: Different initialization methods
init_methods = ['k-means++', 'random']
n_clusters = 3  # 使用一個固定的群集數
silhouette_scores = []
ari_scores = []

for init in init_methods:
    kmeans = KMeans(n_clusters=n_clusters, init=init, random_state=int(time.time()))
    labels = kmeans.fit_predict(X)
    silhouette = silhouette_score(X, labels)
    ari = adjusted_rand_score(true_labels, labels)
    silhouette_scores.append(silhouette)
    ari_scores.append(ari)
    print(f"initialization={init}:")
    print(f"Silhouette Score: {silhouette:.3f}")
    print(f"Adjusted Rand Index: {ari:.3f}")
    print("-" * 30)

plot_categorical_scores(init_methods, silhouette_scores, ari_scores,
                        'Initialization Method', 'KMeans - Initialization Methods vs Scores')
```

2. Implements DBSCAN with sklearn package to conduct experiments.

```python
def experiment_dbscan(X, true_labels):
    """Experiment with DBSCAN parameters"""
    print("\nDBSCAN Experiments:")
    print("-" * 50)

    # Experiment 1: Different eps values
    eps_range = np.linspace(0.1, 2.0, 20)
    silhouette_scores = []
    ari_scores = []
    valid_eps = []

    for eps in eps_range:
        dbscan = DBSCAN(eps=eps, min_samples=32)
        labels = dbscan.fit_predict(X)
        n_clusters = len(set(labels)) - (1 if -1 in labels else 0)

        if n_clusters > 1:  # 確保至少有兩個群集
            silhouette = silhouette_score(X, labels)
            ari = adjusted_rand_score(true_labels, labels)
            silhouette_scores.append(silhouette)
            ari_scores.append(ari)
            valid_eps.append(eps)
            print(f"eps={eps:.2f}:")
            print(f"Number of clusters: {n_clusters}")
            print(f"Silhouette Score: {silhouette:.3f}")
            print(f"Adjusted Rand Index: {ari:.3f}")
            print("-" * 30)

    if valid_eps:
        plot_scores(valid_eps, silhouette_scores, ari_scores,
                    'Epsilon', 'DBSCAN - Epsilon vs Scores')
```

```python
def experiment_dbscan(X, true_labels):
    """Experiment with DBSCAN parameters"""
    print("\nDBSCAN Experiments:")
    print("-" * 50)

    # Experiment 1: Different eps values
    eps_range = np.linspace(0.1, 2.0, 20)
    silhouette_scores = []
    ari_scores = []
    valid_eps = []

    for eps in eps_range:
        dbscan = DBSCAN(eps=eps, min_samples=32)
        labels = dbscan.fit_predict(X)
        n_clusters = len(set(labels)) - (1 if -1 in labels else 0)

        if n_clusters > 1:  # 確保至少有兩個群集
            silhouette = silhouette_score(X, labels)
            ari = adjusted_rand_score(true_labels, labels)
            silhouette_scores.append(silhouette)
            ari_scores.append(ari)
            valid_eps.append(eps)
            print(f"eps={eps:.2f}:")
            print(f"Number of clusters: {n_clusters}")
            print(f"Silhouette Score: {silhouette:.3f}")
            print(f"Adjusted Rand Index: {ari:.3f}")
            print("-" * 30)

    if valid_eps:
        plot_scores(valid_eps, silhouette_scores, ari_scores,
                    'Epsilon', 'DBSCAN - Epsilon vs Scores')
```

3. Implements Agglomerative Clustering with sklearn package to conduct experiments

```python
    # Experiment 1: Different linkage criteria
    linkages = ['ward', 'complete', 'average']
    n_clusters = 8  # 固定群集數
    silhouette_scores = []
    ari_scores = []

    for linkage in linkages:
        hierarchical = AgglomerativeClustering(n_clusters=n_clusters, linkage=linkage)
        labels = hierarchical.fit_predict(X)
        silhouette = silhouette_score(X, labels)
        ari = adjusted_rand_score(true_labels, labels)
        silhouette_scores.append(silhouette)
        ari_scores.append(ari)
        print(f"linkage={linkage}:")
        print(f"Silhouette Score: {silhouette:.3f}")
        print(f"Adjusted Rand Index: {ari:.3f}")
        print("-" * 30)

    plot_categorical_scores(linkages, silhouette_scores, ari_scores,
                            'Linkage Criterion', 'Hierarchical - Linkage Methods vs Scores')
```

```python
    # Experiment 2: Different number of clusters
    n_clusters_list = range(2, 21, 2)
    silhouette_scores = []
    ari_scores = []

    for n_clusters in n_clusters_list:
        hierarchical = AgglomerativeClustering(n_clusters=n_clusters, linkage='average')
        labels = hierarchical.fit_predict(X)
        silhouette = silhouette_score(X, labels)
        ari = adjusted_rand_score(true_labels, labels)
        silhouette_scores.append(silhouette)
        ari_scores.append(ari)
        print(f"n_clusters={n_clusters}:")
        print(f"Silhouette Score: {silhouette:.3f}")
        print(f"Adjusted Rand Index: {ari:.3f}")
        print("-" * 30)

    plot_scores(n_clusters_list, silhouette_scores, ari_scores,
                'Number of Clusters', 'Hierarchical - Number of Clusters vs Scores')
```

4. Implements meanshift clustering with sklearn package to conduct experiments

```python
def experiment_meanshift(X, true_labels):
    """Experiment with Mean Shift Clustering parameters"""
    print("\nMean Shift Experiments:")
    print("-" * 50)

    # Experiment with different bandwidth values
    # 先用estimate_bandwidth得到一個基準值
    bandwidth_base = estimate_bandwidth(X, quantile=0.2)
    print(f"Bandwidth base: {bandwidth_base}")

    # 測試不同的bandwidth倍數
    bandwidth_multipliers = [0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0]
    silhouette_scores = []
    ari_scores = []

    for mult in bandwidth_multipliers:
        bandwidth = bandwidth_base * mult
        meanshift = MeanShift(bandwidth=bandwidth)
        labels = meanshift.fit_predict(X)

        # 確保至少有兩個群集
        n_clusters = len(set(labels))
        if n_clusters > 1:
            silhouette = silhouette_score(X, labels)
            ari = adjusted_rand_score(true_labels, labels)
            silhouette_scores.append(silhouette)
            ari_scores.append(ari)
            print(f"bandwidth_multiplier={mult:.2f}:")
            print(f"Number of clusters: {n_clusters}")
            print(f"Silhouette Score: {silhouette:.3f}")
            print(f"Adjusted Rand Index: {ari:.3f}")
            print("-" * 30)

    plot_scores(bandwidth_multipliers, silhouette_scores, ari_scores,
                'Bandwidth Multiplier', 'MeanShift - Bandwidth vs Scores')
```

5. Implements Gaussian Mixture Model clustering with sklearn package to conduct experiments

```python
def experiment_gmm(X, true_labels):
    """Experiment with Gaussian Mixture Model parameters"""
    print("\nGaussian Mixture Model Experiments:")
    print("-" * 50)

    # Experiment 1: Different number of components
    n_components_list = [5, 10, 15]
    silhouette_scores = []
    ari_scores = []

    for n_components in n_components_list:
        gmm = GaussianMixture(n_components=n_components, random_state=42)
        labels = gmm.fit_predict(X)
        silhouette = silhouette_score(X, labels)
        ari = adjusted_rand_score(true_labels, labels)
        silhouette_scores.append(silhouette)
        ari_scores.append(ari)
        print(f"n_components={n_components}:")
        print(f"Silhouette Score: {silhouette:.3f}")
        print(f"Adjusted Rand Index: {ari:.3f}")
        print("-" * 30)

    plot_scores(n_components_list, silhouette_scores, ari_scores,
                'Number of Components', 'GMM - Number of Components vs Scores')

    # Experiment 2: Different covariance types
    covariance_types = ['full', 'tied', 'diag', 'spherical']
    n_components = 10  # 固定組件數
    silhouette_scores = []
    ari_scores = []

    for covariance_type in covariance_types:
        gmm = GaussianMixture(n_components=n_components, covariance_type=covariance_type,
        labels = gmm.fit_predict(X)
        silhouette = silhouette_score(X, labels)
        ari = adjusted_rand_score(true_labels, labels)
        silhouette_scores.append(silhouette)
        ari_scores.append(ari)
        print(f"covariance_type={covariance_type}:")
        print(f"Silhouette Score: {silhouette:.3f}")
        print(f"Adjusted Rand Index: {ari:.3f}")
        print("-" * 30)

    plot_scores(covariance_types, silhouette_scores, ari_scores,
                'Covariance Type', 'GMM - Covariance Types vs Scores')
```

6. My own implementation of K-means (key parts):

```python
def fit(self, X):
    if self.random_state is not None:
        np.random.seed(self.random_state)

    # Initialize centroids randomly
    n_samples, n_features = X.shape
    idx = np.random.choice(n_samples, self.n_clusters, replace=False)
    self.centroids = X[idx].copy()

    for _ in range(self.max_iters):
        # Assign samples to nearest centroid
        distances = np.sqrt(((X - self.centroids[:, np.newaxis])**2).sum(axis=2))
        self.labels_ = np.argmin(distances, axis=0)

        # Update centroids
        new_centroids = np.array([X[self.labels_ == k].mean(axis=0)
                                  for k in range(self.n_clusters)])

        # Check convergence
        if np.all(np.abs(new_centroids - self.centroids) < 1e-6):
            break

        self.centroids = new_centroids

    return self
```

7. My own implementation of DBSCAN (key parts):

```python
for point_idx in range(n_samples):
    if self.labels_[point_idx] != -1:  # 跳過已經被訪問的點
        continue

    neighbors = self._get_neighbors(X, point_idx)

    if len(neighbors) < self.min_samples:  # 不是核心點
        continue

    # 發現新的群集
    self.labels_[point_idx] = cluster_label

    # 擴展群集
    seed_points = neighbors.tolist()
    while seed_points:
        current_point = seed_points.pop()
        if self.labels_[current_point] == -1:  # 未訪問的點
            self.labels_[current_point] = cluster_label
            current_neighbors = self._get_neighbors(X, current_point)
            if len(current_neighbors) >= self.min_samples:
                seed_points.extend([p for p in current_neighbors if self.labels_[p] == -1])

    cluster_label += 1
```

Thank you for reading, hope you have a nice day. :)