Third Home Assignment - Data Mining - Group 2

May 15, 2023

Ana Silva nº 39074, 15 hours; **Cláudia Afonso** nº 36273, 15 hours; **Martim Silva** nº 51304, 15 hours; **Rita Rodrigues** nº 54859 15 hours

---

[1]:
```
import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN, OPTICS
from sklearn.metrics import silhouette_score, silhouette_samples
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.cluster import calinski_harabasz_score, homogeneity_score, completeness_score,␣
↪v_measure_score
import time
```

0.1 Objective 1 - Building Clustering Models

0.1.1 Determining the best clustering approach to classify the "train" dataset

Pre-Processing of the Data

The dataset's features are numerical with different ranges. With this in mind, the data was scaled using StandardScaler from Scikit-learn. The StandardScaler scales the features in such a way that they have zero mean and unit variance along each dimension, thus ensuring that each feature is on a similar scale. Since clustering algorithms often rely on distance measures between data points, scaling is important to ensure that each feature contributes equally to the distance calculations and prevent dominance by any particular feature.

[3]:
```
train = pd.read_csv("train.csv")
temp = train["critical_temp"]
train = train.drop(columns = ['critical_temp'])
```

[4]:
```
scaler = StandardScaler()
scaled_train = scaler.fit_transform(train)
```

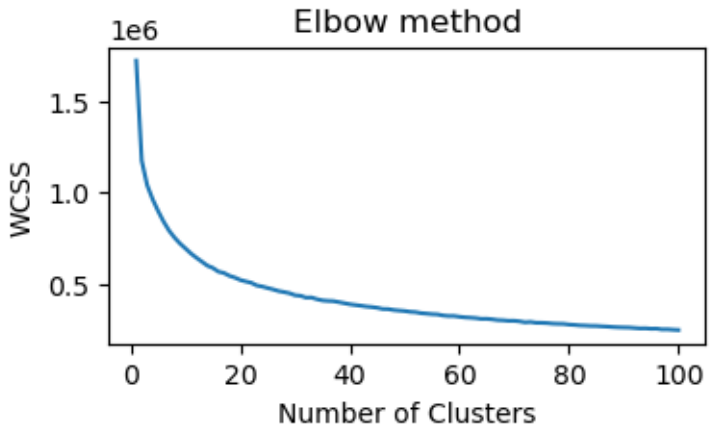
K-Means

The Elbow method was employed to heuristically derive the optimal number of clusters in the K-Means algorithm for the "train" dataset.

[5]:
```
def kmeans_optimization(max_clusters, dataset):
    wcss = []
    for k in range(1, max_clusters):
        kmeans = KMeans(n_clusters=k, random_state=0, n_init=10)
        kmeans.fit(dataset)
        wcss.append(kmeans.inertia_)
    plt.figure(figsize=(4,2))
    plt.plot(range(1, max_clusters), wcss)
```

```
    plt.xlabel('Number of Clusters')
    plt.ylabel('WCSS')
    plt.title('Elbow method')
    plt.show()
```

[6]:
```
kmeans_optimization(101, scaled_train)
```



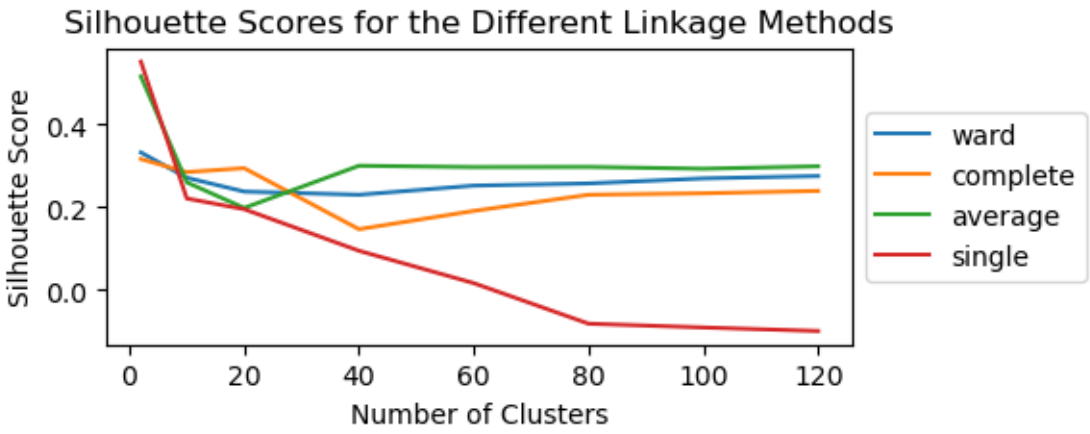
From the plot above comparing the within-cluster sum of squares (WCSS) as a function of the number of clusters, the optimal number of clusters (which corresponds to the elbow point) was considered to be 20.

Hierarchical Agglomerative Clustering

The Silhouette coefficient was employed to heuristically derive the optimal number of clusters and the most appropriate linkage method for Hierarchical Agglomerative Clustering (HAC). As shown below, the Silhouette coefficient was computed for each linkage method at various discrete values for the number of clusters within a certain interval. Unfortunately, it was not possible to compute a continuous range of cluster values due to computational constraints imposed by the size of the dataset.

[7]:
```
def agglomerative_clustering_optimization(linkage_methods, cluster_values, dataset):
    silhouette_scores = {method: [] for method in linkage_methods}
    for k_cluster in cluster_values:
        for linkage in linkage_methods:
            hac = AgglomerativeClustering(n_clusters=k_cluster, linkage=linkage)
            hac.fit(dataset)
            labels = hac.labels_
            score = silhouette_score(dataset, labels)
            silhouette_scores[linkage].append(score)
    plt.figure(figsize=(5,2))
    for linkage, scores in silhouette_scores.items():
        plt.plot(cluster_values, scores, label=linkage)
    plt.xlabel('Number of Clusters')
    plt.ylabel('Silhouette Score')
    plt.title('Silhouette Scores for the Different Linkage Methods')
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.show()
```

[8]:
```
linkage_methods = ['ward', 'complete', 'average', 'single']
cluster_values = [2, 10, 20, 40, 60, 80, 100, 120]
agglomerative_clustering_optimization(linkage_methods, cluster_values, scaled_train)
```



As observed from the plot above, the Silhouette scores for all linkage methods are highest for a number of clusters equal to 2 and decrease from there until this parameter reaches the value of 10. Since the dataset is quite large, it does not make sense to group the data points into only 2 clusters. Thus, to select an appropriate linkage method and the respective number of clusters, we checked where the slope becomes positive again and reaches a local maximum (when considering all four linkage methods). Thus, it seems that the average linkage method (in green) with a number of clusters equal to 40 is the more appropriate choice of for this dataset.

DBSCAN

Unlike partitioning and hierarchical methods, which are designed to find spherical-shaped clusters, density-based clustering methods model clusters as dense regions in the data space separated by sparse regions. These methods can thus discover clusters of non-spherical shape. DBSCAN, a representative density-based clustering algorithm, was employed in this project. The Silhouette coefficient was used to heuristically derive the optimal values for the eps and min_samples parameters of DBSCAN.

[9]:
```
def DBSCAN_optimization(eps_values, min_samples_values, dataset):
    silhouette_scores = {(eps, min_samples): [] for eps in eps_values for min_samples in␣
 ↪min_samples_values}
    for eps in eps_values:
        for min_samples in min_samples_values:
            dbscan = DBSCAN(eps=eps, min_samples=min_samples)
            dbscan.fit(dataset)
            current_labels = dbscan.labels_
            if len(np.unique(current_labels)) > 1:
                silhouette = silhouette_score(dataset, current_labels)
                silhouette_scores[(eps, min_samples)].append(silhouette)
    eps_values = sorted(set(key[0] for key in silhouette_scores.keys()))
    min_samples_values = sorted(set(key[1] for key in silhouette_scores.keys()))
    df = pd.DataFrame(columns=eps_values, index=min_samples_values)
    for key, value in silhouette_scores.items():
        eps, min_samples = key
        df.loc[min_samples, eps] = value[0]
    df = df.sort_index()
    return df
```

[10]:
```
eps_values = [0.10, 0.75, 1.50, 3.00, 6.00, 9.00, 10.00]
min_samples_values = [2, 4, 10, 25, 50, 75]
DBSCAN_optimization(eps_values, min_samples_values, scaled_train)
```

[10]:

	0.10	0.75	1.50	3.00	6.00	9.00	10.00
2	0.125052	0.500385	0.38034	0.179301	-0.055697	0.252088	0.495315
4	-0.145484	0.31103	0.254065	0.161902	0.034333	0.393789	0.495315

```
10 -0.308779  0.041531  0.061157  0.103735  0.130989  0.413694  0.499329
25 -0.279813 -0.115112 -0.077252  0.061048  0.214817  0.413266  0.475434
50 -0.258467 -0.165656 -0.103531  0.014892  0.310419   0.48726  0.543508
75 -0.263077 -0.194953 -0.116101  0.062022  0.155113  0.473565  0.536574
```

From the results above, it is possible to observe that the best Silhouette scores were obtained for the following value pairs of eps and min_samples: (0.75, 2), (10.00, 75) and (10.00, 50). However, the number of obtained clusters for the former case is 2224 and that obtained for the latter is 2 (as shown below), quite extreme values that do not make much sense considering the size of our dataset. Thus, we decided to investigate the number of obtained clusters for other value pairs of the eps and min_samples parameters in which the Silhouette scores were still reasonable. As observed from the results below, possibly the best value pair for the eps and min_samples parameters would be 6.00 and 25, respectively, which would give a total of 10 clusters.

[11]:
```
def calculate_dbscan_clusters(eps_min_samples_list, dataset):
    results = []
    for eps, min_sample in eps_min_samples_list:
        db = DBSCAN(eps=eps, min_samples=min_sample).fit(dataset)
        cls = 1 + db.labels_
        unique_values = np.unique(cls)
        num_unique_values = len(unique_values)
        results.append({'eps': eps, 'min_samples': min_sample, 'number of clusters':␣
↪num_unique_values})

    df_results = pd.DataFrame(results)
    return df_results
```

[12]:
```
eps_min_samples_list = [(0.75, 2), (1.5, 4), (6.00, 25), (6.00, 50), (9.00, 25), (10.00, 50)]
calculate_dbscan_clusters(eps_min_samples_list, scaled_train)
```

[12]:

	eps	min_samples	number of clusters
0	0.75	2	2224
1	1.50	4	884
2	6.00	25	10
3	6.00	50	3
4	9.00	25	3
5	10.00	50	2

Evaluating the Best Clustering Approach

To evaluate the best clustering approach among K-Means, HAC and DBSCAN for the "train" dataset, intrinsic methods such as the Silhouette coefficient and the Calinski-Harabasz metric were employed. The Calinski-Harabasz metric is a measure of cluster cohesion and separation. It is used to evaluate the quality of a clustering result, with higher values indicating more tight clusters as the distances between instances and their cluster assignments are smaller than the inter cluster centroids.

[13]:
```
def evaluate_intrinsic_methods(dataset, kmeans_num_clusters, hac_num_clusters, linkage_method, eps,␣
↪min_samples):
    kms = KMeans(n_clusters=kmeans_num_clusters, random_state=0, n_init=10).fit(dataset)
    hac = AgglomerativeClustering(linkage=linkage_method, n_clusters=hac_num_clusters).fit(dataset)
    dbs = DBSCAN(eps=eps, min_samples=min_samples).fit(dataset)
    kmeans_c = calinski_harabasz_score(dataset, kms.labels_)
    kmeans_s = silhouette_score(dataset, kms.labels_)
    hac_c = calinski_harabasz_score(dataset, hac.labels_)
    hac_s = silhouette_score(dataset, hac.labels_)
    dbscan_c = calinski_harabasz_score(dataset, dbs.labels_)
    dbscan_s = silhouette_score(dataset, dbs.labels_+1)
    eval_metrics = {"Algorithm": ["K-Means", "HAC", "DBSCAN"], "Silhouette": [kmeans_s, hac_s,␣
↪dbscan_s],
                    "Calinski-Harabasz": [kmeans_c, hac_c, dbscan_c]}
    return pd.DataFrame(eval_metrics)
```

[14]:
```
evaluate_intrinsic_methods(scaled_train, 20, 40, 'average', 6, 25)
```

[14]:

	Algorithm	Silhouette	Calinski-Harabasz
0	K-Means	0.208346	2590.148142
1	HAC	0.300708	662.429315
2	DBSCAN	0.214817	198.482285

From the results above, it is possible to observe that the best clustering algorithm for this dataset is K-Means. While its Silhouette score is the lowest of the three algorithms, it is almost the same as that obtained for DBSCAN. Furthermore, it should noted that the parameters of DBSCAN and HAC were optimized around higher values of the Silhouette coefficient, while K-Means was not. Thus, a more important metric to consider when comparing all three clustering algorithms would be the Calinski-Harabasz one. Here, it is possible to see that K-Means displays significantly higher values than either HAC or DBSCAN.

0.1.2 Determining the best clustering approach to classify the "unique" dataset

The general approach followed previously to determine the best clustering algorithm for the "train" dataset was also used to reach the same goal for the "unique" dataset. Again, the relevant parameters for the K-Means, HAC and DBSCAN clustering algorithms were first optimized for this dataset using the Elbow method and the Silhouette coefficient metric. The optimized parameters were then used to find the best clustering approach.

Pre-Processing of the Data

Since the dataset's numerical features have different ranges, the data was scaled using StandardScaler from Scikit-learn.

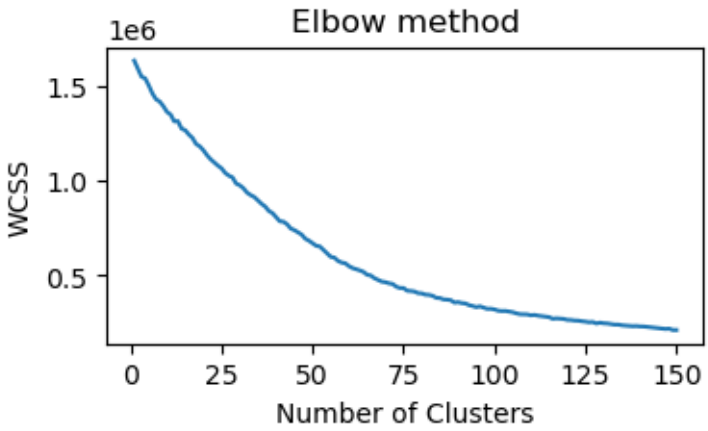
[15]:
```
unique = pd.read_csv("unique_m.csv")
unique = unique.drop(columns = ['critical_temp', 'material'])
```

[16]:
```
scaler = StandardScaler()
scaled_unique = scaler.fit_transform(unique)
```

K-Means

Similarly to what was described previously for the "train" dataset, the Elbow method was again employed to determine the optimal number of clusters in the K-means algorithm for the "unique" dataset.

[17]:
```
kmeans_optimization(151, scaled_unique)
```

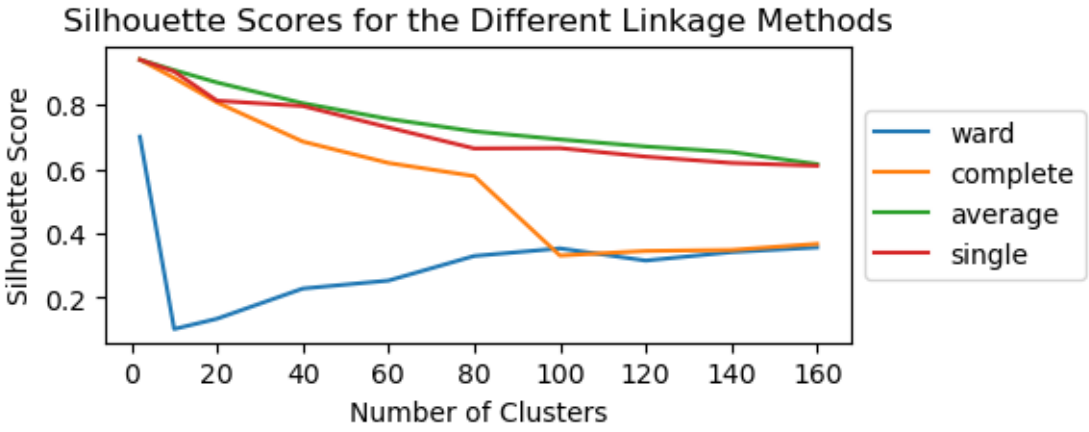


From the plot above comparing the within-cluster sum of squares (WCSS) as a function of the number of clusters, the optimal number of clusters (which corresponds to the elbow point) was considered to be 60.

Hierarchical Agglomerative Clustering

Similarly to what was described previously for the "train" dataset, the Silhouette coefficient was again employed to determine the optimal number of clusters and the most appropriate linkage method for HAC when using the "unique" dataset.

[18]:
```
linkage_methods = ['ward', 'complete', 'average', 'single']
cluster_values = [2, 10, 20, 40, 60, 80, 100, 120, 140, 160]
agglomerative_clustering_optimization(linkage_methods, cluster_values, scaled_unique)
```



As observed from the plot above, the Silhouette scores for all linkage methods are highest for a number of clusters equal to 2 and decrease from there until this parameter reaches the value of 10. Since the dataset is quite large, it does not make sense to group the data points into only 2 clusters. Thus, to select an appropriate linkage method and the respective number of clusters, we checked where the slope becomes positive again and reaches a local maximum (when considering all four linkage methods). From this viewpoint, the linkage method to consider would be the single with a number of clusters equal to 40. However, the average linkage method has a higher Silhouette score for this same number of clusters than the single linkage method. Thus, we considered the average linkage method (in green) with a number of clusters equal to 40 as the more appropriate choice for this dataset.

DBSCAN

Similarly to what was described previously for the "train" dataset, the Silhouette coefficient was again employed to heuristically determine the optimal values for the eps and min_samples parameters of DBSCAN when using the "unique" dataset.

[19]:
```
eps_values = [0.10, 0.60, 0.90, 3.00, 6.00, 9.00, 10.00]
min_samples_values = [2, 4, 10, 25, 50, 75]
DBSCAN_optimization(eps_values, min_samples_values, scaled_unique)
```

[19]:

	0.1	0.6	0.9	3.0	6.0	9.0	10.0
2	0.116558	0.22054	0.092481	0.001474	0.39351	0.478777	0.552183
4	-0.057368	0.180765	0.073431	0.036176	0.410325	0.483221	0.562151
10	-0.182172	0.143035	0.095499	0.155055	0.500186	0.562945	0.563361
25	-0.282907	0.076324	0.060267	0.269747	0.498298	0.657521	0.658902
50	-0.326083	0.002853	0.015666	0.206679	0.673593	0.707638	0.708999
75	-0.356636	-0.052527	0.011434	0.217315	0.669141	0.724104	0.732777

From the results above, it is possible to observe that the best Silhouette scores were obtained for the following value pairs of eps and min_samples: (10.0, 75) and (9.0, 75). However, the number of obtained clusters for both cases is 2 as shown below, a value that does not make sense considering the size of our dataset. Again, we decided to investigate the number of obtained clusters for other value pairs of the eps and min_samples parameters in which the Silhouette scores were still reasonable. As observed from the results below, possibly the best value pair for the eps and min_samples parameters would be 6.0 and 10, respectively, which would give a total of 25 clusters.

[20]:
```
eps_min_samples_list = [(10.0, 75), (9.0, 75), (9.0, 10), (6.0, 10)]
calculate_dbscan_clusters(eps_min_samples_list, scaled_unique)
```

[20]:

	eps	min_samples	number of clusters
0	10.0	75	2
1	9.0	75	2

```
2   9.0            10                   19
3   6.0            10                   25
```

Evaluating the Best Clustering Approach

To evaluate the best clustering model among K-Means, HAC and DBSCAN for the "unique" dataset, intrinsic methods such as the Silhouette coefficient and the Calinski-Harabasz metric were employed.

[21]:
```
evaluate_intrinsic_methods(scaled_unique, 60, 40, "average", 6.0, 10)
```

[21]:

	Algorithm	Silhouette	Calinski-Harabasz
0	K-Means	0.282610	709.548398
1	HAC	0.804567	214.958278
2	DBSCAN	0.500186	178.038151

From the results above, it is again possible to observe that the best clustering algorithm for this dataset is K-Means. While its Silhouette score is the lowest of the three algorithms, it should be noted that the parameters of DBSCAN and HAC were optimized around higher values of the Silhouette coefficient, while K-Means was not. Thus, a more important metric to consider when comparing all three clustering algorithms would be the Calinski-Harabasz one. Here, the K-Means algorithm displays a much higher value for the Calinski-Harabasz metric than HAC or DBSCAN.

0.1.3 Comparing both approaches in computational performance and clustering quality

To compare the chosen clustering approach for the two datasets, which was the K-Means for both, intrinsic methods such as the Silhouette coefficient and the Calinski-Harabasz metric were also employed.

[22]:
```
def kmeans_clustering_performance(num_clusters, dataset):
    start_time = time.time()
    kmeans = KMeans(n_clusters=num_clusters, random_state=0, n_init=10).fit(dataset)
    end_time = time.time()
    elapsed_time = end_time - start_time
    kmeans_s = silhouette_score(dataset, kmeans.labels_)
    kmeans_c = calinski_harabasz_score(dataset, kmeans.labels_)
    return elapsed_time, kmeans_s, kmeans_c
```

[23]:
```
elapsed_time_train, kmeans_s_train, kmeans_c_train = kmeans_clustering_performance(20, scaled_train)
elapsed_time_unique, kmeans_s_unique, kmeans_c_unique = kmeans_clustering_performance(60,␣
 ↪scaled_unique)
performance = pd.DataFrame({"Dataset": ["Train", "Unique"], "Algorithm": ["K-Means", "K-Means"],
                            "Silhouette": [kmeans_s_train, kmeans_s_unique],
                            "Calinski-Harabasz": [kmeans_c_train, kmeans_c_unique],
                            "Performance": [elapsed_time_train, elapsed_time_unique]})
performance
```

[23]:

	Dataset	Algorithm	Silhouette	Calinski-Harabasz	Performance
0	Train	K-Means	0.208346	2590.148142	1.159318
1	Unique	K-Means	0.282610	709.548398	1.621372

Here, it is possible to conclude that the results obtained in terms of Silhouette score and computational performance are quite similar for both datasets. In terms of computational performance, the approach using the K-Means algorithm proved to be quite fast, taking approximately between 1 to 2 seconds to run in both datasets. However, the values for the Calinski-Harabasz metric are quite different between both datasets and are higher in the "train" when compared to the "unique" dataset. This indicates that the obtained clusters in the former are better-defined and more well-separated than in the latter dataset.

0.2 Objective 2 - Evaluating clustering with extrinsic methods

On this second part we are going to consider the **critical temperature** class as our ground truth. Then, in the following code we are going to map the dependent variable, `critical_temp`, into classes using the provided intervals.

[24]:
```
bins = [0.0, 1.0, 5.0, 20.0, 100.0, 200.0]
labels = ["VeryLow", "Low", "Medium", "High", "VeryHigh"]
temp_class = pd.cut(temp, bins = bins, labels = labels, right = False, include_lowest = True)
```

0.2.1 For the "Train" Dataset

[25]:
```
def evaluate_extrinsic_methods(y_labels, dataset, kmeans_num_clusters, hac_num_clusters,␣
↪linkage_method, eps, min_samples):
    kms = KMeans(n_clusters=kmeans_num_clusters, random_state=0, n_init=10).fit(dataset)
    kmeans_h = homogeneity_score(y_labels, kms.labels_)
    kmeans_c = completeness_score(y_labels, kms.labels_)
    kmeans_v = v_measure_score(y_labels, kms.labels_)
    hac = AgglomerativeClustering(linkage=linkage_method, n_clusters=hac_num_clusters).fit(dataset)
    hac_h = homogeneity_score(y_labels, hac.labels_)
    hac_c = completeness_score(y_labels, hac.labels_)
    hac_v = v_measure_score(y_labels, hac.labels_)
    dbs = DBSCAN(eps=eps, min_samples=min_samples).fit(dataset)
    dbscan_h = homogeneity_score(y_labels, dbs.labels_)
    dbscan_c = completeness_score(y_labels, dbs.labels_)
    dbscan_v = v_measure_score(y_labels, dbs.labels_)
    eval_extrinsic_metrics = {"Algorithm": ["K-Means", "HAC", "DBSCAN"], "Homogeneity": [kmeans_h,␣
↪hac_h, dbscan_h],
                              "Completeness": [kmeans_c, hac_c, dbscan_c], "V-measure": [kmeans_v,␣
↪hac_v, dbscan_v]}
    return pd.DataFrame(eval_extrinsic_metrics)
```

[26]:
```
evaluate_extrinsic_methods(temp_class, scaled_train, 20, 40, "average", 6, 25)
```

[26]:

	Algorithm	Homogeneity	Completeness	V-measure
0	K-Means	0.359300	0.173963	0.234424
1	HAC	0.308565	0.269341	0.287622
2	DBSCAN	0.028678	0.158978	0.048591

When ground truth is available for the "train" dataset, again we see that DBSCAN is the worst performing clustering model, since its homogeneity, completeness and V-measure scores are the lowest among the three. K-Means displays a higher value for the homogeneity metric than HAC, indicating that the former model produces more "pure" clusters (containing members of only one data class) than the latter. In contrast, HAC exhibits higher values for the completeness metric than K-Means, suggesting that the former model produces clusters that better capture the entire class members. When assessing the V-measure metric, which corresponds to the harmonic mean of completeness and homogeneity measures, it appears that the model resulting from the HAC algorithm is superior to the one produced by K-Means. This result is different from the one obtained previously in Objective 1 in the absence of ground truth, for which the model arising from the K-Means clustering algorithm appeared to be more suitable.

[27]:
```
def show_clustering_results(dataset, clusters, labels=None):
    results = {}
    sil_score = silhouette_score(dataset, clusters)
    results['Silhouette'] = [sil_score]
    ch_score = calinski_harabasz_score(dataset, clusters)
    results['Calinski-Harabasz'] = [ch_score]
    if labels is not None:
        hom_score = homogeneity_score(clusters, labels)
        results['Homogeneity'] = [hom_score]
        cmp_score = completeness_score(clusters, labels)
        results['Completeness'] = [cmp_score]
        vms_score = v_measure_score(clusters, labels)
        results['V-measure'] = [vms_score]
    return pd.DataFrame(results)

def random_perfect_clustering_results(random_df, perfect_df):
```

```
    df = pd.concat([random_df, perfect_df], keys=['Random', 'Perfect']).reset_index()
    df = df.rename(columns={'level_0': 'Model'})
    df = df[['Model', 'Silhouette', 'Calinski-Harabasz',
             'Homogeneity', 'Completeness', 'V-measure']]
    return df
```

[28]:
```
# Constructing random and perfect models for the train dataset
K = 5
N = len(scaled_train)
rand_clust_train = np.random.randint(K, size=N)
random_train = show_clustering_results(scaled_train, rand_clust_train, temp_class)
perfect_train = show_clustering_results(scaled_train, temp_class, temp_class)
random_perfect_clustering_results(random_train, perfect_train)
```

[28]:

	Model	Silhouette	Calinski-Harabasz	Homogeneity	Completeness	V-measure
0	Random	-0.008239	1.100141	0.000333	0.000416	0.00037
1	Perfect	-0.001726	1615.780671	1.000000	1.000000	1.00000

A random clustering model provides a baseline for evaluating the performance of other clustering algorithms. It represents a scenario in which there is no meaningful structure or pattern in the data. When comparing the Silhouette and Calinski-Harabasz scores obtained for a random model to the ones from Objective 1, it is possible to see that the values for the former (which are quite close to zero) are significantly much lower than those obtained for the three clustering models (K-Means, HAC and DBSCAN). This means that these three built clustering models are actually finding meaningful clusters in the data and their results are not due to chance. When looking at the homogeneity, completeness and V-measure scores obtained for a random model to the ones from Objective 2, it is possible to observe that the values for the former are again practically zero and significantly much lower than those obtained for K-Means, HAC and DBSCAN. This indicates that, in the presence of ground truth, all three clustering models are producing clusters that are more pure and better at grouping together all data points of a particular class within a cluster than what would be expected by chance.

A perfect clustering model represents an ideal scenario in which the true underlying structure of the data is known and is perfectly captured by the clusters. Thus, comparing the results obtained for distinct clustering models with those of the perfect model can help to select the most appropriate algorithm for a given dataset. When comparing the Silhouette and Calinski-Harabasz scores obtained for the perfect model to the ones from Objective 1, it is possible to see that the clustering model that comes closest is the one produced by K-Means. This result is in line with our registered observations for Objective 1. Indeed, the K-Means model displays the highest Calinski-Harabasz score of all three, even higher than the one registered by the perfect model. Surprinsingly, both the random and perfect models display similar Silhouette scores, perhaps indicating that there is an issue or inconsistency with the ground truth labels that were used to build the perfect model.

0.2.2 For the "unique" dataset

[29]:
```
evaluate_extrinsic_methods(temp_class, scaled_unique, 60, 40, "average", 6.0, 10)
```

[29]:

	Algorithm	Homogeneity	Completeness	V-measure
0	K-Means	0.252399	0.161063	0.196643
1	HAC	0.007413	0.172780	0.014217
2	DBSCAN	0.025795	0.099828	0.040996

When ground truth is available for the "unique" dataset, we now see that HAC appears to be the worst performing clustering model, since its V-measure score is the lowest among the three. This is in contrast with the results obtained for Objective 1 in the absence of ground truth, for which the worst performing model seemed to be DBSCAN. In the presence of ground truth, the best performing clustering model resulted from the K-Means algorithm, since it displays the highest value for the V-measure metric. This result is in line with the one obtained previously in Objective 1 in the absence of ground truth.

[30]:
```
# Constructing random and perfect models for the unique dataset
K = 5
N = len(scaled_unique)
rand_clust_unique = np.random.randint(K, size=N)
random_unique = show_clustering_results(scaled_unique, rand_clust_unique, temp_class)
perfect_unique = show_clustering_results(scaled_unique, temp_class, temp_class)
random_perfect_clustering_results(random_unique, perfect_unique)
```

[30]:

	Model	Silhouette	Calinski-Harabasz	Homogeneity	Completeness	V-measure
0	Random	-0.012487	1.128954	0.000255	0.000319	0.000284
1	Perfect	-0.022206	118.311137	1.000000	1.000000	1.000000

When comparing the Silhouette and Calinski-Harabasz scores obtained for a random model to the ones from Objective 1, it is possible to see that the values for the former (which which are quite close to zero) are significantly much lower than those obtained for the three clustering models (K-Means, HAC and DBSCAN). Thus, similarly to what was described previously for the "train" dataset, the three built clustering models are finding meaningful clusters in the data of the "unique" dataset and the obtained results are not due to chance. When looking at the homogeneity, completeness and V-measure scores obtained for a random model to the ones from Objective 2, it is possible to observe that the values for the former are again practically zero and significantly much lower than those obtained for K-Means, HAC and DBSCAN. This indicates that, in the presence of ground truth, all three clustering models are producing clusters that are more pure and better at grouping together all data points of a particular class within a cluster than what would be expected by chance.

When comparing the Silhouette and Calinski-Harabasz scores obtained for the perfect model to the ones from Objective 1 using the "unique" dataset, it is possible to see that the clustering model that comes closest is the one produced by DBSCAN. This result is different from the one registered in Objective 1, where the more appropriate model for our dataset was deemed to be K-Means. However, neither the Silhouette or the Calinski-Harabasz scores for the perfect model are satisfactory. The Calinski-Harabasz score is not particularly high and the Silhouette score is negative, which might suggest that there is a problem with the ground truth labels that were used to build the perfect model.

0.3 Conclusions

To sum up, this project focused on the application of various clustering algorithms, namely K-Means, HAC and DBSCAN to analyze and group together data points in two related but distinct datasets. By using intrinsic methods, such as the Elbow method and the Silhouette coefficient, the relevant parameters for each clustering algorithm were optimized in both datasets. With the optimized parameters, the clustering models were then built and the best approach chosen according to the values obtained using intrinsic methods such as the Silhouette coefficient and the Calinski-Harabasz score. Then, in the presence of ground truth, the clustering quality of the obtained models was evaluated using extrinsic methods. These results were then compared with those obtained previously in the absence of ground truth, as well as with those arising from random and perfect clustering models.

Initially, in the absence of ground truth, the best clustering models for both datasets were those obtained using the K-Means algorithm, as determined by the higher values of the Calinski-Harabasz metric. In terms of computational performance, this method proved to be quite fast, taking approximately between 1 to 2 seconds to run in both datasets. However, in the presence of ground truth, the best model proved to be the one derived using the HAC algorithm for the "train" dataset, while for the "unique" dataset the most appropriate was still the one obtained using the K-Means algorithm. Thus, ground truth can influence the selection of the best clustering model for a dataset. Surprisingly, the perfect model displayed negative values for the Silhouette coefficient in both datasets and the value for the Calinski-Harabasz metric was quite low for the "unique" dataset. This could suggest that there is an issue with the ground truth labels which were used to build this model.

0.4 Bibliography

[1] Han J, Kamber M, Pei J. Data mining concepts and techniques third edition. University of Illinois at Urbana-Champaign Micheline Kamber Jian Pei Simon Fraser University. 2012.