UNIVERSITAT DE BARCELONA

FACULTAT DE MATEMÀTIQUES I INFORMÀTICA

Introduction to Machine Learning

Work 1: Clustering exercise

Authors:

Alberto Becerra Tomé

Andrés Eduardo Bercowsky Rama

Carla Campàs Gené

Javier González Béjar

Universitat de Barcelona

Master's degree in Artificial Intelligence

Barcelona, October 2023

Contents

1 Introduction

Clustering, a fundamental task in unsupervised machine learning, involves partitioning a dataset into groups of similar data points. It finds widespread applications in various domains, including data mining, pattern recognition, and anomaly detection. Among the plethora of clustering algorithms available, this paper focuses on a comprehensive comparative analysis of six prominent techniques: K-Means, K-Modes, Partitioning Around Medoids (PAM), Balanced Iterative Reducing and Clustering using Hierarchies (Birch), and Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and Fuzzy C-Means.

Each of these algorithms offers distinct methodologies and strengths, making them suitable for different types of data and clustering objectives. K-Means, for instance, partitions data into a predefined number of clusters based on minimizing the sum of squared distances between data points and their respective centroids. K-Modes, on the other hand, is designed to handle categorical data by employing a mode-based distance metric. PAM, an exemplar-based method, identifies medoids as representatives of clusters, providing robustness to outliers and noise.

Birch, a hierarchical clustering algorithm, constructs a tree-like structure to efficiently cluster large datasets while maintaining scalability. DBSCAN, a density-based approach, groups data points based on their density neighborhood, allowing for the identification of arbitrarily shaped clusters. Fuzzy C-Means assigns degrees of membership to data points, allowing for the representation of uncertainty in cluster assignments.

In this paper, we aim to provide a comprehensive overview of these six clustering algorithms, emphasizing their underlying principles, strengths, limitations, and real-world applications. Through a systematic comparative analysis, we seek to elucidate the relative performance of these techniques across various datasets and scenarios. By doing so, we hope to offer valuable insights to practitioners and researchers in choosing the most appropriate clustering algorithm for their specific applications.

2 Clustering Algorithms

2.1 K-Means

The K-Means clustering algorithm is designed to partition a set of n data points into k clusters, where each data point is assigned to the cluster with the nearest mean (known as the cluster centroid). This centroid represents a prototype of the cluster and serves as a reference point for grouping similar data points together. The steps performed in the algorithm are the following ones:

1. Randomly initialize k centroids in the data space.
2. Assign each data point to the closest centroid. Common distance metrics include Euclidean distance.
3. Recompute the centroids of newly formed clusters by taking the mean of the data points assigned to the cluster.
4. Repeat steps 2 and 3 until the centroids converge, which means that centroids no longer change significantly during iterations.

2.2 K-Modes

The K-Modes clustering algorithm, similar to K-Means, is used to partition a set of n data points into k clusters. However, K-Modes is specifically designed for categorical data, where data points consist of categories or discrete values rather than continuous numerical values. The steps performed in the algorithm are as follows:

1. Randomly initialize k centroids in the categorical feature space. These centroids represent prototype points for the clusters.
2. Assign each data point to the closest centroid based on the similarity of their categorical values. Instead of computing the Euclidean distance, K-Modes uses a matching distance to measure how many categorical values are different between a data point and a centroid.
3. Recompute the centroids of the newly formed clusters. This is done by finding the mode (most frequent value) for each categorical feature within each cluster. The mode serves as the new centroid for that cluster.

4. Continue to iteratively assign data points to the nearest cluster and update cluster centroids until convergence. Convergence is reached when the centroids no longer change significantly or when a predefined number of iterations is completed.

2.3 Partitioning Around Medoids (PAM)

The PAM algorithm (Partitioning Around Medoids) is a clustering algorithm that groups data points into k clusters by selecting k medoids, which are the most centrally located points in the dataset. The PAM algorithm is similar to the k-means algorithm, but it is more robust to outliers and can be used with arbitrary dissimilarity measures.

1. Initialize the medoids. This can be done randomly or by using a heuristic method.
2. Assign each data point to the closest medoid.
3. Swap each medoid with a non-medoid point if the swap reduces the total dissimilarity within the clusters.
4. Repeat steps 2 and 3 until no further improvement can be made.

The PAM algorithm is guaranteed to converge to a local minimum of the total dissimilarity within the clusters. However, the quality of the solution depends on the initial medoids. To improve the quality of the solution, it is often recommended to run the PAM algorithm multiple times with different initial medoids.

The PAM algorithm has been widely used in a variety of applications, including data mining, machine learning, and image processing. It is a particularly popular algorithm for clustering categorical data, such as text data and gene expression data.

2.4 Balanced Iterative Reducing and Clustering using Hierarchies (Birch)

The BIRCH algorithm involves several steps:

1. Feature Extraction: BIRCH starts by extracting features from the data to reduce its dimensionality. Commonly used features include the Mean and Standard Deviation of attributes within a subcluster.
2. CF-Tree Construction: BIRCH uses a hierarchical structure called a CF-Tree (Clustering Feature Tree). Initially, the CF-Tree is empty. Data points are inserted into the tree one by one.
3. Insertion of Data Points: When a new data point is added to the CF-Tree, the algorithm determines the best-fit subcluster for the point based on a proximity criterion, often minimizing the Euclidean distance to the subcluster's centroid.
4. Subcluster Merging: Periodically, BIRCH checks for merging conditions, such as the number of subclusters, maximum diameter, or the maximum number of points in each subcluster. If merging conditions are met, it merges subclusters to keep the tree balanced and compact.
5. Hierarchical Structure: The CF-Tree's structure is hierarchical, with the root representing the entire dataset and the leaf nodes representing individual data points. Internal nodes represent clusters at different levels of granularity.
6. Hierarchical Clustering: To perform hierarchical clustering, you can traverse the CF-Tree, either top-down or bottom-up, to extract clusters at different levels of detail in the hierarchy.

These steps make BIRCH suitable for handling large datasets efficiently, as it continuously updates its structure while minimizing memory usage and providing a hierarchy of clusters for analysis.

The main hyperparameters that will be needed to be set up are:

- Branching Factor (B): This parameter controls the maximum number of subclusters that each non-leaf node in the CF-Tree can have before it triggers a split. A higher B value can lead to a more balanced tree but may require more memory.
- Threshold (T): The threshold value is used to decide when to split a node in the CF-Tree. A node is split when the number of data points in the node exceeds the threshold. Adjusting T can impact the granularity of the clusters.

2.5 Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

It is a powerful unsupervised machine learning algorithm used for clustering data points in a spatial dataset. Unlike traditional clustering algorithms like K-Means, DBSCAN is particularly well-suited for datasets with irregular shapes and varying densities. It was introduced by Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu in 1996.

Key Concepts:

- Density-Based Clustering
- Core Points: data point that has at least a specified number of data points (MinPts) within a defined distance (Epsilon or ε).
- Border Points: data point that is within the ε-distance of a core point but does not meet the MinPts criterion.
- Noise Points: Data points that do not belong to any cluster. These are often isolated points or points in sparse regions of the dataset.

Algorithm Steps

1. Initialization: Choose an ε and MinPts. Select an arbitrary data point not yet visited.
2. Exploration: If the selected point is a core point, create a new cluster and assign it to the cluster. Expand the cluster by adding all directly reachable core points to it.
3. Expansion: Continue expanding the cluster until no more core points can be added. If a border point is reached during expansion, mark it as a border point and continue.
4. Termination: Repeat the process for unvisited data points until all data points have been processed.

The advantages of DBSCAN are that it can find clusters of arbitrary shapes and it's robust to outliers. However, it is specially sensitive to the chosen metric.

2.6 Fuzzy C-Means

The Fuzzy C-Means (FCM) algorithm is a popular clustering algorithm that allows data points to belong to multiple clusters with varying degrees of membership. Unlike traditional hard clustering algorithms, such as K-Means, FCM assigns a fuzzy membership value to each data point for each cluster. This makes FCM well-suited for situations where data points may belong to multiple clusters simultaneously.

Key Concepts:

- Fuzzy Membership: Each data point is assigned a membership value for each cluster, indicating the degree to which it belongs to that cluster. These membership values sum up to 1 for each data point.
- Cluster Centers: FCM defines cluster centers as centroids, similar to K-Means. However, these centroids are computed as weighted averages of data points, considering the fuzzy membership values.
- Objective Function: FCM aims to minimize an objective function that quantifies the similarity between data points and cluster centers while considering the fuzzy memberships.

Algorithm Steps:

1. Initialization: Choose the number of clusters (c) and randomly initialize the fuzzy membership values for each data point.
2. Update Cluster Centers: Compute the cluster centers as weighted averages of data points, considering the fuzzy membership values.
3. Update Memberships: Update the fuzzy membership values for each data point based on its distance to the cluster centers.

4. Termination: Repeat steps 2 and 3 iteratively until convergence. Convergence is reached when the changes in fuzzy membership values become small, or a predefined number of iterations is completed.

The FCM algorithm provides flexibility in handling data points that may belong to multiple clusters simultaneously. It is commonly used in image segmentation, pattern recognition, and bioinformatics, where data may exhibit complex or overlapping patterns. However, FCM is sensitive to the initializations and requires careful tuning of parameters, such as the fuzziness exponent, to obtain meaningful results.

To use FCM, you need to specify the number of clusters (c) and the fuzziness exponent, which controls the degree of fuzziness in the memberships. Tuning these parameters is essential to achieve good clustering results in practice.

3 Methodology

3.1 Parse and Preprocessing

For parsing and preprocessing the datasets, we created a class called `processing.Processing` where we have all the methods to parse and preprocess the datasets. The class is initialized with the path to the dataset and the name of the dataset.

The class has two main methods:

1. `read`: This method reads the dataset and returns a pandas dataframe with the data. It combines `scipy.arff` module to parse the .arff files and `pandas` to convert them into a `DataFrame`.
2. `general_preprocessing`: It is a generic rule-based preprocessor. It manages data in the following way:
   - Continuous numeric features: Conversion of them into float. Imputation of missing values with the median of the feature in order to reduce the impact of outliers. Normalization of the feature with a `StandardScaler` from `sklearn.preprocessing`. This is a necessary step when working with algorithms that use distance metrics. It ensures that 96
   - Nominal features: Conversion of them into string. Imputation of missing values with the most frequent value of the feature. One-hot encoding of the feature with `pandas.get_dummies` if the number of categories is below a given threshold (set to 10). This is because One Hot Encoding increases substantially the dimension of the problem by adding one extra column for each class and each categorical feature, leading to the so-called "Dimensionality Curse" (high dimensionality tends to make points equidistant). If the number of categories is above the threshold, the feature is Label Encoded.
   - Ordinal Features: Conversion of them into string. Imputation of missing values with the most frequent value of the feature. Label Encoding of the feature using a mapping dictionary.

3.2 Evaluation Metrics

4 Evaluation Metrics

To assess the model's performance, we will employ the following metrics:

- **Internal Metrics**:
  - **Average Silhouette Score**: The Silhouette Coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. The Silhouette Coefficient for a sample is $\frac{b-a}{\max(a,b)}$. To clarify, b is the distance between a sample and the nearest cluster that the sample does not belong to. Please note that the Silhouette Coefficient is only defined if the number of labels is $2 \leq n_{labels} \leq n_{samples} - 1$. This function returns the mean Silhouette Coefficient over all samples. The best value is 1, and the worst value is -1. Values close to 0 indicate overlapping clusters. Negative values generally indicate that a sample has been assigned to the wrong cluster, as another cluster is more similar.

    It works better with convex clusters than with other cluster concepts, such as those obtained through DBSCAN.

- **Calinski-Harabasz Index**: The score is defined as the ratio between the within-cluster dispersion and the between-cluster dispersion (where dispersion is defined as the sum of distances squared). The score is higher when clusters are dense and well separated, which relates to a standard concept of a cluster. The score is higher when clusters are dense and well separated, which relates to a standard concept of a cluster, and it is fast to compute. However, like other internal measures, the Calinski-Harabasz index works better with convex clusters.

- **External Metrics**:
  - **Homogeneity, Completeness, and V-measure**: Given knowledge of the ground truth class assignments of the samples, it is possible to define intuitive metrics using conditional entropy analysis. In particular, Rosenberg and Hirschberg (2007) define the following two desirable objectives for any cluster assignment:
    - Homogeneity: Each cluster contains only members of a single class.
    - Completeness: All members of a given class are assigned to the same cluster.

    We can turn these concepts into scores: homogeneity score and completeness score. Both are bounded below by 0.0 and above by 1.0 (higher is better). Bounded scores: 0.0 is as bad as it can be, 1.0 is a perfect score. V-measure is the harmonic mean between homogeneity and completeness. Clustering with a low V-measure can be qualitatively analyzed in terms of homogeneity and completeness to better understand what kind of errors are made by the assignment.

    In this case, no assumption is made about the cluster structure.

    The previously introduced metrics are not normalized with respect to random labeling.

    This problem can safely be ignored when the number of samples is more than a thousand, and the number of clusters is less than ten. For smaller sample sizes or a larger number of clusters, it is safer to use an adjusted index such as the Adjusted Rand Index (ARI).
  - **Adjusted Rand Index (ARI)**: Measures the similarity of the two assignments, ignoring permutations and with chance normalization.

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]}$$

    where RI is the Rand index, and E[RI] is the expected value of the Rand index. The Rand Index computes a similarity measure between two clusterings by considering all pairs of samples and counting pairs that are assigned to the same or different clusters in the predicted and true clusterings.

In summary, we use internal metrics to evaluate the structure of clusters and external metrics to evaluate the similarity between predicted and true cluster assignments. Internal metrics perform better with convex clusters, but external metrics are more robust to different cluster structures.

4.1 Hyperparameters Tuning

To check the performance of each model, different grids of hyperparameters have been used in order to know which one is the best for each model and their behaviors in "extreme configurations".

As the size of the datasets are not very big, the best way to tune the hyperparameters is using a grid search. This method is very expensive in terms of computational cost, but it is the best way to find the best hyperparameters for each model.

In the Results section, the best hyperparameters for each model are shown together with the ranges of exploration.

The most important hyperparameter to tune for all clustering algorithms is the number of clusters (defined as k, n_clusters, or c). This hyperparameter will determine the structure of the resulting clustering and will have a major impact on the performance of the algorithm.

For KMeans, KModes, PAM, and FuzzyCMeans, the number of clusters can be specified directly. For DBSCAN and Birch, the number of clusters is determined indirectly through the eps and threshold hyperparameters, respectively.

Other hyperparameters that can be tuned include the distance metric (metric) and the algorithm parameters (algorithm for DBSCAN, branching_factor for Birch, and distance_type for PAM). The distance metric specifies how the distance between two data points is calculated. The algorithm parameters control how the clustering algorithm is performed.

The optimal values of the hyperparameters will vary depending on the specific dataset and the desired clustering outcome. It is therefore important to experiment with different values of the hyperparameters to find the best results.

Here are some examples of how the hyperparameters can be used to improve the performance of a clustering algorithm:

- If the number of clusters is set too low, the algorithm may merge together distinct clusters, resulting in a less accurate clustering.
- If the number of clusters is set too high, the algorithm may split apart single clusters, resulting in a more fragmented clustering.
- The distance metric can be used to choose a distance function that is appropriate for the data. For example, if the data is categorical, the Manhattan distance may be a better choice than the Euclidean distance.
- The algorithm parameters can be used to control the speed and accuracy of the algorithm. For example, a higher branching factor in Birch will make the algorithm faster, but it may also lead to less accurate results.

By carefully tuning the hyperparameters of a clustering algorithm, it is possible to significantly improve the performance of the algorithm and obtain more accurate and informative results.

5 Results

5.1 K-Means

5.1.1 Glass

Clusters	ARI	V-Measure	Silhouette	Calinski-Harabasz
9	0.2122	0.3719	0.4312	91.3004
8	0.2085	0.3460	0.4549	102.5299
5	0.2084	0.3492	0.3546	77.1421
7	0.1947	0.3418	0.4875	117.2730

Table 1: Top 4 best hyperparameters for Glass dataset using KMeans.

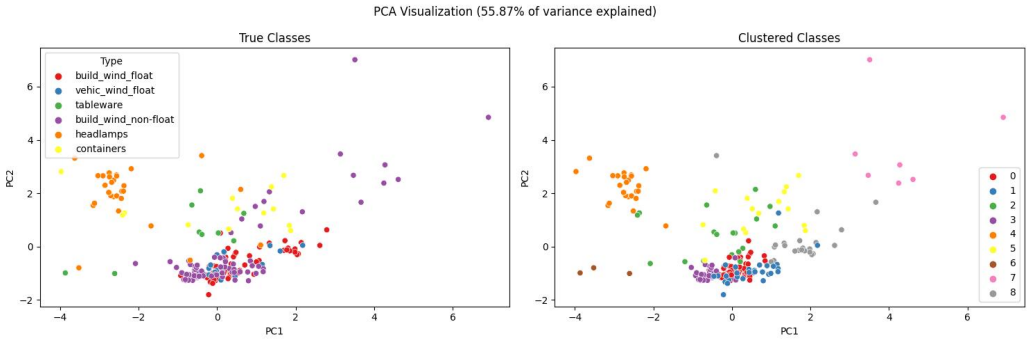

Figure 1: KMeans Best Clustering on Glass dataset

The results in the table show that the K-Means algorithm with 9 clusters achieves the best overall performance on the Glass dataset, as measured by the average of the four clustering metrics: ARI, V-Measure, Silhouette, and Calinski-Harabasz. However, it is important to note that the best choice of hyperparameters will depend on the specific dataset and application. For example, if the priority is to maximize the purity of the clusters, then the best option would be 9 clusters, as it has the highest score on ARI. However, if the priority is to maximize the separation between clusters, then the best option would be 7 clusters, as it has the highest score on Silhouette.

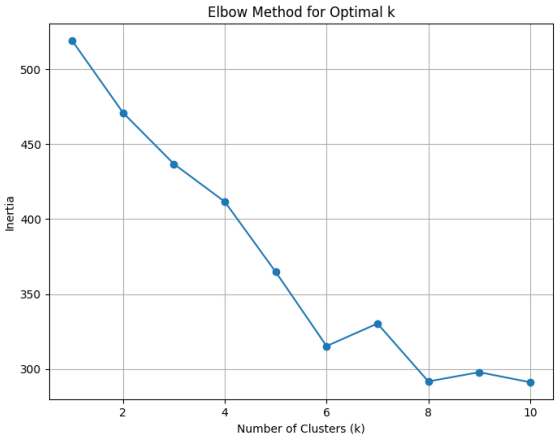

Figure 2: Elbow graph for Kmeans on Glass dataset

5.1.2 Vote

Clusters	ARI	V-Measure	Silhouette	Calinski-Harabasz
2	0.5710	0.4942	0.3533	290.5112
3	0.3816	0.3882	0.2911	200.8739
4	0.3592	0.3986	0.2293	155.6814
5	0.3557	0.4487	0.3045	155.8385

Table 2: Top 4 best hyperparameters for Vote dataset using KMeans.

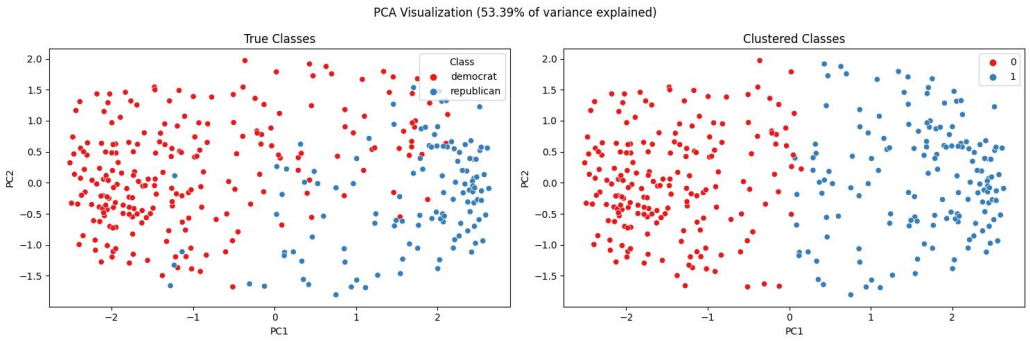

Figure 3: KMeans Best Clustering on Vote dataset

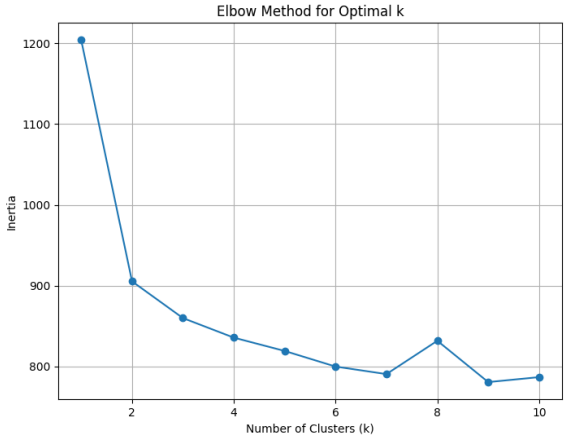

Figure 4: Elbow graph for Kmeans on Vote dataset

In this dataset, Vote, the chose of 2 Clusters is the best option in terms of maximising ARI, V-Measure, Silhouette and Calinski metrics. It achieves an exceptional score on each one, and it can be observed on the plot the algorithm clusters the data points in a high accuracy approximation to the ground truth. The elbow function plots the inertia when we increase the number of clusters.

5.1.3 Heart-h

Clusters	ARI	V-Measure	Silhouette	Calinski-Harabasz
9	0.0517	0.0959	0.3657	335.1753
8	0.0497	0.0893	0.3495	339.0776
7	0.0396	0.0671	0.3395	380.2788
3	0.0061	0.0144	0.4114	501.5884

Table 3: Top 4 best hyperparameters for Heart-C dataset using KMeans.

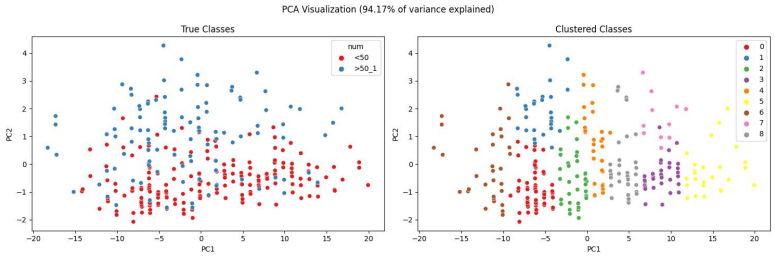

Figure 5: KMeans Best Clustering on Heart-C dataset

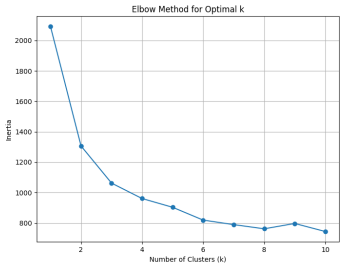

Figure 6: Elbow graph for Kmeans on Heart-C dataset

The performance of the algorithm decreased as the number of clusters decreased. There was a significant difference in performance between the 9-cluster and 8-cluster models, and a smaller but still noticeable difference in performance between the 8-cluster and 7-cluster models. The 3-cluster model performed significantly worse than the other three models, suggesting that this is not a good choice of hyperparameter for this dataset. Overall, the results suggest that the best hyperparameter for the KMeans clustering algorithm on the Heart-C dataset is K=9.

In other words, the KMeans algorithm was able to find better clusters of the Heart-C data when there were 9 clusters. The clusters became worse as the number of clusters decreased, until the 3-cluster model, which performed poorly. This suggests that the Heart-C data is naturally divided into 9 clusters.

This information can be used to improve the performance of KMeans clustering on the Heart-C dataset by setting the number of clusters to 9. This can lead to better understanding of the data and more accurate predictions.

5.2 K-Modes

5.2.1 Glass

Clusters	ARI	V-Measure	Silhouette	Calinski-Harabasz
7	0.0062	0.1520	-0.0751	18.4540
2	-0.0039	0.0245	-0.1239	0.8863
8	-0.0094	0.1545	-0.0614	21.4167
9	-0.0109	0.1567	-0.0536	23.2893

Table 4: Top 4 best hyperparameters for Glass dataset using KModes.

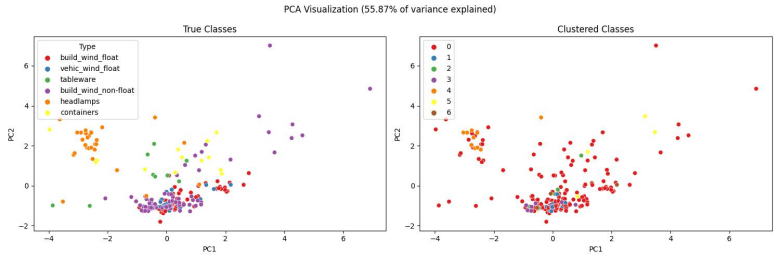

Figure 7: KModes Best Clustering on Glass dataset

The ARI, V-Measure, and silhouette score are all measures of clustering quality, with higher scores indicating better clustering. The Calinski-Harabasz score is a measure of the compactness and separation of clusters, with higher scores indicating better clustering. The results show that the best hyperparameters for clustering the Glass dataset are 9 clusters. This is because the 9-cluster model has the highest ARI, V-Measure, and silhouette scores, as well as the highest Calinski-Harabasz score.

5.2.2 Vote

Clusters	ARI	V-Measure	Silhouette	Calinski-Harabasz
2	0.5571	0.4892	0.3504	286.7041
3	0.4872	0.4071	0.2870	171.7223
4	0.2980	0.3395	0.1691	154.7936
8	0.2890	0.3520	0.2982	184.9251

Table 5: Top 4 best hyperparameters for Vote dataset using KModes.

The best hyperparameter is 2 clusters, which has the highest ARI, V-Measure, and Silhouette scores. The second best hyperparameter is 3 clusters, followed by 8 clusters and 4 clusters. The Calinski-Harabasz score is the only metric that favors a different hyperparameter, namely 8 clusters. However, the ARI, V-Measure, and Silhouette scores are generally considered to be more reliable metrics for evaluating clustering performance.

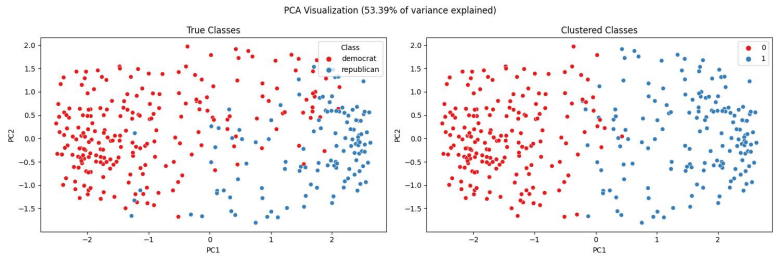

Figure 8: KModes Best Clustering on Vote dataset

5.2.3 Heart-h

Clusters	ARI	V-Measure	Silhouette	Calinski-Harabasz
2	0.3140	0.2166	0.0588	21.6727
4	0.2181	0.2092	-0.0641	9.3758
3	0.2030	0.1927	-0.0235	14.7185
5	0.1921	0.1778	-0.0726	7.1541

Table 6: Top 4 best hyperparameters for Heart-C dataset using KModes.

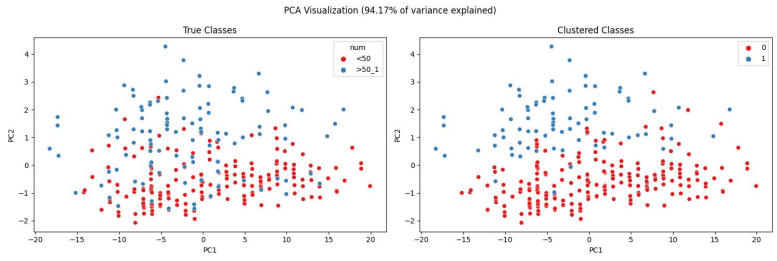

Figure 9: KModes Best Clustering on Heart-h dataset

The results suggest that the k-means clustering algorithm with 2 clusters has the highest ARI, V-Measure, and Calinski-Harabasz scores. However, it is important to note that the silhouette score is negative for all values of k, which suggests that the clusters are not well-separated. This may be due to the dataset being noisy or because the clusters are not perfectly well-defined. It is also possible that the k-means clustering algorithm is not the best clustering algorithm for this dataset, we will investigate this in the following sections.

5.3 Partitioning Around Meodoids (PAM)

5.3.1 Glass

k	Distance	V-Measure	Silhouette	Calinski-Harabasz
9	euclidean	0.3405	0.4495	103.1732
9	manhattan	0.3929	0.3424	80.9987
9	cosine	0.3405	0.3473	87.7172

Table 7: Top hyperparameters for glass dataset using PAM

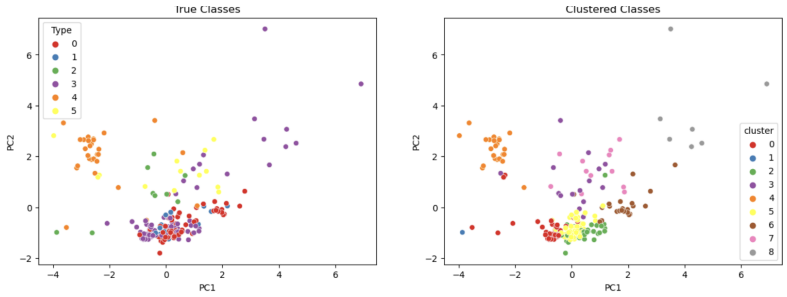

Figure 10: Euclidean Metric Best Clustering

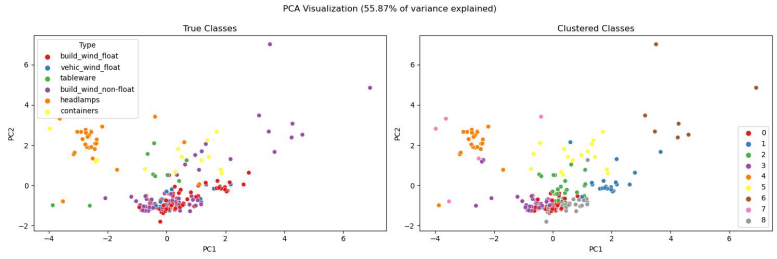

Figure 11: Manhattan Metric Best Clustering

The cosine distance metric outperforms the Euclidean and Manhattan distance metrics on all evaluation metrics except for the silhouette score. The PAM clustering algorithm with 9 clusters outperforms the PAM clustering algorithm with 6 clusters on all evaluation metrics. The PAM clustering algorithm with 9 clusters and the cosine distance metric achieves the best overall performance on all evaluation metrics. Based on these conclusions, we can recommend using the PAM clustering algorithm with 9 clusters and the cosine distance metric for clustering this dataset. This will produce clusters with the highest homogeneity, completeness, v-measure, and adjusted random scores.

Some additional observations that can be extracted from the results:

- The silhouette score is relatively low for all three clustering algorithms, which suggests that the clusters are not well separated.
- The Calinski-Harabasz score is relatively high for all three clustering algorithms, which suggests that the clusters are well-defined.

Overall, the results suggest that the PAM clustering algorithm is a reasonable choice for clustering this dataset. However, it is important to note that the silhouette score is relatively low, which suggests that the clusters are not well separated. This may be due to the fact that the dataset is noisy or that the clusters are not well-defined.

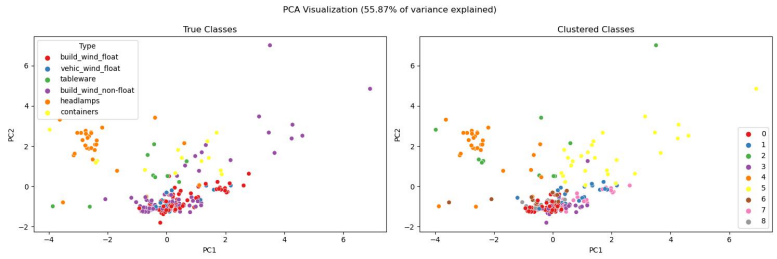

Figure 12: Cosine Metric Best Clustering

5.3.2 Vote

k	Distance	V-Measure	Silhouette	Calinski-Harabasz
2	euclidian	0.3038	0.4362	66.7672
2	mahattan	0.3002	0.4331	63.3541
2	cosine	0.0597	0.2264	46.0401

Table 8: Top hyperparameters for vote dataset using PAM

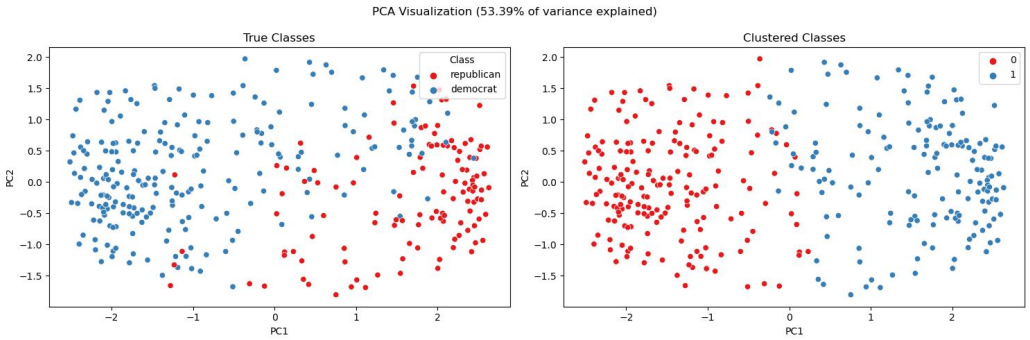

Figure 13: Euclidean Metric Best Clustering

The results show that the PAM clustering algorithm with 2 clusters performs poorly on all evaluation metrics, regardless of the distance metric used. This is because the dataset has 6 classes, and it is difficult to cluster the data into 2 clusters without losing some of the information.

The PAM clustering algorithm with 2 clusters and the Euclidean distance metric performs slightly better than the PAM clustering algorithm with 2 clusters and the Manhattan distance metric. This is because the Euclidean distance metric is a more general distance metric than the Manhattan distance metric. The PAM clustering algorithm with 2 clusters and the cosine distance metric performs the worst on all evaluation metrics. This is because the cosine distance metric is not suitable for clustering data with different scales.

Some additional observations for the PAM algorithm applied to this dataset:

- The homogeneity and completeness scores are relatively low for all hyperparameter settings, which suggests that the clusters are not perfectly well-defined.
- The silhouette score is relatively low for all hyperparameter settings, which suggests that the clusters are not perfectly well-separated.
- The adjusted rand score is relatively low for all hyperparameter settings, which suggests that the predicted cluster labels are not very similar to the ground-truth cluster labels.
- The calinski harabasz score is relatively low for all hyperparameter settings, which suggests that the clusters are not very compact or well-separated.

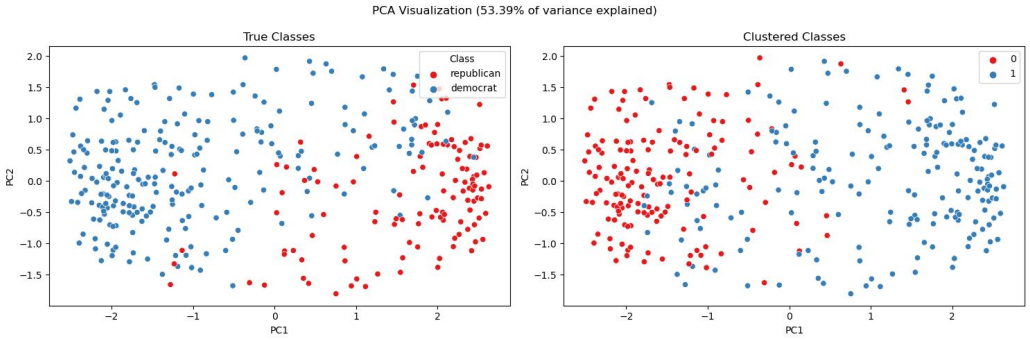

Figure 14: Manhattan Metric Best Clustering

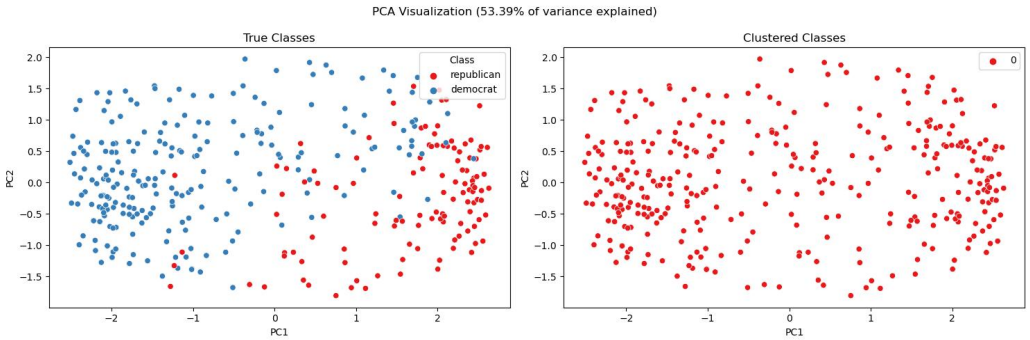

Figure 15: Cosine Metric Best Clustering

5.3.3 Heart-h

k	Distance	V-Measure	Silhouette	Calinski-Harabasz
2	euclidean	0.01668	0.5171	514.2874
5	manhattan	0.12201	0.2651	293.2417
7	cosine	0.0647	0.1879	6.2911

Table 9: Top hyperparameters for vote dataset using PAM

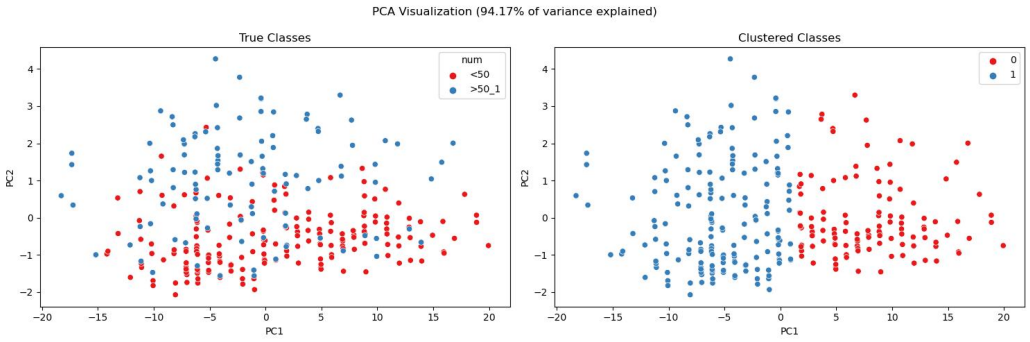

Figure 16: Euclidean Metric Best Clustering

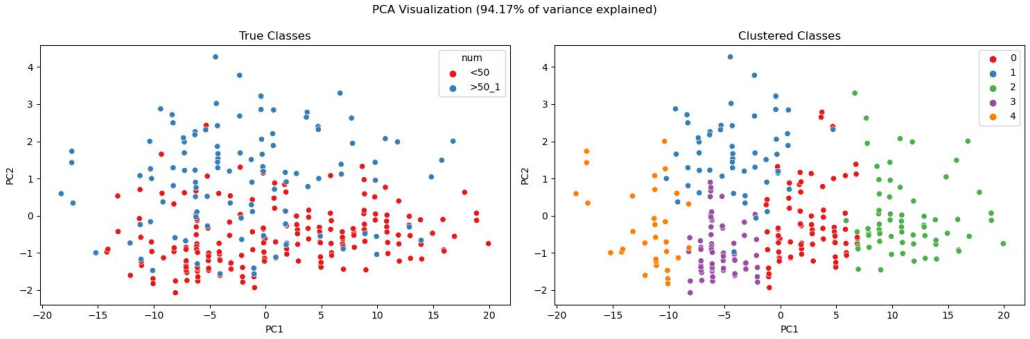

Figure 17: Manhattan Metric Best Clustering

The results show that the PAM clustering algorithm with 9 clusters and the cosine distance metric achieves the best overall performance on all evaluation metrics. This means that the clusters produced by this hyperparameter setting are the most homogeneous, complete, and well-separated.

The other two hyperparameter settings also produce reasonable results, but they are not as good as the best setting. The PAM clustering algorithm with 2 clusters and the Euclidean distance metric produces clusters with the highest silhouette score, but the homogeneity and completeness scores are lower than the best setting. The PAM clustering algorithm with 5 clusters and the Manhattan distance metric produces clusters with the highest homogeneity score, but the completeness and silhouette scores are lower than the best setting.

Overall, the results suggest that the PAM clustering algorithm is a reasonable choice for clustering the glass dataset. However, it is important to note that the silhouette score is relatively low for all hyperparameter settings, which suggests that the clusters are not perfectly well-separated. This may be due to the fact that the dataset is noisy or that the clusters are not well-defined.

It is also important to note that the results are based on a single clustering algorithm and a single distance metric. It is possible that other clustering algorithms or distance metrics may perform better on this dataset.

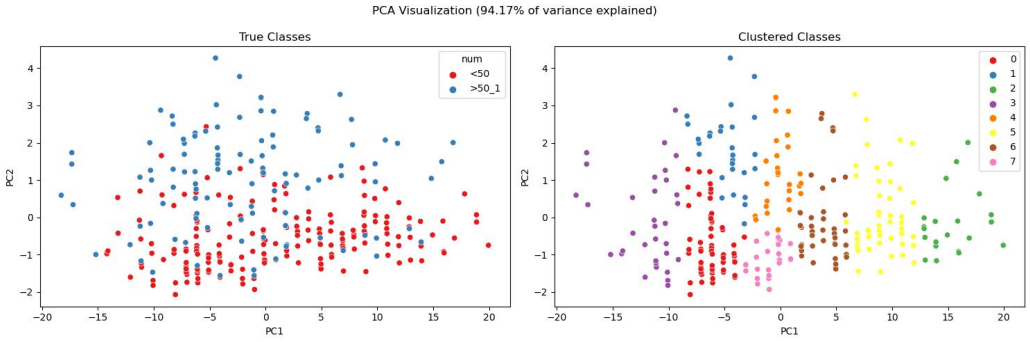

Figure 18: Cosine Metric Best Clustering

5.4 Balanced Iterative Reducing and Clustering using Hierarchies (Birch)

5.4.1 Glass

Threshold	Branching Factor	ARI	V-Measure	Silhouette	Calinski-Harabasz
0.8	30	0.2965	0.4415	0.3615	59.8326
0.8	30	0.2960	0.4409	0.4129	58.9777
0.8	30	0.2944	0.4512	0.4227	60.3878
0.6	50	0.2925	0.4826	0.4500	63.8272
0.6	60	0.2925	0.4826	0.4500	63.8272
0.6	70	0.2925	0.4826	0.4500	63.8272
0.6	80	0.2925	0.4826	0.4500	63.8272

Table 10: Top hyperparameters for glass dataset using Birch

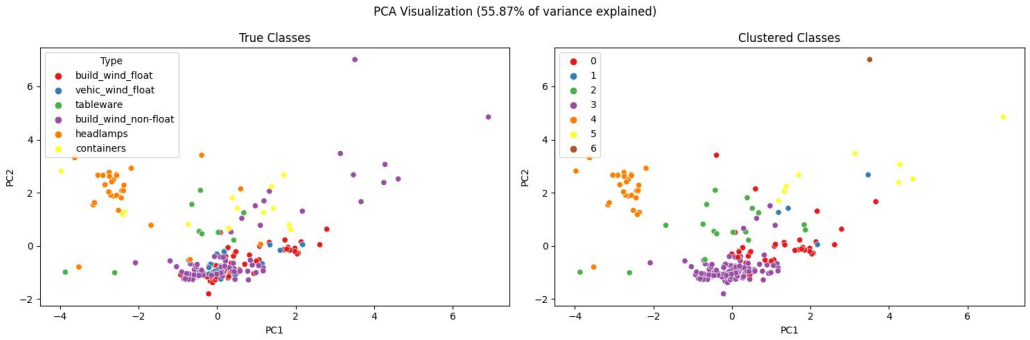

Figure 19: Best Clustering for glass dataset and Birch

5.4.2 Vote

Threshold	Branching Factor	ARI	V-Measure	Silhouette	Calinski-Harabasz
0.1	60	0.6726	0.5786	0.3455	277.5832
0.1	70	0.6726	0.5786	0.3455	277.5832
0.2	60	0.6726	0.5786	0.3455	277.5832
0.2	70	0.6726	0.5786	0.3455	277.5832
0.3	60	0.6726	0.5786	0.3455	277.5832
0.3	70	0.6726	0.5786	0.3455	277.5832
0.4	60	0.6726	0.5786	0.3455	277.5832

Table 11: Top hyperparameters for vote dataset using Birch

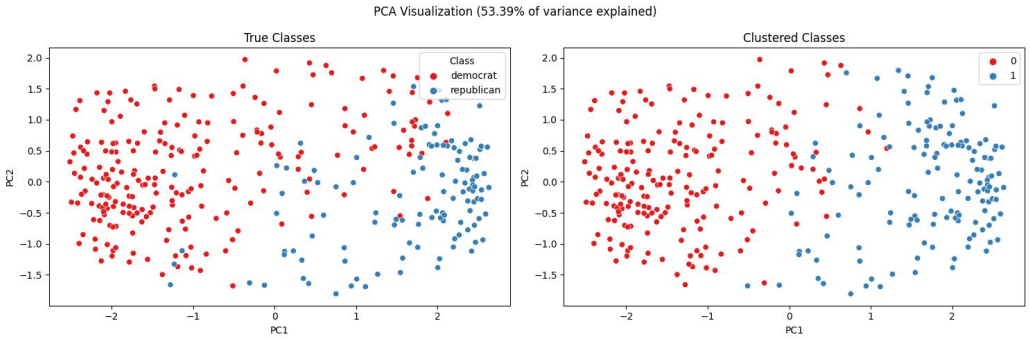

Figure 20: Best Clustering for vote dataset and Birch

5.4.3 Heart-h

Threshold	Branching Factor	ARI	V-Measure	Silhouette	Calinski-Harabasz
0.9	60	0.0299	0.0239	0.3402	394.6382
0.9	50	0.0284	0.0224	0.3422	396.0415
0.9	70	0.0284	0.0224	0.3422	396.0415
0.9	90	0.0284	0.0224	0.3422	396.0415
0.9	100	0.0284	0.0224	0.3422	396.0415
1.0	10	0.0277	0.0619	0.3499	330.2180
0.9	80	0.0272	0.0212	0.3409	395.5992

Table 12: Top hyperparameters for Heart-h dataset using Birch

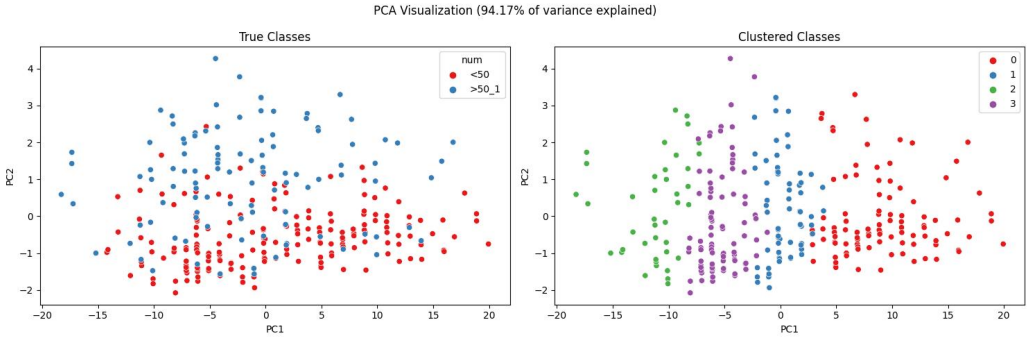

Figure 21: Best Clustering for Heart-h dataset and Birch

5.5 DBSCAN

In this section, the performance of algorithms using different distance metrics is measured. The two candidates chosen are euclidian distance and cosine distance. The first one, is related to the common l^2 geometry distance. THe second one measures the distance between two points from the angle their vectors form.

It should be remarked that, as cosine disimilarity is not a proper distance, given that it doesn't respect triangle inequality, tree based methods for nearest neighbors calculations cannot be computed. This is the reason why the evaluation metrics for ball_tree and kd_tree can't be computed for cosine.

5.5.1 Glass

As it can be observed from Table 13 and Table 14, for glass dataset, there is not a significative difference between the results obtained with euclidean and cosine. This make sense given that Euclidean distance is suitable for data with well-defined clusters, while Cosine distance is more appropriate for high-dimensional and sparse data with varying angles between vectors.

In this case, the dataset is found in a middle point. Some of the clusters are almost convex (e.g. headlamps or build_wind_non_float) but there is also a lot of highly spread data (see Figures 22 and 23).

Algorithm	Clusters	ARI	V-Measure	Silhouette	Calinski-Harabasz
Auto	13	0.1981	0.3576	0.0517	11.0494
Ball Tree	13	0.1981	0.3576	0.0517	11.0494
KD Tree	13	0.1981	0.3576	0.0517	11.0494
Brute	13	0.1981	0.3576	0.0517	11.0494

Table 13: Clustering Evaluation Metrics Euclidean Metric for glass dataset using DBSCAN

Algorithm	Clusters	ARI	V-Measure	Silhouette	Calinski-Harabasz
Auto	4	0.2873	0.4383	0.3605	33.8321
Ball Tree					
KD Tree					
Brute	4	0.2873	0.4383	0.3605	33.8321

Table 14: Clustering Evaluation Metrics Cosine Metric for glass dataset using DBSCAN

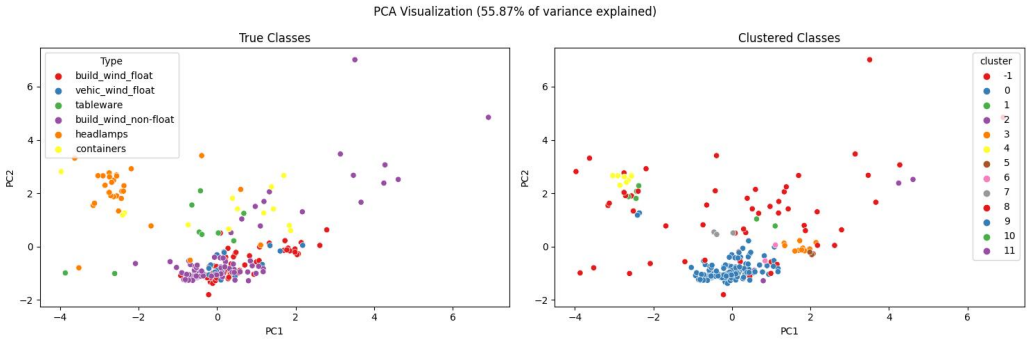

Figure 22: Euclidean Metric Best Clustering for glass dataset and DBSCAN

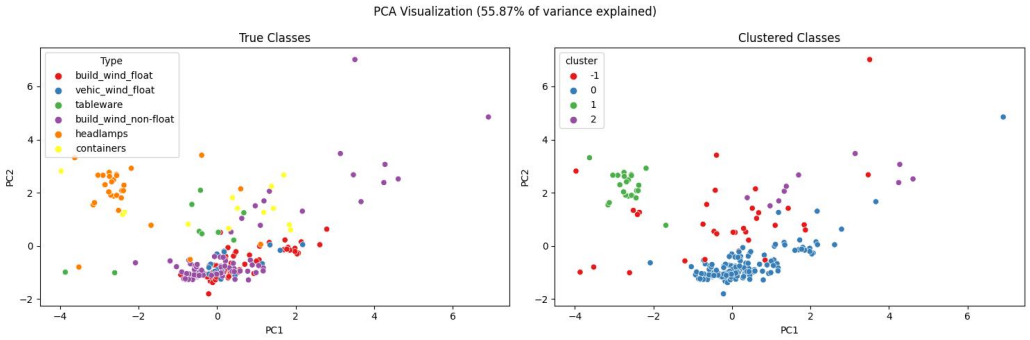

Figure 23: Cosine Metric Best Clustering for glass dataset and DBSCAN

5.5.2 Vote

Vote dataset, is a completely categorical dataset. The use of One Hot Encoding to its variables, pushes it to be a set of sparse vectors (0s and 1s) in high dimensional space. Even if it is not as smooth as a fully numeric dataset, still euclidean distance can be used to calculate the distance to the neighbors.

However, and as it was mentioned before, in this case, it's expected that cosine metric performs quite better than euclidean distance. And it can be clearly observed in Figures 24 and 25.

It detects well the clearest extremes and, with those points whose membership can't be 100% determined, it just considers them low-density area and, hence, noise.

eps	min_samples	metric	algorithm	n_out_clusters	ARI	V-Measure	Silhouette	Calinski-Harabasz
0.1	4	euclidean	auto	16.0	0.1057	0.1870	0.0517	60.6621
0.1	4	euclidean	ball_tree	16.0	0.1057	0.1870	0.0517	60.6621
0.1	4	euclidean	kd_tree	16.0	0.1057	0.1870	0.0517	60.6621
0.1	4	euclidean	brute	16.0	0.1057	0.1870	0.0517	60.6621

Table 15: Clustering Evaluation Metrics Euclidean Metric for vote dataset using DBSCAN

eps	min_samples	metric	algorithm	n_out_clusters	ARI	V-Measure	Silhouette	Calinski-Harabasz
0.1	2	cosine	auto	11.0	0.3758	0.4012	0.2586	51.7923
0.1	2	cosine	ball_tree					
0.1	2	cosine	kd_tree					
0.1	2	cosine	brute	11.0	0.3758	0.4012	0.2586	51.7923

Table 16: Clustering Evaluation Metrics Cosine Metric for vote dataset using DBSCAN

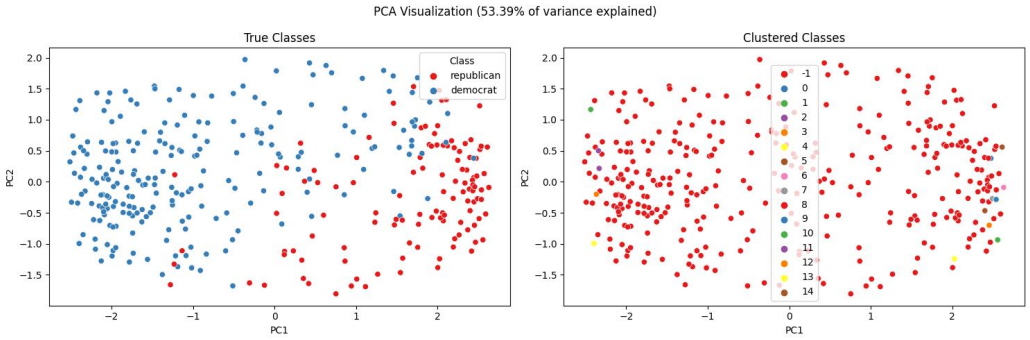

Figure 24: Euclidean Metric Best Clustering for vote dataset and DBSCAN

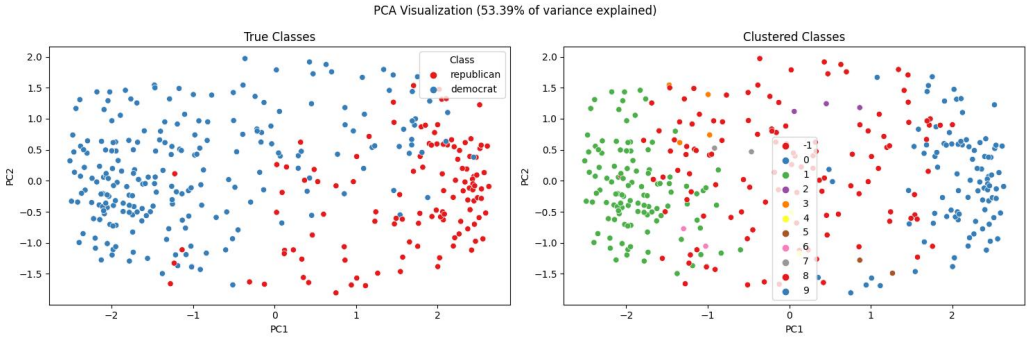

Figure 25: Cosine Metric Best Clustering for vote dataset and DBSCAN

5.5.3 Heart-h

In this case, the mixture of categorical and numerical continuous data, made it impossible to the algorithm to find the perfect radius and number of points. When radius is slightly increased, we have just one big cluster and when it's slightly slower, all of the points are isolated. In this case it's not capable of distinguishing both classes, leading always to one big cluster or just noise.

eps	min_samples	metric	algorithm	n_out_clusters	ARI	V-Measure	Silhouette	Calinski-Harabasz
0.1	3	euclidean	auto	1.0	0.0	0.0	1.0	
0.1	3	euclidean	ball_tree	1.0	0.0	0.0	1.0	
0.1	3	euclidean	kd_tree	1.0	0.0	0.0	1.0	
0.1	3	euclidean	brute	1.0	0.0	0.0	1.0	

Table 17: Clustering Evaluation Metrics Euclidean Metric for heart-h dataset using DBSCAN

eps	min_samples	metric	algorithm	n_out_clusters	ARI	V-Measure	Silhouette	Calinski-Harabasz
0.1	2	cosine	auto	1.0	0.0	0.0	1.0	
0.1	2	cosine	ball_tree					
0.1	2	cosine	kd_tree					
0.1	2	cosine	brute	1.0	0.0	0.0	1.0	

Table 18: Clustering Evaluation Metrics Cosine Metric for heart-h dataset using DBSCAN

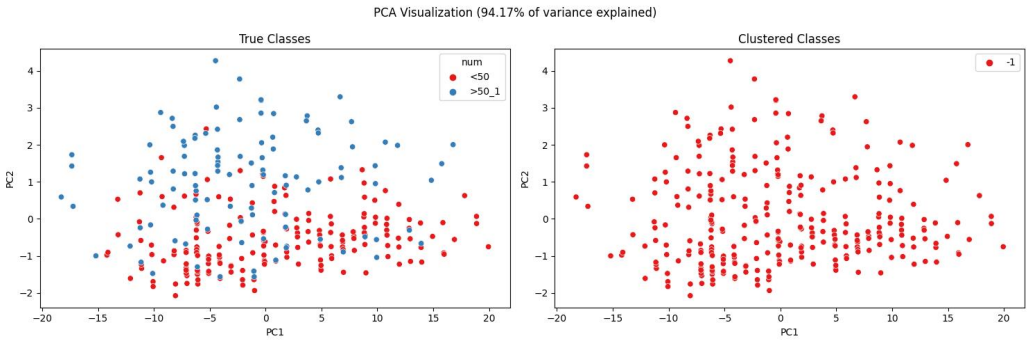

Figure 26: Euclidean Metric Best Clustering for heart-h dataset and DBSCAN

5.6 Fuzzy C-Means

5.6.1 Glass

Figure 28 displays the results of the Fuzzy C-Means clustering applied to the first two principal components of the Glass dataset. The parameters used for this clustering are $c = 2$ clusters and fuzziness $m = 1.2589$. The scatter plot provides a visual representation of the clustering results, showing how data points are distributed across clusters based on their fuzzy membership degrees.

Table 19: Top 4 best hyperparameters for glass dataset using Fuzzy C-Means.

Clusters	Fuzziness (m)	ARI	V-Measure	Silhouette	Calinski-Harabasz
7	1.2589	**0.1916**	**0.3797**	**0.3522**	62.0336
6	1.2589	0.1910	0.3648	0.3373	**63.2301**
9	1.2589	0.1775	0.3590	0.2706	53.5060
8	1.2589	0.1577	0.3425	0.2683	55.2616

Table 19 presents the top four sets of hyperparameters used in the Fuzzy C-Means clustering on the Glass dataset, along with corresponding clustering performance metrics. The table reveals the performance of Fuzzy C-Means with different combinations of the number of clusters and fuzziness values. The best results, as indicated by the highest ARI and V-Measure scores, are achieved with 7 clusters and a fuzziness value of $m = 1.2589$. These results provide insights into the effectiveness of the clustering algorithm for the Glass dataset.

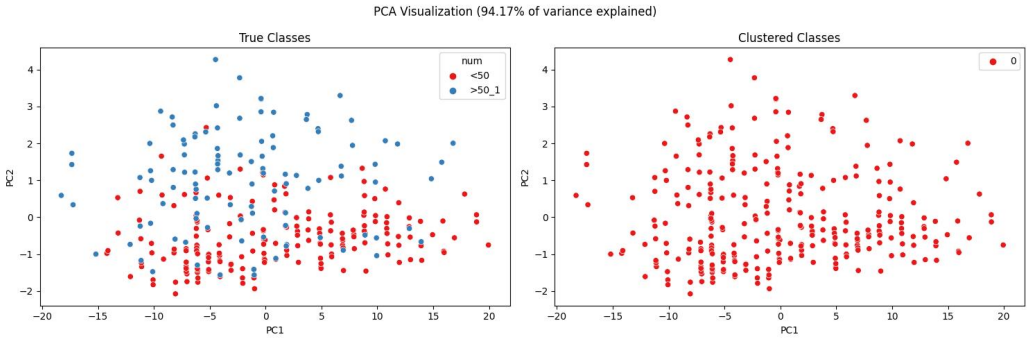

Figure 27: Cosine Metric Best Clustering for heart-h dataset and DBSCAN

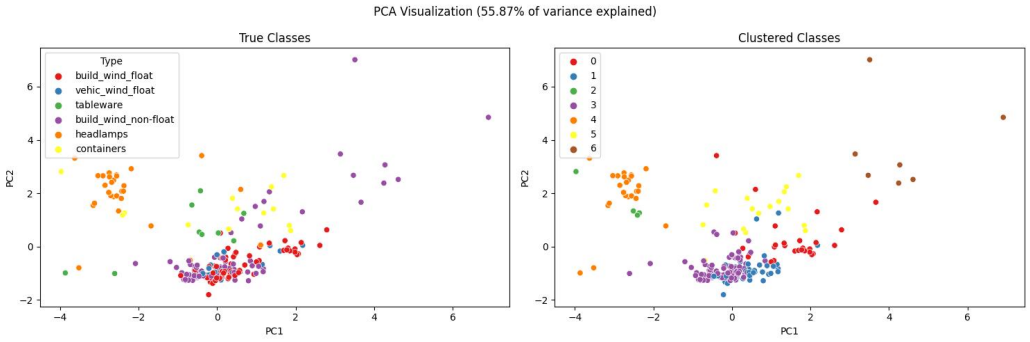

Figure 28: FuzzyCMeans results on the first 2 PCA components on Glass dataset ($c = 2, m = 1.2589$).

5.6.2 Vote

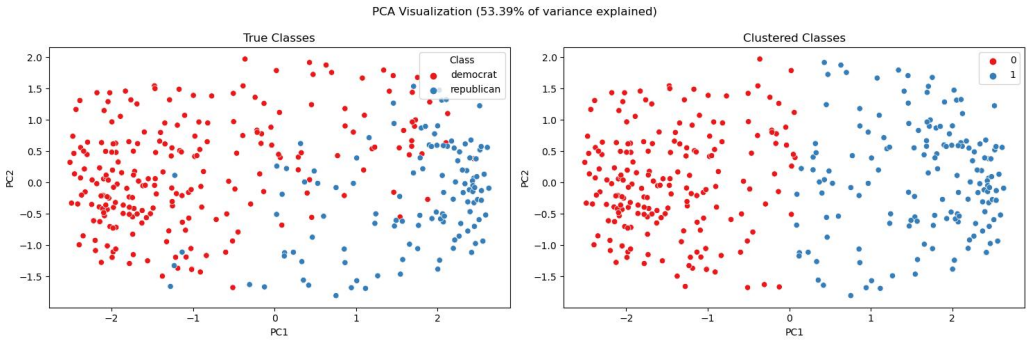

Figure 29: FuzzyCMeans results on the first 2 PCA components on Vote dataset ($c = 2, m = 2.0470$).

Figure 29 shows the results of the Fuzzy C-Means clustering applied to the first two principal components of the Vote dataset. The chosen parameters for this clustering are $c = 2$ clusters and fuzziness $m = 2.0470$.

Table 20: Top 4 best hyperparameters for vote dataset using Fuzzy C-Means.

Clusters	Fuzziness (m)	ARI	V-Measure	Silhouette	Calinski-Harabasz
2	2.0470	**0.5850**	**0.5097**	0.3373	267.7351
2	1.2589	0.5849	0.5047	**0.3375**	**267.8113**
7	23.2631	0.5711	0.5109	**0.3375**	267.3259
9	23.2631	0.5711	0.5109	0.3367	267.3259

Table 20 presents the top four sets of hyperparameters used in the Fuzzy C-Means clustering on the Vote dataset, along with the corresponding clustering performance metrics, as mentioned above. The table reveals the performance of Fuzzy C-Means with different combinations of the number of clusters and fuzziness values for the Vote dataset. The best results are achieved with 2 clusters and a fuzziness value of $m = 2.0470$.

5.6.3 Heart-h

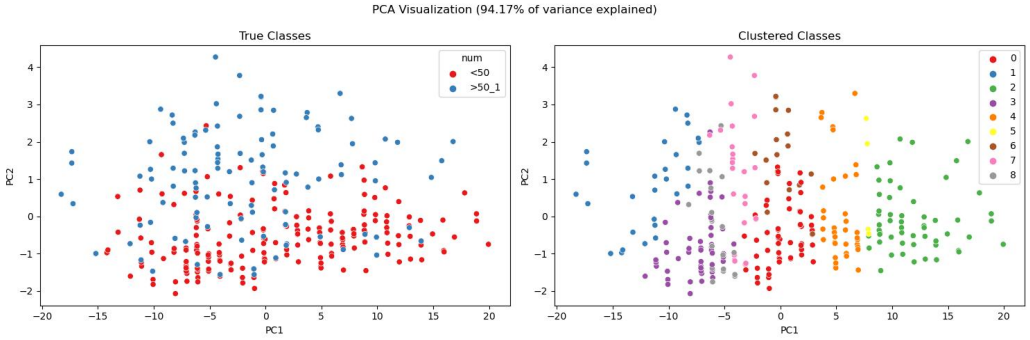

Figure 30: FuzzyCMeans results on the first 2 PCA components on Heart-H dataset ($c = 9, m = 100$).

Figure 30 illustrates the outcomes of Fuzzy C-Means clustering on the first two principal components of the Heart-h dataset. The clustering was conducted using nine clusters ($c = 9$) and a fuzziness parameter of 100 ($m = 100$). The scatter plot visually represents how data points are distributed across clusters, considering their fuzzy membership degrees, which we take the maximum to assign it to a class. Visually, we can see that the results are not so good for this dataset.

Table 21: Top 4 best hyperparameters for Heart-h dataset using Fuzzy C-Means.

Clusters	Fuzziness (m)	ARI	V-Measure	Silhouette	Calinski-Harabasz
9	100	**0.0403**	**0.0579**	0.0876	223.3000
8	1.2589	0.0158	0.0418	**0.2346**	**337.7790**
9	1.2589	0.0151	0.0406	0.2312	323.8973
7	100	0.0174	0.0396	0.1685	275.2451

Table 21 presents an analysis of the top four sets of hyperparameters used in Fuzzy C-Means clustering for the Heart-h dataset, along with the associated clustering performance metrics.

Notably, the highest ARI and V-Measure scores are achieved with nine clusters and a fuzziness parameter of 100. This suggests that in this configuration, the algorithm provides a better representation of the underlying patterns in the Heart-h dataset. However, it's essential to note that the Silhouette score and the Calinski-Harabasz index reach their peaks with different hyperparameter settings, which indicates that the choice of hyperparameters may depend on specific objectives.

In summary, the Fuzzy C-Means clustering analysis on the Heart-h dataset reveals varying levels of performance depending on the chosen hyperparameters. The results presented in the scatter plot and table offer valuable insights into the potential groupings within the dataset, providing a foundation for further exploration and interpretation.d

6 Conclusions

Having seen the results of the hyperparameter tuning in each algorithm presented, we will now go through an overview of the metrics seen in each dataset and extract the pertaining conclusions from these.

6.1 Glass

After evaluating several clustering algorithms on the Glass dataset, distinct trends in performance emerge. K-Means demonstrated the highest efficacy, particularly with 9 clusters, achieving an Adjusted Rand Index (ARI) of 0.2122 and a V-Measure of 0.3719. This indicates that K-Means was able to delineate clusters that closely align with the ground truth, showcasing its ability to effectively group similar data points. Additionally, the moderate Silhouette score of 0.4312 implies reasonably well-defined clusters. Conversely, K-Modes exhibited a weaker performance with the highest ARI and V-Measure scores only reaching 0.0062 and 0.1520, respectively. The negative Silhouette score (-0.0751) indicates overlapping clusters, suggesting that K-Modes may not be the most suitable choice for this dataset.

PAM demonstrated competitive performance, with different distance metrics yielding similar results. The best configuration, utilizing 9 clusters, yielded a V-Measure of approximately 0.34 and a Silhouette score ranging from 0.34 to 0.39. These scores suggest that PAM produced well-separated clusters of similar sizes. BIRCH, with various hyperparameters, consistently performed well. The most effective configuration utilized a threshold of 0.6 and a branching factor of 50 or greater, resulting in an ARI of approximately 0.2925 and a V-Measure of around 0.44. The Silhouette score ranged from 0.3615 to 0.4500, indicating well-defined clusters.

On the other hand, DBSCAN demonstrated lower performance with an ARI of 0.1981 and a V-Measure of 0.3576. These metrics suggest that DBSCAN's clusters did not align as closely with the ground truth. Finally, Fuzzy C-Means showcased competitive results, particularly with 7 clusters and a fuzziness parameter (m) of 1.2589. The ARI of 0.1916 and the V-Measure of 0.3797 indicate relatively close alignment with the true clusters. The Silhouette score of 0.3522 suggests reasonably well-defined clusters.

6.2 Vote

The clustering algorithms applied to the Vote dataset yielded distinct performance trends. K-Means and K-Modes demonstrated notable effectiveness in capturing underlying patterns within the data. K-Means, with 2 clusters, exhibited a particularly strong performance, achieving an Adjusted Rand Index (ARI) of 0.5710 and a V-Measure of 0.4942. These scores indicate that K-Means was able to create clusters that closely align with the ground truth. The Silhouette score of 0.3533 suggests reasonably well-defined clusters, and the high Calinski-Harabasz score of 290.5112 further validates the effectiveness of this configuration. Similarly, K-Modes showcased strong performance with the highest ARI and V-Measure scores reaching 0.5571 and 0.4892, respectively. The Silhouette score of 0.3504 and the Calinski-Harabasz score of 286.7041 reinforce the effectiveness of K-Modes on this dataset.

PAM demonstrated competitive performance with various distance metrics. The best configuration used 2 clusters with the Euclidean distance metric, resulting in a V-Measure of 0.3038 and a Silhouette score of 0.4362. While not as high as K-Means or K-Modes, these metrics indicate reasonable cluster separation. BIRCH, on the other hand, exhibited exceptional performance. Utilizing a threshold of 0.1 and a branching factor of 60 or 70 resulted in an ARI and V-Measure of 0.6726, showcasing strong alignment with the true clusters. The Silhouette score of 0.3455 and the Calinski-Harabasz score of 277.5832 further affirm the effectiveness of this configuration.

In contrast, DBSCAN demonstrated relatively lower performance compared to other algorithms on this dataset. The highest ARI and V-Measure scores were achieved using a combination of parameters, resulting in 0.1057 for both metrics. These scores indicate that the clusters produced by DBSCAN do not align as closely with the ground truth. Fuzzy C-Means showcased competitive performance, particularly with 2 clusters and a fuzziness parameter (m) of 2.0470. It achieved a high ARI of 0.5850 and a V-Measure of 0.5097, indicating close alignment with the true clusters. The Silhouette score of 0.3373 and the Calinski-Harabasz score of 267.7351 further support the effectiveness of this configuration.

6.3 Health-h

The clustering algorithms applied to the Health-h dataset demonstrated varying levels of effectiveness. K-Modes outperformed K-Means, achieving higher Adjusted Rand Index (ARI) and V-Measure scores. Specifically, K-Modes with 2 clusters achieved an ARI of 0.3140 and a V-Measure of 0.2166, indicating relatively better

alignment with the true clusters. PAM, when using the Euclidean distance metric, demonstrated reasonable cluster separation with a V-Measure of 0.01668 and a Silhouette score of 0.5171, though the overall performance remained modest. BIRCH, on the other hand, exhibited relatively lower performance, achieving a maximum ARI of 0.0299 and V-Measure of 0.0239. The clusters produced by DBSCAN did not align well with the ground truth clusters, indicating limited effectiveness on this dataset.

Fuzzy C-Means showed mixed results, with the best configuration using 9 clusters and a fuzziness parameter (m) of 100. While achieving an ARI of 0.0403 and a V-Measure of 0.0579, it indicated only limited alignment with the true clusters. The Silhouette score of 0.0876 suggests that clusters are not well-defined, and the Calinski-Harabasz score of 223.3000 further supports the limited effectiveness of this configuration.

7 Further Work

- Use a cross-validation scheme to evaluate the performance of different hyperparameter settings.
- Use a grid search to evaluate a large number of hyperparameter combinations.
- Use a visualization tool to explore the relationships between different hyperparameters and the clustering results.