# Implementation, Evaluation, and Comparison of K-Means and Other Clustering Algorithms

Carlos Jiménez, Sheena Lang, Zachary Parent, and Kacper Poniatowski

December 1, 2024

# 1 Introduction

This report focuses on evaluating various clustering techniques across three distinct datasets: Mushroom, Vowel, and Hepatitis, as detailed in Section 2. The clustering algorithms examined include K-Means, Fuzzy K-Means, OPTICS, Spectral Clustering, Global K-Means, and GMeans, as described in Sections 4.1 to 4.5 . We perform multiple experimental runs with different hyperparameters and assess the performance of each method using four evaluation metrics: two internal metrics (Davies-Bouldin Index and Calinski-Harabasz Index) and two external metrics (Adjusted Rand Index and F-Measure) as seen in 4.6. The results and analysis of these experiments are presented in Section 5.

# 2 Data

This section describes the datasets used in our study and details the preprocessing steps applied to prepare them for analysis.

# 3 Dataset Selection and Characteristics

This section outlines our dataset selection criteria and analyzes the characteristics of the chosen datasets for evaluating clustering algorithms.

## 3.1 Dataset Selection

In this project, we aimed to select three datasets that offer substantial variability across different aspects to evaluate the performance of various clustering algorithms, such as K-Means, G-Means, OPTICS, gs-FCM, etc.

The criteria for dataset selection were centered around diversity in dataset size, attribute types, class distribution, and missing data patterns. These factors allowed us to thoroughly analyze the efficiency and effectiveness of each clustering algorithm under different scenarios.

**Dataset size variation** To examine the scalability of the clustering algorithms, we selected one small dataset (Hepatitis), one medium-sized dataset (Vowel), and one large dataset (Mushroom). This setup enables us to assess the computational requirements and clustering performance of the algorithms across varying dataset sizes.

**Attribute type diversity** We chose datasets with diverse attribute types, including nominal, numerical, and mixed attributes. This ensures that the clustering algorithms are evaluated for their ability to handle varying data representations and preprocessing requirements.

**Class distribution** Although clustering is an unsupervised learning task, the class distributions of the datasets can serve as a baseline for validating the quality of clusters. We included one dataset with balanced classes (Vowel), one with slightly imbalanced classes (Mushroom), and one with a heavily imbalanced distribution (Hepatitis).

**Missing data patterns** We considered datasets with varying levels of missing data. The Hepatitis dataset contains missing values, challenging the algorithms to handle incomplete information, while the Vowel and Mushroom datasets are complete, ensuring a comparison of clustering performance in the absence of missing data.

## 3.2 Dataset Characteristics

- **Mushroom Dataset**
  - 8,124 instances
  - 22 nominal attributes
  - Binary classification (edible vs. poisonous, used for validation purposes)
  - Nearly balanced classes
  - No missing values
- **Hepatitis Dataset**
  - 155 instances
  - Mixed attribute types (13 nominal and 6 numerical)
  - Binary classification (survive vs. die, used for validation purposes)
  - Imbalanced classes (79.35% majority class)
  - 6.01% missing values
- **Vowel Dataset**
  - 990 instances
  - Mixed attribute types (3 nominal and 10 numerical)
  - Multi-Class classification (11 classes, used for validation purposes)
  - Balanced classes
  - No missing values

## 3.3 Selection Criteria

We selected the Mushroom, Hepatitis, and Vowel datasets to provide complementary characteristics for evaluating clustering algorithms. Our selection criteria focused on:

- **Dataset size variation** (small, medium, large)
- **Attribute type diversity** (nominal, numerical, mixed)

## 3.4 Data Preprocessing

This section outlines the preprocessing steps applied to prepare the three datasets (*Hepatitis*, *Mushroom*, and *Vowel*) for clustering analysis.

### 3.4.1 Data Cleaning and Feature Handling

We began by replacing all missing values, denoted as `?`, with `NaN` to facilitate appropriate imputation techniques. The class label column, required only for post-clustering evaluation, was temporarily removed during preprocessing.

To account for the varying characteristics of the datasets, we differentiated between categorical and numerical features using a heuristic-based approach. Columns were classified as categorical if their data type was `object` or if the proportion of unique values was below 5% of the dataset's size. All other columns were treated as numerical.

### 3.4.2 Numerical Feature Processing

Numerical features were processed using a pipeline that:

- Imputed missing values with the column mean to preserve overall data distribution.
- Rescaled features to the [0, 1] range using Min-Max scaling, ensuring equal contribution of features during distance-based clustering.

### 3.4.3 Categorical Feature Processing

Categorical features were handled as follows:

- Missing values were imputed with the most frequent category (mode), preserving the dominant data patterns.
- Binary categorical features (those with two unique values) were encoded using label encoding.
- Non-binary categorical features were one-hot encoded, creating separate columns for each category. This approach was selected based on literature recommendations for clustering algorithms like K-Means, where numerical representations avoid introducing arbitrary ordinal relationships.

### 3.4.4 Final Dataset Preparation

The processed numerical and categorical features were concatenated to form the final dataset. Additionally, the class labels were label-encoded for use in evaluation metrics such as Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI).

**Preprocessing Pipeline Overview**:

- Missing values: Imputed with mean (numerical) and mode (categorical).
- Numerical features: Min-Max scaling.
- Categorical features: Label encoding (binary) and one-hot encoding (non-binary).

This approach ensures that the preprocessing pipeline is tailored to the unique requirements of clustering algorithms while maintaining data integrity and minimizing potential biases.

# 4 Methods

This section provides an overview of the clustering algorithms used in our analysis, detailing their mechanisms, key parameters, and parameter variations. The methods include K-Means (Section 4.1), Improved K-Means (Section 4.2), Fuzzy C-Means (Section 4.3), OPTICS (Section 4.4), and Spectral Clustering (Section 4.5).

## 4.1 K-Means

K-Means is one of the most widely used clustering algorithms due to its simplicity and efficiency. It partitions a dataset into a predefined number of clusters by iteratively refining cluster centroids based on the distance between data points and the centroids.

### 4.1.1 Mechanism

The K-Means algorithm operates by minimizing the within-cluster variance, which is defined as the sum of squared distances between each point and the centroid of its assigned cluster. The process begins with the initialization of $k$ centroids, where $k$ represents the number of clusters. These centroids can either be selected randomly or provided as input.

Following initialization, each data point $x_i$ is assigned to the nearest centroid $c_j$ based on a distance metric, typically the Euclidean distance. This assignment is computed using the formula:

$$\text{Cluster}(x_i) = \arg\min_j ||x_i - c_j||_2^2.$$

After assigning clusters, the centroids are updated by recalculating their positions as the mean of all points assigned to each cluster. The new centroid $c_j$ is determined using the equation:

$$c_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i,$$

where $C_j$ is the set of points in cluster $j$, and $|C_j|$ is the number of points in that cluster.

Finally, the algorithm checks for convergence by comparing the updated centroids to the previous ones. If the difference between the new and old centroids is less than a predefined tolerance ($\epsilon$), or if the maximum number of iterations is reached, the algorithm terminates. The difference is computed as:

$$\Delta = \sum_{j=1}^{k} ||c_j^{(t)} - c_j^{(t-1)}||_2^2 < \epsilon.$$

K-Means is computationally efficient but sensitive to the initial positions of centroids, which can cause it to converge to a local minimum.

### 4.1.2 Parameter Grid

K-Means involves several important parameters, which include:

- **Number of Clusters**: Determines the number of clusters ($k$) to form.
- **Maximum Iterations**: The maximum number of iterations allowed for convergence.
- **Tolerance** ($\epsilon$): The convergence threshold based on the change in centroids.
- **Random State**: Controls the random seed for reproducibility of results.

The parameter variations used in our experiments are summarized in Table 1.

## 4.2 Improved K-Means

This section provides an overview of the improved K-Means algorithms implemented, Global K-Means and G-Means.

Table 1: K-Means Parameter Grid

| **Parameter** | **Values** |
|---|---|
| Number of Clusters | 2, 3, 4, 5, 6, 8, 10 |
| Maximum Iterations | 100, 300, 500 |
| Tolerance ($\epsilon$) | $1 \times 10^{-5}, 1 \times 10^{-4}, 1 \times 10^{-3}$ |
| Random State | 1, 2, 3, 4, 5 |

### 4.2.1 Global K-Means

The global K-Means algorithm is an advanced clustering approach that improves upon the traditional K-Means by introducing an intelligent centroid initialization and incremental clustering strategy. Our implementation focuses on addressing key limitations of standard clustering techniques through a novel candidate selection and optimization mechanism.

#### Mechanism

The Global K-Means algorithm operates through a progressive clustering process with several key improvements:

The clustering process starts with a single cluster by using standard K-Means with $k = 1$, and incrementally adds clusters. For each iteration, the optimal location for the new centroid is determined by minimising the Within-Cluster Sum of Squares (WCSS). Candidate points for the new centroid are selected based on their minimum distance from existing centroids, with the number of candidates dynamically adjusted based on the current cluster count. An efficient selection method is used via `np.argpartition`, ensures scability.

The algorithm refines centroid placement by using vecotorised WCSS computation, reducing the computational complexity of the algorithm by simultaneously calculating the WCSS for all candidates. The configuration with the lowest WCSS is selected as the optimal solution for the current cluster count.

A unique aspect of this implementation is the caching mechanism. Intermediate results such as cluster labels, centroids, and distance matrices are stored persistently. This allows the algorithm to resume from cached states for higher values of $k$, thus reducing computational overhead dramatically by preventing recomputations of lower values of $k$. The caching mechanism is hash-based, ensuring compatibility with different datasets and configurations.

The implementation also includes an adaptive candidate reduction strategy. As the number of clusters increases, the number of candidate points are reduced to prevent the algorithm from becoming computationally infeasible. This adaptive strategy ensures that the algorithm remains efficient and scalable for large datasets, while maintaining high-quality clustering results.

### Parameter Grid

The Global K-Means algorithm is highly configurable, with several key parameters that can be tuned to optimize performance:

- **Number of Clusters ($n_{\textbf{clusters}}$)**: Determines the number of clusters ($k$) to form.
- **Maximum Iterations ($max_{\textbf{iterations}}$)**: The maximum number of iterations allowed for convergence.
- **Tolerance ($\epsilon$)**: The convergence threshold based on the change in centroids.
- **Random State**: Controls the random seed for reproducibility of results.

The parameter grid for Global K-Means is shown in Table 2.

Table 2: Global K-Means Parameter Configuration

| **Parameter** | **Values** |
|---|---|
| Number of Clusters | [2, 3, 5, 10, 11, 12] |
| Maximum Iterations | [100, 300, 500] |
| Convergence Tolerance | $\{1\times10^{-5}, 1\times10^{-4}, 1\times10^{-3}\}$ |
| Random State | [1, 2, 3, 4, 5] |

### 4.2.2 G-Means

The G-Means algorithm is an advanced clustering techniques that improves upon the standard K-Means algorithm by addressing a fundamental limitation: the need to specify a predetermined number of clusters. G-Means dynamically determines the optimal number of clusters by recursively splitting clusters based on statistically validating their Gaussian distribution.

### Mechanism

The algorithm begins the clustering process by initialising a single cluster, obtained using standard K-Means with $k = 1$. For each cluster, the data points are split into two subclusters by applying K-Means with $k = 2$. The resulting clusters are then evaluated using Anderson-Darling Gaussianity test. This test evaluates whether the data points in the cluster are Gaussian distributed.

If the test indicates both subclusters are Gaussian distributed, the split is rejected and the original cluster remains unchanged. If at least one of the subclusters are not Gaussian distributed, the split is accepted and the process is repeated recursively for each subcluster until all clusters are Gaussian distributed, or the user-defined maximum depth is reached.

Before applying the Gaussianity test, the algorithm projects the data onto the principal components using Principal Component Analysis (PCA) to ensure the Anderson-Darling test is applied to the most significant directions in the data.

To prevent over-segmentation, a minimum number of observations ($\min_{\text{obs}}$) is enforced for each cluster. If a cluster has fewer observations than the minimum threshold, it is not split further.

### Parameter Grid

The G-Means algorithm offers several configurable parameters to fine-tune its clustering behavior:

- **Strictness ($s$)**: Controls the sensitivity of the Gaussianity test.
- **Minimum Observations ($\min_{\textbf{obs}}$)**: Prevents splitting clusters with too few data points.
- **Maximum Depth ($\max_{\textbf{depth}}$)**: Limits the recursive splitting process.
- **Random State**: Ensures reproducibility of results.

The parameter grid for Global K-Means is shown in Table 3.

Table 3: G-Means Parameter Configuration

| **Parameter** | **Values** |
|---|---|
| Strictness | [0, 1, 2, 3, 4] |
| Minimum Observations | [1, 5, 10] |
| Maximum Depth | [5, 10, 15] |
| Random State | [1, 2, 3, 4, 5] |

## 4.3 Fuzzy C-Means

To be more precise, we implemented the ***Generalized Suppressed Fuzzy C-Means*** (gs-FCM) algorithm, which extends the Suppressed Fuzzy C-Means (s-FCM) approach by introducing a time-invariant, context-sensitive suppression rule [5, 4].

The s-FCM algorithm itself incorporates a constant suppression factor to enhance clustering performance. Intuitively, the gs-FCM algorithm retains the same primary objective as the original Fuzzy C-Means (FCM) algorithm: to partition a dataset into a predefined number of clusters, allowing data points to belong to multiple clusters with varying degrees of membership [2]. The addition of the suppression mechanism improves convergence efficiency and robustness, particularly in scenarios with imbalanced or noisy data.

### 4.3.1 Mechanism

Initially, the user specifies the number of clusters $c$ and sets the fuzzy exponent $m > 1$, which controls the degree of fuzziness. Cluster prototypes are then initialized, either by applying intelligent initialization principles or by randomly selecting input vectors. A suppression rule and its corresponding parameter are chosen from predefined options, typically referenced in a lookup table. For this implementation, the suppression rule chosen is defined as follows:

$$\alpha_k = \frac{1}{1 - u_w + u_w \cdot (1 - \text{param})^{\frac{2}{1-m}}},$$

where $u_w$ is the fuzzy membership of the winning cluster, param is the suppression parameter, and $m$ is the fuzzy exponent.

At each iteration, fuzzy memberships are calculated using the standard FCM formula. For each data point $\mathbf{x}_k$, the algorithm determines the winning cluster (i.e., the cluster prototype closest to $\mathbf{x}_k$) and calculates the suppression rate $\alpha_k$ using the chosen rule. Suppressed fuzzy memberships are then computed using a modified formula that incorporates $\alpha_k$, effectively reducing the influence of over-represented data points.

Cluster prototypes are updated using the suppressed memberships, ensuring that the influence of individual data points on the cluster centers aligns with the suppression mechanism. These steps are repeated iteratively until convergence, typically measured by the norm of the variation in cluster prototypes between successive iterations.

This suppression mechanism enhances the robustness of the clustering process, addressing imbalances in data distribution and improving the interpretability and quality of the clustering results.

### 4.3.2 Parameter Grid

gs-FCM involves the following parameters:

- **Number of Clusters**: Determines the number of clusters ($k$) to form.
- **Fuzzyness**: Controls the degree of fuzziness in the membership function.

* It's important to add that there's a suppression factor $\alpha$ that varies for each data point (i.e.m it is context/data sensitive) following a pre-defined suppresion rule.

The parameter variations used in our experiments are summarized in Table 4.

Table 4: gs-FCM Parameter Grid

| **Parameter** | **Values** |
|---|---|
| Number of Clusters | 2, 3, 5, 10, 11, 12 |
| Fuzzyness | 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0 |
| Suppression Factor | Varies for each data point (context/data sensitive) |

By tuning these parameters, gs-FCM can adapt to various data distributions and noise levels, making it a powerful tool for clustering complex datasets.

## 4.4 OPTICS

OPTICS (Ordering Points To Identify the Clustering Structure) is a density-based clustering algorithm that creates an augmented ordering of data points to identify cluster structures [1].

### 4.4.1 Mechanism

OPTICS works by computing a special ordering of points in the dataset based on their density-reachability relationships. The algorithm introduces two key concepts: core-distance and reachability-distance.

The core-distance represents how dense the neighborhood around a point is, measured as the minimum radius needed to classify a point as a core point (containing a minimum number of neighbors). If a point doesn't have enough neighbors, its core-distance is undefined.

The reachability-distance between two points measures how close they are from a density-connectivity perspective. It considers both the direct distance between points and the density of the neighborhood they're in, ensuring that points in dense regions are considered more closely connected than those in sparse regions.

The algorithm processes points in a specific order, maintaining a priority queue ordered by reachability-distance. For each point:

1. The core-distance is computed
2. For each unprocessed neighbor, the reachability-distance is calculated
3. Points are added to the priority queue based on their reachability-distance
4. The process continues with the point having the smallest reachability-distance

This ordering produces a reachability plot, where valleys in the plot represent clusters. The $\xi$ parameter is used to identify significant drops in reachability that indicate cluster boundaries. The minimum cluster size parameter ensures that identified clusters have a meaningful number of points.

Unlike traditional clustering algorithms, OPTICS does not explicitly produce clusters but rather provides a density-based ordering that can be used to extract clusters of varying density. This makes it particularly effective at finding clusters of arbitrary shape and identifying noise points in the dataset.

### 4.4.2 Parameter Grid

OPTICS involves several important parameters, which include:

- **Metric**: The distance metric used for calculating point distances.
- **Algorithm**: The algorithm used to compute the nearest neighbors.
- **Minimum Samples** ($min_{\textbf{samples}}$): The number of samples in a neighborhood for a point to be considered a core point.
- **Xi** ($\xi$): The minimum steepness on the reachability plot that constitutes a cluster boundary.
- **Minimum Cluster Size** ($min_{\textbf{cluster}}$): The minimum number of samples in a cluster.

The parameter variations used in our experiments are summarized in Table 5.

Table 5: OPTICS Parameter Grid

| **Parameter** | **Values** |
|---|---|
| Metric | euclidean, manhattan |
| Algorithm | auto, ball_tree |
| Minimum Samples | 5, 10, 20 |
| Xi | 0.01, 0.05, 0.1 |
| Minimum Cluster Size | 5, 10, 20 |

## 4.5 Spectral Clustering

Spectral clustering is a technique that performs dimensionality reduction before clustering by using the eigenvectors of a similarity matrix [3].

### 4.5.1 Mechanism

The algorithm proceeds in three main steps:

1. Constructs a similarity matrix using either RBF kernel or k-nearest neighbors to capture relationships between data points
2. Computes the normalized Laplacian matrix and extracts its eigenvectors, creating a lower-dimensional representation that emphasizes cluster structure
3. Applies either k-means or cluster_qr to this transformed space to obtain the final clustering

This approach is particularly effective at identifying clusters of arbitrary shape, as it does not make assumptions about the geometric structure of the clusters.

### 4.5.2 Parameter Grid

Spectral Clustering involves several important parameters:

- **Number of Neighbors** ($n_{\textbf{neighbors}}$): The number of neighbors for affinity matrix construction.
- **Affinity**: The type of affinity matrix to construct.
- **Eigen Solver**: The eigenvalue decomposition strategy.
- **Assign Labels**: The strategy for assigning labels in the embedding space.
- **Random State**: Controls reproducibility of results.

The parameter variations used in our experiments are summarized in Table 6.

Table 6: Spectral Clustering Parameter Grid

| **Parameter** | **Values** |
|---|---|
| Number of Neighbors | 5, 10, 20 |
| Affinity | nearest_neighbors, rbf |
| Eigen Solver | arpack, lobpcg |
| Assign Labels | kmeans, cluster_qr |
| Random State | 1, 2, 3, 4, 5 |

### 4.6 Evaluation Metrics

We evaluated our clustering approaches using two internal and two external metrics to ensure a comprehensive assessment by capturing different aspects of clustering performance. The **Davies-Bouldin Index (DBI)** measures the compactness and separation of clusters, where lower values indicate better clustering. The **Calinski-Harabasz Index (CHI)** evaluates the ratio of between-cluster variance to within-cluster variance, favoring solutions with well-separated, compact clusters; higher scores are better. These two internal metrics together provide insights into the geometric quality of clusters.

On the other hand, external metrics assess alignment with ground truth labels. The **Adjusted Rand Index (ARI)** quantifies the agreement between true and predicted labels while correcting for chance, making it robust for comparing different clustering solutions. The **F-measure**, derived from precision and recall, evaluates how well the predicted clusters capture the true ones, focusing on class overlap. Together, these external metrics assess clustering accuracy and consistency, while complementing the internal metrics by incorporating the dataset's known structure.

## 5 Results and Analysis

### 5.1 K-Means

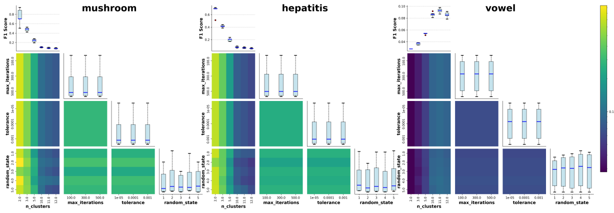

Figure 1: Parameter Interactions for K-Means Clustering.

K-Means demonstrated strong performance on the Mushroom and Hepatitis datasets. For Mushroom, it achieved an F-measure of 0.94 with the optimal parameters, highlighting its effectiveness. However, its performance on the Vowel dataset was poor, with a best F-measure of only 0.17. This low performance can likely be attributed to the dataset's higher optimal cluster count (11). As depicted in Figure 1, the optimal number of clusters varied across datasets. For Mushroom and Hepatitis, the highest F-measure was obtained with 2 clusters, matching the number of classes in these datasets. In contrast, for the Vowel dataset, the optimal result was achieved with 8 clusters, falling short of the actual number of classes (11). The inability to correctly identify the full number of clusters may explain the lower performance.

While variations in parameters such as the maximum number of iterations and tolerance had minimal effect on the F-measure, the choice of the random state significantly influenced the results. For instance, in the Mushroom dataset, the F-measure reached 0.94 with one random state but dropped dramatically to around 0.46 with another. This underscores the critical role of testing different initializations in K-Means clustering to ensure robust results.

### 5.2 Improved K-Means

This section provides an overview of the results obtained from the improved K-Means algorithms implemented, Global K-Means and G-Means.

#### 5.2.1 Global K-Means

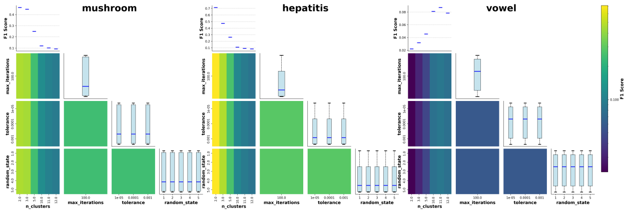

Figure 2: hyperparameter interactions for Global K-Means

We found that the number of clusters had the most significant impact on the F-measure, as shown in Figure 2. For both the Mushroom and Hepatitis datasets, the F1 score was highest when number of clusters set to 2, and steadily decreased as the number of clusters increased. For the Vowel dataset, the F1 score was lowest with a small number of clusters, and increased as the number of clusters increased, peaking at number of clusters set to 11. These results are consistent with the number of classes in each dataset—2 for Mushroom and Hepatitis, and 11 for Vowel.

The tolerance and $max_iteration$ parameters both had a negligable impact on the F1 score. One possible explanation for this is that, in Global K-Means, the tolerance parameter primarily influences the convergence criteria of the K-Means subroutine used at each step of the algorithm. As long as K-Means converges successfully for a given number of clusters, the overall clustering results remain largely unaffected. Similarly, the $max_iterations$ parameter acts as a safeguard to prevent excessively long runtimes but does not directly influence the final cluster placements, provided that convergence is reached within the allowed iterations. These findings suggest that factors such as the initial placement of centroids and the maximum number of clusters are more critical to the performance of Global K-Means than the tolerance or max iterations parameters.

#### 5.2.2 G-Means

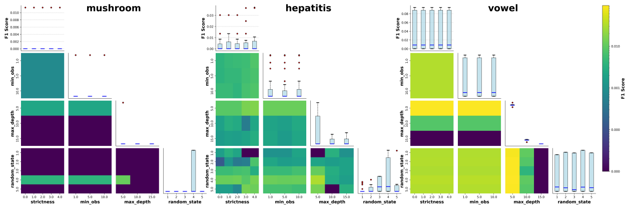

Figure 3: hyperparameter interactions for G-Means

Compared to Global K-Means, the results are more varied for G-Means as shown in Figure 3.

The F1 score for the Mushroom dataset remains consistently low across all configurations of the hyperparameters. While there are some outliers, the overall clustering performance is poor, with F1 scores close to zero for most parameter combinations. The heatmaps show little variation across hyperparameter values, suggesting that changes to $max_depth$, strictness, or $min_obs$ have negligible impact on the clustering outcome. The bar plot for $random_state$ indicates some sensitivity to initialization, but even the best cases fail to produce meaningful clusters. These results suggest that G-Means is ill-suited for the Mushroom dataset.

For the Hepatitis dataset, the F1 score shows greater variability across hyperparameter configurations. The heatmaps reveal that $max_depth$ and $random_state$ have more noticeable impacts on clustering performance, with specific combinations producing better results. The box plots indicate significant outliers, particularly for strictness and $max_depth$, suggesting these parameters introduce instability. Overall, these results show G-Means hyperparameter combinations are sensitive for this dataset, highlighting the need for careful selection to optimize performance. Despite this, the F1 scores remain low, indicating G-Means cannot accurately predict the clusters in the Hepatitis dataset.

The F1 score remains consistently low across all hyperparameter configurations for the Vowel dataset. The heatmaps suggest that $max_depth$ is the most significant hyperparameter, with lower levels producing the best results. Having said this, the results are still poor.The box plots show negligible variability for most parameters, with no significant outliers. These results indicate that G-Means struggles to produce meaningful clusters for this dataset, likely due to the dataset's complexity and the algorithm's limitations in handling high-class scenarios.

### 5.3 Fuzzy C-Means

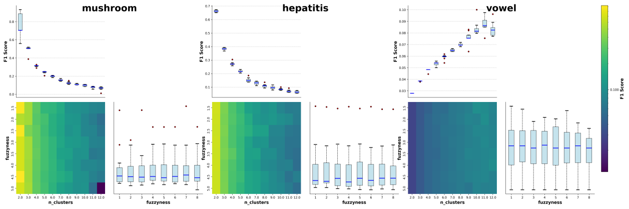

Figure 4: Parameter interactions for gs-FCM

Figure 4 presents the interaction effects between the key parameters of the Fuzzy C-Means clustering algorithm, namely the number of clusters (*n_clusters*) and the fuzziness coefficient, across three datasets: *hepatitis*, *mushroom*, and *vowel*. Each grid provides two types of visualizations: boxplots illustrating the variation in F1 score across different *n_clusters* values and heatmaps depicting the interaction between *n_clusters* and the fuzziness coefficient.

The boxplots indicate that, for all datasets, the performance of the clustering algorithm (measured in terms of F1 score) is sensitive to the choice of *n_clusters*. For the *hepatitis* dataset, the F1 score decreases steadily as the number of clusters increases, suggesting that simpler models (with fewer clusters) yield better results. This trend is less pronounced for the *mushroom* dataset, where the F1 score stabilizes at a relatively high level as *n_clusters* increases. Conversely, the *vowel* dataset demonstrates an improvement in F1 score with larger *n_clusters*, indicating that more complex clustering structures are better suited for this dataset.

The heatmaps reveal nuanced interactions between *n_clusters* and the fuzziness parameter. For the *hepatitis* dataset, lower values of fuzziness (closer to

1.5) combined with fewer clusters yield the highest F1 scores. This suggests that crisp clusters are more appropriate for this dataset. In contrast, for the *mushroom* dataset, the heatmap indicates less sensitivity to the fuzziness parameter, as high F1 scores are maintained across a range of parameter combinations. The *vowel* dataset, however, benefits from higher fuzziness values when paired with larger *n_clusters*, suggesting that soft cluster boundaries improve performance for this dataset's characteristics.

Overall, the results highlight the importance of dataset-specific tuning of Fuzzy C-Means parameters. The *hepatitis* dataset favors simple models with low fuzziness and fewer clusters, while the *vowel* dataset benefits from greater complexity in terms of both parameters. The *mushroom* dataset demonstrates robustness across a wider range of parameter combinations, underscoring its relative insensitivity to these hyperparameters.

These findings emphasize the need for careful hyperparameter optimization and underscore the variability of clustering behavior across datasets with differing characteristics.

## 5.4 OPTICS

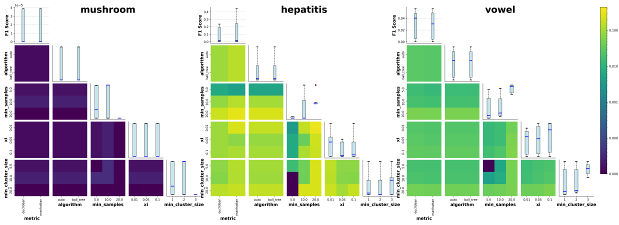

Figure 5: Parameter interactions for OPTICS

OPTICS demonstrated varying performance across the three datasets. For Hepatitis, it achieved moderate results with an F-measure around 0.23 at optimal parameters. The Mushroom dataset showed very poor performance with F-measures near 0, despite its simpler binary classification nature. The Vowel dataset achieved F-measures around 0.05, which while low, was better than the Mushroom results.

As shown in Figure 5, the algorithm's performance was significantly influenced by parameter choices. The *min_samples* parameter had the strongest effect across all datasets, while the *xi* parameter showed notable impact particularly on the Hepatitis dataset. The metric choice (euclidean vs manhattan) and algorithm implementation (auto vs ball_tree) had relatively minor effects on the results.

The *min_cluster_size* parameter demonstrated interesting interactions with other parameters, particularly visible in the Hepatitis dataset where higher values generally led to better F-measures when combined with appropriate *min_samples* settings. This suggests that OPTICS performs better when allowed to form larger, more stable clusters rather than numerous small ones.

## 5.5 Spectral Clustering

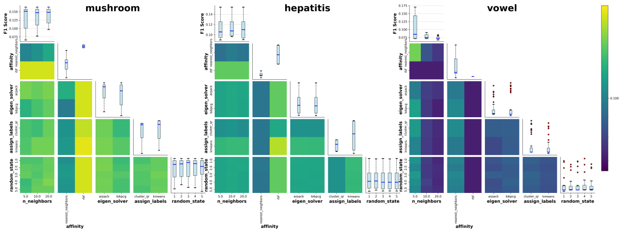

Figure 6: Parameter interactions for Spectral Clustering

As seen in 6, spectral clustering showed varying performance across different parameter configurations and datasets. It performed best on the mushroom dataset, achieving ARI scores of 0.35 and F-measures of 0.16 using RBF affinity with ARPACK solver and assigning labels using kmeans. The vowel dataset showed more modest results, with nearest-neighbors affinity performing better than RBF, though still achieving relatively low ARI scores (typically between -0.007 and 0.03).

The algorithm's performance was significantly influenced by several key parameters:

- **Affinity Matrix**: Nearest-neighbors consistently outperformed RBF kernel for the vowel dataset, while RBF showed superior results for the mushroom and hepatitis datasets
- **Solver Choice**: LOBPCG solver generally provided faster execution times (0.03-0.08s) compared to ARPACK (0.1-0.4s), and the results were similar
- **Number of Neighbors**: Lower values (n_neighbors=5) produced more stable results for the vowel dataset, but it mattered less for the other datasets
- **Assignment Method**: cluster_qr and kmeans assignments showed similar clustering quality
- **Random State**: The results were very similar across different random states, indicating that

the results are consistent and not highly dependent on the initialization

We saw interesting interactions between the parameters. For example, using a Kmeans assignment with the rbf affinity matrix showed particular effectiveness on the hepatitis dataset.

### 5.6 Summary

## 6 Conclusion

### 6.1 Key Findings

- K-means demonstrated consistent performance with moderate execution times (0.03-0.07s) and balanced clustering metrics

- Fuzzy C-means showed particularly strong performance on the hepatitis dataset, achieving high ARI scores (0.14-0.17) for lower cluster numbers

- Global K-means exhibited longer execution times but provided more stable clustering than K-means results across different initializations

- Gmeans did not demonstrate competitive results generally, but exhibited decent F1 scores on the vowel dataset

- OPTICS generally showed weak performance, probably due to the lack of natural density variations in the datasets

- Spectral clustering performed well with nearest-neighbors affinity, especially on the vowel dataset, achieving competitive ARI scores

### 6.2 Practical Implications

- For well-separated clusters: K-means offers the best balance of speed and accuracy

- For overlapping clusters: Fuzzy C-means provides more nuanced cluster assignments

- For datasets with varying densities: OPTICS or Spectral clustering are more suitable

- When cluster number is unknown: Mean Shift or OPTICS are preferable

- When stability is crucial: Global K-means offers more consistent results

### 6.3 Limitations and Future Work

- Current analysis is limited to specific parameter ranges and could be expanded

- Computational efficiency could be improved, particularly for Global K-means and OPTICS

- The impact of data preprocessing and normalization needs further investigation

- Integration of multiple algorithms in an ensemble approach could potentially yield better results

### 6.4 Final Remarks

Each clustering algorithm demonstrates distinct strengths and weaknesses, suggesting that the choice of algorithm should be primarily driven by specific application requirements and data characteristics. While K-means and Fuzzy C-means offer good general-purpose solutions, specialized algorithms like OPTICS and Spectral clustering provide advantages for specific data distributions. Future work should focus on developing hybrid approaches that can combine the strengths of multiple algorithms while mitigating their individual weaknesses.

## References

[1] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. *SIGMOD Rec.*, 28(2):49–60, June 1999.

[2] James C Bezdek, Robert Ehrlich, and William Full. Fcm: The fuzzy c-means clustering algorithm. *Computers & geosciences*, 10(2-3):191–203, 1984.

[3] Anil Damle, Victor Minden, and Lexing Ying. Simple, direct and efficient multi-way spectral clustering. *Information and Inference: A Journal of the IMA*, 8(1):181–203, 06 2018.

[4] Jiu-Lun Fan, Wen-Zhi Zhen, and Wei-Xin Xie. Suppressed fuzzy c-means clustering algorithm. *Pattern Recognition Letters*, 24(9-10):1607–1612, 2003.

[5] László Szilágyi and Sándor M Szilágyi. Generalization rules for the suppressed fuzzy c-means clustering algorithm. *Neurocomputing*, 139:298–309, 2014.

# 7 Appendix