TITLE

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

November 29, 2024

1 Introduction

1.1 Background

1.2 Research Objectives

1.

1.3 Methodology Overview

•

2 Data

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

2.1 Dataset Selection and Characteristics

This section outlines our dataset selection criteria and analyzes the characteristics of the chosen datasets.

2.1.1 Dataset Selection

In this project, we aimed to select two datasets that offer substantial variability across different aspects to evaluate the performance of Support Vector Machine (SVM) and k-Nearest Neighbors (KNN) algorithms. The criteria for dataset selection were centered around having one dataset that is small and another that is large, allowing us to analyze both the efficiency and effectiveness of these models across differing dataset sizes.

Smaller datasets

Larger datasets

Attribute types We wanted to assess the models' ability to handle datasets with different types of attributes. For this purpose, we wanted to select one dataset with primarily nominal attributes and another with numerical features. This allows us to evaluate how well each algorithm handles the representation and processing of different data types.

Class distribution By choosing one dataset with a balanced distribution and another with a notable class imbalance, we aim to observe how SVM and KNN, particularly with reduction, perform in scenarios where the data is skewed.

Missing data Lastly, we prioritized finding datasets with varying levels of missing data to examine how effectively the algorithms manage incomplete information.

Based on these criteria, we selected the Hepatitis and Mushroom datasets, as they provide the most distinct and complementary combination for our evaluation.

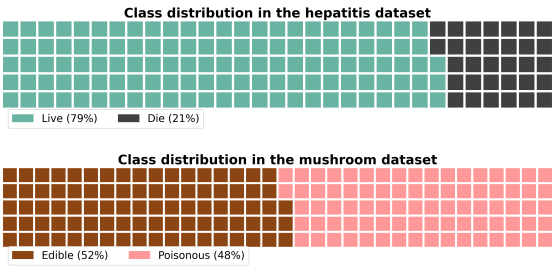

Figure 1: Class distribution comparison between datasets

2.1.2 Selection Criteria

We selected two datasets with contrasting characteristics to evaluate the performance of Support Vector Machine (SVM) and k-Nearest Neighbors (KNN) algorithms across different scenarios. Our selection criteria focused on:

- Dataset size variation (small vs. large)
- Attribute type diversity (nominal vs. numerical)
- Class distribution (balanced vs. imbalanced)
- Missing data patterns

A smaller dataset enables rapid experimentation and initial algorithm validation, while a larger dataset allows evaluation of reduction methods' effectiveness, particularly for KNN's storage and computational requirements.

2.1.3 Dataset Characteristics

- **Mushroom Dataset**
  - 8,124 instances
  - 22 nominal attributes
  - Binary classification (edible vs. poisonous)
  - Nearly balanced classes
- **Hepatitis Dataset**
  - 155 instances
  - Mixed attribute types (both nominal and numerical)
  - Binary classification (survive vs. die)
  - Imbalanced classes (79.35% majority class)
  - 6.01% missing values

These datasets provide complementary characteristics for evaluating our algorithms' performance across different scenarios. The Mushroom dataset challenges the algorithms with its size and categorical nature, while the Hepatitis dataset tests their ability to handle mixed data types and class imbalance.

2.2 Data Preprocessing

This section details the preprocessing steps applied to prepare both datasets for analysis.

2.2.1 Handling Different Data Types and Ranges

To manage the varying types and ranges of attributes in our datasets, we implemented specific preprocessing techniques. For the nominal attributes in both the Mushroom and Hepatitis datasets, we used label encoding. This technique converts categorical values into numerical labels, enabling the algorithms to interpret the data correctly. While we considered one-hot encoding to avoid implying any ordinal relationship among categories, we opted for label encoding due to its simplicity and reduced dimensionality, as one-hot encoding would significantly increase dimensions and lead to a sparse space, making accurate predictions more challenging, particularly for KNN with the Mushroom dataset's numerous nominal features [1].

For the numerical attributes in the Hepatitis dataset, we applied min-max scaling to rescale the data to a fixed range of [0, 1]. This normalization is crucial for distance-based algorithms like KNN and SVM, ensuring all features contribute equally to model performance. We evaluated other scaling methods, such as standardization, but chose min-max scaling for its effectiveness in maintaining the original data distribution [2].

We implemented specific preprocessing techniques based on the characteristics of each dataset:

- **Nominal Attributes**
  - Applied label encoding to both datasets
  - Chose label encoding over one-hot encoding to prevent dimensionality explosion
  - Particularly important for the Mushroom dataset's numerous categorical features
- **Numerical Attributes**
  - Applied min-max scaling to the Hepatitis dataset's numerical features
  - Normalized all values to [0, 1] range

  - Essential for distance-based algorithms (KNN and SVM)

2.2.2 Missing Value Treatment

Addressing missing values is a critical step in preparing our datasets for analysis, as they can significantly impact model performance. In the case of the nominal attributes in both the Mushroom and Hepatitis datasets, we opted to impute missing values with the majority class. This method is straightforward and effective for maintaining dataset integrity. However, it can also introduce bias, particularly in the Hepatitis dataset, where the majority class represents 79.35% of instances. Relying on this method may lead to a situation where the imputed values disproportionately favor the majority class, thereby affecting the overall distribution and potentially skewing the results [2].

For the numerical attributes in the Hepatitis dataset, we used the mean of the available data to fill in missing values. This approach preserves the overall data distribution and is easy to implement, but it is not without its drawbacks. The mean can be heavily influenced by outliers, which might distort the data and lead to less accurate predictions. This is especially important in medical datasets, where extreme values may carry significant meaning.

We also considered employing K-Nearest Neighbors (KNN) for imputing missing values, as it could provide a more nuanced approach by considering the nearest data points for each instance. However, we ultimately decided against this option to avoid introducing bias into our evaluation. Since KNN is one of the algorithms we are testing, using it for imputation could influence its performance and lead to skewed results. Therefore, we chose the more straightforward methods of majority class imputation for nominal values and mean imputation for numerical values, allowing for a clearer assessment of the models' effectiveness without confounding factors.

We employed different strategies for handling missing values based on attribute type:

- **Nominal Attributes**
  - Imputed with mode (majority class)
  - Applied to both datasets
  - Potential limitation: May reinforce majority class bias
- **Numerical Attributes**
  - Imputed with mean values
  - Applied only to Hepatitis dataset
  - Preserves overall distribution while handling missing data

Alternative approaches such as KNN-based imputation were considered but rejected to avoid introducing bias into our evaluation of KNN as a classifier. Our chosen methods provide a balance between simplicity and effectiveness while maintaining data integrity.

3 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 3.1), Improved K-Means (Section 3.2), Fuzzy C-Means (Section 3.3), OPTICS (Section 3.4), and Spectral Clustering (Section 3.5).

3.1 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.

3.1.1 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 (ϵ), 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.

3.1.2 Parameter Grid

K-Means involves several important parameters, which include:

- **Number of Clusters (n_{clusters})**: Determines the number of clusters (k) to form.
- **Maximum Iterations ($max_{\text{iterations}}$)**: The maximum number of iterations allowed for convergence.
- **Tolerance (ϵ)**: 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.

Table 1: K-Means Parameter Grid

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

3.2 Improved K-Means

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

3.2.1 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** (ϵ): 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]

3.2.2 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** ($\min_{\mathbf{obs}}$): Prevents splitting clusters with too few data points.
- **Maximum Depth** ($\max_{\mathbf{depth}}$): 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]

3.3 Fuzzy C-Means

3.4 OPTICS

3.5 Spectral Clustering

4 Results and Analysis

4.1 K-Means

4.2 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.

4.2.1 Global K-Means

4.2.2 G-Means

4.3 Fuzzy C-Means

4.4 OPTICS

4.5 Spectral Clustering

4.6 Summary

5 Conclusion

5.1 Key Findings

-

5.2 Practical Implications

-

5.3 Limitations and Future Work

-

5.4 Final Remarks

References

[1] Suad A Alasadi and Wesam S Bhaya. Review of data preprocessing techniques in data mining. *Journal of Engineering and Applied Sciences*, 12(16):4102–4107, 2017.

[2] Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 international conference on management of data*, pages 2201–2206, 2016.

6 Appendix