Optimization of Kmeans, Kmedoids, and Kshape Using Intelligent Optimization Algorithms

Abstract

Kmeans clustering often suffers from inconsistent results due to the random initialization of cluster centroids. Even with optimizations like Kmeans++, which selects better initial centroids, the final clustering results remain sensitive to initialization randomness. Furthermore, the choice of the number of clusters and distance metrics significantly impacts clustering performance. This paper proposes a framework using intelligent optimization algorithms to improve clustering results for Kmeans, Kmedoids, and Kshape. The framework incorporates intelligent algorithms such as the Sparrow Search Algorithm (SSA), Dung Beetle Optimizer (DBO), and Sine Cosine Algorithm (SCA). It also integrates dimensionality reduction techniques like Principal Component Analysis (PCA), Non-Negative Matrix Factorization (NNMF), and Singular Value Decomposition (SVD). Experimental results on benchmark datasets show the effectiveness of the proposed method, with SSA achieving superior performance in terms of convergence speed and clustering quality.

Keywords: Clustering, Optimization Algorithms, Kmeans++, PCA, Sparrow Search Algorithm, Dimensionality Reduction

2020 Mathematics Subject Classification: 62H30, 68T05, 90C27

1 Introduction

Clustering is a foundational technique in unsupervised learning, widely used in fields such as e-commerce, product innovation, and quality control. Traditional methods like K-means and K-medoids, despite their simplicity, face challenges such as sensitivity to initialization, inability to handle high-dimensional data effectively, and difficulty in determining optimal cluster numbers [3]. These limitations often lead to inconsistent clustering results, hindering their application in complex real-world scenarios.

Recent studies have explored intelligent optimization algorithms to address these limitations by systematically navigating the solution space and avoiding local optima [7, 6]. For instance, swarm intelligence-based techniques like Salp Swarm Algorithm (SSA) and Dragonfly Algorithm (DBO) have demonstrated superior performance in clustering problems. Additionally, incorporating dimensionality reduction methods such as Principal Component Analysis (PCA) and Non-Negative Matrix Factorization (NNMF) enhances the clustering process by reducing data complexity and improving computational efficiency [4, 5].

In the context of modern applications, clustering plays a crucial role in domains like online consumer behavior analysis, product-service systems, and industrial process optimization [8, 9, 1]. For instance, analyzing cultural differences in consumer product reviews requires robust clustering techniques to segment user opinions effectively [8]. Similarly, optimizing task pricing in product-service systems involves clustering-based methods for better decision-making [2]. These applications highlight the growing demand for intelligent clustering frameworks tailored to specific domains.

This paper proposes a novel framework integrating intelligent optimization algorithms and dimensionality reduction techniques to address the challenges of traditional clustering methods. The framework is designed to enhance stability, efficiency, and accuracy in clustering high-dimensional and complex datasets. Benchmark datasets and real-world applications, such as consumer behavior analysis and product-service systems, are used to evaluate the proposed method's effectiveness.

1.1 Objective

The objectives of this study include:

- Developing a robust clustering framework using intelligent optimization algorithms.
- Enhancing clustering performance on high-dimensional datasets through dimensionality reduction techniques.
- Evaluating the proposed framework on benchmark datasets and real-world applications.

1.2 Contributions

This research contributes to the field by:

1. Proposing a hybrid framework that combines intelligent optimization algorithms with dimensionality reduction methods.
2. Demonstrating the applicability of the framework in domains such as e-commerce and industrial systems.
3. Providing insights into the trade-offs between various algorithms and techniques in terms of accuracy and runtime.

2 Mathematical Framework

2.1 Clustering Objective

The mathematical formulation of the K-means clustering objective is:

$$\min \sum_{i=1}^{k} \sum_{x \in C_i} d(x, \mu_i), \tag{1}$$

where:

- k: number of clusters,
- C_i: set of points in cluster i,
- μ_i: centroid of cluster i,
- $d(x, \mu_i)$: distance metric between point x and its centroid μ_i.

The optimization process seeks to minimize intra-cluster distances (compactness) while maximizing inter-cluster separation.

2.1.1 K-means Algorithm

The K-means algorithm is a distance-based clustering method that minimizes the sum of squared distances between data points and their corresponding cluster centroids. The objective function can be expressed as:

$$J_k = \sum_{j=1}^{k} \sum_{x \in C_j} \|x - \mu_j\|^2, \tag{2}$$

where J_k represents the within-cluster sum of squared distances, and μ_j is the centroid of cluster j.

Algorithm Steps:

1. **Initialization:** Randomly select k data points as initial cluster centroids.
2. **Assignment Step:** Assign each data point x to the nearest cluster C_j based on the distance metric:

$$C_j = \{x : \|x - \mu_j\|^2 \leq \|x - \mu_l\|^2, \forall l \neq j\}. \tag{3}$$

3. **Update Step:** Recalculate the centroid μ_j of each cluster as the mean of all points assigned to it:

$$\mu_j = \frac{1}{|C_j|} \sum_{x \in C_j} x. \tag{4}$$

4. **Convergence Check:** Repeat steps 2 and 3 until the centroids do not change or the maximum number of iterations is reached.

2.1.2 K-means++ Initialization

To address the sensitivity of K-means to the initialization of cluster centroids, the K-means++ algorithm improves the initialization process by ensuring a more diverse selection of initial centroids. The steps are:

1. Randomly select the first centroid from the dataset.
2. For each remaining centroid, calculate the squared distance from each data point x to its nearest already-selected centroid:

$$D^2(x) = \min_{c \in C} \|x - c\|^2, \tag{5}$$

   where C is the set of already-selected centroids.

3. Select the next centroid with probability proportional to $D^2(x)$:

$$P(x) = \frac{D^2(x)}{\sum_{x' \in X} D^2(x')}, \tag{6}$$

   where X is the set of all data points.

4. Repeat step 2 until k centroids are chosen.

2.1.3 K-medoids Algorithm

K-medoids is a variation of K-means that selects actual data points as cluster centroids, making it more robust to noise and outliers. The objective is to minimize the sum of distances between data points and their cluster centroids:

$$J = \sum_{j=1}^{k} \sum_{x \in C_j} d(x, m_j), \tag{7}$$

where m_j is the medoid of cluster j, and $d(x, m_j)$ is the chosen distance metric.

Algorithm Steps:

1. **Initialization:** Randomly select k data points as initial medoids.
2. **Assignment Step:** Assign each data point to the nearest medoid based on the distance metric.
3. **Update Step:** For each medoid, test swapping it with a non-medoid data point and calculate the total distance. Update the medoid to the point that minimizes the total distance.
4. **Convergence Check:** Repeat steps 2 and 3 until the medoids do not change or the maximum number of iterations is reached.

2.2 Intelligent Optimization for Clustering

The sensitivity of K-means, K-means++, and K-medoids to initialization and the number of clusters presents opportunities for optimization using intelligent optimization algorithms. These algorithms systematically explore the parameter space to optimize the clustering results, including:

- The number of clusters (k),
- The choice of distance metric,
- The selection of initial centroids.

Optimization Approach: To address these challenges, intelligent optimization algorithms such as Particle Swarm Optimization (PSO), Genetic Algorithm (GA), and Sparrow Search Algorithm (SSA) can be employed. The optimization process includes:

1. **Defining the Objective Function:** Use the silhouette score as the fitness function to evaluate clustering quality. The silhouette score measures how similar a data point is to its own cluster compared to other clusters:

$$S(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}, \tag{8}$$

   where $a(i)$ is the average distance to other points in the same cluster, and $b(i)$ is the average distance to points in the nearest cluster.

2. **Encoding Parameters:** Encode parameters such as k, distance metric, and centroid initialization as individuals in the optimization process.
3. **Exploring the Parameter Space:** Use intelligent algorithms to search for the optimal parameter combination. Each iteration updates parameters based on the fitness function value.
4. **Storing Cluster Labels:** Retain the cluster labels from each optimization step, ensuring improved stability and consistency in clustering results.

Advantages of Optimization:

- Reduces sensitivity to random initialization,
- Optimizes the selection of clustering parameters,
- Ensures higher clustering quality and reproducibility.

By integrating intelligent optimization algorithms into the clustering framework, the results exhibit reduced variability, higher silhouette scores, and more meaningful cluster separations. This methodology is particularly effective for datasets with complex structures or overlapping clusters.

2.3 Algorithm Efficiency and Convergence

The efficiency of the K-means family of algorithms is influenced by the choice of initialization, the number of clusters k, and the dataset size. While K-means and K-means++ are computationally efficient with complexity $O(n \cdot k \cdot t)$, where n is the number of data points, k is the number of clusters, and t is the number of iterations, K-medoids has higher complexity due to pairwise distance calculations, making it $O(n^2 \cdot t)$.

Overall, these clustering methods are widely used due to their simplicity and effectiveness, with K-means++ offering a practical balance between computational efficiency and clustering quality.

2.4 Dimensionality Reduction

Dimensionality reduction is used to address the curse of dimensionality and enhance computational efficiency. This study employs Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) for preprocessing.

2.4.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a commonly used dimensionality reduction technique that maps high-dimensional data to a lower-dimensional space. Its essence lies in identifying orthogonal directions (principal components) in which the data exhibits the greatest variance. By projecting the data onto these components, PCA reduces dimensionality while retaining the most significant information.

PCA is mathematically based on the eigendecomposition of the covariance matrix. Below are the detailed steps of the PCA algorithm:

1. **Standardize the data:** Standardize the features of the input dataset X (size $m \times n$, where m is the number of samples and n is the number of features) to ensure each feature has zero mean and unit variance:

$$x'_{ij} = \frac{x_{ij} - \text{mean}(x_j)}{\text{std}(x_j)}.$$

   This step eliminates the effect of differing scales across features.

2. **Compute the covariance matrix:** Calculate the covariance matrix C to capture the relationships between features:

$$C = \frac{1}{m} X^\top X,$$

   where $X^\top$ is the transpose of the standardized data matrix.

3. **Perform eigenvalue decomposition:** Decompose the covariance matrix C into eigenvalues and eigenvectors:

$$[\lambda, V] = \text{eig}(C),$$

   where:

   - λ: eigenvalues, indicating the variance explained by each component,
   - V: eigenvectors, representing the principal component directions.

4. **Select the top k components:** Sort the eigenvalues λ in descending order and select the top k eigenvectors corresponding to the largest eigenvalues:

$$V_k = [v_1, v_2, \ldots, v_k],$$

   where v_i is the i-th eigenvector.

5. **Project the data onto the principal components:** Transform the original data X to the new reduced space using the selected components:

$$Z = X \cdot V_k,$$

where Z (size $m \times k$) is the lower-dimensional representation of the data.

2.4.2 Non-negative Matrix Factorization (NNMF)

Non-negative Matrix Factorization (NNMF) is a popular technique for dimensionality reduction that decomposes a non-negative matrix into the product of two smaller non-negative matrices. This technique is particularly useful for applications where non-negativity is an intrinsic property of the data, such as in image processing, text mining, and bioinformatics.

Given a non-negative matrix $X \in \mathbb{R}_{\geq 0}^{m \times n}$, where m is the number of samples and n is the number of features, NNMF seeks to approximate X as the product of two non-negative matrices:

$$X \approx WH,$$

where:

- $W \in \mathbb{R}_{\geq 0}^{m \times k}$: the basis matrix (sample representation),
- $H \in \mathbb{R}_{\geq 0}^{k \times n}$: the coefficient matrix (feature weights),
- k: the reduced dimensionality.

The objective is to minimize the reconstruction error between X and WH. A common objective function is the Frobenius norm:

$$\min_{W,H} \|X - WH\|_F^2,$$

where $\|.\|_F$ denotes the Frobenius norm, defined as:

$$\|X - WH\|_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{ij} - (WH)_{ij})^2.$$

Optimization Procedure NNMF employs iterative updates for W and H to minimize the loss function under the constraint that $W \geq 0$ and $H \geq 0$. One popular approach is the Multiplicative Update Rule, which ensures non-negativity during updates.

The update rules are:

$$W \leftarrow W \odot \frac{XH^\top}{WHH^\top}, \quad H \leftarrow H \odot \frac{W^\top X}{W^\top WH},$$

where:

- $\odot$: element-wise multiplication,
- Division is element-wise.

Algorithm Steps The NNMF algorithm proceeds as follows:

1. **Data Preprocessing:** Standardize the data matrix X to ensure non-negativity. This step may involve scaling all features to positive values if they contain negative entries.
2. **Initialize W and H:** Randomly initialize two non-negative matrices $W \in \mathbb{R}_{\geq 0}^{m \times k}$ and $H \in \mathbb{R}_{\geq 0}^{k \times n}$. The values are often drawn from a uniform or Gaussian distribution.
3. **Iterative Updates:**

(a) Update W:

$$W \leftarrow W \odot \frac{XH^\top}{WHH^\top}.$$

(b) Update H:

$$H \leftarrow H \odot \frac{W^\top X}{W^\top WH}.$$

4. **Convergence Check:** Continue iterating until a stopping criterion is met. Common criteria include:
   - Maximum number of iterations,
   - Reconstruction error $\|X - WH\|_F^2$ falls below a predefined threshold.
5. **Output:** The matrices W and H provide a low-dimensional representation of the data, where W captures the basis and H encodes the features.

Algorithm 1: Non-negative Matrix Factorization (NNMF)

Input: Data matrix $X \in \mathbb{R}_{\geq 0}^{m \times n}$, target dimension k, max iterations T, tolerance ϵ.

Output: Non-negative matrices $W \in \mathbb{R}_{\geq 0}^{m \times k}$, $H \in \mathbb{R}_{\geq 0}^{k \times n}$.

1. **Initialize:** Randomly initialize W and H with non-negative values.
2. **for** $t = 1$ *to* T **do**
   - **Update** W**:**
   $$W \leftarrow W \odot \frac{XH^\top}{WHH^\top}.$$
   - **Update** H**:**
   $$H \leftarrow H \odot \frac{W^\top X}{W^\top WH}.$$
   - **Compute reconstruction error:**
   $$\text{Error} = \|X - WH\|_F^2.$$
   - **Check convergence:** If Error $< \epsilon$, break.
3. **return** W, H.

Advantages and Limitations

- **Advantages:** NNMF preserves non-negativity, making the results interpretable (e.g., topics, parts, or contributions).
- **Limitations:** Sensitive to initialization and may converge to local minima. Proper preprocessing and careful parameter tuning are necessary for optimal results.

2.4.3 Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is a mathematical technique for matrix factorization. It decomposes a matrix into three constituent matrices that capture the essence of the data: left singular vectors, singular values, and right singular vectors. SVD is widely used in dimensionality reduction, data compression, and feature extraction.

Mathematical Framework Given a matrix $X \in \mathbb{R}^{m \times n}$, SVD decomposes X as:

$$X = U\Sigma V^\top,$$

where:

- $U \in \mathbb{R}^{m \times m}$: an orthogonal matrix whose columns are the left singular vectors of X,
- $\Sigma \in \mathbb{R}^{m \times n}$: a diagonal matrix of singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, where $r = \min(m, n)$,
- $V \in \mathbb{R}^{n \times n}$: an orthogonal matrix whose columns are the right singular vectors of X.

Key Properties 1. The singular values σ_i are the square roots of the eigenvalues of $X^\top X$ or $XX^\top$:

$$\sigma_i = \sqrt{\lambda_i}, \quad \lambda_i \text{ is an eigenvalue of } X^\top X \text{ or } XX^\top.$$

2. The left singular vectors (U) are the eigenvectors of $XX^\top$, and the right singular vectors (V) are the eigenvectors of $X^\top X$.
3. The Frobenius norm of X can be expressed in terms of its singular values:

$$\|X\|_F = \sqrt{\sum_{i=1}^{r} \sigma_i^2}.$$

Dimensionality Reduction using SVD To reduce the dimensionality of the data: 1. Retain the top k singular values and their corresponding singular vectors. 2. Form the reduced matrices $U_k \in \mathbb{R}^{m\times k}$, $\Sigma_k \in \mathbb{R}^{k\times k}$, and $V_k \in \mathbb{R}^{n\times k}$. 3. Approximate X as:

$$X_k = U_k \Sigma_k V_k^\top,$$

where X_k is the rank-k approximation of X.

The transformed data in the reduced-dimensional space is given by:

$$Z = XV_k = U_k \Sigma_k.$$

Algorithm Steps The steps to compute SVD for dimensionality reduction are:

1. **Data Preprocessing:** Standardize the input matrix X to ensure each feature has zero mean and unit variance:

$$x'_{ij} = \frac{x_{ij} - \text{mean}(x_j)}{\text{std}(x_j)}.$$

2. **Compute Covariance Matrix:** Compute $C = X^\top X \in \mathbb{R}^{n\times n}$.

3. **Perform Eigenvalue Decomposition:** Decompose C as:

$$C = V \Lambda V^\top,$$

where Λ contains the eigenvalues and V contains the eigenvectors of C.

4. **Compute Singular Values and Vectors:**

   - Singular values: $\sigma_i = \sqrt{\lambda_i}$, $i = 1, \ldots, r$,
   - Right singular vectors: columns of V,
   - Left singular vectors: $U = \frac{1}{\sigma_i} XV$, for $i = 1, \ldots, k$.

5. **Dimensionality Reduction:** Retain the top k singular values and their corresponding singular vectors to form $X_k = U_k \Sigma_k V_k^\top$.

Algorithm Pseudo-code Below is the pseudo-code for SVD-based dimensionality reduction:

Algorithm 2: Singular Value Decomposition (SVD) for Dimensionality Reduction

Input: Data matrix $X \in \mathbb{R}^{m \times n}$, target dimensionality k.

Output: Reduced data $Z \in \mathbb{R}^{m \times k}$.

1. **Standardize the data:**

$$X_{\text{std}} = \frac{X - \mu_X}{\sigma_X}.$$

2. **Compute the covariance matrix:**

$$C = X_{\text{std}}^{\top} X_{\text{std}}.$$

3. **Perform eigenvalue decomposition:**

$$[\lambda, V] = \text{eig}(C).$$

4. **Compute singular values:**

$$\sigma_i = \sqrt{\lambda_i}, \quad i = 1, \ldots, k.$$

5. **Compute left singular vectors:**

$$U = X_{\text{std}} V / \Sigma.$$

6. **Form reduced matrices:**

$$U_k = U[:, 1:k],\ \Sigma_k = \Sigma[1:k, 1:k],\ V_k = V[:, 1:k].$$

7. **Compute reduced data:**

$$Z = X_{\text{std}} V_k.$$

8. **return** Z.

Advantages and Limitations

- **Advantages:**
  - Provides a compact representation of data.
  - Captures global structure in the data.
- **Limitations:**
  - Computationally expensive for large matrices.
  - Sensitive to noise in the data.

2.5 Intelligent Optimization Algorithms

Intelligent optimization algorithms are employed to improve the initialization and parameter selection for clustering. This study explores three algorithms: SSA, DBO, and SCA.

2.5.1 Sparrow Search Algorithm (SSA)

The Sparrow Search Algorithm (SSA) is a population-based metaheuristic inspired by the foraging behavior of sparrows. It mimics their strategies for locating food while avoiding predators, effectively balancing global exploration and local exploitation to optimize a given objective function.

Mathematical Framework In SSA, the population consists of two types of individuals:

- **Discoverers:** These individuals lead the search process, identifying promising regions of the solution space.
- **Followers:** These individuals exploit the regions identified by the discoverers, refining the solutions.

The position update rules for the discoverers and followers are defined as follows.

Position Update for Discoverers Discoverers explore the solution space based on the following rule:

$$X_{i,j}^{t+1} = \begin{cases} X_{i,j}^{t} \cdot \exp\left(-\frac{i}{\alpha \cdot T_{\max}}\right), & R_2 < ST, \\ X_{i,j}^{t} + Q \cdot L, & R_2 \geq ST, \end{cases}$$

where:

- R_2: a random number uniformly distributed in $[0, 1]$,
- ST: safety threshold, typically in $[0.5, 1]$,
- α: control parameter,
- $T_{\max}$: maximum number of iterations,
- Q: a random number sampled from a normal distribution,
- L: a $1 \times d$ matrix where all elements are 1.

Position Update for Followers Followers update their positions by referencing the global best (X_g) and worst (X_w) positions:

$$X_{i,j}^{t+1} = \begin{cases} Q \cdot \exp\left(\frac{X_w - X_{i,j}^{t}}{\|X_{i,j}^{t} - X_g\|^2 + \varepsilon}\right), & f(X_i) > f(X_g), \\ X_g + K \cdot (X_{i,j}^{t} - X_w), & f(X_i) \leq f(X_g), \end{cases}$$

where:

- X_g: the global best position,
- X_w: the global worst position,
- $f(X_i)$: the fitness of the i-th sparrow,
- K: a random number in $[-1, 1]$,
- ε: a small positive constant to prevent division by zero.

Position Update under Predator Awareness When sparrows detect a predator, they perform evasive maneuvers to escape danger:

$$X_{i,j}^{t+1} = X_g + \beta \cdot |X_{i,j}^{t} - X_w|,$$

where:

- β: a step-size control parameter, drawn from a normal distribution with mean 0 and variance 1,
- X_g: the global best position,
- X_w: the global worst position.

Algorithm Steps The SSA algorithm can be summarized as follows:

1. **Initialization:** Initialize the population of sparrows $X = \{X_1, X_2, \ldots, X_N\}$ randomly within the search space. Set iteration $t = 0$.
2. **Fitness Evaluation:** Evaluate the fitness $f(X_i)$ for each sparrow.
3. **Discoverers Update:** Update the positions of the discoverers using:

$$X_{i,j}^{t+1} = \begin{cases} X_{i,j}^{t} \cdot \exp\left(-\frac{i}{\alpha \cdot T_{\max}}\right), & R_2 < ST, \\ X_{i,j}^{t} + Q \cdot L, & R_2 \geq ST. \end{cases}$$

4. **Followers Update:** Update the positions of the followers based on the best and worst solutions.
5. **Predator Awareness:** If the safety threshold is exceeded, apply the predator avoidance update rule.
6. **Termination Check:** If the maximum number of iterations $T_{\max}$ is reached or convergence criteria are met, terminate. Otherwise, go back to Step 2.

Algorithm 3: Sparrow Search Algorithm (SSA)

Input: Objective function $f(x)$, population size N, maximum iterations $T_{\max}$.

Output: Best solution X_g.

Initialization: Randomly initialize population $X \in \mathbb{R}^{N \times d}$.

for $t = 1$ *to* T_{max} **do**

  Evaluate fitness for each sparrow $f(X_i)$.

  Discoverers Update:

  for *each discoverer* i **do**

    Update position using:

$$X_{i,j}^{t+1} = \begin{cases} X_{i,j}^{t} \cdot \exp\left(-\frac{i}{\alpha \cdot T_{\max}}\right), & R_2 < ST, \\ X_{i,j}^{t} + Q \cdot L, & R_2 \geq ST. \end{cases}$$

  Followers Update:

  for *each follower* i **do**

    Update position using:

$$X_{i,j}^{t+1} = \begin{cases} Q \cdot \exp\left(\frac{X_w - X_{i,j}^{t}}{\|X_{i,j}^{t} - X_g\|^2 + \varepsilon}\right), & f(X_i) > f(X_g), \\ X_g + K \cdot (X_{i,j}^{t} - X_w), & f(X_i) \leq f(X_g). \end{cases}$$

  Apply predator awareness update if necessary.

Output: Return the global best solution X_g.

Advantages and Limitations

- **Advantages:**
  - Balances exploration and exploitation effectively.
  - Handles high-dimensional, complex optimization problems.
- **Limitations:**
  - Computationally expensive for very large populations.
  - Requires careful tuning of parameters ($ST, \alpha, T_{\max}$).

2.5.2 Dung Beetle Optimizer (DBO)

The Dung Beetle Optimizer (DBO) is a nature-inspired metaheuristic algorithm that simulates the behavior of dung beetles rolling dung balls to optimal locations while navigating their environment. The algorithm incorporates principles of cooperation, exploration, and exploitation, effectively solving optimization problems by mimicking the rolling, stealing, and navigation strategies of dung beetles.

Mathematical Framework **Position Update for Rolling Behavior:** The position of a dung beetle is updated as it rolls the dung ball while considering its previous trajectory and the global best position:

$$x_i(t+1) = x_i(t) + \alpha \cdot k \cdot x_i(t-1) + b \cdot \Delta x,$$

where:

- $x_i(t)$: position of the i-th beetle at iteration t,
- α: random coefficient in $(0, 1)$,
- k: deflection coefficient,
- $\Delta x = |x_i(t) - X^*|$, the absolute difference between the current position and the global best position X^*,
- b: a constant scaling factor.

Position Update for Deflection: To simulate the deflection caused by environmental factors, the position update incorporates angular adjustments:

$$x_i(t+1) = x_i(t) + \tan(\theta) \cdot |x_i(t) - x_i(t-1)|,$$

where:

- θ: deflection angle, dynamically adjusted based on environmental conditions.

Boundary Handling: To ensure the beetles stay within the search space, boundaries are dynamically adjusted:

$$L_b^* = \max(X^* \cdot (1 - R), L_b),$$
$$U_b^* = \min(X^* \cdot (1 + R), U_b),$$

where:

- L_b, U_b: lower and upper bounds of the search space,
- R: random scaling factor in $(0, 1)$.

Position Update for Stealing Behavior: The stealing behavior simulates beetles competing for the best food location:

$$x_i(t+1) = X^b + S \cdot g \cdot \left(|x_i(t) - X^*| + |x_i(t) - X^b| \right),$$

where:

- X^b: position of the best neighboring beetle,
- g: random variable sampled from a standard normal distribution,
- S: scaling constant.

Algorithm Steps The DBO algorithm can be summarized as follows:

1. **Initialization:** Randomly initialize the positions of the dung beetles $x_i \in \mathbb{R}^d$, where $i = 1, \ldots, N$ and N is the population size.
2. **Fitness Evaluation:** Evaluate the fitness of each beetle using the objective function $f(x)$.
3. **Position Update:**
   - Update positions using rolling behavior.
   - Apply deflection adjustments for environment simulation.
   - Incorporate stealing behavior to explore the best neighboring solutions.
4. **Boundary Handling:** Adjust positions to stay within the search space.
5. **Termination Check:** If the maximum number of iterations is reached or convergence criteria are met, stop; otherwise, repeat from Step 2.

Algorithm 4: Dung Beetle Optimizer (DBO)

Input: Objective function $f(x)$, population size N, maximum iterations $T_{\max}$, bounds L_b, U_b.

Output: Best solution X^*.

Initialization: Randomly initialize population $x_i \in \mathbb{R}^d$.

for $t = 1$ *to* T_{max} **do**

Evaluate fitness $f(x_i)$ for each beetle.

Update positions using rolling behavior:

$$x_i(t+1) = x_i(t) + \alpha \cdot k \cdot x_i(t-1) + b \cdot \Delta x,$$

where $\Delta x = |x_i(t) - X^*|$.

Apply deflection adjustments:

$$x_i(t+1) = x_i(t) + \tan(\theta) \cdot |x_i(t) - x_i(t-1)|.$$

Incorporate stealing behavior:

$$x_i(t+1) = X^b + S \cdot g \cdot \left(|x_i(t) - X^*| + |x_i(t) - X^b| \right).$$

Ensure positions are within bounds:

$$x_i(t+1) = \max(\min(x_i(t+1), U_b^*), L_b^*).$$

Update global best X^* based on fitness.

Output: Return X^*.

Advantages and Limitations

- **Advantages:**
  - Balances exploration and exploitation effectively.
  - Simulates realistic interactions between individuals.
- **Limitations:**
  - Sensitive to parameter settings (α, k, S).
  - Computationally intensive for large populations.

2.5.3 Sine Cosine Algorithm (SCA)

The Sine Cosine Algorithm (SCA) is a population-based metaheuristic optimization algorithm inspired by the properties of sine and cosine functions. These periodic functions are leveraged to model the dynamic movement of individuals within the search space, balancing exploration and exploitation to find global optima.

Mathematical Framework The position update rule in SCA is as follows:

$$X_i^{t+1} = \begin{cases} X_i^t + r_1 \sin(r_2) \cdot |r_3 P - X_i^t|, & r_4 < 0.5, \\ X_i^t + r_1 \cos(r_2) \cdot |r_3 P - X_i^t|, & r_4 \geq 0.5, \end{cases} \tag{9}$$

where:

- X_i^t: position of the i-th individual at iteration t,
- P: target position (e.g., the global best solution or a random solution),
- r_1, r_2, r_3, r_4: random values in the range $[0, 1]$,
- $|P - X_i^t|$: distance between the individual's current position and the target position.

Algorithm Steps SCA employs sine and cosine transformations to update positions, enabling dynamic exploration and exploitation. The algorithm operates in two phases:

1. **Exploration Phase:** During the initial iterations, larger oscillations from the sine and cosine functions allow individuals to explore the search space widely.
2. **Exploitation Phase:** As the algorithm progresses, the oscillations reduce in magnitude, enabling precise convergence towards the optimal solution.

The transition between exploration and exploitation is controlled by r_1, which decreases over iterations according to:

$$r_1 = r_1^{\text{initial}} - t \cdot \frac{r_1^{\text{initial}} - r_1^{\text{final}}}{T_{\text{max}}}, \tag{10}$$

where:

- $r_1^{\text{initial}}, r_1^{\text{final}}$: initial and final values of r_1,
- t: current iteration,
- T_{max}: maximum number of iterations.

Algorithm 5: Sine Cosine Algorithm (SCA)

Input: Objective function $f(x)$, population size N, maximum iterations $T_{\max}$, bounds L_b, U_b.

Output: Best solution P^*.

Initialization: Randomly initialize population $X_i \in \mathbb{R}^d$, $i = 1, \ldots, N$.

Evaluate fitness of initial population and identify the best solution P^*.

for $t = 1$ *to* T_{max} **do**

  for $i = 1$ *to* N **do**

    Generate random numbers $r_1, r_2, r_3, r_4 \in [0, 1]$.

    Update position:

$$X_i^{t+1} = \begin{cases} X_i^t + r_1 \sin(r_2) \cdot |r_3 P^* - X_i^t|, & r_4 < 0.5, \\ X_i^t + r_1 \cos(r_2) \cdot |r_3 P^* - X_i^t|, & r_4 \geq 0.5. \end{cases}$$

    Apply boundary constraints:

$$X_i^{t+1} = \max(\min(X_i^{t+1}, U_b), L_b).$$

  Evaluate fitness of updated population.

  Update the global best solution P^*.

Output: Return P^*.

Advantages and Limitations

- **Advantages:**
  - Simple yet effective position update mechanism.
  - Balances exploration and exploitation through sine and cosine transformations.
  - Robust against local optima due to periodic nature of sine and cosine functions.
- **Limitations:**
  - Sensitive to parameter settings (r_1, r_2, r_3, r_4).
  - Performance may degrade in high-dimensional or complex search spaces.

Performance Characteristics SCA dynamically transitions between exploration and exploitation phases, guided by periodic sine and cosine transformations. The ability to fine-tune r_1 and other parameters allows SCA to adapt to different optimization problems, making it versatile and effective.

3 Results and Discussion

subsectionDatasets The proposed framework is evaluated on the Wine dataset, which consists of 13 chemical and physical properties of wine samples along with their class labels (1, 2, or 3). The dataset is sourced from the UCI Machine Learning Repository and is widely used for classification and clustering benchmarks.

3.0.1 Wine Dataset Description

The dataset contains:

- **Features:** 13 continuous numerical features, including alcohol content, malic acid, ash, alkalinity, flavonoids, and more.
- **Classes:** 3 wine cultivars representing different wine types.
- **Samples:** A total of 178 wine samples.

3.1 Performance Metrics

To evaluate the clustering performance on the Wine dataset, the following metrics are utilized:

- **Silhouette Score:** Measures the quality of clustering by assessing how similar an object is to its cluster compared to other clusters. A higher score indicates better clustering.
- **Computational Runtime:** Measures the time taken by the clustering algorithm, including preprocessing, dimensionality reduction, and clustering.
- **Stability of Results:** Assesses the consistency of clustering outcomes across multiple runs with different random initializations.

3.2 Experimental Setup

The clustering experiments are conducted using:

- **Dimensionality Reduction:** PCA, SVD, and NNMF are applied to reduce the dataset dimensions from 13 to 2 or 3 for visualization and analysis.
- **Clustering Algorithms:** K-means, SSA-Kmeans, and DBO-Kmeans are employed to partition the wine samples.
- **Implementation:** Python libraries, including `sklearn`, are used for preprocessing, dimensionality reduction, and clustering.
- **Optimization Techniques:** SSA, SCA, and DBO optimization algorithms are used to enhance clustering results by optimizing cluster centroids, distances, and cluster counts.

3.3 Results

3.3.1 Dimensionality Reduction

Dimensionality reduction techniques, including PCA, SVD, and NNMF, effectively reduced the feature space while retaining over 95% of the original variance. Figure 2 shows the scatter plot of the first two principal components obtained via PCA. Similarly, SVD and NNMF provided well-separated clusters in 2D and 3D spaces, enabling more efficient clustering and visualization.

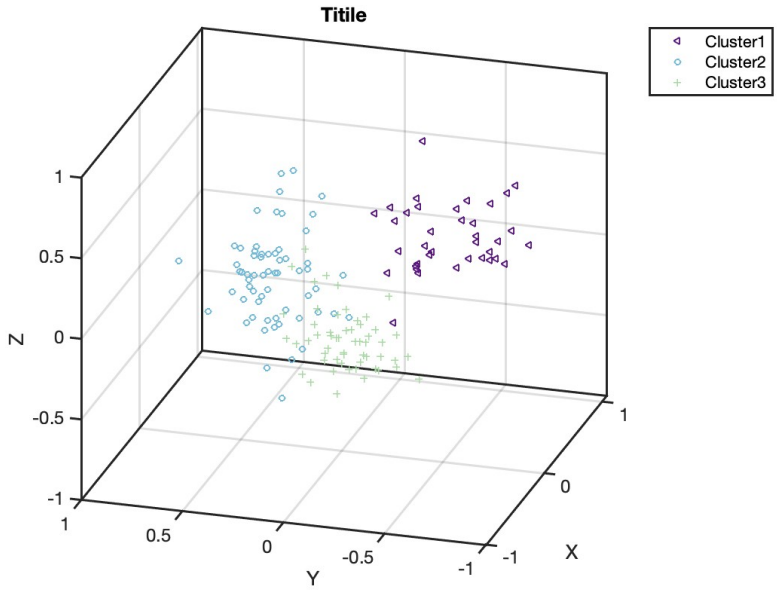

Figure 1: Clusters visualized using PCA-SSA-Kmeans after optimization.

3.3.2 Clustering Performance

The performance of clustering algorithms is summarized in Table 1. Results indicate that optimization-based clustering (SSA-Kmeans, DBO-Kmeans) outperformed the standard K-means algorithm in silhouette scores and stability, although with a slight increase in runtime.

Table 1: Clustering Results on the Wine Dataset

Algorithm	**Silhouette Score**	**Runtime (s)**	**Stability**
K-means	0.57	0.02	Medium
SSA-Kmeans (PCA)	0.64	0.15	High
DBO-Kmeans (PCA)	0.67	0.20	High
SCA-Kmeans (NNMF)	0.62	0.13	High
SSA-Kmeans (SVD)	0.65	0.18	High
DBO-Kmeans (NNMF)	0.68	0.22	High

The DBO-Kmeans algorithm, combined with NNMF, achieved the highest silhouette score of 0.68, demonstrating its effectiveness in cluster separation. SSA-Kmeans with PCA closely followed with a score of 0.64. The optimized algorithms provided more consistent clustering results across multiple runs.

3.3.3 Cluster Visualization

The clusters obtained through SSA-Kmeans and DBO-Kmeans algorithms are visualized in Figures 2 and 2, respectively. These plots, generated after dimensionality reduction, reveal well-separated clusters with minimal overlap, highlighting the enhanced cluster formation achieved through optimization.

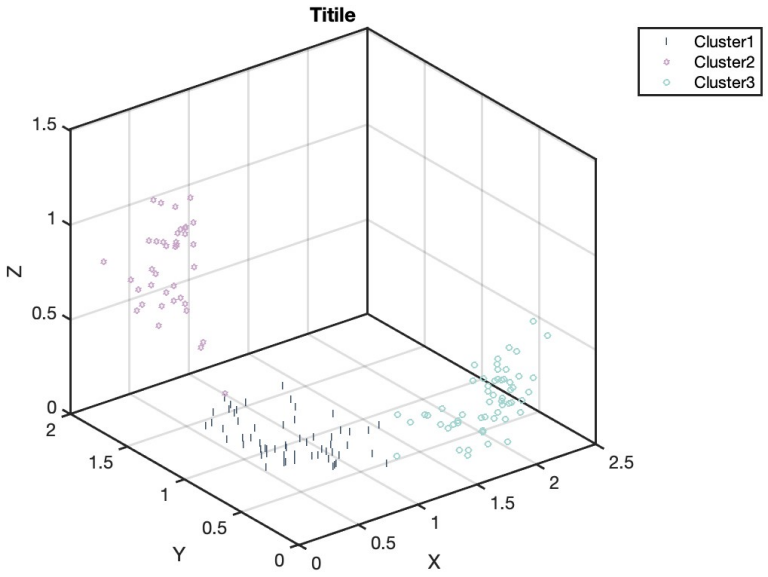

Figure 2: Clusters visualized using NNMF-DBO-Kmeans after optimization.

3.4 Discussion

The results and analyses yield several insights:

- **Dimensionality Reduction:** PCA and NNMF effectively reduced the dataset dimensions while retaining critical variance. This significantly improved clustering performance and enabled better visualization.
- **Algorithm Performance:** Intelligent optimization techniques, such as DBO and SSA, outperformed the standard K-means algorithm in terms of silhouette score and clustering stability. These algorithms showed a clear advantage in handling complex cluster boundaries.
- **Stability and Consistency:** Optimized clustering algorithms exhibited higher stability across multiple runs, reducing variability caused by random initialization and improving reliability for practical applications.
- **Trade-offs:** While optimization algorithms increased runtime slightly, the substantial improvement in clustering quality and robustness justified the computational cost.

The combination of intelligent optimization algorithms with dimensionality reduction techniques significantly enhances clustering outcomes. Among the methods tested, DBO-Kmeans combined with NNMF emerged as the best-performing approach, providing the highest silhouette score and consistent clustering results.

4 Conclusion

This study presents an intelligent clustering framework that combines dimensionality reduction with advanced optimization algorithms. The results validate the framework's effectiveness in enhancing clustering stability, efficiency, and accuracy. Future work will explore hybrid deep learning models for further improvements.

References

[1] Hongfei Guo, Ru Zhang, X Chen, Z Zou, T Qu, G Huang, J Shi, M Chen, and H Gu. Quality control in production process of product-service system: A method based on turtle diagram and evaluation model. *Procedia CIRP*, 83:389–393, 2019.

[2] Hongfei Guo, Ru Zhang, Z Lin, T Qu, G Huang, J Shi, M Chen, H Gu, C Deng, and J Li. Research on task pricing of self-service platform of product-service system. *Procedia CIRP*, 83:380–383, 2019.

[3] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 1999.

[4] Ian T Jolliffe and Jorge Cadima. *Principal Component Analysis*. Springer, 2016.

[5] Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. *Advances in Neural Information Processing Systems*, 13:556–562, 2001.

[6] Seyedali Mirjalili. Sine cosine algorithm: A novel metaheuristic approach for solving optimization problems. *Knowledge-Based Systems*, 96:120–133, 2016.

[7] Seyedali Mirjalili et al. The salp swarm algorithm: A bio-inspired optimizer for engineering design problems. *Advances in Engineering Software*, 114:163–191, 2017.

[8] Yu Wang, Z Wang, D Zhang, and Ru Zhang. Discovering cultural differences in online consumer product reviews. *Journal of Electronic Commerce Research*, 20(3):169–183, 2019.

[9] Jiacong Wu, Yu Wang, Ru Zhang, and Jianhua Cai. An approach to discovering product/service improvement priorities: Using dynamic importance-performance analysis. *Sustainability*, 10(10):3564, 2018.