Comparison of the efficiency of autoencoders for solving clustering problems

Volodymyr Lytvynenko[1], Serge Olszewski[2], Volodymyr Osypenko[3], Violeta Demchenko[4], Daria Dontsova[4]

[1]*Kherson National Technical University, st. Instytutska, 11 Khmelnytskyi 29016 Ukraine*
[2] *Taras Shevchenko National University of Kyiv, 64/13, Volodymyrska Street, 01601 Kyiv, Ukraine*
[3]*Kyiv National University of Technology and Design, Mala Shyianovska str., 2,01011 Kyiv, Ukraine*
[4] *State Institution «Kundiiev Institute of Occupational Health of the National Academy of Medical Sciences of Ukraine», Saksaganskogo Str., 75, 01033 Kyiv, Ukraine*

Abstract
This study compares the effectiveness of autoencoders and variational autoencoders for clustering tasks, using the Iris dataset with k-means, spectral clustering, Affinity Propagation, and Gaussian Mixture Model. Clustering quality was assessed with metrics like the Silhouette Index, Davies-Bouldin Index, Adjusted Rand Index, and Mutual Information. Results showed that classical autoencoders performed more reliably and effectively, particularly with k-means, while variational autoencoders excelled with Affinity Propagation. The Gaussian Mixture Model was the least effective for both types. The study underscores the importance of choosing the right autoencoder and clustering algorithm based on the task and data structure, paving the way for future research on more complex datasets.

Keywords
autoencoders, variational autoencoders, k-means, affinity propagation, spectral clustering, Gaussian mixture model, silhouette score, Davis-Bouldin index, adjusted rand index, mutual information, latent space

1. Introduction

In recent decades, data volume has grown exponentially across various fields, becoming complex, multidimensional, and irregular, necessitating advanced analysis methods. Classical clustering methods like k-means often struggle with such data, especially with nonlinear dependencies or noise. Classical autoencoders, a type of neural network for dimensionality reduction, address this by compressing input data into a lower-dimensional latent space and reconstructing it, enabling more accurate clustering. Autoencoders automate feature selection by representing data in a low-dimensional space where clusters are more defined. They are used in fields such as image and text analysis, time series, and biomedical data, improving the extraction of valuable insights.

Autoencoders can integrate with other methods like variational autoencoders and GANs, enhancing adaptability for complex tasks. They are especially effective for high-dimensional data, addressing the "curse of dimensionality" by compressing data for better clustering. The growth in GPU and computing power has made training these models more accessible, boosting their relevance in modern data analysis. The need to analyze complex data, extract features, and

ITTAP'2024: 4th International Workshop on Information Technologies: Theoretical and Applied Problems, October 23-25, 2024, Ternopil, Ukraine, Opole, Poland
* Corresponding author.
† These authors contributed equally.
immun56@gmail.com (V.Lytvynenko); Olszewski.Serge@gmail.com (S.Olszewski); vvo7@ukr.net (V. Osypenko); chemioh@ukr.net (V. Demchenko); vigovskadasha@gmail.com (D. Dontsova)
0000-0003-4499-8485 (*V. Lytvynenko*); 0000-0003-4499-8485 (*S. Olszewski*); 0000-0002-1077-1461 (*V. Osypenko*); 0000-0001-6239-0882 (*V. Demchenko*); 0000-0003-3676-1672 (*D. Dontsova*)

integrate deep learning methods highlights the importance of autoencoders for achieving accurate results.

The main contributions of this paper are as follows: a)A comprehensive comparative analysis of the effectiveness of classical (AE) and variational (VAE) autoencoders for clustering tasks, applying various algorithmic approaches; b) Empirical evidence of the superiority of classical autoencoders in terms of stability and clustering quality, especially in combination with the k-means algorithm; c)The high efficiency of the Affinity Propagation algorithm in combination with VAE, indicating its potential for specific types of data;d)The implementation of a multi-criteria approach to clustering quality assessment, integrating several metrics, which provided a deep understanding of the effectiveness of the studied methods.

The rest of the paper is structured as follows. Section 2 provides a literature review, Section 3 presents the problem statement. Section 4 describes the materials and methods used in this study. Section 5 presents the results of testing the proposed clustering methods. Finally, Section 6 concludes the study.

2. Review of the literature

Autoencoders are effectively used for clustering by leveraging their latent representations. Clustering groups objects based on similarities, and autoencoders help extract latent data features and reduce dimensionality before clustering. Hinton and Salakhutdinov [1] introduced a neural network methodology for dimensionality reduction, showing that autoencoders create compact latent representations suitable for clustering. This foundational work advanced the use of autoencoders in data clustering. The Deep Embedded Clustering (DEC) model [2] uses an autoencoder to learn latent data representations and cluster them, jointly optimizing data reconstruction and cluster distribution for more accurate and stable results. Xie and colleagues' work was a significant advancement in autoencoder-based clustering methods. Variational autoencoders (VAEs) [3] are key in data analysis for modeling complex distributions and uncertainty. They provide a theoretical foundation for creating latent representations, enabling effective data clustering and making them popular for generative modeling and clustering tasks. In [4], the authors proposed a method combining convolutional autoencoders with deep learning for clustering, enhancing feature extraction and clustering quality, especially for complex data like images. In [5], Adversarial Autoencoders (AAEs) were introduced, combining autoencoders with GANs to create more structured latent representations, enhancing clustering and generative capabilities for data analysis. The paper [6] introduces the Deep Embedded Clustering (DEC) method, noting its limitations in preserving local data structure. An improved method, IDEC, addresses this by combining clustering with local structure preservation through a new objective function. Results demonstrate IDEC's superiority across metrics, with visualizations highlighting enhanced local structure retention and clustering performance. This work underscores the importance of local structure in deep clustering and offers a practical enhancement. The article [7] introduces the Deep Clustering Network (DCN), which combines deep learning with the K-means algorithm to perform nonlinear data mapping and clustering simultaneously. DCN outperforms classical and modern methods by transforming data into a more "K-means-friendly" space, improving clustering quality and advancing data analysis through the integration of deep learning with traditional clustering. The article [8] introduces a clustering approach combining autoencoders and traditional algorithms. It employs

autoencoders for nonlinear dimensionality reduction, applies K-means to the hidden representation, and iteratively optimizes parameters. The method excels in handling complex, nonlinear data, automatically extracting relevant features and outperforming traditional clustering techniques. The study highlights the potential of integrating autoencoders into unsupervised analysis and discusses variational autoencoders (VAEs), which model probabilistic distributions and create latent representations for clustering and generative tasks. Variational Autoencoders (VAEs), introduced by Kingma and Welling [9], model the latent space as a probabilistic distribution, enabling data generation and effective organization in the latent space. This makes VAEs ideal for clustering complex, high-dimensional data. The Variational Deep Embedding (VaDE) method [10] combines VAEs with clustering by modeling latent variables as a Gaussian mixture, enabling direct clustering in the latent space. It outperforms traditional methods like K-means, particularly on complex datasets. The Gaussian Mixture Variational Autoencoder (GMVAE) [11] combines variational autoencoders with Gaussian mixtures, modeling the latent space as a Gaussian mixture to enhance clustering and handle complex data structures. In [12], the authors enhance VaDE by combining variational autoencoders with deep embedding, improving clustering for high-dimensional data like images and texts. Autoencoders effectively extract and utilize latent representations for clustering. This study compares classical and variational autoencoders for clustering, analyzing their integration with algorithms like K-means, spectral clustering, GMM, and Affinity Propagation. It aims to identify their strengths, limitations, and suitability for different data types and requirements. Clustering quality will be evaluated using metrics such as silhouette, Davies-Bouldin Index, Adjusted Rand Index, and Mutual Information, providing objective insights and practical recommendations.

3. Problem statement

A formal statement of the problem can be formulated as follows:

Given:1. Data set $D = \{x_1, x_2, ..., x_n\}$, where $x_i \in \mathbb{R}^m$ is m-dimensional vector of features.

2. Multiple clustering methods based on autoencoders $M = \{M_1, M_2, ..., M_k\}$, where each method M_i can be a classical or variation autoencoder.

3. Set of clustering algorithms A = {K-means, Spectral Clustering, Gaussian Mixture Model, Affinity Propagation}.

4. Set of clustering quality metrics Q = {Silhouette Score, Davies-Bouldin Index, Adjusted Rand Index, Mutual Information, Adjusted Mutual Information}.

Required: 1) For each method $M_i \in M$ and algorithm $A_j \in A$: a) Learn the model M_i on the data D; b) Apply the algorithm A_j to the latent representation of the data obtained by the M_i; c) Calculate the values of all quality metrics $q \in Q$ for the resulting clustering; 2) Carry out a comparative analysis of the results obtained: a) Evaluate the effectiveness of each combination (M_i, A_j) across all metrics $q \in Q$; b) Identify the advantages and limitations of each method $M_i \in M$; c) Identify the most effective combinations of methods and algorithms for different types of data and clustering tasks; 3) To formulate recommendations for selecting the optimal autoencoder based clustering method and clustering algorithm depending on data characteristics and clustering quality requirements.

Limitations and assumptions: 1. The true cluster labels for the dataset D are assumed to be unknown (unsupervised learning problem); 2. The number of clusters K is assumed to be given

or determined automatically depending on the clustering algorithm used; 3. Computational resources and model training time are not considered as limiting factors in this problem formulation. This formal problem statement covers all aspects mentioned in the research objective and provides a clear structure for a comparative analysis of autoencoder based clustering methods.

4. Materials and Methods

4.1. Data

The IRIS dataset (https://www.kaggle.com/uciml/iris) was chosen for comparing classical (AE) and variational (VAE) autoencoders in clustering due to its known structure, interpretability, and manageable size, enabling efficient training and testing. Its explicit clustering structure (three classes) allows assessment of the models' ability to extract informative latent representations, making it an ideal baseline for such analyses.

4.2. Autoencodes

A classical autoencoder encodes data into a compact representation and reconstructs it, extracting features for clustering. Below is its mathematical description and application in clustering:**Autoencoder**

Architecture. The autoencoder consists of two main parts:-

***Encoder*:** Function $f_{encoder}$ converts the input data $x \in \mathbf{Rd}$ in the hidden view $z \in \mathbf{Rd}$.

$$z = f_{encoder}(x;\theta), \tag{1}$$

where θ — encoder parameters.

Decoder: Function $f_{decoder}$ reconstructs the original data $\hat{\mathbf{x}}$ from the hidden view $\mathbf{z}$.

$$\hat{\mathrm{x}} = f_{decoder}(\mathrm{z};\theta) \tag{2}$$

where $\hat{\mathbf{x}}$ is refurbished entrance, and θ is decoder parameters.

Loss function. The autoencoder is trained using a loss function that measures the difference between the original data $\mathbf{x}$ and recovered data $\hat{\mathbf{x}}$. Typically, the standard error of mean square error is used (MSE):

$$L_{recon} = \frac{1}{n}\sum_{i=1}^{n}\left\|\mathbf{x}_i - \hat{\mathbf{x}}_i\right\|^2 \tag{3}$$

where n is number of examples in the batches.

Application for clustering. After training the autoencoder and obtaining hidden representations for all the data, one can use these representations for clustering. The process can be described as follows*:*

a) **Obtaining latent representations:** Skip the entire dataset X through the encoder to get the hidden views Z:

$$Z = \left\{ z_i = f_{\text{encodet}}\left(x_i ; \theta \right) \mid x_i \in X \right\} \tag{4}$$

b) **Clustering:** Apply a clustering algorithm such as K-means or other method to the hidden representations Z for cluster tagging C:

$$C = Cluster\left(Z ; \phi \right) \tag{5}$$

where ϕ is parameters of the clustering algorithm.

Thus, a classical autoencoder can be described as a model that first encodes data into a hidden representation and then reconstructs data from that representation. Hidden representations obtained from the encoder are used for clustering, which are then clustered using a clustering algorithm.

Algorithm 1: Improved Autoencoder-based Clustering

Input: Dataset $X = \{x_1, \ldots, x_n\}$, $x_i \in \mathbb{R}^d$
Output: Cluster assignments $C = \{c_1, \ldots, c_n\}$
Data: Encoding dim m, clusters k, epochs E, batch size B, threshold ϵ
Initialize autoencoder params θ, clustering params ϕ;
$epoch \leftarrow 0$, $prev\_loss \leftarrow \infty$, $converged \leftarrow false$;
while *not converged and epoch* $< E$ **do**
  $total\_loss \leftarrow 0$;
  for *batch b of size B from X* **do**
    $z \leftarrow f_{\text{encoder}}(b; \theta)$;
    $\hat{b} \leftarrow f_{\text{decoder}}(z; \theta)$;
    $L_{\text{recon}} \leftarrow \text{MSE}(b, \hat{b})$;
    Update θ to minimize L_{recon};
    $total\_loss \leftarrow total\_loss + L_{\text{recon}}$;
  end
  if $|prev\_loss - total\_loss| < \epsilon$ **then**
    $converged \leftarrow true$;
  end
  $prev\_loss \leftarrow total\_loss$, $epoch \leftarrow epoch + 1$;
end
Encode dataset $Z \leftarrow f_{\text{encoder}}(X; \theta)$;
Apply clustering on Z with params ϕ to get initial C;
while *clustering not converged and iter* < 100 **do**
  Update cluster centroids ϕ, reassign points to clusters to get C;
  $iter \leftarrow iter + 1$;
end
Compute clustering quality metric Q;
return *Cluster assignments C, quality metric Q*

4.3. Variation Autoencoder

Variational Autoencoder (VAE) is a probabilistic model that encodes data as a probability distribution rather than a fixed representation. In clustering, VAE generates compact, informative data representations for clustering, with an encoder and decoder, differing from the classical autoencoder. instead of a pointwise latent representation $\mathbf{z}$ distribution is used $q(\mathbf{z}|\mathbf{x})$.

Encoder. The encoder converts the input data into distribution parameters (it's usually an average μ) and standard deviation σ):

$$z \sim q(z|x) = \mathsf{N}\left(\mu(x;\theta) \right), \tag{6}$$

where $\mu(x;\theta)$ and $\log q(z|x)$ are parameters that depend on the input data and network parameters θ.

Decoder. The decoder recovers data from a hidden view $\mathbf{z}$, which was sampled from the distribution $q(\mathbf{z}|\mathbf{x})$:

$$\hat{\mathrm{x}} \sim p(\mathrm{x}|\mathrm{z};\theta), \tag{7}$$

where $p(\mathrm{x}|\mathrm{z};\theta)$ is a distribution (e.g., Gaussian) defined by the decoder.
The variational autoencoder is trained by minimizing the two components of the loss function:
1. **Reconstructive error** (RMS error or cross-entropy) between the original data and the reconstructed data $\hat{\mathbf{x}}$:

$$L_{recon} = \mathrm{E}_{q(\mathbf{z}|x)}\left[-\log p(\mathbf{x}\,|\,\mathbf{z})\right] \tag{8}$$

2. **Kulbak-Leibler divergence** (KL- divergence) between the posterior distribution $q(\mathrm{z}|\mathrm{x})$ and a priori distribution $p(\mathrm{z})$, which is usually assumed to be the standard normal distribution $\mathrm{N}(0,\mathrm{I})$:

$$L_{KL} = D_{KL}\left(q(\mathbf{z}\,|\,\mathbf{x})\|p(\mathbf{z})\right) \tag{9}$$

Complete loss function:

$$L = L_{\text{recon}} + \beta L_{KL} \tag{10}$$

where β is the weighting factor that can be adjusted to control for the effect of KL-divergence (e.g. in the model β-VAE).

After training VAE to produce probabilistic representations of the data, these representations can be used for clustering:

a) **Obtaining latent representations:** Skip the entire dataset X through the encoder to get the distribution parameters $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$ and then sample the latent representations Z:

$$Z = \left\{\mathbf{z}_i \sim q(\mathbf{z}\,|\,\mathbf{x}_i)\,|\,\mathbf{x}_i \in X\right\} \tag{11}$$

b) **Clustering:** We apply the clustering algorithm to the sampled hidden representations Z for cluster tagging C:

$$C = Cluster(Z;\phi) \tag{12}$$

where ϕ is parameters of the clustering algorithm (e.g., K-means).

A variational autoencoder (VAE) is a probabilistic model that uses latent variables to represent data. After training, it generates latent representations for clustering and effectively handles complex, multi-level data structures.

Algorithm 2: VAE-Based Clustering Algorithm

Input: Dataset $X = \{x_1, \ldots, x_n\} \in \mathbb{R}^d$
Output: Cluster assignments $C = \{c_1, \ldots, c_n\}$, clustering quality Q
Data: Encoding dim. m, clusters k, epochs E, batch size B, threshold ϵ
Initialize VAE parameters θ, clustering parameters ϕ, $epoch \leftarrow 0$,
$prev\_loss \leftarrow \infty$;
while *not converged* ***and*** $epoch < E$ **do**
  $total\_loss \leftarrow 0$;
  for *each batch* b *of size* B *from* X **do**
    $z \leftarrow f_{\text{encoder}}(b; \theta)$, $\hat{b} \leftarrow f_{\text{decoder}}(z; \theta)$;
    $L_{\text{recon}} \leftarrow \text{MSE}(b, \hat{b})$, Update θ to minimize L_{recon},
    $total\_loss \leftarrow total\_loss + L_{\text{recon}}$;
  end
  if $|prev\_loss - total\_loss| < \epsilon$ **then**
    $converged \leftarrow$ true;
  end
  $prev\_loss \leftarrow total\_loss$, $epoch \leftarrow epoch + 1$;
end
$Z \leftarrow f_{\text{encoder}}(X; \theta)$, Apply clustering on Z using ϕ to obtain C;
for $iter \leftarrow 0$ **to** 100 ***and*** *not converged* **do**
  Update centroids ϕ, Reassign points to clusters C;
end
Compute clustering quality metric Q;
return C, Q

4.4. Internal Clustering Algorithms to Hidden Representations

The following algorithms were used to cluster the hidden representations: k-means, spectral clustering, Affinity Propagation and Gaussian Mixture Model.

k-means [14]. K-means clustering is a vector quantization method that divides n observations into k clusters by assigning each to the nearest centroid, which represents the cluster. This partitions the data space into Voronoi cells. The algorithm minimizes intra-cluster variance (sum of squared Euclidean distances), unlike the more complex Weber problem, which minimizes Euclidean distances. For greater accuracy in Euclidean distance minimization, k-median or k-medoid methods can be used.

Algorithm 3: K-means algorithm

Input: Data points X, number of clusters k
Output: Cluster assignments

for $i \leftarrow 1$ **to** k **do**
  $centroids[i] \leftarrow$ randomly selected data point from X;
end
while *not converged* **do**
  for $x \in X$ **do**
    $closest\_centroid \leftarrow$ find nearest centroid to x;
    assign x to cluster of $closest\_centroid$;
  end
  for $i \leftarrow 1$ **to** k **do**
    $centroids[i] \leftarrow$ calculate mean of points in cluster i;
  end
end

Explanation:

Step 1: Initialises the centroids by randomly selecting k points from the data. Cycle While: Iteratively performs the following steps until convergence or until the maximum number of iterations is reached: Step 2: Assigns each data point to the nearest centroid, forming clusters.

Step 3: Updates the centroids by computing the average of all points assigned to each cluster. Step 4: Checks convergence by evaluating whether the change in centroids is less than a given tolerance ϵ If the change is small enough or the maximum number of iterations is reached, the loop terminates. Return: Outputs the final clusters and their corresponding centroids.

Spectral Clustering [15]. Spectral clustering transforms data into a lower-dimensional space using eigenvectors of the Laplace matrix derived from a similarity graph, then applies standard clustering methods like k-means.

Algorithm 4: Spectral Clustering

Input: Data points X, number of clusters k
Output: Cluster assignments

Construct similarity matrix W: Calculate pairwise similarity between data points in X;
Construct Laplacian matrix L: Compute the Laplacian matrix from W;
Compute eigenvalues and eigenvectors of L: Find the k smallest non-zero eigenvalues and corresponding eigenvectors of L;
Form data matrix X': Use the k eigenvectors to form a new data matrix X';
Apply k-means clustering to X': Cluster the data points in X' using the k-means algorithm;

Step 1: Constructs the Laplacian matrix from the similarity matrix.
Step 2: Computes the first k eigenvectors of the Laplacian matrix.
Step 3: Forms a matrix U from these eigenvectors.
Step 4: Normalizes the rows of U.
Step 5: Applies the k-means algorithm to cluster the rows of U.
While loop: Iteratively refines the clustering by recalculating centroids and reassigning points until clusters stabilize.

Gaussian Mixture Model [16]. A mixture model is a probabilistic approach for representing subpopulations in data without identifying their membership. In model-based clustering, such as Gaussian Mixture Models (GMMs), data are modeled as a mixture of parametric distributions, with each cluster represented by a separate Gaussian distribution.

Algorithm 5: Gaussian Mixture Model (GMM)

Input: Data points $X = \{x_1, x_2, \ldots, x_n\}$, number of components k, tolerance ϵ, maximum iterations *max_iter*
Output: Parameters of the Gaussian components: $\{\pi_j, \mu_j, \Sigma_j\}_{j=1}^{k}$

Step 1: Initialize the parameters $\{\pi_j, \mu_j, \Sigma_j\}_{j=1}^{k}$:
- Mixing coefficients π_j, such that $\sum_{j=1}^{k} \pi_j = 1$.
- Means μ_j for each Gaussian component.
- Covariance matrices Σ_j for each Gaussian component.

while *not converged* ***and*** *iteration* $<$ *max_iter* **do**
E-Step: Compute the responsibility γ_{ij} that component j takes for data point x_i:

$$\gamma_{ij} \leftarrow \frac{\pi_j \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}{\sum_{l=1}^{k} \pi_l \mathcal{N}(x_i \mid \mu_l, \Sigma_l)}$$

M-Step: Update the parameters $\{\pi_j, \mu_j, \Sigma_j\}_{j=1}^{k}$ based on the responsibilities γ_{ij}:

- Update the mixing coefficients:

$$\pi_j \leftarrow \frac{1}{n} \sum_{i=1}^{n} \gamma_{ij}$$

- Update the means:

$$\mu_j \leftarrow \frac{\sum_{i=1}^{n} \gamma_{ij} x_i}{\sum_{i=1}^{n} \gamma_{ij}}$$

- Update the covariance matrices:

$$\Sigma_j \leftarrow \frac{\sum_{i=1}^{n} \gamma_{ij} (x_i - \mu_j)(x_i - \mu_j)^T}{\sum_{i=1}^{n} \gamma_{ij}}$$

Check Convergence: Compute the log-likelihood and check if the change is less than the tolerance ϵ.
end
Return the parameters $\{\pi_j, \mu_j, \Sigma_j\}_{j=1}^{k}$.

Explanation:
Step 1: Initializes the parameters of the Gaussian components:
Mixing coefficients π_j determine the proportion of each component.
Means μ_j represent the center of each Gaussian component.
Covariance matrices $\sum_j$ describe the spread of each component.
While loop: Iteratively performs the Expectation-Maximization (EM) steps until convergence or the maximum number of iterations is reached.

E-Step: Calculates the responsibilities γ_{ij}, which represent the probability that data point x_j belongs to component j.

M-Step: Updates the parameters $\{\pi_j, \mu_j, \Sigma_j\}_{j=1}^{k}$ based on the current responsibilities.

Check Convergence: The algorithm checks if the change in log-likelihood is below a certain tolerance ϵ

Return: After convergence, returns the final parameters of the Gaussian components.

The Affinity Propagation (AP) clustering algorithm is a method that identifies cluster centers (exemplars) through message passing between data points. Unlike traditional methods such as K-means, AP does not require the number of clusters to be specified in advance.

The Affinity Propagation (AP) clustering algorithm automatically finds clusters by determining exemplars (cluster centers) through message passing between data points. It uses a similarity matrix S, where $S(i, j)$ measures how similar points i and j are. The diagonal values $S(i, j)$ indicate the preference of each point to be chosen as an exemplar.

The process involves updating two matrices: the "responsibility" matrix R, which shows how suitable point j is as a center for point i, and the "availability" matrix A, which reflects how

Algorithm 6: Affinity Propagation for Clustering

Data: Similarity matrix $S(i,k)$ for all data points i and k
Input: Preference values $S(k,k)$ for all data points k
Result: Cluster centers and assignments for each data point
Initialize responsibility $r(i,k) \leftarrow 0$ and availability $a(i,k) \leftarrow 0$ for all i,k;
while *not converged* **do**
  // Update responsibility
  foreach *data point* i **do**
    foreach *candidate exemplar* k **do**
      $r(i,k) \leftarrow S(i,k) - \max_{k' \neq k}\{a(i,k') + S(i,k')\}$;
    end
  end
  // Update availability
  foreach *data point* i **do**
    foreach *candidate exemplar* k **do**
      if $i \neq k$ **then**
        $a(i,k) \leftarrow \min\left(0, r(k,k) + \sum_{i' \notin \{i,k\}} \max(0, r(i',k))\right)$;
      else
        $a(k,k) \leftarrow \sum_{i' \neq k} \max(0, r(i',k))$;
      end
    end
  end
end
// Identify exemplars
foreach *data point* i **do**
  Assign i to the exemplar k that maximizes $a(i,k) + r(i,k)$;
end

appropriate point j is as a center considering all other points. These matrices are updated iteratively until convergence. The number of clusters is determined automatically based on the data, making AP useful for tasks where the number of clusters is not known in advance.

4.5. Clustering Quality Assessment

Silhouette Score. The silhouette index evaluates cluster compactness and separation, ranging from -1 (misclassified) to 1 (well-separated), with 0 indicating overlap. It is defined for each object i as:

(1) Average internal distance $a(i)$:

This is the average distance from the object i to all other objects in the same cluster C.

$$a(i) = \frac{1}{|C|-1} \sum_{\substack{j \in C \\ i \neq j}} d(i,j) \tag{13}$$

where $d(i,j)$ is object spacing i and j (for example, the Euclidean distance).

(2) Average inter-cluster distance $b(i)$:

This is the average distance from the object i to all facilities in the nearest cluster C', into which the object i is not included.

$$b(i) = \min_{C' \neq C} \frac{1}{|C'|} \sum_{j \in C'} d(i,j) \tag{14}$$

Silhouette index $s(i)$:

For each object i its silhouette is defined as:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{15}$$

If $s(i)$ close to 1, then the object i is well categorised if is close to 0, then the object is on the boundary between clusters. If $s(i)$ close to -1, then the object was probably misclassified.

(3) Average silhouette index for the entire dataset:

The overall silhouette for the entire clustering is calculated as the average of the $s(i)$ across all facilities:

$$S = \frac{1}{n} \sum_{i=1}^{n} s(i) \tag{16}$$

where n is the total number of objects in the dataset.

The Silhouette Plot evaluates clustering quality by showing silhouette values for objects within clusters. Positive values (close to 1) indicate well-separated clusters, while negative values suggest misclassified objects.

Davis-Bouldin Index, (DBI) is a metric for assessing the quality of clustering [18]. This index measures the average "similarity" of each cluster to its most similar cluster, and the smaller the DBI value, the better separated the clusters are. The Davis-Bouldin index is based on two key concepts: the spread within a cluster and the distance between clusters.

(1) Scatter intra-cluster S_i:

Cluster spread C_i measures the average distance between all points within a cluster and its centroid $\mathbf{c}_i$.

$$S_i = \frac{1}{|C_i|} \sum_{x_j \in C_i} d\left(\mathbf{x}_j, \mathbf{c}_i\right) \quad (17)$$

where $d\left(\mathbf{x}_j, \mathbf{c}_i\right)$ is distance (e.g., Euclidean) between the point $\mathbf{x}_j$ and the centroid $\mathbf{x}_j$.

(2) Cluster Distance $d\left(C_i, C_j\right)$:

This is the distance between the centroids of the two clusters C_i and C_j.

$$d\left(C_i, C_j\right) = d\left(\mathbf{c}_i, \mathbf{c}_j\right) \quad (18)$$

(3) Similarity Index between two clusters R_{ij}:

It is defined as the ratio of the sum of the spreads of two clusters to the distance between them.

$$R_{ij} = \frac{S_i + S_j}{d\left(C_i, C_j\right)} \quad (19)$$

(4) Davis-Bouldin index for the cluster C_i:

This is the maximum of the similarity index R_{ij} with any other cluster.

$$R_i = \max_{j \neq i} R_{ij} \quad (20)$$

(5) Davis-Bouldin Index (DBI) for the entire clustering:

This is the average R_i across all clusters.

$$DBI = \frac{1}{k} \sum_{j=1}^{k} R_i \quad (21)$$

where k is total number of clusters.

Adjusted Rand Index (ARI). The Rand Index (RI) evaluates clustering quality by comparing predicted clusters to true labels. Adjusted for random matches, RI ranges from -1 to 1, with 1 indicating perfect clustering [19].

The Adjusted Rand Index (ARI) corrects for this shortcoming by providing a normalized value that accounts for random matches.

Formal definition of adjusted Rand index

1. Matching Matrix is Suppose we have two partitions of a dataset: U - true cluster labels (ground truth) and V is predicted cluster labels.

2.Let's construct a matching matrix, where each element of n_{ij} shows the number of points falling simultaneously into the cluster i partitioned U and cluster j partitioned V .2. **Rand Index (RI) Formula:** The Rand index measures the proportion of point pairs that either belong to the same clusters or different clusters in both partitions [21].

$$RI = \frac{\text{number of matched pairs}}{\text{total number of pairs}} \tag{22}$$

Adjusted Rand Index (ARI):

$$ARI = \frac{\sum_{ij}\binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}\right] / \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}\right] / \binom{n}{2}} \tag{23}$$

ARI accounts for the probability of random matches by normalizing the Rand index. The formula is as follows: where: n is total number of data points; a_i is line amount i matching matrix (i.e. the number of points in the cluster i partitioned U ; b_j is sum on the column of the matching matrix (i.e. the number of points in the cluster j partitioned V ; $\binom{n}{2}$ is total number of object pairs.

ARI = 1: Complete correspondence between predicted and true partitioning (perfect clustering); ARI = 0: Clustering is no better than random partitioning; ARI < 0: The result is worse than random clustering.

Mutual Information (MI) and its corrected version, **Adjusted Mutual Information (AMI)**, are metrics used to assess the quality of clustering. They measure how well one partitioning of data (clustering) agrees with another, taking into account information common to both partitions [16].

Mutual Information between two partitions U and V (e.g., true partitioning and predicted partitioning) measures the amount of information common to both partitions.

Definition: Reciprocal information measures the extent to which knowledge of partitioning U reduces the uncertainty about the partitioning V .

$$MI(U,V) = \sum_{i=1}^{|U|}\sum_{j=1}^{|V|} P(i,j)\log\frac{P(i,j)}{P(i)P(j)} \tag{24}$$

where: $P(i,j)$ is probability that a randomly selected object belongs simultaneously to the cluster i to U and cluster j to V ; $P(i)$ and $P(j)$ is marginal probabilities that the object belongs to the cluster i to U and cluster j to V respectively.

MI takes values from 0 to $\log\left(\min\left(|U|\right),|V|\right)$.The value 0 means that the partitions U and V are independent (no common information). Higher MI values mean more dependence between partitions, i.e. better cluster matching.

Adjusted Mutual Information (AMI)

AMI — is a corrected version of MI that accounts for the probability of random matches between partitions. It is a normalized metric that removes the positive bias of MI. AMI is calculated as follows:

$$AMI\left(U,V\right)=\frac{MI\left(U,V\right)-E\left[MI\left(U,V\right)\right]}{\max\left(H\left(U\right),H\left(V\right)\right)-E\left[MI\left(U,V\right)\right]} \quad (25)$$

where: $E\left[MI\left(U,V\right)\right]$ is mathematical expectation of mutual information between random partitions; $H\left(U\right)$ **и** $H\left(V\right)$ is partition entropies U and V respectively.

AMI takes values from -1 to 1.

AMI = 1: Full correspondence between the partitions.

AMI = 0: Conformity is no better than casual conformity.

AMI < 0: Conformity is worse than random (which is extremely rare).

5. Experiments and Results

Experimental results obtained on clustering when applying autoencoders

Table1.

When applying internal clustering algorithms

Index name	k-means	AP	SP	GMM
Silhouette Score	0,5869039	0,5779381	0,5832608	0,3563197
Davis-Bouldin Index, (DBI)	1,097961	0,967877	1,009742	0,9561066
Adjusted Rand Index (ARI)	0,8681109	0,839829	0,8342589	0,3622746
Mutual Information, (MI)	0,9330692	0,8971948	0,9043042	0,4296664
Adjusted Mutual Information (AMI)	0,8681109	0,8339829	0,8342589	0,3933723

Autocoders

Silhouette Score: The values range from 0.3563 to 0.5869.The best result is for k-means clustering algorithm, indicating good cluster separation.The lowest result is for Gaussian Mixture Model (GMM), indicating weaker separation.
Davis-Bouldin Index (DBI): The values range from 0.9561 to 1.0979.GMM has the lower value which indicates better clustering.
Adjusted Rand Index (ARI): he values range from 0.3622 to 0.8681.The best result is again for k-means, which confirms the high quality of clustering compared to true labels.
Mutual information indices (MI) and AMI: MI: 0.4296 до 0.9331. AMI: 0.3933 до 0.8681. The in-house k-means algorithm performs better on both MI and AMI.

Experimental results obtained on IRISDATA clustering using VAE variational autoencoders

Table 2.
When applying the internal clustering algorithm of the variational autoencoder

Index name	k-means	AP	SP	GMM
Silhouette Score	0,6732217	0,1494412	0,6951292	0,1494412
Davis-Bouldin Index, (DBI)	0,9988075	0,9601669	0,4153072	1,1864
Adjusted Rand Index (ARI)	0,6813073	0,3901078	0,6516407	0,4078
Mutual Information, (MI)	0,7900837	0,4664558	0,7766257	0,4599
Adjusted Mutual Information (AMI)	0,6813073	0,4253811	0,7272799	0,4323

Variational Autoencoders (VAE)
Silhouette Index: The values range from 0.0149 to 0.6951 Affinity Propagation has the best value which shows good cluster separation.
Davis-Bouldin Index: Values from 0.0415 to 1.1864. Affinity Propagation has the best DBI value indicating better clustering results.
Adjusted Rand Index (ARI): Values from 0.0408 to 0.6813. k-means and Affinity Propagation show the best results.
Mutual Information Index (MI) and AMI:MI: 0.0459 to 0.7901. AMI: 0.0432 to 0.6813. k-means shows the best score for MI and AMI.
Autoencoders show better clustering performance, especially when using k-means, as shown by high Silhouette Score, ARI, MI and AMI values.

Variational autoencoders show competitive results, especially when using Affinity Propagation, although in some cases (e.g., GMM) the quality of clustering is lower, as indicated by the low values of the metrics. Thus, when comparing the two approaches, autoencoders (especially when combined with k-means) show more stable and better results in clustering tasks compared to variational autoencoders.

Experimental results obtained on IRISDATA clustering using autoencoders

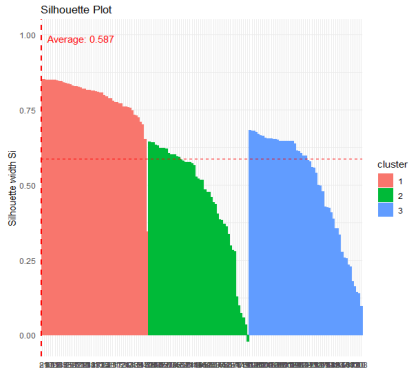

Figure1: When applying the internal k-means clustering algorithm

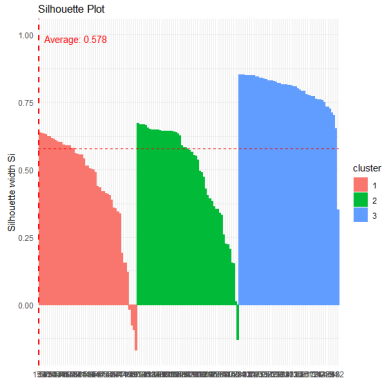

Figure 2: When applying the internal spectral clustering algorithm

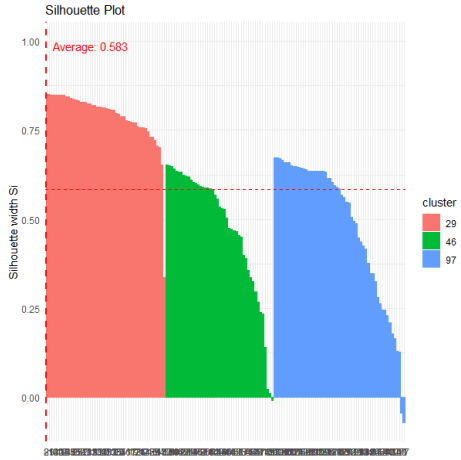

Figure 3: When applying Affinity Propagation's internal clustering algorithm

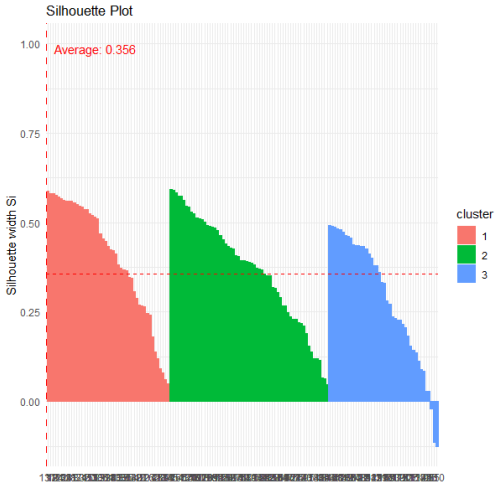

Figure 4: When applying the Gaussian Mixture Model internal clustering algorithm

Experimental results obtained on IRISDATA clustering using VAE variational autoencoders

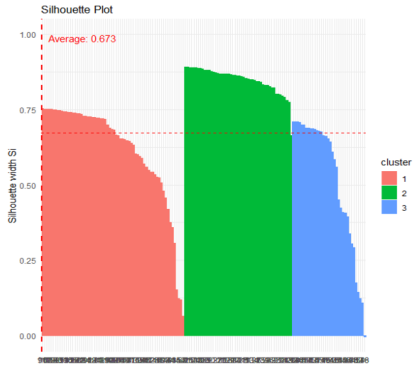

Figure 5: When applying the internal k-means clustering algorithm of the variational autoencoder

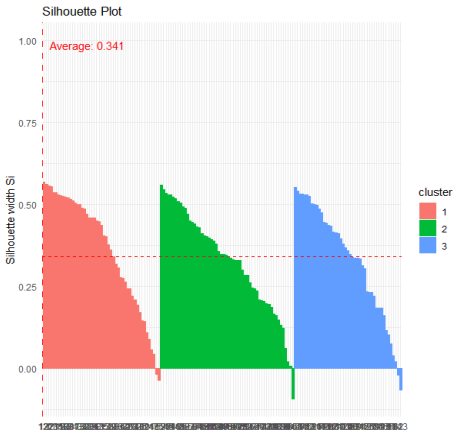

Figure 6: When applying the internal algorithm of spectral clustering of the variational autoencoder

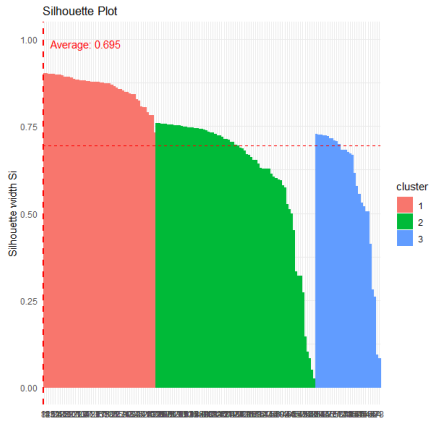

Figure 7: When applying the internal Affinity Propagation clustering algorithm of the variational autoencoder

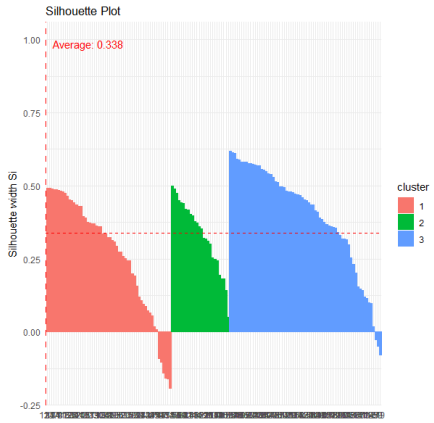

Figure 8: When applying the internal Gaussian Mixture Model clustering algorithm of the variational autoencoder

Evaluation of clustering results based on Silhouette Index on graphical representations Autocoders:

Figure 1: k-means. The k-means silhouette index indicates well-separated, high-quality clusters, with most samples showing high positive values and balanced cluster sizes.

Figure 2: Spectral clustering. Spectral clustering shows less distinct separation and less balanced cluster sizes than k-means, with a lower Silhouette Index but still good quality.

Figure 3: Affinity Propagation. Clustering results using Affinity Propagation show a low Silhouette Index, indicating less clear separation of clusters. The clusters overlap, making them difficult to interpret.

Figure 4: Gaussian Mixture Model (GMM). The GMM silhouette index is lower than k-means and spectral clustering, indicating poor separation with significant overlap and many negative values, particularly in one cluster.

Variational autoencoders (VAE):

Figure 5: k-means. The VAE with k-means silhouette plot shows moderate clustering quality, with less distinct separation than autoencoders, though the clusters are relatively balanced in size.

Figure 6: Spectral clustering. Spectral clustering using VAE shows low Silhouette Index values. Demonstrates very poor cluster separation. Graphical representation shows significant overlap of clusters, indicating poor clustering quality.

Figure. 7: Affinity Propagation. The silhouette index for Affinity Propagation using VAE shows the lowest quality among all the methods considered. The graph shows strong overlapping of clusters, indicating poor data separation.

Figure 8: Gaussian Mixture Model (GMM). The GMM with VAE graph shows a low Silhouette Index, indicating poor clustering with weak, overlapping clusters and many negative values.

Autocoders k-means and Spectral Clustering show the best results on Silhouette Index, showing clear and qualitative separation of clusters. While GMM and Affinity Propagation show less clear separation.

c Variational autoencoders (VAE) VAE-based methods generally perform worse than classical autoencoders, especially with Affinity Propagation and GMM, where clusters are almost indistinguishable. GMM gives the poorest results for both. Autoencoders offer more stable, better cluster separation, while VAE shows extreme results, excelling with some methods (e.g., Affinity Propagation) and struggling with others (e.g., spectral clustering).

6. Conclusion

Classical autoencoders (AE) generally perform more stably and yield better clustering results than variational autoencoders (VAE). K-means is most effective with AE, while Affinity Propagation works best with VAE, particularly for cluster separation. GMM shows the worst performance for both. Silhouette Score graphs confirm AE's better separation, especially with k-means and spectral clustering, while VAE shows more variability, performing well with some algorithms (e.g., Affinity Propagation) and poorly with others (e.g., spectral clustering). The study highlights the need for tailored choices of autoencoder and clustering algorithm, especially for small datasets like Iris, and underscores the importance of using multiple metrics for a comprehensive evaluation.

7. FUTURE RESEARCH

Ours will be directed toward researching ref data and problems. In particular, we plan to use these algorithms to predict the toxic properties of different pesticides depending on their structural 3D formulas.

References

[1] Hinton G.E, Salakhutdinov R.R. “Reducing the dimensionality of data with neural networks”. Science. 2006 Jul 28; 313(5786), (2006): 504-7. https://doi.org/10.1126/science.1127647. PMID:16873662

[2] Xie, J., Girshick, R., & Farhadi, A. Unsupervised deep embedding for clustering analysis, in: International conference on machine learning. PMLR. 2016, pp. 478-487. https://doi.org/10.48550/arXiv.1511.06335

[3] Kingma, D. P., & Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.2013. https://doi.org/10.48550/arXiv.1312.6114

[4] Dizaji, K. G., Herandi, A., Deng, C., Cai, W., & Huang, H. Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization, in: Proceedings of the IEEE international conference on computer vision, 2017, pp. 5736-5745. https://doi.org/10.48550/arXiv.1704.06327

[5] Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., & Frey, B. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*. 2015. https://doi.org/10.48550/arXiv.1511.05644

[6] Guo, X., Gao, L., Liu, X., & Yin, J. Improved Deep Embedded Clustering with Local Structure Preservation, in: International Joint Conference on Artificial Intelligence (IJCAI). 2017, pp. 1753-1759. URL https://www.ijcai.org/proceedings/2017/243

[7] Yang, B., Fu, X., Sidiropoulos, N. D., & Hong, M. Towards K-means-friendly spaces: Simultaneous deep learning and clustering, in: International Conference on Machine Learning, PMLR.. 2017, pp. 3861-3870. URL: http://proceedings.mlr.press/v70/yang17b.html

[8] Song, C., Liu, F., Huang, Y., Wang, L., & Tan, T. Auto-encoder based data clustering, in: Iberoamerican Congress on Pattern Recognition. Springer, Berlin, Heidelberg, pp. 117-124. URL: https://link.springer.com/chapter/10.1007/978-3-642-41822-8_15

[9] Kingma, D. P., & Welling, M. Auto-Encoding Variational Bayes. 2014. https://arxiv.org/abs/1312.6114

[10] Jiang, Z., Zheng, Y., Tan, H., Tang, B., & Zhou, H. Variational deep embedding: An unsupervised and generative approach to clustering, 2017. https://arxiv.org/abs/1611.05148

[11] Dilokthanakul, N., Mediano, P. A. M., Garnelo, M., Lee, M. C. H., Salimbeni, H., Arulkumaran, K., & Shanahan, M. Deep unsupervised clustering with Gaussian mixture variational autoencoders, 2016. https://arxiv.org/abs/1611.02648

[12] Zhang, L., Jiang, Z., Zhang, Y., & Zhou, H. Variational Autoencoder with Deep Embedding: A Generative Approach for Clustering, 2019. https://arxiv.org/abs/1906.11242

[13] Hartigan, J. A., & Wong, M. A. Algorithm AS 136: A k-means clustering algorithm. Journal of the Royal Statistical Society. Series C (Applied Statistics), 28(1), (1979): 100-108. https://doi.org/ 10.2307/2346830

[14] Ng, A. Y., Jordan, M. I., & Weiss, Y. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems,* 14, (2002): 849-856. https://doi.org/10.7551/mitpress/1198.003.0104

[15] Fruhwirth-Schnatter, S. Finite Mixture and Markov Switching Models. Springer. 2006. ISBN 978-0-387-32909-3.

[16] Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer. (Chapter 9: Mixture Models and EM), 2006. https://doi.org/10.5555/1162264

[17] Peter J. Rousseeuw. "Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis". Computational and Applied Mathematics. 20, (1987): 53–65. https://doi:10.1016/0377-0427(87)90125-7.
[18] Davies, David L.; Bouldin, Donald W. "A Cluster Separation Measure". IEEE Transactions on Pattern Analysis and Machine Intelligence. PAMI-1 (2): 1987, pp. 224–227. https://doi:10.1109/TPAMI.1979.4766909. S2CID 13254783.
[19] Gates A. J., Ahn Y. Y. The impact of random models on clustering similarity. Journal of Machine Learning Research. T. 18. №. 87. (2017): 1-28. https://doi.org/10.48550/arXiv.1701.06508
[20] Vinh, N. X.; Epps, J.; Bailey, J. "Information theoretic measures for clusterings comparison". Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09. 2009, p. 1. https://doi.org/10.1145/1553374.1553511. ISBN 9781605585161