Cureus Journal of Computer Science

Open Access Review Article

Unsupervised Learning: Techniques, Applications, and Testing Methodologies

Protyasha Roy [1] , Sajjan G. Shiva [1]

1. Computer Science, University of Memphis, Memphis, USA

Corresponding author: Protyasha Roy, proy1@memphis.edu

Received 04/30/2025
Review began 05/18/2025
Review ended 10/20/2025
Published 10/23/2025

DOI:
https://doi.org/10.7759/s44389-025-06198-1

Abstract

Finding hidden patterns in data to reveal meaningful relationships is one of the major uses of unsupervised learning (UL). With the increasing volumes of raw, unlabeled data across applications such as anomaly detection, fraud detection, recommender systems, genetic research, etc., UL techniques have become widely used for discovering meaningful features in datasets without supervision. Various UL algorithms including clustering algorithms, Apriori algorithm, autoencoders, etc., support these applications, each with their own strengths and limitations. It is important to evaluate UL algorithms to ensure the reliability and accuracy of their outcomes. In this paper, we aim to address three key questions: (i) Which domains have successfully applied UL techniques? (ii) What are the strengths and limitations of UL algorithms? (iii) What testing methodologies exist for UL?

Categories: Evaluation Methods, Algorithm Analysis, Machine Learning (ML)
Keywords: unsupervised learning, clustering algorithm, testing approach, real applications, autoencoder

Introduction And Background

Unsupervised learning (UL) is a type of machine learning (ML) that operates without specific human guidance or instruction. It learns from large amounts of unlabeled data to uncover hidden patterns. For example, consider a task where the goal is to categorize cats and dogs using a large dataset of animals with various characteristics such as color, size, tail length, etc. However, the dataset does not contain any labels indicating whether an animal is a cat or a dog. Therefore, it is not possible to guide the learning process directly by specifying the category to which each data instance belongs. This is where UL becomes useful. Using UL, data with similar characteristics are organized into distinct groups, and based on these groups, one can infer which group represents cats and which represents dogs.

In real world, datasets often consist of raw, unlabeled data, making them well suited for UL tasks. UL can identify and extract relevant patterns or features, including previously unknown structures or relationships. That is why UL techniques are applied in a wide range of tasks, including anomaly detection in medical records, fraud detection in financial transactions, customer segmentation in marketing, etc.

UL primarily involves three key approaches: clustering, association, and dimensionality reduction [1]. Clustering is a widely used approach. In clustering, data points are grouped according to their similarities or differences by clustering algorithms. Some popular algorithms for clustering include K-means (KM), X-means (XM), and Expectation-Maximization (EM). Association is a rule-based approach. It is helpful in finding similar patterns and connections among various dataset features. Apriori algorithms and Eclat algorithms are used for association. Dimensionality reduction reduces the number of irrelevant features and extracts important features in a dataset. Principal component analysis (PCA) and singular value decomposition (SVD) are examples of dimensionality reduction techniques.

Besides UL, the other two types of ML approaches are supervised learning (SL) and reinforcement learning (RL). SL learns the relationship between input and output data to categorize them and make predictions on new, unseen data. Unlike UL, SL uses labeled datasets. On the other hand, through a process of trial and error, RL identifies the optimal path to achieve the best outcomes by learning from the feedback of each action. An RL system consists of several key components: the agent (the decision-making algorithm or system), states (the environment at a particular time), actions (steps taken to navigate the environment), rewards (feedback received based on taken actions), and a policy (a strategy that maps an agent's state to actions).

The primary aim of this review is to examine testing approaches for UL systems, while placing the discussion in the broader context of UL algorithms and their applications. To guide this analysis, three research questions are considered. First, *which domains have successfully applied UL techniques?* This highlights the practical use of UL in different areas. Second, *what are the strengths and limitations of UL algorithms?* This helps to assess current methods of UL with both their capabilities and challenges. Third, *what testing methodologies exist for UL?* This draws attention to a new yet important aspect of UL study -

how UL systems can be evaluated when ground truth is missing.

The above questions define the scope of the article: survey of UL applications, assessment of the strengths and weaknesses of current methods, and review of available testing approaches. By organizing the review of these questions, the paper provides a clear picture of the field and identifies gaps and opportunities where further research is needed.

The next section presents an overview of key UL algorithms, explores primary domains where UL has been effectively applied, and reviews existing testing methodologies of UL. The Review section analyzes and discusses the research questions and the last section concludes with future research directions.

Background

Unsupervised Learning Algorithms

Clustering: Clustering is the process of grouping data elements according to their similarities or differences. It can be classified into four types: centroid-based, density-based, distribution-based, and hierarchical [2]. In Centroid-based Clustering, the centroid, or cluster center, is the arithmetic mean or median of all the points in the cluster. Data points are assigned to the cluster with the nearest centroid, resulting in the formation of non-hierarchical clusters. Examples of centroid-based clustering include KM and XM.

KM [3] works by minimizing the sum of distances between the data points and their cluster centroids and assigns data points closest to a centroid to the same cluster. KM clustering is simple, scalable, and suitable for large datasets. However, the performance of KM depends on the centroid initialization. A poor initialization can result in longer convergence time and low-quality clusters. Also, KM tends to perform poorly when clusters vary in size or when outliers are present. The numbers of clusters need to be predefined for KM, but XM does not require this. XM [4] applies KM to identify initial cluster assignments and then perform cluster splitting to determine the optimal number of clusters. XM provides a partial solution to the local optimum issue by searching for the true number of clusters in a predefined range. Also, XM has better performance than KM in cases where clusters were largely separated. However, XM is sensitive to initial conditions and noisy data just like KM.

In density-based clustering, data points located within the densely populated regions of data space are grouped to the same cluster. This enables the identification of clusters with varying shapes and sizes. Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [5] is one of the density-based clustering algorithms. In DBSCAN, data points are classified as core, border, or noise point. Core points within a certain distance and with enough nearby points are grouped into same cluster. Border points are assigned to the cluster of closest core point. Points that do not fit any cluster are outliers or noise points. The main advantage of DBSCAN is that it can identify clusters of arbitrary shapes while effectively filtering out noise. It also works well when the data contains clusters of similar density. However, it struggles when there is high dimensional data, and the clusters have different density.

Distribution-based clustering groups together data points according to the probability distribution, such as Gaussian distributions. EM algorithm [6] is a distribution-based clustering approach for performing maximum likelihood estimation when latent variables are involved. The EM algorithm performs the expectation or E-step and maximization or M-step until convergence. The E-step estimates the missing or latent variables, and the M-step optimizes the parameters of the model. EM algorithm needs good initial parameter estimates, as it may fail to converge and can become trapped in local minima.

Hierarchical clustering builds a tree of clusters by grouping data based on similarities and then repeatedly merging them according to their hierarchical relationships. Examples of hierarchical clustering algorithms include agglomerative nesting (AN) and farthest-first traversal (FF).

AN [7] uses a bottom-up approach, starting with each data point as its own cluster, and then repeatedly merging clusters into larger ones until only one overall cluster remains. To merge two similar clusters, there are several linkage criterions - complete (maximum distance between points in each cluster), single (minimum distance between the clusters), and average (average distance between all pairs of points across the two clusters). AN does not require a predefined number of clusters and can handle clusters with different shapes and sizes. However, a limitation of AN is that once two data points are merged into a cluster, they cannot be separated even when it is discovered that they are not actually similar.

FF [8] is an approximation algorithm for solving the k-center problem, which aims to find an optimal k-clustering by minimizing maximum cluster radius. It begins by selecting an arbitrary point, then repeatedly chooses the point farthest from the current set until k points are chosen. Because FF selects each new centroid as the point farthest from the existing ones, it creates a large difference between clusters and tends to form well-separated groups. However, it is very sensitive to initialization conditions

[9].

Association: Association rule mining is a method that reveals relationship between variables in a given dataset. Apriori algorithm is one of the common association rule approaches that [10] reflects the use of prior knowledge. It performs a breadth-first search and finds frequent item combinations by building on smaller frequent sets. It assumes that adding items to a frequent set can only make it less frequent. If an itemset is infrequent, then all its supersets are infrequent as well. Apriori algorithms are used in recommendation engines and online retail applications. They analyze data to find item groups that often appear together and estimate the chance of selecting an item based on the selection of another. Apriori algorithms are simple and adaptable; however, they are not efficient at working with large datasets. Also, they are computationally expensive due to the generation of many itemset candidates.

Dimensionality reduction: Dimensionality reduction reduces the number of irrelevant or random features, or dimensions, and extracts important features in a dataset. PCA [11] is a dimensionality reduction technique that provides a simplified version of the data while minimizing information loss. In addition to dimensionality reduction, PCA can also reveal underlying patterns and produce a more informative representation that is less affected by noise. PCA helps to reduce overfitting in regression-based algorithms by lowering the dimensionality of the training dataset beforehand. However, after applying PCA, it is challenging to identify which features are most significant in the dataset. Also, information loss is inevitable during the dimensionality reduction process of PCA.

Another common UL algorithm is autoencoders [12], which are neural networks that compress input data into lower dimensional representation and then reconstructs the original input. Multiple autoencoders are stacked together to train DNNs. Besides dimensionality reduction, autoencoders are also capable of noise removal. Autoencoders can be trained to remove noise or unnecessary information from input data. However, autoencoders have certain disadvantages, including sensitivity to hyperparameters such as the number and size of layers, learning rate, and others. When trained on small datasets, they face overfitting issues. Also, autoencoders often capture only low-level features. They may have difficulty representing high-level features in the data.

Table *1* summarizes the strengths and weaknesses of the UL algorithms discussed.

Algorithms	Strengths	Weaknesses
KM	Simple and easy to implement	Poor centroid initialization leads to slower convergence and low-quality clusters; performs poorly with varying cluster sizes or outliers
XM	No need for predefined clusters; provides a partial solution to the local optimum; outperforms KM on largely separated clusters	Sensitive to initial conditions and noisy data
DBSCAN	Identifies clusters of arbitrary shapes; filters noise; performs well with similar-density clusters	Struggles with high-dimensional data, and varying cluster densities
EM	Performs well with latent variables	Needs good initial parameter estimates, or it may fail to converge and can get stuck in local minima
AN	Does not require a predefined number of clusters; can handle clusters with different shapes and sizes	Once merged, data points cannot be separated, even if later found dissimilar
FF	Selects each new center as the farthest point, leading to well-separated clusters	Very sensitive to initialization conditions
Apriori algorithm	Simple and adaptable	Not efficient with large datasets; computationally expensive
PCA	Reduces overfitting in regression-based algorithms	Difficulty in identifying significant features in the dataset; information loss
Autoencoder	Dimensionality reduction; noise removal	Sensitive to hyperparameters such as the number and size of layers, the learning rate, etc.; overfitting with small datasets; difficulty in representing high-level features in the data

TABLE 1: Strengths and weaknesses of UL algorithms

AN, Agglomerative Nesting; DBSCAN, Density-Based Spatial Clustering of Applications with Noise; EM, Expectation-Maximization; FF, Farthest-First Traversal; KM, K-Means; PCA, Principal Component Analysis; UL, Unsupervised Learning; XM, X-Means

Unsupervised Learning Applications

Compared to unlabeled data, acquiring and storing labeled data are more time consuming and expensive. One of the main advantages of UL is that it uses unlabeled data. Most real-world word applications employ unlabeled data. UL is capable of analyzing large data sets and discovering novel relationships or structures within them. These findings can help identify significant trends or groupings, leading to a deeper understanding of the data. A variety of applications, such as anomaly detection, recommendation engines, generation of customer persona profiles, fraud detection, natural language processing applications, genetic research, medical imaging, etc., use UL. We provide brief descriptions of typical UL applications and algorithms employed next.

Anomaly or outlier is anything that is not normal or expected. Anomaly detection, or outlier detection, identifies unexpected and abnormal observations, events, or data points within a dataset. DBSCAN clustering was used by Nanehkaran et al. [13] to diagnose anomalies in heart patients and accurately detect those with heart conditions. By employing adaptive parameters, DBSCAN improved the clustering accuracy of normal instances and effectively identified abnormal cases. The accuracy for predicting heart patients is approximately 95%. This is attributed to the adjustment of density-based clustering parameters based on the number of samples in the cluster and the average distances between clusters. To detect anomalies in the shuttle and satellite datasets, Shriram and Sivasankar [14] applied four UL techniques. The shuttle dataset provided insights into the appropriate control actions for space shuttle landings under different external conditions. And the satellite dataset contained spectral values of pixels in a 3 × 3 satellite image neighborhood. The UL techniques used include One-Class Support Vector Machine (OneClassSVM), local outlier factor (LOF), isolation forest, and elliptic envelope.

Recommendation engines use ML algorithms to identify patterns in user behavior data and suggest items that are most relevant to each user. By identifying group similarities among users to create a movie recommendation system, Cintia Ganesha Putri et al. [15] employ different clustering algorithms, including KM algorithm, birch algorithm, mini-batch KM algorithm, mean-shift algorithm, affinity propagation

algorithm, AN clustering algorithm, and spectral clustering algorithm. They used Genre and Tags for grouping movies. Among the experimented algorithms, Birch demonstrated the best performance. Addagarla and Amalanathan [16] proposed a similar image-based recommender system, where a dimensionality reduction technique using PCA through SVD was applied to transform extracted features into lower-dimensional space. To address the initialization sensitivity of initial centroid selection, they proposed K-mean++ algorithm [17], which optimizes the selection of the initial cluster centroids. Their approach, PCA-SVD with K-mean++, was compared with five other UL algorithms - Minibatch, K-Mediod, AN, Brich, and the Gaussian Mixture Model. Experimental results show that K-mean++ algorithm was more effective in grouping PCA-transformed images and outperformed others for recommending similar image products.

Customer segmentation is the process of dividing customers into distinct groups based on common characteristics like behaviors, preferences, etc. This improves efficiency of marketing campaigns, sales, and customer services. Using KM clustering algorithm, Shen [18] grouped customers based on an online transaction platform dataset. The author also used Apriori algorithm to analyze product purchase patterns. The identified customer groups provide insights into consumer behaviors or product preferences to help the business adjust the marketing strategies appropriately. Du Toit et al. [19] also used KM algorithm to group similar daily energy load profiles based on metering data. Power utilities can utilize this information for planning outages, making network investment decisions, predicting future load increase, and performing predictive maintenance.

Fraud detection is the process of identifying suspicious activities or anomalies, especially in financial transactions to reveal potential instances of fraudulent behavior that may indicate theft of money, data, or resources. Two deep UL models, autoencoder and variational autoencoder (VAE), were incorporated by Gomes et al. [20] to learn more about the components like marital status, education, etc., that lead to insurance fraud. VAE [21] is one kind of generative deep learning model used to generate new data in the form of variations of input data. Like autoencoder, VAE also has two independent parameterized models - the encoder or recognition model and the decoder or generative model. Monamo et al. [22] used trimmed KM clustering to detect fraudulent activities on the Bitcoin network based on transaction patterns. Trimmed KM [23] is a technique used to make KM more robust. Based on credit card transactions, Rezapour [24] detected fraudulent activities using autoencoder, OneClassSVM, and Mahalanobis outlier detection.

Genetic research is the study of genes and environmental factors that contribute to the progression of disease. The majority of chronic illnesses differ in their genomic basis, disease progression, clinical manifestations, and response to treatment. This variability highlights the importance of identifying meaningful traits within patient groups. Lopez et al. [25] used AN clustering algorithm to cluster patients based on their genomic similarity. They also conducted a gene pathway analysis to explore potential connections among the identified clusters.

Testing Methodologies for Machine Learning Systems

As mentioned earlier, testing methodologies for ML systems are an active area of research. We provide brief outline of testing methods in this section, followed by highlighting the testing methodologies for UL, SL, and RL systems. Our focus is to evaluate SL and RL testing methodologies for their applicability or adoption to UL system testing.

Symbolic Analysis with Statistical Hypothesis Testing: Symbolic execution is a program analysis technique where inputs are considered as symbolic variables rather than actual numbers [26]. After covering all possible execution paths in a program, expressions that describe the conditions needed to follow each path are created. A constraint solver then finds the actual inputs that satisfy these conditions. This way symbolic execution generates test cases that cover different paths.

For example,

If x>0 Then

Print “Positive”

Else

Print “Negative”

Instead of assigning concrete or actual values to x, symbolic execution treats it as symbolic variable and explores different execution paths.

Path 1: If x>0, output “Positive” ; path constraint {x>0}

Path 2: If x<0, output “Negative” ; path constraint {x<0}

Then, test inputs will be generated that can satisfy each path constraint, such as for path 1, x = 1 will give output “Positive” and for path 2, x = -1 will give output “Negative”.

Hypothesis testing [27] uses statistical methods to determine whether the probability of a given hypothesis is true or not. The first step is to create a null hypothesis and alternative hypothesis. The null hypothesis suggests that the observed data is the result of random chance and alternative hypothesis says that observed data shows real effect combined with the chance variation. Then, to evaluate the null hypothesis, a test statistic needs to be selected. Next step is to compute p-value and compare the p-value with a significance level. Assuming the null hypothesis is true, p-value measures the probability of the observed results occurring by chance. If the p-value is smaller than the significance level, it indicates that observed data is unlikely to have occurred by chance, leading to the rejection of the null hypothesis and accepting alternative hypothesis.

Metamorphic testing: Oracle refers to the technique of verifying the correctness of system under test. Metamorphic testing is used to handle this oracle problem. In the metamorphic testing technique, transformations are applied to test inputs to verify whether the resulting output shows the expected behavior or not [28]. So, by creating a metamorphic relation (MR) that identifies the relation between input and output, correctness of a program is determined.

For example, given a random set of numbers {1, 4, 2, 8, 1} to be sorted in ascending order, the expected output is {1, 1, 2, 4, 8}. Now, an MR for this example can be reversing the inputs, which is {1,8,2,4,1}. If the sorting algorithm is correct, then reversing the inputs should not change the output. However, if the MR is violated, that is the same output is not produced, it implies there is a potential issue with sorting algorithm.

Mutation testing: Test adequacy evaluation techniques are used to find whether or not existing tests can reveal faults. One type of test adequacy evaluation technique is mutation testing [29] where the source is intentionally altered with small changes called mutations, and it is observed whether the test suite can detect these changes and distinguish between correct and faulty program behavior. Using different mutation operators, this testing simulates a variety of errors that the system might experience in real life.

In ML, there are two main approaches to applying mutations [30]. One method involves injecting faults into the training data, then retraining the model with the mutated training data to create mutant model. Another approach is to directly introduce faults into the model without requiring retraining. A test dataset is used to detect faults in those mutated models. For example, a dataset can be mutated by adding a small amount of noise (changing some values). After running a clustering algorithm on the original and mutated dataset, the results are compared to check how the model behaves to the altered data.

Testing Approaches for UL

As discussed earlier UL techniques are utilized in various domains, such as anomaly detection in medical records, fraud detection in financial transactions, genetic research, etc. In these applications, the quality of the model's output can have significant consequences. For example, in the healthcare sector, anomaly detection is used for monitoring patient health data, which includes vital signs, test results, or medical imaging reports. With the use of these data, a patient's condition can be diagnosed, allowing for timely prevention. It is possible to detect credit card fraud, money laundering, and other fraudulent conduct in the banking and insurance industries by identifying suspicious activity from transaction patterns. Ensuring the accuracy of the output generated by the specific UL algorithms is important for these significant tasks. Therefore, it is important to evaluate the performance of the UL models to verify and validate their quality and effectiveness. Potential flaws or inaccuracies in these models can be discovered through testing, and the models can then be fixed to improve the accuracy of the results.

Although UL models are used in numerous applications, there has not been much work on testing them. Among the reviewed papers, three main UL testing approaches have been found, which are symbolic analysis and statistical hypothesis testing, metamorphic testing, and mutation testing. The majority of current research is concerned with various SL testing techniques with few works on RL. Table *2* shows the comparison of these three testing techniques across the three ML methods.

UL techniques are utilized in various domains, such as anomaly detection in medical records, fraud detection in financial transactions, genetic research, etc. In these applications, the quality of the model's output can have significant consequences. For example, in the healthcare sector, anomaly detection is used for monitoring patient health data, which includes vital signs, test results, or medical imaging reports. With the use of these data, a patient's condition can be diagnosed, allowing for timely prevention.

It is possible to detect credit card fraud, money laundering, and other fraudulent conduct in the banking and insurance industries by identifying suspicious activity from transaction patterns. Ensuring the accuracy of the output generated by the specific UL algorithms is important for these significant tasks. Therefore, it is important to evaluate the performance of the UL models to verify and validate their quality and effectiveness. Potential flaws or inaccuracies in these models can be discovered through testing, and the models can then be fixed to improve the accuracy of the results.

Although UL models are used in numerous applications, there has not been much work on testing them. Among the reviewed papers, three main UL testing approaches have been found, which are symbolic analysis and statistical hypothesis testing, metamorphic testing, and mutation testing. The majority of current research is concerned with various SL testing techniques with few works on RL. Table *2* shows the comparison of these three testing techniques across the three ML methods.

Ramanathan et al. [31] has proposed a combination of symbolic analysis and statistical hypothesis testing to identify test cases to reveal bugs in the algorithms. Automated symbolic approaches were used to find test scenarios and they used statistical hypothesis testing to identify the test cases that can differentiate between accurate and inaccurate intelligent systems. Using this combinational approach, they could successfully identify test cases where even if there is slight changes in the inputs, the system provides the right result, and these test cases were used to detect even minor bugs in the KM algorithm. However, they doe not discuss how the sample size affects the possibility of false positives or false negatives, which can impact reliable bug detection.

Murphy et al. [32] worked with both SL and UL applications. When analyzing the metamorphic properties of the SL application, they found six metamorphic properties: additive, multiplicative, permutative, invertive, inclusive, and exclusive. For additive and multiplicative properties, we modify the input data by adding or multiplying a constant. Permutative property means changing the order of the examples, and invertive property takes the opposite of the input. Inclusion of a new element in the input and exclusion of an element means inclusive and exclusive metamorphic property, respectively. Modification made by these properties, however, should not affect the output. That is, the result should stay the same.

They also found that these same six metamorphic features are present in the UL application as well [32]. Using these metamorphic properties, they tested the UL application PAYL. A set of Transmission Control Protocol/Internet Protocol network payloads, or streams of bytes, without any corresponding labels or classification make up PAYL's training data. They found two errors in PAYL through applying the exclusive metamorphic property where incorrect alerts were raised; however, the paper did not explain the reasons behind them.

Xie et al. [9] mainly focused on identifying unusual patterns in clustering systems from the viewpoint of the user. They also created 11 generic MRs that correspond to six distinct clustering system features. These MRs add or remove attributes, rotate, or scale the coordinate system, add outliers, and modify the order of the objects as well as the distinctness and object density of clusters in the dataset. With the help of these MRs, users can validate and test how changes to the input dataset for a particular clustering system affect the clustering outcome. Six clustering systems were evaluated using these 11 MRs. They are KM, XM, EM, AN, FF, and DBSCAN. They also discussed three user-specific MRs based on participant input: transforming geodetic coordinates to cartesian coordinates, feature reduction with PCA, and Z-score normalization.

However, Xie et al. [9] used synthetic 2D data, which does not capture real-world scenarios. They also used 15 users to investigate whether users were able to use their framework using their own requirements. This made the user study sample size small. Also, this study [9] tested only well-formed clusters to ensure compatibility with the six clustering systems.

Rehman and Izurieta [33] suggested 22 MRs that fall into 14 categories to evaluate behavior from the standpoints of developers and users. They are mainly data instance duplication and replacement, attribute addition, feature duplication, removal, order modification, shifting and scaling, addition of new data point between two cluster centroids, data point transformation, and reflection transformation. Two algorithms KM and AN were tested for both verification (from the developer's perspective) and validation (from the user's perspective) purposes using these MRs using a customer segmentation application.

Rehman and Izurieta [34] also worked with DBSCAN algorithm. They tested their proposed 21 MRs on an anomaly detection system, which uses the DBSCAN algorithm to detect the noise. Apart from the MR, duplicating the cluster centroids in [33], all the MRs described in [33] are same as [34].

Yang et al. [35] proposed seven hypothesized metamorphic relations (HMRs), which include translation, order manipulation and flipping of data points, addition of duplicate point and dimension, point relocation towards the cluster center, and x- and y-coordinates swapping. These seven HMRs were used to test the KM algorithm, and two of the seven HMRs were violated by KM. However, the authors mentioned

that these MRs are not evaluated correctly and so named as hypothesized. They also used 2D data that does not reflect real-world. The dataset was also small with 150 instances.

Lu et al. [30] proposed MTUL, which has two types of mutation operators, data level and algorithm level, for testing clustering system and generative adversarial network (GAN). In the case of data level mutation operators, for clustering system, using point, position, and dimensional mutation operators, it is ensured that the dataset size remains constant both before and after the mutation. Again, using outlier, noise point, and density mutation operators, the input datasets are modified by adding or deleting data in clustering systems. For GAN, data level mutation operators consist of noise injection, disturb preprocessing, randomness reduction, and hyperparameter operators. Similarity change and cluster mutation operators are two of the algorithm level mutation operators for clustering systems; confusing discriminator, change sampling, and mishandling gradient operator are the algorithm level operators for GAN.

Lu et al. [30] also proposed two frameworks, MUAE and MTGAN. MUAE is a combination of MTUL and autoencoder. Autoencoder is an unsupervised artificial neural network where the input is first encoded into lower dimensional latent representation and then decoded back to the latent representation of the original input. The purpose of this MUAE is to identify possible adversarial attacks inside a given dataset and to provide adversarial examples. Based on low-dimensional features, MUAE produced high-dimensional adversarial examples that could lead to instability in clustering outcomes. With the MNIST dataset embedded, MTGAN is proposed as a tool for mutation testing for GANs using the MTUL. To do mutation testing, users can incorporate their own GAN model and utilize alternative datasets in addition to MNIST.

However, the study [30] did not show calculation of mutation scores on clustering systems, making it difficult to verify how well the mutation score quantifies clustering stability. The authors mention high or low mutation score but does not define what is considered high or low score. Also, the results from IRIS and MNIST dataset (real world datasets) before and after mutations on clustering systems were not shown. They presented results from Apple dataset which is a synthetic dataset.

Testing Approaches for SL

Unlike UL systems, research on SL has been extensive. A hybrid approach named concolic testing that combines the execution of concrete inputs of a deep neural network (DNN) with a symbolic analysis technique to design the DeepConcolic tool to test and debug DNNs was introduced by Sun et al. [36]. Given a specific coverage requirement, the concolic engine generates test cases. To meet test conditions, combination of concrete inputs and a symbolic analysis technique is executed, and users can also choose between two symbolic techniques - linear programming approach and global optimization approach. There is also another engine for test case generation, GA search engine, which includes GA search algorithm. The tool can be used to analyze different behaviors of a DNN by testing its components.

Ding et al. [37] proposed three levels of MRs - system level, dataset level, and data item level to validate a deep learning framework. The framework includes deep learning architecture AlexNet, deep learning execution environment Caffe, and a dataset of biological cell images. In total, 11 MRs were used to perform metamorphic testing. One MR was developed at the system level, and it was built on the relationship between Support Vector Machine (SVM) and deep learning's classification accuracy. According to the MR, for the classification accuracy of the diffraction of biological cell images, deep learning classifier should perform better than the corresponding SVM classifier. On the dataset level, some of the MRs were used to add 10% new images into the training, validation, and test datasets. Overall, the main purpose for the MRs in this level was to show that modifications should not have an impact on the classification accuracy when they are applied. Finally at the data item level, for example, the MRs proposed that the classification accuracy of the classifier trained on the datasets cropped from original images using different stride distances or different moving directions should be nearly the same.

Similarly, Xie et al. [38] suggested some MRs that classifications algorithms may display, such as permutating the class labels, adding attributes, or removing sample data. They used these MRs for testing K-Nearest Neighbors and Naïve Bayes classifiers.

Ma et al. [39] presented a DeepMutation framework for mutation testing and evaluated this framework on two datasets - MNIST and CIFAR-10 - and three DL models to analyze the dataset quality. They mainly focused on DL systems for classification problems. The framework consists of eight source-level mutation operators and eight model-level mutation operators. In source-level, there are two kinds - data mutation operators, which duplicate or add noise to the training data, falsify data findings, etc., and program mutation operators, which add or remove layers, for example. The model-level mutation operators mutate the structure and parameters of deep learning models using operators like Neuron Activation Inverse, Layer Addition, etc. They also introduced two metrics (Mutation Score and Average Error Rate) that are deep learning specific to test the quality of the dataset.

Based on real deep learning faults, Humbatova et al. [40] proposed 35 mutation operators (24 of them were implemented). Their proposed mutation operators are applied on the system before the start of the training process, and they consist of training data operators, hyperparameters operators, activation function operators, regularization operators, weights operators, loss function operators, optimization operators, and validation operators. Based on the selected operator, this technique modifies labels or removes training data; alters batch size or number of epochs; and adds, removes, or changes activation function, dropout rate parameter, or the loss function. To evaluate these operators, they used five different deep learning systems, which cover both classification and regression problems.

Testing Approaches for RL

In the case of testing RL systems, few papers were found. Among them, Lu et al. [41] focused on creating mutation operators to introduce potential faults into three elements of RL system - states, rewards, and actions. They proposed element-level and agent-level mutation operators. For example, element-level operators modify the rewards and create wrong state-action connection, and the agent-level operators mutate the policy of the agent by removing neurons in the input or output layer, etc.

In another work on mutation testing for RL by Tambon et al. [42], they proposed first-order mutations (FOM), which are environment, agent, and policy mutation operators, to introduce faults such as adding noise to true reward, reversing the order of received reward wrongly, changing default activation function, etc. Additionally, by generating higher order mutations based on the combinations of FOM, they showed that higher order mutations are more difficult to kill than the FOMs that made them up.

Table *2* compares the testing techniques for UL, SL and RL systems.

Topic	Type of Testing	Criteria	SL	UL	RL
Test Input Generation	Symbolic	Design	Combination of the execution of concrete inputs with a symbolic analysis technique [36]	Combination of symbolic analysis and statistical hypothesis technique [31]	
		Implementation	Analyze different behaviors of a DNN by testing its components	Detect minor bugs in the KM algorithm	
Test Oracle	Metamorphic	Metamorphic Properties	Additive, multiplicative, permutative, invertive, inclusive, and exclusive [32]	Additive, multiplicative, permutative, invertive, inclusive, and exclusive [32]	
		Example of MRs	Adding 10% of new images into the training, validation, and test dataset [37]	Add or removing attributes [9], duplicating and replacing data instance [33]	
		Validated Algorithms	Classification algorithms [37,38]	Clustering algorithms	
Test Adequacy	Mutation	Example of Mutation Operators	Duplicate training data, Add noise to training data, Falsify data findings [39], Modify labels, Remove training data [40]	Add or delete input data, Reduce randomness, Inject noise, Change gradient of model [30]	Modify the rewards, Create wrong state-action connection [41], Add noise to true reward [42]
		Implementation	Deep learning systems that cover both classification and regression problems	Testing clustering system and GAN	Q-Learning, Deep Q-Network, Advantage Actor-Critics (A2C), and Proximal Policy Optimization

TABLE 2: Comparison of testing techniques for SL, UL and RL systems

DNN, Deep Neural Networks; GAN, Generative Adversarial Network; KM, K-Means; MRs, Metamorphic Relations; RL, Reinforcement Learning; SL, Supervised Learning; UL, Unsupervised Learning

Paper Collection Methodology

The primary objective of this paper is to explore the testing of UL systems - a relatively underexplored

area within ML testing. The paper collection process was therefore designed to prioritize works on UL testing, while also incorporating related literature on UL algorithms and applications to provide the necessary background and context. We also explored literature on SL and RL testing methods mainly to understand the common testing techniques in these areas as applicable to UL systems.

Our paper collection process began after reviewing the work by Zhang et al. [26], who conducted a survey on ML testing. Their study highlighted the limited research on UL testing, which inspired us to focus on this specific area. We first examined the papers they reviewed, mainly those on UL testing, along with some paper on SL and RL. By doing so, we identified three existing testing methods, symbolic analysis with statistical hypothesis testing, metamorphic testing, and mutation testing. Next, we searched for recent works on UL testing across multiple academic databases, including Google Scholar, ACM Digital Library, IEEE Xplore, and Scopus. The following keywords were used: UL testing, metamorphic testing (unsupervised/reinforcement) learning, and mutation testing (unsupervised/reinforcement) learning.

Since Zhang et al. [26] already reviewed most of the SL testing papers, we selectively collected some papers on SL from their study. At the same time, we searched for new papers on UL testing and a few on RL testing. Our search focused on publications from 2018 to the present to ensure we captured recent advancements. For each keyword, the first 100 results were retrieved from each database to ensure consistency. After that, title and abstract screening followed. We applied title and abstract screening to exclude duplicates, papers on general software testing, works where SL/UL/RL algorithms were used to test other systems (e.g., quantum computation, blockchain, DevOps), and papers that addressed testing challenges but not specific testing approaches. In this way, we gathered relevant papers on testing for our study.

In the included papers, we observed a concentration on SL or deep learning testing than UL and RL. The testing methods explored include metamorphic testing, mutation testing, etc., with some work addressing the framework, challenges, and issues related to the testing approaches. In contrast, excluded papers mainly discuss software testing, with a small mention of ML. Some papers use SL, UL, or RL algorithms to automate the tests or testing bugs in their system or software rather than focusing on testing SL, UL, or RL systems. Table *3* presents the selected testing papers along with the reasons for their inclusion. We have created a GitHub page to include the papers used in this study. This page will be periodically updated as new papers on UL, particularly those related to testing, are published: https://github.com/ProtyashaRoy/Unsupervised-Learning-Techniques-Applications-and-Testing

	Paper	Reason for Selection
SL	Sun et al. [36]	DeepConcolic demonstrates a hybrid testing framework to automatically generate test cases that could inspire UL testing research.
	Ding et al. [37]	This discusses a multi-level metamorphic testing framework; provides UL an opportunity to expand UL testing frameworks.
	Xie et al. [38]	Suggested MRs can help develop metamorphic testing for UL.
	Ma et al. [39]	Their mutation operators and quantitative metrics can inspire the development of UL mutation testing framework and evaluation metrics.
	Humbatova et al. [40]	Their broad taxonomy of mutation operators based on real faults can guide UL research in developing richer, fault-based mutation operators.
RL	Lu et al. [41]	Illustrates how mutation testing can be tailored to the internal mechanics of a learning system, inspires the development of UL-specific operators.
	Tambon et al. [42]	It can guide UL to explore compound operators that simulate more complex, realistic faults.
UL	Ramanathan et al. [31]	It demonstrates a hybrid method to uncover subtle bugs in clustering algorithms like K-means.
	Xie et al. [9]	It introduces both generic and user-driven MRs and shows their applicability across multiple clustering algorithms.
	Murphy et al. [32]	It establishes that MRs can transfer across ML paradigms (SL and UL). This broadens the generality of metamorphic testing and based on these MRs can be expanded later.
	Rehman and Izurieta [33]	This work provided a large and structured set of MRs for clustering algorithms to address both verification and validation.
	Rehman and Izurieta [34]	They adapted their previously defined MRs to test the DBSCAN algorithm in the context of anomaly detection.
	Yang et al. [35]	It highlights the potential and the challenges of MR-based UL testing.
	Lu et al. [30]	It introduces mutation testing for UL systems, establishing both data- and algorithm-level operators.

TABLE 3: Selected testing papers and their reasons for inclusion

DBSCAN, Density-Based Spatial Clustering of Applications with Noise; MRs, Metamorphic Relations; RL, Reinforcement Learning; SL, Supervised Learning; UL, Unsupervised Learning

Review

Analysis of the results

Comparison of SL, UL, and RL Testing Approaches

Table *2* shows a comparison of SL, UL, and RL systems in terms of the three types of testing. Symbolic analysis is used as a test input generation technique. The comparison in symbolic testing demonstrates how symbolic analysis has been combined with other techniques and applied to SL and UL systems. The table also shows metamorphic testing, which serves as a test oracle generation technique. It lists the metamorphic properties and provides examples of MRs used in SL and UL systems, along with the SL and UL algorithms that have been validated through metamorphic testing. In addition, the table presents examples of mutation operators and indicates the SL, UL, and RL systems in which these operators were applied for mutation testing.

In symbolic execution as a test input generation, for SL, a combination of concrete execution and symbolic execution is used. In UL, apart from symbolic execution, statistical hypothesis testing was used.

In a study by Murphy et al. [32], six metamorphic properties (additive, multiplicative, permutative, invertive, inclusive, and exclusive) that are present in both SL and UL were mentioned. Most of these properties can be seen in the mentioned papers on metamorphic testing here. From Table *4*, we can determine which MRs from SL and UL correspond to specific metamorphic properties.

A study by Xie et al. [9] proposes several MRs, such as permuting the order of objects and adding

informative attributes; these correspond to the permutative and inclusive metamorphic properties, respectively. Likewise, Rehman and Izurieta [33] define MRs such as removing instances from clusters and changing the order of features, which correspond to the exclusive and permutative metamorphic properties, respectively. For testing clustering systems, we can observe some common MRs across studies, for example, manipulating the order of sample objects (as seen in [9,33-35]) and duplicating instances (as seen in [33] and [35]). Some MRs can be seen to be common in both SL and UL, such as addition of informative and uninformative attributes [33,37]. Having class labels, SL has some MRs involving it - for example, permutation of class labels or removal of classes [38], making the MRs not the same as UL. However, the MRs in both UL and SL follow the six metamorphic properties.

It should be noted that some MRs cannot be perfectly assigned to the six mentioned metamorphic properties, such as replacement of instances [33], random shuffling [33], and cases where a deep learning classifier outperforms an SVM classifier in classifying diffraction images [37]. As stated by Murphy et al. [32], additional metamorphic properties may exist beyond the six identified ones, which aligns with the findings of our study.

While certain concepts, such as the metamorphic properties from SL described by Murphy et al. [32], have been applied to evaluate UL in metamorphic testing, it was not possible to do the same for mutation testing. The fact that UL has the ability to identify hidden patterns from unlabeled data while SL depends on dataset labels was one of the main issues. Since SL systems use labels and UL systems do not, mutation operators involving labels, such as injecting faults through mislabeling [39], are not applicable to UL. Furthermore, because the functions of SL and UL algorithms are also different, the operators that mutate the elements in model level or algorithm level are also different. For instance, using the gaussian distribution to mutate a given weight value [39] or changing activation function [40] cannot be applied to clustering algorithms.

Likewise, RL systems, which are based on agents and their interaction with environment through a feedback system, also differ from UL. As such, mutation operators proposed by Lu et al. [41], like Reward Reduction/Increase operator, cannot be applied to UL systems. However, mutation operators involved in noise injection and hyperparameter changes are found to be similar across SL and UL, despite possible differences in how they function.

So, based on the analysis, it can be said that with some structural changes, testing techniques such as symbolic execution, metamorphic testing, and mutation testing, used in SL and RL, can also be applied to UL.

Properties	SL	UL
Additive	Duplicating 10% of images, one existing category of data, all samples	Duplication of single/multiple instance or cluster centroids
Multiplicative	Datasets cropped and pooled from the original images, datasets pooled with different functions, applying affine transformation function	Shrinking one or more clusters towards their centroids, rotating/scaling the coordinate system, shifting/scaling features with constant k, translating 2D points along a line parallel to the x- or y-axis, moving an existing point towards the cluster center
Permutative	Data sets cropped using different moving directions, permutating class labels/attributes	Changing the object/features order, reversing the order of datapoints/rows
Invertive		Data mirroring, Data reflection transformation, swapping the x- and y-coordinates of each point, flipping the data points along one (x- or y-) axis
Inclusive	Adding 10% of new images into training/validation/test dataset, adding informative/uninformative attributes, Feeding the test case's predicted output back into the training dataset, adding classes by duplicating /re-labeling samples	Adding sample objects around cluster centers/boundary, adding informative/uninformative attributes, inserting outliers, duplicating features, adding new instance with informative/uninformative attributes, adding a duplicate point
Exclusive	Removing one category of the data, datasets cropped with varying stride distances, Removing classes/samples	Removing redundant attributes, data normalization, removing instance(s) from one/different cluster(s)

TABLE 4: Common metamorphic properties shared between metamorphic relations of SL and UL

SL, Supervised Learning; UL, Unsupervised Learning

Challenges of the testing approaches

Symbolic analysis is challenging due to the large number of data points and the computational cost of distance calculations. Ramanathan et al. [31] proposed a distance-theoretic abstraction to avoid the expensive computations by representing sets of points canonically, avoiding redundant path analysis under translations or rotations. However, this approach does not fully address high-dimensional feature spaces, relies on approximations, and may overlook certain errors due to limited precision in calculations. Statistical hypothesis testing assumes, within a certain confidence level, that some errors are acceptable due to the stochastic nature of ML, where outputs may vary even for the same sample size. This variability can affect false positives and false negatives, thereby impacting reliable bug detection.

Metamorphic testing was introduced to mitigate, though not fully resolve, the oracle problem [43]. Rather than verifying the correctness of individual outputs against a perfect oracle, it checks whether or not the system under test maintains expected relationships when inputs are systematically altered based on predefined MRs. So, metamorphic testing can detect the inconsistencies, but it is unable to determine if an incorrect output is indeed wrong or rather an unexpected but valid result. Also, the effectiveness of metamorphic testing relies on the quality of the identified MRs and the tests derived from them. If an MR is violated, it indicates potential defects in the system under test; however, satisfying an MR does not mean the system is free from defects.

Designing effective MRs, which capture all possible fault scenarios, is a challenge as well. The generic MRs from the reviewed studies do not cover all possible properties of clustering systems, requiring more relevant MRs. However, to verify a single MR, the ML model needs to be trained twice - one for the source inputs and another for the follow-up inputs, which leads to high execution costs. So, a large number of MRs further increases this challenge. Also, while studies claim their proposed MRs to be broadly applicable, proper verification and validation are needed to confirm their generalizability across different algorithms.

Many clustering algorithms introduce randomness, such as selecting initial centroids. This can lead to variations in results across different runs. Also, clustering algorithms are sensitive to parameter configurations. So, Xie et al. [9] do not consider parameter variations, instead fixing them during experiments. Assessing the impact of parameter changes on clustering validity remains a challenge.

Based on our literature review, metamorphic testing has been explored extensively in SL, with limited research on UL and none on RL. So, defining effective MRs for UL remains difficult, especially when borrowing from RL. And although the proposed MRs of SL and UL share common properties, most of the MR do not match as seen from Table *4*, making it challenging to derive MRs from SL.

Mutation testing requires retraining the model with the mutated training data. So, one of the common challenges in mutation testing across the three ML systems is the high computational cost because of the retraining. Again, the effectiveness of mutation testing depends on the quality of the test dataset. A poor-quality test dataset can result in a high mutation score, which can be misleading as the model may not actually be robust. Ensuring that the test dataset representing real-world faults is important for assessing whether the model can generalize well. Moreover, a high or low mutation score does not necessarily indicate the quality of the test suite or its effectiveness in finding faults. Lu et al. [30] use IRIS dataset who has predefined number of clusters (three). However, in real-world applications, the number of clusters is not always known. So, when cluster mutation operator modified the number of clusters from 3 to 4, it resulted in a high mutation score [30]. Even the original dataset had a very high mutation score as well. This implies that algorithmic sensitivity, rather than presence of actual faults in data, may cause mutation scores to be inflated. Further, while a mutated dataset with a higher mutation score is more likely to reveal potential vulnerabilities, there is no specific threshold to distinguish between a high or low mutation score.

Severe mutations can degrade image quality during mutation testing on GANs, such as distortion, blurring, or other visual imperfections. This will result in a higher mutation score, and the probability of images being flagged as fake by the discriminator of the original GAN will increase. Also, a higher mutation score will indicate less adaptability and robustness, making the GAN less reliable in generating high-quality images.

As certain classes contribute more to mutation scores than others, a test dataset yielding a high mutation score on one model may perform poorly on another. As a result, mutation testing results may not generalize across applications. For UL, the existing study [30] does not address performing mutation testing on other clustering algorithms apart from KM. Although, they proposed mutation operators for all clustering algorithms, it is still necessary to verify if the same operators can be applied to other clustering systems as well. To the best of our knowledge, the existing study on mutation testing for UL [30] proposed many mutation operators. However, it is challenging to design mutation operators that reflect real-world faults. Consequently, there is no up-to-date list that covers all possible instances of faults.

Equivalent mutants analysis [44] is also necessary for mutation testing, as these mutants differ syntactically but behave identically to the original program. Not even a perfect test suite can kill these mutants, so they should be excluded from mutation analysis. However, identifying equivalent mutants is difficult, making it a major challenge.

As UL systems use unlabeled data, the learning process is random and cannot be predicted. Compared to SL, mutation testing in UL and RL is a relatively new research area, and so the lack of the established techniques presents another challenge.

Discussion

The discussion in this section synthesizes results derived from the studies outlined based on the research questions we outlined earlier.

RQ1: Which domains have successfully applied UL techniques?

There are several main applications where UL has been effectively implemented. In anomaly detection, DBSCAN was used to detect anomalies in heart patients [13]. OneClassSVM, local outlier factor, isolation forest, and elliptic envelope were used to detect anomalies in the shuttle and satellite datasets [14]. For recommendation systems, Cintia Ganesha Putri et al. [15] use KM algorithm, birch algorithm, mean-shift algorithm, etc., and found Birch as the best performing. On the other hand, Addagarla and Amalanathan [16] utilized PCA-SVD with K-mean++ for effective results.

KM was used by both Arthur and Vassilvitskii [17] and Shen [18] for customer segmentation. Besides KM, Arthur and Vassilvitskii [17] also employed Apriori algorithm. For insurance fraud detection, autoencoder and VAE were applied by Gomes et al. [20]. Trimmed KM was used by Monamo et al. [22] to detect fraud in the Bitcoin network. Also, autoencoder, OneClassSVM, and Mahalanobis outlier detection were used by Rezapour [24] for credit card fraud detection. The AN algorithm was used by Lopez et al. [25] to cluster patients based on their genomic similarity.

This addresses our first research question, as UL techniques have been effectively applied across various domains such as anomaly detection, recommender systems, customer segmentation, fraud detection, and genetic research.

RQ2: What are the strengths and limitations of UL algorithms?

We discussed some of the key UL algorithms, mainly clustering algorithms (KM, XM, DBSCAN, EM, AN, and FF), association (Apriori algorithm), dimensionality reduction (PCA), and deep UL algorithm (autoencoder). Table *1* also lists the strengths and weaknesses of each algorithm, which directly addresses our second research question.

Centroid-based clustering algorithms, particularly KM, are easy to implement but are sensitive to initial conditions and noise, just like XM. However, XM outperforms KM by providing partial solutions for local optima problem and performing better on well-separated clusters. Density-based clustering algorithm or DBSCAN performs well with similar-density clusters, but struggles with high dimensional data, and varying cluster densities. Distribution-based clustering algorithm or EM performs well with missing variables, but it needs good initial parameter estimates, or it may fail to converge and can get stuck in local minima. Hierarchical-based clustering algorithms, especially AN, can handle clusters with different shapes and sizes, but its limitation is that once merged, data points cannot be separated, even if later they are found to be dissimilar. FF can also create well-separated clusters, but it is sensitive to initialization conditions. Although Apriori algorithm is simple and adaptable, they are computationally expensive and inefficient with large datasets. PCA can reduce overfitting but can cause information loss and sometimes fail to identify significant features in the dataset. Autoencoder can reduce dimensionality and noise, but it is sensitive to hyperparameters, overfits with small datasets, and has difficulty in representing high-level features in the data.

RQ3: What testing methodologies exist for UL?

We found three main testing methods for UL: symbolic analysis with statistical hypothesis, metamorphic testing, and mutation testing.

For effective test input generation, Ramanathan et al. [31] used symbolic analysis with statistical hypothesis testing. By using symbolic decision procedures, they searched for test cases where minor input perturbations, such as bit-flips, lead to significant differences in clustering outcomes. And statistical hypothesis testing was then used to determine, with a certain confidence level, whether a test case reliably differentiates between correct and erroneous system behavior.

This approach proved to be more efficient than randomized testing, which failed to expose bit-flip errors even after observing 100 samples, whereas symbolic analysis successfully distinguished such errors within the same sample limit. Using the metric abstraction method, the test case generation was limited to 100 samples. Even with this constraint, the method consistently differentiated between correct and erroneous clustering implementations, with at least a 70% probability.

Additionally, the distance-theoretic abstraction methodology helps to analyze the relationship between nonlinear data manipulations and clustering behaviors. This is important for UL models like KM, where small input variations can cause changes in cluster assignments. By combining symbolic decision procedures with statistical hypothesis testing, the approach ensures that errors detected in clustering models are statistically significant issues.

Metamorphic testing helps address the oracle problem, enabling users to validate clustering systems even without prior data knowledge. Since clustering models learn from vast data, assessing their correctness is challenging. Xie et al. [9] defined 11 generic MRs that users can adopt or extend to assess a clustering system's suitability for specific applications. By evaluating compliance with these MRs, users can determine a system's robustness.

Xie et al. [9] assigned two metrics to each clustering system to assess the degree of violations when the 11 MRs were implemented. The first is the Violation Rate (VR), which is calculated as the proportion of trials that have been violated to all trials. And the second is Reclustering Percentage (RP), which calculates the variation between source output and the related follow-up output. The results show that KM violated nine MRs. Both FF and DBSCAN violated seven MRs. XM and EM violated five and four MRs, respectively. AN violated three MRs and had the best performance. In summary, KM and FF performed the worst according to both VR and RP, whereas EM and AN remained mostly stable.

Rehman and Izurieta [33,34] defined in total 22 MRs for both the verification and validation purposes. They also use two metrics, number of violated MRs and VR, which represents the percentage of instances where programs exhibit inconsistent behavior.

Violations of MRs, which represent necessary properties of the algorithms, indicate the presence of defects, and are used for verification. On the other hand, violations of MRs that are not necessary properties do not imply implementation flaws but offer insights to for validation. For example, adding a duplicate instance to any of the clusters in a study by Rehman and Izurieta [33] is not used for verification because its violation does not indicate a defect in the KM implementation. The change in cluster centroids due to the addition of duplicate instances is an expected behavior based on how KM calculates centroids. So, the violation in this MR does not reveal a bug in the implementation. However, this MR was used for validation because its violation provides useful insights into how the clustering results change in response to data transformations.

Ten of the 22 MRs are violated for the KM and AN combined [33]. However, KM is considered more stable than AN, as AN is seen to be more sensitive to slight changes, and a slight change results in a larger violation rate. The results indicate that 11 out of 14 of the MRs related to validation were violated by DBSCAN, with the highest violation rate of 94.5% observed for the MR involving feature scaling [34]. However, the MRs targeting the verification aspect have not been violated.

Murphy et al. [32] found two defects in PAYL's behavior through applying the exclusive metamorphic property. When removing 274-byte payloads from the training data, an incorrect anomaly alert was raised. Also, for 1448-byte payloads, PAYL raised both anomaly alerts and "length-never-seen-before" alerts, which are not supposed to occur.

For KM, Rehman and Izurieta [33] used equation 1 for calculating new centroids

$$c_i^{(t+1)} = \frac{1}{\left|c_i^{(t)}\right|} \sum_{j=1}^{n} x_j \qquad (1)$$

here,

$c_i^{(t+1)} = i^{\text{th}}$ new centroid

$x_j = j^{\text{th}}$ instance belonging to cluster c_i

For AN, Rehman and Izurieta's [33] equation 2 defines the average linkage method, which is used to merge two similar clusters

$$d(c_i, c_j) = \frac{1}{|c_i|\,|c_j|} \sum_{x_s \in c_i} \sum_{x_s \in c_j} d(x_r, x_s) \quad (2)$$

here,

$d(c_i,c_j)$ = distance between clusters c_i and c_j

x_r = r^{th} data point belonging to cluster c_i

x_s = s^{th} data point belonging to cluster c_j

For KM, to verify the proposed MRs, the authors [33] also gave three other criteria - clusters, centroid, and nearest point to each centroid. Other papers or other algorithms [34,35] verified the effectiveness of their proposed MRs by checking if the cluster assignments are same or not.

Table 5 presents the MRs violated by UL algorithms in metamorphic testing, along with specific violations and the reasons behind them. It shows that, for all the algorithms, rotating the coordinate system causes reassignment of boundary data points and the reason for this is min-max normalization. Rotating the coordinate system also causes other types of violations as well, such as two source clusters merging, with another source cluster splitting into two smaller clusters, changing the number of clusters in the follow-up dataset, etc. Other violations revealing MRs that cause multiple types of violation are changing object order, modifying object density, adding instance with informative/uninformative attributes, etc.

KM shows the most violations as this was tested most across papers. XM displayed similar weaknesses to KM, due to the random initialization of starting centroids. XM offers a partial remedy for the local optimum problem. However, according to Xie et al. [9], XM only outperforms KM when the original dataset is highly separated. So, MRs such as manipulating cluster distinctness and attributes did not show any violation. On the other hand, when data groups were well clustered but had a lower degree of separation, both XM and KM had similar clustering results.

For EM, violations caused by adding sample objects around cluster centers or near a cluster's boundary, rotating the coordinate system had low RP values, which means that only a small number of data samples were affected. EM initializes estimators by running KM [9]. So, the clustering result produced by EM partially depends on KM. As KM is prone to suboptimal solution due to its sensitivity to initialization and local optima, these effects carry over to EM as well. However, Xie et al. [9] found that EM is less sensitive to local optima than KM.

Xie et al. [9] observed that compared to FF, AN is more robust to data transformation with only boundary points being affected sometimes. However, according to Rehman and Izurieta [33], KM can be considered more stable than AN, as AN is seen to be more sensitive to slight changes, and a slight change results in a larger violation rate.

FF is highly sensitive to data transformation including object order changes, data mirroring, attribute manipulation, and outlier insertion. While FF effectively detects and assigns outliers to distinct clusters, data transformations may still cause non-outlier data points to be reassigned. Additionally, even when data samples are well separated, FF sometimes fails to produce clear and accurate clusters.

DBSCAN is fairly robust to the data transformation - changing the object order - for well-separated data samples. However, DBSCAN showed violations for MRs like data mirroring, adding sample objects around cluster centers or near a cluster's boundary, etc. Rehman and Izurieta [34] found that if a border point of a cluster is duplicated, replaced, or a new instance with uninformative attributes is added in the follow-up input, the border point may transform into a core point. This newly formed core point can further expand the cluster, potentially causing border points from different clusters to be reassigned to the same cluster (the cluster where the newly formed core point resides), ultimately altering the final clustering outcome. Duplication of all attributes doubles the distance between data points, leading to different core, border, and noisy points. Scaling features increased distances, sometimes moving most points into a single noisy/anomalous cluster. Removing instances could shift core points into border or noisy points, further changing clustering outcomes.

From the reviewed papers, we can see that most of the testing methods (symbolic analysis and statistical hypothesis testing, and metamorphic testing) were implemented on the KM clustering algorithm. Besides KM, Xie et al. [9] also applied metamorphic testing on other clustering algorithms collected from the open-source software Weka and validated them from the user's perspective to find faults within the algorithms. Additionally, Yang et al. [35] take KM algorithm from Weka tool to validate the metamorphic testing for user's expectation. However, Rehman and Izurieta [33] collected KM and AN clustering

algorithm from Scikit-learn and implemented metamorphic testing from both the developer's and user's perspective. DBSCAN algorithm was also collected from Scikit-learn in a study by Rehman and Izurieta [34].

Violations	Violated Algorithms	Violation Revealing MRs	Reason for Violation
Reassignment of boundary data points	KM, XM, EM, AN, FF, DBSCAN	Rotating the coordinate system	Min-max normalization
	KM, XM, EM, AN, FF	Modifying object density	Random initialization of starting centroids
			EM - Deviations in parameter estimation
			AN - Deviations in distance computation
	KM, XM, FF, DBSCAN	Changing object order	Random initialization of starting centroids
			DBSCAN - Chronological sequence of detecting clusters
	KM, XM, FF	Inserting outliers	Random initialization of starting centroids
			FF - Deviation in distance computation
	FF	Data mirroring, adding informative attributes	Changes in the traversal sequence
Two source clusters merge, another source cluster splits into two smaller clusters	KM, XM, FF	Rotating the coordinate system	Tendency to converge to local optima
			FF - Min-max normalization
	KM, XM	Changing object order, modifying object density, inserting outliers	Tendency to converge to local optima
	KM	Altering cluster distinctness, manipulating attributes	Tendency to converge to local optima
Reclustering occurs for objects regarded as noise	DBSCAN	Inserting outliers	Decrease in noisy objects that is increase in core objects
		Adding sample objects around cluster centers	
		Data mirroring	Points labeled as noisy become density-reachable
		Adding sample objects near a cluster's boundary	Increase in the number of density-reachable points
		Removing redundant attributes	
One or more source clusters were split into smaller follow-up clusters	EM, AN	Rotating the coordinate system	EM - Suboptimal initialization of KM
			AN - Min-max normalization
Changes the number of clusters in the follow-up dataset	DBSCAN	Rotating the coordinate system	Min-max normalization
	AN	Duplicating single or multiple instances, removing instance(s) from one or different clusters, replacing single, multiple or all instances, adding instance with informative or uninformative attributes	Average distance between clusters (Equation 2) differs

Different cluster assignments	DBSCAN	Duplicating single or multiple instances, replacing single, multiple or all instances, adding instance with uninformative attributes	Border point may transform into a core point
		Removing instance(s) from one or different clusters	A core point may become a border point, or a border point may turn into a noisy point
		Duplicating features, scaling of data	Increases the distance between data points
	KM	Duplicating single instance	Recalculation of Euclidean distances when re-locating the new cluster centroids
Different cluster assignments and centroid	KM	Duplicating single/multiple instances, removing instance from one or different cluster, replacing all instances, adding new instance with informative attributes	The centroid, calculated using Equation 1, differs
Different centroids and the nearest point to centroid(s)	KM	Adding an informative attribute/a new instance with uninformative attributes	The centroid, calculated using Equation 1, differs
Different centroids	KM	Replacing single instance	The centroid, calculated using Equation 1, differs
Different cluster assignments, centroids, and nearest point to centroid(s)	KM	Removing multiple instances from a single cluster	The centroid, calculated using Equation 1, differs

TABLE 5: Types of violations, their causes, corresponding MRs, and affected algorithms for UL in metamorphic testing

AN, Agglomerative Nesting; DBSCAN, Density-Based Spatial Clustering of Applications with Noise; EM, Expectation-Maximization; FF, Farthest-First Traversal; KM, K-Means; UL, Unsupervised Learning; XM, X-Means

Lu et al. [30] worked on mutation testing and proposed data-level and algorithm-level mutation operators, which simulate faults in UL. Their framework for data-level mutation testing improves test suites by creating and filtering datasets that significantly impact clustering, while also detecting test data that cause abnormal behavior in GANs. Additionally, the algorithm-level mutation testing framework evaluates UL system resilience against structural and parameter mutations. This involves modifying similarity criteria in clustering and structural parameters in GANs and identifying weaknesses and vulnerabilities by applying the mutations.

They [30] also provided mutation scores for quantitative evaluation. The mutation score increases when the dataset experiences minimal perturbation, yet the clustering results are significantly affected. They gave an example of an effective mutation operator - position mutation operator, which caused huge variations in clustering results despite the mutated dataset appearing indistinguishable from the original. Mutated datasets expose potential system instability, with higher mutation scores indicating greater vulnerability detection. Targeted adjustments, such as modifying the initial cluster centroids, can help mitigate the issues revealed by the mutated dataset corresponding to the mutation point operator.

Their framework [30] can provide quantitative analysis of potential GAN errors, where mutation score positively correlates with the degree of potential error. The combination of MTUL and autoencoder - MUAE - helps detect adversarial examples and approximate class boundaries, improving system robustness.

Metrics for mutation testing used in a study by Lu et al. [30]:

$$X = \text{original dataset}$$

$$X' = \text{mutated dataset}$$

$$m = \text{mutated data from the original dataset } X$$

$$T = \text{original UL system}$$

T' = mutated UL system

killed = number of data whose clustering results change against UL system T

n = size of original dataset

n' = size of mutated dataset

$p_1, \ldots, p_k$ = parameters in T

$p'_1, \ldots, p'_k$ = parameters in T'

D = discriminator of original GAN

S = image sets generated by the original generator

S' = image sets generated by the mutated generator

f = proportions of images labeled as fake by D in S

f' = proportions of images labeled as fake by D in S'

M = changes in characteristics between T and T'

c = groups of changed parameters

Then, when using point, position, and dimensional mutation operators, the mutation score is [30]

$$\text{MutationScore}_{un}(X, X', T) = \frac{killed}{m} \tag{3}$$

When there is outlier, noise point, and density mutation operators, the mutation score is [30]

$$\text{MutationScore}_{c}(X, X', T) = \begin{cases} \text{killed}/(n' - n), & n' > n \\ \text{killed}/(n - n'), & n' < n \end{cases} \tag{4}$$

For algorithm-level mutation operators for clustering system, the mutation score is [30]

$$\text{MutationScore}_{p}(X, T, T') = \frac{M}{\sum_{i=1}^{c} \sum_{j=k_{i-1}+1}^{k_i} \log_{a_i} (max(p_j/p'_j, p'_j/p_j))} \tag{5}$$

For GAN, the mutation score is [30]

$$\text{MutationScore}_{G}(S, S', D) = \frac{|f - f'|}{f} \tag{6}$$

Conclusions

This paper analyzed the current state of research in UL, focusing on its key techniques along with their strengths and limitations. It also highlighted some of the main application domains like anomaly detection, recommendation systems, genetic research, etc., where UL has been effectively utilized. While UL has shown strong potential in uncovering hidden structures in data, challenges remain in algorithm sensitivity, scalability, and the absence of ground truth for evaluation. The paper also reviewed three testing approaches - symbolic analysis and statistical hypothesis testing, metamorphic testing, and mutation testing - which have been effective mainly in clustering systems. The paper also highlighted the common SL and RL testing methodologies as possible candidates for application or adoption to UL system testing. Our findings suggest that UL has already been applied in diverse real-world applications, but reliable outcomes depend heavily on algorithm choice and evaluation. Current UL testing methods remain limited in scope and should be applied to other UL algorithms such as Apriori, PCA, SVD, etc. Other key limitation inferred from our review is that most testing frameworks have only been validated in controlled settings, which may reduce their generalizability. Further research is needed, which will focus on developing broader validation strategies and designing new mutation operators and metamorphic relations that can identify every possible error occurrence in real-life high-dimensional data samples.

For future research, a framework for UL testing can be developed where users provide datasets and receive suggestions on appropriate testing methods, while still having the flexibility to select their own methods, algorithms, and evaluation criteria. Research should also aim at creating broader validation strategies that generalize across different UL approaches. Such strategies would allow us to compare and evaluate all UL algorithms such as clustering, dimensionality reduction, etc., under a common validation framework, helping build greater trust in the outcomes of UL systems. In addition, new mutation operators and metamorphic relations that can identify every possible error occurrence in real-life high-dimensional data samples need to be designed.

Additional Information

Author Contributions

All authors have reviewed the final version to be published and agreed to be accountable for all aspects of the work.

Concept and design: Protyasha Roy, Sajjan G. Shiva

Acquisition, analysis, or interpretation of data: Protyasha Roy

Drafting of the manuscript: Protyasha Roy

Critical review of the manuscript for important intellectual content: Sajjan G. Shiva

Supervision: Sajjan G. Shiva

Disclosures

Conflicts of interest: In compliance with the ICMJE uniform disclosure form, all authors declare the following: **Payment/services info:** All authors have declared that no financial support was received from any organization for the submitted work. **Financial relationships:** All authors have declared that they have no financial relationships at present or within the previous three years with any organizations that might have an interest in the submitted work. **Other relationships:** All authors have declared that there are no other relationships or activities that could appear to have influenced the submitted work.

References

1. What is unsupervised learning?. Accessed: April 30, 2025: https://cloud.google.com/discover/what-is-unsupervised-learning.
2. Clustering algorithms. Accessed: April 30, 2025: https://developers.google.com/machine-learning/clustering/clustering-algorithms.
3. Chong B: K-means clustering algorithm: a brief review. Academic Journal of Computing & Information Science. 2021, 4:37-40. 10.25236/ajcis.2021.040506
4. Pelleg D, Moore A: X-means: extending K-means with efficient estimation of the number of clusters. ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning. 2000, 727-734.
5. Bhattacharjee P, Mitra P: A survey of density based clustering algorithms. Frontiers of Computer Science. 2021, 15:151308. 10.1007/s11704-019-9059-3
6. Tyagi K, Rane C, Sriram R, Manry M: Unsupervised learning. Artificial Intelligence and Machine Learning for EDGE Computing. Pandey R, Khatri SK, Singh NK, Verma P (ed): Academic Press, London; 2022. 33-52. 10.1016/B978-0-12-824054-0.00012-5
7. Oyelade J, Isewon I, Oladipupo O, et al.: Data clustering: algorithms and its applications. 2019 19th International Conference on Computational Science and Its Applications (ICCSA), St. Petersburg, Russia. 2019, 71-81. 10.1109/ICCSA.2019.000-1
8. Dasgupta S, Long PM: Performance guarantees for hierarchical clustering. Journal of Computer and System Sciences. 2005, 70:555-569. 10.1016/j.jcss.2004.10.006
9. Xie X, Zhang Z, Chen TY, Liu Y, Poon P-L, Xu B: METTLE: a METamorphic Testing approach to assessing and validating unsupervised machine LEarning systems. IEEE Transactions on Reliability. 2020, 69:1293-1322. 10.1109/tr.2020.2972266
10. Fergus P, Chalmers C: Un-supervised learning. Applied Deep Learning. Computational Intelligence Methods and Applications. Springer, Cham; 2022. 95-113. 10.1007/978-3-031-04420-5_4
11. Naeem S, Ali A, Anam S, Ahmed MM: An unsupervised machine learning algorithms: Comprehensive review. International Journal of Computing and Digital Systems. 2023, 13:911-921.
12. Berahmand K, Daneshfar F, Salehi ES, Li Y, Xu Y: Autoencoders and their applications in machine learning: a survey. Artificial Intelligence Review. 2024, 57:28. 10.1007/s10462-023-10662-6
13. Nanehkaran YA, Licai Z, Chen J, et al.: Anomaly detection in heart disease using a density-based unsupervised approach. Wireless Communications and Mobile Computing. 2022, 2022:1-14. 10.1155/2022/6913043
14. Shriram S, Sivasankar E: Anomaly detection on shuttle data using unsupervised learning techniques. 2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE), Dubai, United Arab Emirates. 2019, 221-225. 10.1109/ICCIKE47802.2019.9004325
15. Cintia Ganesha Putri D, Leu J-S, Seda P: Design of an unsupervised machine learning-based movie recommender system. Symmetry. 2020, 12:185. 10.3390/sym12020185
16. Addagarla SK, Amalanathan A: Probabilistic unsupervised machine learning approach for a similar image

recommender system for E-commerce. Symmetry. 2020, 12:1783. 10.3390/sym12111783
17. Arthur D, Vassilvitskii S: k-means++: The advantages of careful seeding. Technical Report. Stanford, 2006.
18. Shen B: E-commerce customer segmentation via unsupervised machine learning. CONF-CDS 2021: The 2nd International Conference on Computing and Data Science. 2021, 1-7. 10.1145/3448734.3450775
19. Du Toit J, Davimes R, Mohamed A, Patel K, Nye JM: Customer segmentation using unsupervised learning on daily energy load profiles. Journal of Advances in Information Technology. 2016, 7:69-75. 10.12720/jait.7.2.69-75
20. Gomes C, Jin Z, Yang H: Insurance fraud detection with unsupervised deep learning. Journal of Risk and Insurance. 2021, 88:591-624. 10.1111/jori.12359
21. What is a variational autoencoder?. Accessed: April 30, 2025: https://www.ibm.com/think/topics/variational-autoencoder.
22. Monamo P, Marivate V, Twala B: Unsupervised learning for robust Bitcoin fraud detection. 2016 Information Security for South Africa (ISSA), Johannesburg, South Africa. 2016, 129-134. 10.1109/ISSA.2016.7802939
23. Cuesta-Albertos JA, Gordaliza A, Matrán C: Trimmed k-means: An attempt to robustify quantizers. The Annals of Statistics. 1997, 25:553-576. 10.1214/aos/1031833664
24. Rezapour M: Anomaly detection using unsupervised methods: Credit card fraud case study. International Journal of Advanced Computer Science and Applications. 2019, 10:1-8. 10.14569/ijacsa.2019.0101101
25. Lopez C, Tucker S, Salameh T, Tucker C: An unsupervised machine learning method for discovering patient clusters based on genetic signatures. Journal of Biomedical Informatics. 2018, 85:30-39. 10.1016/j.jbi.2018.07.004
26. Zhang JM, Harman M, Ma L, Liu Y: Machine learning testing: Survey, landscapes and horizons. IEEE Transactions on Software Engineering. 2022, 48:1-36. 10.1109/tse.2019.2962027
27. Hypothesis Testing. Accessed: April 30, 2025: https://mathworld.wolfram.com/HypothesisTesting.html.
28. Chen TY, Cheung SC, Yiu SM: Metamorphic testing: a new approach for generating next test cases. arXiv. 2020, 10.48550/arXiv.2002.12543
29. Woodward MR: Mutation testing—its origin and evolution. Information and Software Technology. 1993, 35:163-169. 10.1016/0950-5849(93)90053-6
30. Lu Y, Shao K, Zhao J, Sun W, Sun M: Mutation testing of unsupervised learning systems. Journal of Systems Architecture. 2024, 146:103050. 10.1016/j.sysarc.2023.103050
31. Ramanathan A, Pullum LL, Hussain F, Chakrabarty D, Jha SK: Integrating symbolic and statistical methods for testing intelligent systems: Applications to machine learning and computer vision. 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany. 2016, 786-791.
32. Murphy C, Kaiser GE, Hu L,Wu L: Properties of machine learning applications for use in metamorphic testing. Proceedings of the Twentieth International Conference on Software Engineering & Knowledge Engineering (SEKE'2008), San Francisco, CA, USA. 2008, 867-872.
33. Rehman FU, Izurieta C: MT4UML: Metamorphic testing for unsupervised machine learning. 2022 9th Swiss Conference on Data Science (SDS), Lucerne, Switzerland. 2022, 26-32. 10.1109/SDS54800.2022.00012
34. Rehman FU, Izurieta C: An approach for verifying and validating clustering based anomaly detection systems using metamorphic testing. IEEE International Conference On Artificial Intelligence Testing (AITest), Newark, CA, USA. 2022, 12-18. 10.1109/AITest55621.2022.00011
35. Yang S, Towey D, Zhou ZQ: Metamorphic exploration of an unsupervised clustering program. 2019 IEEE/ACM 4th International Workshop on Metamorphic Testing (MET), Montreal, QC, Canada. 2019, 48-54. 10.1109/MET.2019.00015
36. Sun Y, Huang X, Kroening D, Sharp J, Hill M, Ashmore R: DeepConcolic: Testing and debugging deep neural networks. 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Montreal, QC, Canada. 2019, 111-114. 10.1109/ICSE-Companion.2019.00051
37. Ding J, Kang X, Hu X-H: Validating a deep learning framework by metamorphic testing. 2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET), Buenos Aires, Argentina. 2017, 28-34. 10.1109/MET.2017.2
38. Xie X, Ho J, Murphy C, Kaiser G, Xu B, Chen TY: Application of metamorphic testing to supervised classifiers. 2009 Ninth International Conference on Quality Software, Jeju, Korea (South). 2009, 135-144. 10.1109/qsic.2009.26
39. Ma L, Zhang F, Sun J, et al.: DeepMutation: Mutation testing of deep learning systems. 2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE), Memphis, TN, USA. 2018, 100-111. 10.1109/ISSRE.2018.00021
40. Humbatova N, Jahangirova G, Tonella P: DeepCrime: mutation testing of deep learning systems based on real faults. ISSTA 2021: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2021, 67-78. 10.1145/3460319.3464825
41. Lu Y, Sun W, Sun M: Mutation testing of reinforcement learning systems. Dependable Software Engineering. Theories, Tools, and Applications. Qin S, Woodcock J, Zhang W (ed): Springer, Cham; 2021. 13071:143-160. 10.1007/978-3-030-91265-9_8
42. Tambon F, Majdinasab V, Nikanjam A, Khomh F, Antoniol G: Mutation testing of deep reinforcement learning based on real faults. 2023 IEEE Conference on Software Testing, Verification and Validation (ICST), Dublin, Ireland. 2023, 188-198. 10.1109/ICST57152.2023.00026
43. Rehman FU, Srinivasan M: Metamorphic testing for machine learning: Applicability, challenges, and research opportunities. 2023 IEEE International Conference On Artificial Intelligence Testing (AITest), Athens, Greece. 2023, 34-39. 10.1109/AITest58265.2023.00014
44. Panichella A, Liem CCS: What are we really testing in mutation testing for machine learning? A critical reflection. 2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), Madrid, ES. 2021, 66-70. 10.1109/ICSE-NIER52604.2021.00022