An Approach For Verifying And Validating Clustering Based Anomaly Detection Systems Using Metamorphic Testing

Faqeer ur Rehman
Gianforte School of Computing
Montana State University
Bozeman, USA
faqeer.rehman@student.montana.edu

Clemente Izurieta
Gianforte School of Computing
Idaho National Laboratories
Montana State University
Bozeman, USA
clemente.izurieta@montana.edu

Abstract*—An oracle or test oracle is a mechanism that a software tester uses to verify the program output. In software testing, the oracle problem arises when either the oracle is not available or it may be available but is so expensive that it is infeasible to apply. To help address this problem in testing machine learning-based applications, we propose an approach for testing clustering algorithms. We exemplify this in the implementation of the award-winning density-based clustering algorithm i.e., Density-based Spatial Clustering of Applications with Noise (DBSCAN). Our proposed approach is based on the 'Metamorphic Testing' technique which is considered an effective approach in alleviating the oracle problem. Our contributions in this paper include, i) proposing and showing the applicability of a broader set of 21 Metamorphic Relations (MRs), among which 8 target the verification aspect, whereas, 14 of them target the validation aspect of testing the algorithm under test, and ii) identifying and segregating the MRs (by providing a detailed analysis) to help both naive and expert users understand how the proposed MRs target both the verification and validation aspects of testing the DBSCAN algorithm. To show the effectiveness of the proposed approach, we further conduct a case study on an anomaly detection system. The results obtained show that, i) different MRs have the ability to reveal different violation rates (for the given data instances); thus, showing their effectiveness, and ii) although we have not found any implementation issues (through verification) in the algorithm under test (that further enhances our trust in the implementation), the results suggest that the DBSCAN algorithm may not be suitable for scenarios (meeting the user expectations a.k.a validation) captured by almost 79% of violated MRs; which show high susceptibility to small changes in the dataset.

***Index Terms*—Machine Learning, Clustering, Metamorphic Testing, Verification, Validation, Anomaly Detection, Oracle Problem**

I. Introduction

The use of machine learning algorithms is growing fast in a number of application domains i.e., document clustering, image processing, social network analysis, recommendation systems, customer segmentation, and anomaly detection. As these applications become pervasive in real-world environments, it becomes crucial to verify their correct behavior.

Software testing is a common approach used to test and verify the quality of software. However, one of the problems it faces is the *oracle problem*. An oracle is a mechanism that a software tester uses to verify the program under test. Software is executed for a given test case and the output produced is compared with the expected outcome. A program is said to be buggy if the output produced by the program does not match the expected output. In real-life scenarios, the oracle may not be available or it may exist but is so expensive that it is infeasible to apply.

It always remains a challenging task to test unsupervised machine learning applications in the absence of a test oracle (i.e., in the form of ground truth/class label). To help address improving the quality of such machine learning-based applications, this work focuses on testing the following award-winning unsupervised clustering algorithm (at the leading data mining conference, ACM SIGKDD [1]): Density-based spatial clustering of applications with noise (DBSCAN) (provided by the leading python library *scikit-learn* [2]). Our approach uses the 'Metamorphic Testing (MT)' [3] technique that has been shown to be an effective approach in alleviating the oracle problem [4] [5]. It uses the necessary characteristics of the program as relations (known as Metamorphic Relations i.e., MRs) to check whether the program under test adheres to specified relations or not. Each MR uses a *source test case* and a *follow-up test case* to verify the output (instead of verifying the individual test case output). The violation of an MR is treated as a bug in the application.

To the best of our knowledge, very little effort has been placed in using the MT technique for quality assurance of clustering algorithms [11] [12], and we are only able to find just one paper in which the authors leveraged the MT approach for testing the DBSCAN algorithm (provided by the WEKA tool) [11]. However, the limitations of their approach are that, ii) it uses synthetic and unrealistic 2D data, i) it lacks targeting the verification aspect of testing the DBSCAN algorithm, and ii) it does not provide any evidence to show whether their approach is also applicable to assess the behaviour

of the DBSCAN algorithm provided by some popular open source python libraries i.e., sci-kit learn, which are equally the motivators for this work.

In this study, we make the following main contributions:

- We propose an MT-based approach for verification and validation of a density based clustering algorithm i.e., DBSCAN.
- We propose a collection of 21 diverse MRs that the clustering algorithm under test is expected to satisfy.
- We further provide the analysis and reasoning for the proposed MRs to show whether each of the MRs target the necessary characteristics of the program; and thus, the MR can be used to serve for *verification* purposes. If the program reveals a violation, it would suggest that there is some bug in the program under test. On the other hand, if the MR does not target the necessary characteristics of the program, it can still be used to show evidence for *validation* purposes (i.e., the 'expected' behaviour). In other words, the proposed MRs target both the verification and validation aspects of testing the DBSCAN algorithm under test.
- To show the applicability of the proposed MRs, we conduct a case study on an open-source anomaly detection system[1] that internally uses the DBSCAN algorithm (to isolate/detect the noise).

We organize the rest of the paper as follows. Section II discusses the background knowledge related to unsupervised machine learning and the DBSCAN algorithm under investigation. Section III talks about related work. Section IV presents the MT approach, the proposed MRs, and the analysis and reasoning to show whether the MR targets the 'necessary characteristics' or the 'expected behavior' of the program under test. Section V presents the results obtained from the conducted case study, Section VI discusses the threats to validity, and last, in Section VII we provide a conclusion made along with future work suggestions.

II. Background

In this section, we provide a brief overview of unsupervised machine learning and the clustering algorithm we investigated i.e., *DBSCAN*.

A. Unsupervised Machine Learning

In unsupervised machine learning, given the unlabeled data set D having x inputs with attributes $< A_o, A_1, A_2, ..., A_n >$, the goal is to partition the data set D into different groups/clusters such as $< C_0, C_1, C_2, ..., C_n >$, in such a way that the instances within the cluster C_i are highly similar to each other, whereas, the instances between the clusters C_i and C_j are highly dissimilar. Some of the widely used clustering techniques include K-means clustering (a prototype-based approach), Agglomerative clustering (a hierarchy-based approach), and DBSCAN (a density-based approach). Prior work has focused on testing the K-means and Agglomerative clustering algorithms [21], whereas, in this work, we focus on testing the popular DBSCAN algorithm (provided by the python scikit-learn library) from both verification and a validation perspective.

[1]https://github.com/matifkhattak/DBScanAnomalyDetection

B. DBSCAN

The DBSCAN algorithm proposed by Ester et al. [19] is designed for a scenario when distributions contain groups of arbitrary shapes. The algorithm has the ability to separate the noise and outliers; which are treated as anomalies. As shown in Figure. 1, this algorithm finds the *core points* that have at least $min\_samples$ (minimum number of data points) in their neighborhood within the radius $epsilon$. The *border points* are within the radius of some core point(s) but do not have the $min\_samples$ in their neighborhood to become a core point. The point is treated as a *noise point*, if there is no data point in its neighborhood within the radius $epsiolon$.

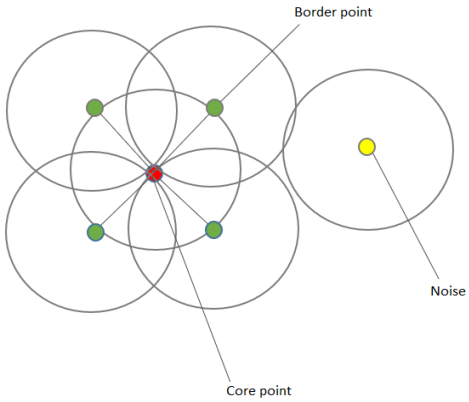

Fig. 1. DBScan Algorithm with min_samples=5

The algorithm starts with an arbitrary core point c_i and it grows the cluster by adding all its neighbor core points (that are within its radius $epsilon$). The cluster continues growing until all the core points and border points (reachable to the nearest core point) are assigned to a single cluster. The same process is repeated and an arbitrary new unvisited core point is selected to form the second cluster, then the third cluster, and so on. The points that neither fall in the category of *core points* or *border points* are treated as *noise or anomalies*.

III. Related Work

MT has been shown to be an effective approach in alleviating the oracle problem in a broad range of applications i.e., forecasting [6], image classification [7], acoustic scene classification [8], intrusion detection systems [9] [10], and machine translation [13]. Xie et al. [14] applied MT to test a category of supervised classifiers that includes K-means and Naive Bayes algorithms (provided by the WEKA tool). The authors proposed 11 MRs to test these two algorithms and identified real faults in the Naive Bayes classifier. Dwarakanath et al. [7] proposed 8 MRs to test an SVM-based digits image

classifier. Their results show that the proposed approach is able to uncover 71% of the implementation faults in the applications under test. Santos et al. [15] applied MT on a breast cancer machine learning classifier and the results obtained show that MT can be considered a useful approach in testing the effectiveness of ML-based classifier in a medical domain.

With the availability of large data sets and higher computing power, deep learning solutions are getting popular and are becoming an integral part of critical applications e.g., autonomous vehicles, health care systems, and intrusion detection systems. Pei et al. [16] used the differential testing technique in combination with MT to test real-world deep learning systems. They leveraged the MRs to generate difference-causing images and found thousands of erroneous behaviors in the programs under test. Zhou et al. [17] used MT to test self-driving cars and found subtle errors eight days before the deadly Uber accident took place. Ding et al. [18] applied the MT approach to validate a deep learning framework that has been used for the classification of biomedical images. The proposed approach was shown to be effective in validating the framework that includes, i) the architecture of a convolutional neural network, ii) the execution environment Caffe, and iii) the data set comprised of cellular images. The most recent work [9] [10] includes taking advantage of machine learning and statistical-based MT techniques to uncover the implementation faults in a DNN-based intrusion detection system and a cancer prediction system.

IV. Our Approach

In this study, we propose Metamorphic Testing (MT) based approach and show how can it be used to target both the *verification* and *validation* aspect of testing the clustering-based ML applications. In our work, we aim to answer the following two Research Questions (RQs):

- **RQ1**: How effective is the proposed approach in testing the application under test?
- **RQ2**: Do all MRs have the same ability to detect the violations?

To date, one of the main challenges faced in the software testing domain is the *oracle problem*. One possible approach to target the oracle problem is the *Differential Testing* technique that uses multiple implementations of the same program as a *pseudo oracle* to verify the output. However, the limitation of this approach that makes it unrealistic in real life is that it is very difficult to obtain multiple copies of the same system, and second, it is possible that multiple implementations have the same bug which causes them to produce the same output for the same given input.

Our approach for testing the DBSCAN algorithm uses the MT technique which is considered an effective approach in alleviating the oracle problem and it also does not require multiple implementations of the program under test. Instead of verifying the individual output, it uses multiple executions of the program to check whether the output is correct or not. The output should adhere to the relation known as *Metamorphic Relation (MR)* that targets the necessary characteristics of the program under test. Each MR uses the *source input* and some valid change/transformation made to the *source input* (using the relation specified in the MR) known as the *follow-up input*. The MR is said to be violated if the output produced for the *source input* does not match with the output produced for the *follow-up input*. As an example, suppose there is a program that calculates the standard deviation for the given inputs. An instance of an MR can be exemplified by multiplying the original inputs with -1 (this transformation will produce the follow-up inputs), the output of the program should remain the same because this change will have no effect on the deviation from the mean.

First, we refer to the official documentation of python based implementation available for the DBSCAN algorithm [22] and propose a broader set of 21 MRs which will enable novice to expert data scientists the ability to test and assess DBSCAN's behavior from multiple perspectives. The list of all the proposed MRs along with their descriptions is provided in Table I. Next, in Table II, we further analyze the proposed MRs and provide the detailed reasoning to show whether each of the MRs targets the verification aspect or the validation aspect of testing the DBSCAN algorithm. This is important because in real-life projects, the data may undergo through some modification and if the user is not aware of the potential consequences on the clustering results, it may lead to misleading decisions and disastrous consequences. Lastly, we show the effectiveness of the proposed MRs on testing an anomaly detection system that utilizes the DBSCAN algorithm, provided by the scikit-learn python library. It is important to highlight that the proposed MRs are not just limited to testing the anomaly detection system under test, instead, they are applicable to number of other real-world applications (i.e., market and customer segmentation, document clustering, search results clustering, biological data clustering, etc.) as long as they use the DBSCAN algorithm for identification of clusters in the data.

V. Experimentation and Evaluation

To show the effectiveness of our proposed MT-based approach, we applied our technique to test an open-source anomaly detection system that internally uses the DBSCAN algorithm (provided by scikit-learn 0.24.1 [2]) to identify noise and anomalies. The scikit-learn is a popular python library that provides a large set of features for performing data preprocessing, classification, and clustering related tasks. Due to its wide range of capabilities, it is very popular among both researchers and practitioners for addressing ML-related problems.

The ingested data to the application under test is comprised of 3093 instances (having 29 features). This data is used to cluster the input data into different groups. Once the clustering process is completed, the instances that are assigned to the -1 cluster are treated as noise/anomaly. All the application code,

TABLE I
PROPOSED METAMORPHIC RELATIONS

Metamorphic Relations (MRs)	Description
MR1 - Data Duplication	**MR1.1 - Duplicating single instance:** For the source input s, we denote the output obtained as O_s. This MR says that if a single instance in the follow-up input f is duplicated, the output O_f should remain similar to the O_s i.e., $O_s = O_f$. **MR1.2 - Duplicating multiple instances:** This MR says that if we duplicate multiple instances (i.e., each belonging to a different cluster) in the follow-up input, the output for both the source and follow-up inputs should remain the same.
MR2 - Data Standardization	For the follow-up input, if the existing standardized data is once again standardized, it should not have any effect on changing the final output i.e., $O_s = O_f$.
MR3 - Features Duplication	For the follow-up input, if the existing feature(s) are duplicated, the output should remain consistent for both the source and follow-up inputs i.e., $O_s = O_f$.
MR4 - Removing Instance(s)	**MR4.1 - Removing an instance from a single cluster:** For the source input s, we denote the clusters found as $C_0, C_1, C_2, ..., C_n$. This MR says that for the follow-up input if an instance is removed from just a single cluster i.e., C_1, the output should remain the same i.e., it should not have any effect on changing the results for the remaining instances. **MR4.2 - Removing an instance from multiple clusters:** For the source input s, we denote the clusters found as $C_0, C_1, C_2, ..., C_n$. This MR says that for the follow-up input if an instance is removed from each of the obtained clusters i.e., $C_1, C_2, C_3, ..., C_n$, the output should remain the same. **MR4.3 - Removing multiple instances from a single cluster:** For the source input s, we denote the clusters found as $C_0, C_1, C_2, ..., C_n$. This MR says that for the follow-up input if multiple instances are removed from just a single cluster i.e., C_1, it should not have any effect on changing the results for the remaining instances.
MR5 - Adding an Uninformative Feature	This MR says that for the follow-up input if we add a new uninformative feature (i.e., having a constant value for all the instances), the output should remain consistent for both the source and follow-up inputs i.e., $O_s = O_f$.
MR6 - Deterministic Output Across Multiple Runs	This MR says that if a new data point is added to the original data set, no matter how many times an algorithm under test is run, the output for this input and other inputs should remain consistent i.e., the output at $time_i$ and $time_{i+1}$ should remain the same.
MR7 - Shifting of Data	This MR says that if we shift the feature(s) with some constant k, the output for both the source and follow-up inputs should remain consistent i.e., $O_s = O_f$.
MR8 - Scaling of Data	This MR says that if we scale the feature(s) with some constant k, the output for both the source and follow-up inputs should remain unchanged i.e., $O_s = O_f$.
MR9 - Data Replacement	**MR9.1 - Replacing single instance:** This MR says that if we replace a single instance in cluster c_i with another instance (belonging to the same cluster c_i), the output should remain unchanged. **MR9.2 - Replacing multiple instances:** This MR says that if we replace multiple instances in cluster c_i with another instance (belonging to the same cluster c_i), the output for both the source and follow-up inputs should remain the same. **MR9.3 - Replacing all instances:** This MR says that if we replace all the instances in cluster c_i with another instance (belonging to the same cluster c_i), it should not have any effect on changing the output for this, and the remaining instances.
MR10 - Shuffling the Features	This MR says that if we shuffle the data features to change their existing order, the output for both the source and follow-up inputs should remain the same.
MR11 - Adding an Informative Attribute	This MR says that for the follow-up input if we add a new informative feature (i.e., its value is strongly associated with each of the clusters), the output should remain consistent for both the source and follow-up inputs i.e., $O_s = O_f$.
MR12 - Transformation of Rows	**MR12.1 - Reversing the order:** This MR says that reversing the order of data instances should not have any effect on changing the final output i.e., the output for both the source and follow-up inputs should remain consistent. **MR12.2 - Random shuffling:** This MR says that if we randomly shuffle the data instances, it should not have any effect on changing the final output i.e., the output for both the source and follow-up inputs should remain the same.
MR13 - Data Reflection	This MR says that if we perform the data reflection transformation on the original inputs i.e., multiply all the features with -1, the output should remain unchanged i.e., $O_s = O_f$.
MR14 - Addition of Instance(s)	**MR14.1 - Adding a new instance with informative attributes:** For the follow-up input, if we add a new data point in the middle of two existing data points of cluster c_i (thus enhancing the density of the cluster), it should not have any effect on changing the results for remaining instances. **MR14.2 - Adding a new instance with uninformative attributes** For the follow-up input, if we add a new data point with uninformative attributes i.e., features with 0 value, it should not have any effect on changing the results for the remaining instances.

the data, and the MRs implementation are provided online[2].

After applying the proposed MRs to the anomaly detection application under test, we present the results in Tables III and IV. The results in Table III shows, i) whether the MR is rejected/violated, and ii) for the given violated MR, what number and percentage of data instances have shown violations. A higher percentage of violation rate suggests that the program is highly susceptible to small changes in the dataset, therefore, it should be avoided in the scenarios captured by the violated MRs. Each of the rejected MRs either shows the deviation from the program specification (verification) or its deviation from general user expectations (validation). If the application under test violates the MR targeting the verification aspect, it would suggest that there is some bug in the program under test. On the other hand, if the MR that does not target the necessary characteristics of the program is violated, it can still be used to show evidence for *validation* purposes (i.e., the 'expected' behaviour). Further analysis of the results in Table III show that in total, the application under test has violated 11 out of 21 MRs (i.e., 52%).

In order to synthesize the granular information provided in

[2]https://github.com/matifkhattak/DBScanAnomalyDetection

TABLE II
DBSCAN ALGORITHM: ANALYSIS FOR THE PROPOSED MRS FROM VERIFICATION (VR) AND VALIDATION (VD) PERSPECTIVE

#	VR?	VD?	Reasoning
MR1	×	✓	For the source input, we denote the clusters identified as C_1, C_2, C_3, ..., C_n. For the follow-up input, if we duplicate a data point x (which is a border point) in cluster C_1, the addition of a new data instance may change this border point x to become a core point. This core point will now be used by the DBSCAN algorithm to further grow the cluster, which can resultantly assign the other border points (belonging to different clusters) to this cluster C_1; thus, changing the final output for the follow-up inputs. A sample example is provided in the excel sheet (available in the shared GitHub repository) that shows the violation of this MR. It is worth mentioning that this MR is not the necessary characteristic of the algorithm under test because the violation of this MR is not due to the bug in the implementation. Instead, the behavior of the program does adhere to the specification but violates the user's potential requirements/expectations; thus, can not be used for verification but can definitely be used for validation purposes. Apart from that, an MR targeting the verification aspect can also be used for validation purposes but not always vice versa. ***Note:*** *We urge the readers to use the same reasoning (as mentioned above) for the rest of the proposed MRs in order to target the verification and validation aspect of testing the algorithm under test.*
MR2	✓	✓	This MR can be used for verification (hence also for validation) purposes because the re-execution of the standardization step will neither change the mean and variance of data points nor the distance between the core points, border points and noisy points. Therefore, this transformation should not have any effect on how the core points, border points, and noisy points are identified by the algorithm to make the clusters. If this MR is violated, it would depict that there is some bug in the program under test.
MR3	×	✓	For the follow-up input, if we duplicate all the existing attributes to the original data points, it will result in doubling the distance between them. As a result, for a data point x_i, the *min_samples (i.e., minimum no of data points)* within the *epsilon* distance may get fewer which will result in the identification of a different number of core points, border points, and noisy points; thus, may assign the data points to different clusters.
MR4	×	✓	For the follow-up input, if we remove any data point(s), it may lead to i) changing the core point to become a border point, and ii) a border point to the noisy point; thus, changing the final output for the program under investigation. Since this violation can not be characterized as a violation of the program specification, so, this MR can be used for validation purposes (but not for verification purposes).
MR5	✓	✓	For the follow-up input, if we add an uninformative feature i.e., a feature with value 0, to all the data points, it will not change the existing relationship between the core points, border points, and noisy points. Therefore, the output should remain consistent for both the source and follow-up inputs.
MR6	✓	✓	If we run the program under test multiple times, it will not have any effect on how the algorithm identifies the core points, border points and noisy points. Furthermore, as we do not make any change in the data, so, the output should remain consistent. On the contrary, if the output for the follow-up input is different from the output obtained for the source input, it would depict that there is some bug in the implementation of the program under test. Therefore, this MR can be used for both verification and validation purposes.
MR7	✓	✓	For the follow-up input, if we shift all the features of the data points with some constant c, it will not change the existing relationship/distance between the data points. To understand this concept, suppose x represents the core point, y represents the border point, and, the distance found between them during the source execution (i.e., $d(x, y)$) is represented as z. Now, after shifting the features for the follow-up inputs, the distance between these two points will remain the same i.e., $d(x, y) = \sqrt{((x+c)-(y+c))^2} => \sqrt{(x-y)^2} => x - y => z$. Therefore, the output should remain unchanged for both the source and follow-up inputs.
MR8	×	✓	If all the features are scaled with some constant c, it will result in increasing the distance between the data points. As a result, for a data point x_i, the *min_samples* within the *epsilon* distance may get fewer which will result in the identification of a different number of core points, border points, and noisy points; thus, leading to the assignment of data points to different clusters. In our conducted experiment (results are shown in Table III), the scaling of features moved the data points so farther away that majority of them got assigned to cluster -1 (treated as noise/anomaly).
MR9	×	✓	For the follow-up input, if we replace the instance x with y (which is a border point), this will add a new data point within the *epislon* distance of y, which may make this instance to become a core point. This core point will now be used by the DBSCAN algorithm to further grow the cluster, which can resultantly assign the other border points (belonging to different clusters) to the cluster to which y belongs; thus, changing the final output for the follow-up execution.
MR10	✓	✓	This MR can be used for verification and validation purposes because changing the location of features will neither change the distance between the data points nor their location in space. Therefore, this transformation should not have any effect on how the core points, border points, and noisy points are identified to make the clusters. The violation of this MR would depict that there is some implementation bug in the program under test.
MR11	✓	✓	For the source input, we denote the clusters identified as C_1, C_2, C_3, ..., C_n. This MR says that for the follow-up input, if we add an informative attribute such that the added attribute is strongly associated with each cluster's instances i.e., having value s for C_1 instances, having value t for C_2 instances, having value u for C_3 instances, and similarly, having value v for C_n instances, this will not change the existing relationship between the data points belonging to the same cluster; thus, the instances should be assigned to the same clusters as identified for the source inputs.
MR12	✓	✓	Same reasoning as provided for MR10. For the follow-up input, if we change the order of data instances, it will neither change the distance between the data points nor their location in space. Therefore, this transformation should not have any effect on how the core points, border points, and noisy points are identified to make the clusters. The violation of this MR would depict that there is some implementation bug in the program under test.
MR13	✓	✓	This MR will also not change the existing relationship/distance between the data points. Therefore, it should not have any effect on changing the final output for the follow-up inputs.
MR14	×	✓	Same reasoning as provided for MR9. If we add a new data point to any of the clusters (identified for the source inputs), it may change the border point y to become a core point. This core point will now be used by the DBSCAN algorithm to further grow the cluster, which can resultantly assign the other border points (belonging to different clusters) to the cluster to which y belongs; thus, changing the final output for the follow-up inputs.

Table II, we present the results of violated MRs at two levels (as shown in Table IV):

i) MRs targeting the *verification* aspect, and
ii) MRs targeting the *validation* aspect.

TABLE III
RESULTS FROM TESTING THE DBSCAN ALGORITHM

MR#	Violation?	No of Violated Instance(s)	Violation Rate
MR1.1	✓	5	0.16%
MR1.2	✓	7	0.23%
MR2	×		
MR3	✓	240	7.76%
MR4.1	✓	1	0.03%
MR4.2	✓	2	0.06%
MR4.3	✓	1937	62.63%
MR5	×		
MR6	×		
MR7	×		
MR8	✓	2926	94.60%
MR9.1	✓	1	0.03%
MR9.2	✓	4	0.13%
MR9.3	✓	171	5.53%
MR10	×		
MR11	×		
MR12.1	×		
MR12.2	×		
MR13	×		
MR14.1	×		
MR14.2	✓	1	0.03%

TABLE IV
MRS SEGGREGATION AND THEIR RESULTS

	Total No. of MRs	No. of Violated MRs	%age of Violated MRs
Verification	8	0	0%
Validation	14	11	79%

In order to understand the possible reasons behind the violated MRs, we refer readers to the detailed analysis and reasoning provided in Table II. The results provided in Table IV show that none of the MRs targeting the *verification* aspect have been violated, which further enhance our trust on the implementation of this algorithm (i.e., its adherence of implementation to specification). However, it is important to note that failure to detect the violations for this type of MRs does not necessarily mean that the program under test is free from bugs. Instead, it is the inherent limitation of every testing technique (i.e., if a testing technique is unable to uncover bugs in a program, it does not necessarily mean that the program does not contain any defects at all), as is with ours. Therefore, we recommend that the proposed MRs should still be used as a supplementary testing technique in the organization's existing built-in testing pipeline. Apart from that, we see a higher number of violated MRs (i.e., 11 out of 14 $=>$ 79%) targeting the *validation* aspect, which ultimately suggests that the program under test is highly susceptible to small valid changes in the input data and may not be suitable for the scenarios (captured by the violated MRs) to meet the general user expectations from this algorithm. This answers our first research question that **the proposed approach is effective in the detection of violations (targeting the validation aspect) in the program under test**. We hope that this type of useful information will help both researchers and practitioners to be aware of possible repercussions in the final results when the data itself may undergo changes in the future (which is very common in real world use cases). If they foresee the possible usage of application in the environment/scenarios captured by the violated MRs then this set of knowledge would significantly help them to perform a comparative analysis of the behaviour of different clustering algorithms, choosing the one best suited for their problem, and making well informed decisions.

To answer the second research question, the results obtained show that different MRs have detected a different percentage of violations for the given instances. It can be seen in Table III that MR8 has detected a higher percentage (94.60%) of violations, showing that the program under test is highly susceptible to small input changes (captured by this MR), whereas, MR4.1, MR9.1, and MR14.2 have the lowest percentage (0.3%) of violations (showing that among the violated MRs, for these three MRs, the program under test is least susceptible to small modifications). Apart from that, MR4.1, MR9.1, and MR14.2 have shown the violation for the same number of instances (i.e., one instance), thus; one may argue that since these all three MRs have the same test effectiveness, using only one of them would be enough. However, this reasoning (without making a deeper analysis) may be subjective and incorrect. Therefore, we further analysed the results for these three MRs and found that the instance for which the program shows a violation is different for each of the violated MRs; thus, revealing the diversity of the proposed MRs and suggesting that they all have different test effectiveness. This answers our second research question that **different MRs have different ability to detect the violation for different number/percentage of data inputs.**

VI. THREATS TO VALIDITY

In this section, we examine the threats to the validity of this research as described by Wohlin et al. [20]. Although the proposed approach seems to be effective in its ability to identify scenarios for which DBSCAN shows inconsistent behavior, we identify the following threats to validity:

- It can be argued that the proposed MRs may not be sufficient to validate a potentially large set of scenarios. This threat refers to the possibility of having unwanted or unanticipated causal relationships as a result of the MRs selected. This poses a potential threat to *internal validity* of this study. We minimize this threat by proposing a sufficient number of diverse MRs whose results show that even with this limited, but diverse set of MRs, we successfully identify the violations for 79% of the scenarios (captured by the MRs targeting the validation aspect).

- *Construct validity* refers to the meaningfulness of measurements made. The threat to *construct validity* may occur because of the selection of the algorithm for this study. We tried to minimize this threat by showing the effectiveness of the proposed MRs on an award-winning DBSCAN clustering algorithm, provided by the leading python library *scikit-learn*.
- *Content validity* refers to how complete the proposed MRs cover the content domain; which in our case is clustering algorithms. Although our focus is on DBSCAN, we have proposed a large set (i.e., 21) diverse MRs that are not only applicable to test this algorithm but has also been applied in our previous work [21] for testing *partitioning-based* and *hierarchical-based* clustering algorithms.
- The DBSCAN algorithm under test belongs to the category of density-based clustering algorithms, thus generalizing the results (obtained from the proposed approach) to other type of density-based algorithms might be speculative and poses a threat to *external validity*. We aim to minimize this threat in the future by showing the applicability of the proposed approach on a collection of other density-based algorithms.

VII. Conclusion And Future Work

Our contribution with this research includes proposing an approach that both naive and expert users can benefit from for testing clustering based applications. This work shows the feasibility of the proposed MT based approach in both the verification and validation of the award-winning DBSCAN algorithm (a density based clustering algorithm), provided by the widely used python library i.e., scikit-learn. We proposed a broader set of 21 MRs and also provided a detailed analysis and reasoning to show whether each of the proposed MRs targets the verification or the validation aspect of testing the algorithm under test. Despite the fact that we have proposed a broader set of 21 MRs, this list is not exhaustive, and instead, we expect researchers to further expand this seminal list with additional MRs that can be leveraged to test a broader set of ML applications.

To show the effectiveness of the proposed approach, we further conducted a case study on an anomaly detection system. The results obtained show that 79% of the MRs (targeting the validation aspect) have been violated by the program under test, with one MR showing the highest violation rate of 94.5%. To show the applicability of the proposed MRs in general, we not only hope to apply them in testing other types of density based algorithms but also aim to further enhance the MR repository in the future.

References

[1] SIGKDD, A. (2014). sigkdd test of time award https://www.kdd.org/News/view/2014-sigkdd-test-of-time-award

[2] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2825-2830.

[3] Chen, T. Y., Cheung, S. C., & Yiu, S. M. (1998). Metamorphic testing: a new approach for generating next test cases. Technical Report HKUST-CS98-01. Hong Kong Univ. of Science and Technology.

[4] Jiang, M., Chen, T. Y., Kuo, F. C., Towey, D., & Ding, Z. (2017). A metamorphic testing approach for supporting program repair without the need for a test oracle. Journal of systems and software, 126, 127-140.

[5] Liu, H., Yusuf, I. I., Schmidt, H. W., & Chen, T. Y. (2014, May). Metamorphic fault tolerance: An automated and systematic methodology for fault tolerance in the absence of test oracle. In Companion Proceedings of the 36th International Conference on Software Engineering (pp. 420-423).

[6] Dwarakanath, A., Ahuja, M., Podder, S., Vinu, S., Naskar, A., & Koushik, M. V. (2019, May). Metamorphic testing of a deep learning based forecaster. In 2019 IEEE/ACM 4th International Workshop on Metamorphic Testing (MET) (pp. 40-47). IEEE.

[7] Dwarakanath, A., Ahuja, M., Sikand, S., Rao, R. M., Bose, R. J. C., Dubash, N., & Podder, S. (2018, July). Identifying implementation bugs in machine learning based image classifiers using metamorphic testing. In Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 118-128).

[8] Moreira, D., Furtado, A. P., & Nogueira, S. (2020, August). Testing acoustic scene classifiers using Metamorphic Relations. In 2020 IEEE International Conference On Artificial Intelligence Testing (AITest) (pp. 47-54). IEEE.

[9] ur Rehman, F., & Izurieta, C. (2021, July). Statistical Metamorphic Testing of Neural Network Based Intrusion Detection Systems. In 2021 IEEE International Conference on Cyber Security and Resilience (CSR) (pp. 20-26). IEEE.

[10] ur Rehman, F., & Izurieta, C. (2021, September). A Hybridized Approach for Testing Neural Network Based Intrusion Detection Systems. In 2021 International Conference on Smart Applications, Communications and Networking (SmartNets) (pp. 1-8). IEEE.

[11] Xie, X., Zhang, Z., Chen, T. Y., Liu, Y., Poon, P. L., & Xu, B. (2020). METTLE: a METamorphic testing approach to assessing and validating unsupervised machine LEarning systems. IEEE Transactions on Reliability, 69(4), 1293-1322.

[12] Yang, S., Towey, D., & Zhou, Z. Q. (2019, May). Metamorphic exploration of an unsupervised clustering program. In 2019 IEEE/ACM 4th International Workshop on Metamorphic Testing (MET) (pp. 48-54). IEEE.

[13] Sun, L., & Zhou, Z. Q. (2018, November). Metamorphic testing for machine translations: MT4MT. In 2018 25th Australasian Software Engineering Conference (ASWEC) (pp. 96-100). IEEE.

[14] Xie, X., Ho, J. W., Murphy, C., Kaiser, G., Xu, B., & Chen, T. Y. (2011). Testing and validating machine learning classifiers by metamorphic testing. Journal of Systems and Software, 84(4), 544-558.

[15] Santos, S. H., da Silveira, B. N. C., Andrade, S. A., Delamaro, M., & Souza, S. R. (2020, October). An Experimental Study on Applying Metamorphic Testing in Machine Learning Applications. In Proceedings of the 5th Brazilian Symposium on Systematic and Automated Software Testing (pp. 98-106).

[16] Pei, K., Cao, Y., Yang, J., & Jana, S. (2017, October). Deepxplore: Automated whitebox testing of deep learning systems. In proceedings of the 26th Symposium on Operating Systems Principles (pp. 1-18).

[17] Zhou, Z. Q., & Sun, L. (2019). Metamorphic testing of driverless cars. Communications of the ACM, 62(3), 61-67.

[18] Ding, J., Kang, X., & Hu, X. H. (2017, May). Validating a deep learning framework by metamorphic testing. In 2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET) (pp. 28-34). IEEE.

[19] Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996, August). A density-based algorithm for discovering clusters in large spatial databases with noise. In kdd (Vol. 96, No. 34, pp. 226-231).

[20] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln, Experimentation in Software Engineering. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. [Online]. Available: http://link.springer.com/10.1007/978-3-642-29044-2

[21] ur Rehman, F., & Izurieta, C. MT4UML: Metamorphic Testing for Unsupervised Machine Learning. In 2022 9th Swiss Conference on Data Science. IEEE (*manuscript accepted*)

[22] https://scikit-learn.org/stable/modules/clustering.html#dbscan