Published by
NIGERIAN SOCIETY OF PHYSICAL SCIENCES
Available online @ https://journal.nsps.org.ng/index.php/jnsps

J. Nig. Soc. Phys. Sci. 5 (2023) 1364

Journal of the Nigerian Society of Physical Sciences

An Empirical Study on Anomaly Detection Using Density-based and Representative-based Clustering Algorithms

Gerard Shu Fuhnwi[a,*], Janet O. Agbaje[b], Kayode Oshinubi[c], Olumuyiwa James Peter[d,**]

[a]*Department of Computer Science, Montana State University, Montana, USA.*
[b]*Department of Mathematical Science, Montana Technological University, Montana, USA.*
[c]*School of Informatics, Computing and Cyber Systems, Northern Arizona University, Arizona, USA.*
[d]*Department of Mathematical and Computer Sciences & Department of Epidemiology and Biostatistics, School of Public Health, University of Medical Sciences, Ondo, Nigeria.*

Abstract

In data mining, and statistics, anomaly detection is the process of finding data patterns (outcomes, values, or observations) that deviate from the rest of the other observations or outcomes. Anomaly detection is heavily used in solving real-world problems in many application domains, like medicine, finance , cybersecurity, banking, networking, transportation, and military surveillance for enemy activities, but not limited to only these fields. In this paper, we present an empirical study on unsupervised anomaly detection techniques such as Density-Based Spatial Clustering of Applications with Noise (DBSCAN), (DBSCAN++) (with uniform initialization, k-center initialization, uniform with approximate neighbor initialization, and k-center with approximate neighbor initialization), and k-means$--$ algorithms on six benchmark imbalanced data sets. Findings from our in-depth empirical study show that k-means$--$ is more robust than DBSCAN, and DBSCAN++, in terms of the different evaluation measures ($F1$-score, False alarm rate, Adjusted rand index, and Jaccard coefficient), and running time. We also observe that DBSCAN performs very well on data sets with fewer number of data points. Moreover, the results indicate that the choice of clustering algorithm can significantly impact the performance of anomaly detection and that the performance of different algorithms varies depending on the characteristics of the data. Overall, this study provides insights into the strengths and limitations of different clustering algorithms for anomaly detection and can help guide the selection of appropriate algorithms for specific applications.

Keywords: Outliers, Noise points, ANN, k-means$--$, DBSCAN, DBSCAN++

Article History :
Received: 22 January 2023
Received in revised form: 18 March 2023
Accepted for publication: 22 March 2023
Published: 19 April 2023

Communicated by: B. J. Falaye

1. Introduction

Anomaly detection is the process of finding data patterns (outcomes, values, or observations) that deviate from the rest of the other observations or outcomes. Anomaly detection is heavily used in solving real-world problems in many application domains like medicine, cybersecurity [1], fraud detection [2], networking, transportation, and military surveillance for enemy activities, but not limited to only these fields, as anomaly detection is classified under deep learning which is applicable in all fields such as in mathematics and statistics [3]. These deviating outcomes or observations are referred to as anomalies (outliers,

*Corresponding author tel. no: +1 931 266 5067
**Corresponding author tel. no: +234803 356 0280
Email addresses: gerard.shufuhnwi@student.montana.edu (Gerard Shu Fuhnwi), peterjames4real@gmail.com (Olumuyiwa James Peter)

deviants, discordant observations, exceptions, surprises, or abnormalities) in different application domains [4].

Anomaly detection algorithms can be categorized into two types: supervised and unsupervised. Supervised anomaly detection involves training a model on labeled data, where anomalies are explicitly labeled or defined. On the other hand, unsupervised anomaly detection involves identifying anomalies without using explicit labels or prior knowledge about the data. Unsupervised methods are more commonly used in practice, as labeled data for training supervised models may be scarce or difficult to obtain.

Various techniques can be used for anomaly detection, including statistical methods, clustering, nearest-neighbor methods, and machine learning algorithms such as support vector machines, decision trees, and neural networks [5]. With the increase in the volume and complexity of data, traditional rule-based approaches have proven to be insufficient in detecting anomalies. As a result, clustering-based techniques have gained popularity as reliable methods for identifying anomalies. In this regard, density-based and representative-based clustering algorithms have been widely used in anomaly detection due to their ability to identify clusters of data points dissimilar to the majority of the data.

There have been many approaches to solving anomaly detection problems, with the unsupervised algorithms being the most widely usedÂ because the techniques involve training the model with unlabeled data. Clustering is the process of grouping a set of observations or data points into multiple groups so that observations within a group or cluster have high similarity but dissimilar to observations from the other clusters. Clustering-based techniques fall under a class of unsupervised anomaly detection techniques that operate on the output of the clustering algorithm and thus turn out to be much faster in general. The clustering based techniques can be grouped into the following categories: representative-based techniques, density-based techniques and hierarchical-based techniques.

Throughout the research community, a lot of work has been done to detect anomalies using clustering-based techniques, see the work of [6-16]. In this paper, we present what is (to the best of our knowledge) the first attempt of an empirical study on anomaly detection using k-means$--$, DBSCAN, and DBSCAN++ using data sets from different domains with varying proportions of outliers. This paper aims to evaluate the performance of representative-based and density-based clustering algorithms detecting anomalies, their computational efficiency, and the effect of varying algorithm parameters on their performance. Our goal is to find out how these methods perform on different data sets with regards to the following evaluation metrics: $F1$ score, False alarm rate, Jaccard coefficient, and Adjusted rand index including the run time of these algorithms on the data sets. Finally, and most importantly, the above mentioned techniques all have the tendency of finding noise points (anomalies or outliers) and assigning labels to them as noise points.

Although the main goal is to evaluate the effectiveness of density-based clustering algorithms like DBSCAN, DBSCAN++ and representative-based clustering algorithm like k-means$--$. Our approach can also provide guidance on how to evaluate and analyze these clustering techniques in solving anomaly detection problems. Also, this method can be used to overcome one of the main challenges of anomaly detection techniques, which is accurate representative labels for normal and abnormal instances, which is a major concern. To overcome this challenge in most anomaly detection problems, our approach can be used as a pre-labeling technique and then apply supervised anomaly detection techniques to solve anomaly detection problems. Overall, our empirical results demonstrate the potential of density-based clustering and representative-based clustering and provide valuable insights for future research in this field.

The rest of this paper is structured as follows: In section 2, we briefly give a description of the algorithms used in this paper. Section 3, analyses the empirical evaluation, where we review data sets used, evaluation metrics description, variations in evaluation metrics, results, and result discussion. Section 4 covers the conclusion and future directions.

2. Methods

This section presents the anomaly detection techniques used in this paper. These anomaly detection techniques are: k-means$--$, and two versions of Density-Based Spatial Clustering of Application with Noise (DBSCAN and DBSCAN++).

2.1. k-means$--$

k-means$--$ [17] is a representative-based clustering technique, which is an extension of the k-means algorithm. k-means$--$ is a more computationally efficient version of the k-means algorithm, which achieves this by updating the cluster centroids more incrementally. In the k-means$--$ algorithm, the centroid of each cluster is initialized as the mean of a randomly selected subset of data points rather than as a randomly selected data point, as in the original k-means algorithm. Then, for each iteration of the algorithm, k-means$--$ updates the centroids by considering only the data points that belong to the cluster being updated rather than all the data points in the data set. This results in faster convergence and improved scalability, particularly for large data sets. The pseudo-code of the k-means$--$ is shown in algorithm 1. We implemented the k-means$--$ in Python using the pseudo-code in algorithm 1, since the implementation was not available in Sklearn.

2.2. Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

DBSCAN is a density-based clustering technique capable of finding arbitrarily shaped clusters. DBSCAN proceeds by computing the empirical densities for each sample point and then designating points whose densities are above a threshold as core points. Then, a neighborhood graph of the core points is constructed, and the clusters are assigned based on the connected components. The pseudo-code of DBSCAN [18] is shown in algorithm 2. We used the Sklearn implementation of DBSCAN in Python. Still, this implementation could not be

ALGORITHM 1: $(k - MEANS - -)$

Input: Set of points $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$
A distance function $d: X \times X \rightarrow \mathbb{R}$
Numbers k and l
Output: A set of k cluster centers C
A set of l outliers $L \subseteq X$
$C_0 \leftarrow \{k$ random points of $X\}$
$i \leftarrow 1$
While (no convergence achieved) **do**
  Compute $d(\boldsymbol{x}|C_{i-1})$, for all $\boldsymbol{x} \in X$
  Re-order the points in X such that $d(\boldsymbol{x}_1|C_{i-1}) \geq \cdots \geq d(\boldsymbol{x}_n|C_{i-1})$
  $L_i \leftarrow \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l\}$
  $X_i \leftarrow X|L_i = \{\boldsymbol{x}_{l+1}, \ldots, \boldsymbol{x}_n\}$
  for $(j \in \{1, \ldots, k\})$ **do**
    $P_j \leftarrow \{\boldsymbol{x} \in X_i | c(\boldsymbol{x}|C_{i-1}) = \boldsymbol{c}_{i-1,j}\}$
    $\boldsymbol{c}_{i,j} \leftarrow \text{mean}(P_j)$
$C_i \leftarrow \{\boldsymbol{c}_{i,1}, \ldots, \boldsymbol{c}_{i,k}\}$
$i \leftarrow i + 1$

Figure 1: k-Means−− Pseudo-code

faster with large data sets due to memory consumption, scalability, parameter tuning, noise sensitivity, and difficulty handling high-dimensional data since it uses the KDTree to build the nearest neighbor tree.

2.3. DBSCAN ++

DBSCAN++ [19] is an extension to the DBSCAN, which runs much faster, more efficient, and less sensitive to hyper-parameter settings. We couldn't find any Python implementation of the DBSCAN++, so we used the pseudo-code provided [19] by using the KDTree scipy in the implementation. This implementation didn't allow us to run it on large data sets due to parameter tuning, noise sensitivity, and difficulty handling high-dimensional data because of the limitation of the KDTree implementation. The pseudo-code is shown in algorithm 3. There were two initialization methods mentioned in this paper.

1. Uniform initialization
2. k-center initialization

Uniform initialization was implemented by uniformly sampling m number of points from the given data set. We only ran KDTree queries for m sampled data points, after running the queries we developed the core point set and then we created the neighbourhood tree by adding edges to points in the radius of the core points.

k-center initialization was more complicated than the uniform initialization, we had to implement part of the greedy k-center clustering algorithm for this initialization. As mentioned in the paper [19], k-center initialization should run faster than DBSCAN algorithm on the same hyper-parameters, but this did not happen in our implementation, k-center initialization took considerable amount of time even though the time complexity was $O(mn)$. To improve the performance of this initialization, we vectorized the computation and used a slightly better algorithm mentioned in Geometric Approximation Algorithms [20]. We saw a massive improvement in initialization time but still, it took longer than DBSCAN algorithm. This happened because in both DBSCAN and DBSCAN++, we had to build the

ALGORITHM 2: DBSCAN

Inputs: X, ε, minPts
$C \leftarrow$core-points in X given ε and minPts
$G \leftarrow$ initialize empty graph
for $c \in C$ **do**
  Add an edge (and possibly a vertex or vertices) in G from c to all points in $X \cap B(c, \varepsilon)$
end for
return connected components of G.

Figure 2: DBSCAN Pseudo-code

ALGORITHM 3: DBSCAN+ +

Inputs: X, m, ε, minPts
$S \leftarrow$ sample m points from X
$C \leftarrow$ all core-points in S $w.r.t$ X, ε and minPts
$G \leftarrow$ empty graph
for $c \in C$ **do**
  Add an edge (and possibly a vertex or vertices) in G from c to all points in $X \cap B(c, \varepsilon)$
end for
return connected components of G.

Figure 3: DBSCAN++ Pseudo-code

KDTree which takes the same time if the input data set is the same. Whereas, in k-center initialization, we had to run the initialization algorithm to pick the m points. Although this m number of points are less than the total number of points, the speed up gain from running m number of queries against running queries for all the points does not exceed the time taken to run the k-center initialization. We suspect that if this was implemented in C++, there might be a difference in result. Since the paper did not mention about any implementation details we cannot be certain about this.

2.4. DBSCAN and DBSCAN++ on Approximate Nearest Neighbour (ANN)

Since we could not run DBSCAN or DBSCAN++ on large data sets, we moved on to implementing the approximate nearest neighbour on DBSCAN and DBSCAN++ algorithms. We used a Python library Annoy [21], which wrapped a C++ implementation of approximate nearest neighbour tree using Python. We saw a massive speed up in creating the nearest neighbour tree after implementing this. This library did not allow us to query the points given in a radius ball what it allowed us to do was to get the approximate k-nearest neighbours, this posed a challenge for us because we need to query the points given in a $\varepsilon - radius$ ball. So what we did was to query $2 * minpts$ number of points for each core point selection query, and check if there are more than $minpts$ number of points that has less than eps distance to the queried point. This allowed us to reduce the uncertainty of not picking all the points in the $\varepsilon - radius$ ball. Since the $minpts$ is a small value, going through $2 * minpts$ was not affecting the performance of the algorithm.

After completing the implementation of these algorithms, we ran experiments on the given data sets. Since we are interested in detecting anomalies, we plotted histograms of ground truth labels of each data sets. Then we decided what class labels are normal instances (expected behavior or pattern of the system or data being analyzed) and what class labels are noise instances (data points or events that deviate significantly from the expected or normal behavior). Some of the data sets already had document explaining what class labels can be identified as normal instances and what class labels are noise labels. Generally, if a class label had a less frequency, we picked them as noise labels. After running the algorithms, we modified the ground truth and cluster label arrays to only contain two class labels. 0 if a data point is a normal instance and 1 if data point is a noise instance. We did this because we are only interested in detecting normal and abnormal instances, we no longer care whether we have the right number of clusters as a result. Then we ran different assessment metrics on both the ground truth and the labels obtained from these algorithms. Explanation of these results are mentioned in the empirical evaluation section.

Also, we made a small modification to DBSCAN algorithm hoping to solve the problem of detecting small outlier clusters. What we did was we added a threshold parameter to the DBSCAN algorithm where it will check the size of clusters before assigning the cluster label, if the size of cluster is smaller than the given threshold, it was marked as a noise cluster. We only made this change to the DBSCAN on an Approximate Nearest Neighbor (ANN) implementation and we tested this on the shuttle data set. This has a very small change, but we got extremely good results for Shuttle data set. This part was done as an extension to what we already did. We could not test this algorithm for all the data sets. It was only tested on the Shuttle data set because it contained small outlier clusters and it was a large data set. We will explain the results in the discussion section.

3. Empirical Evaluation

3.1. Data sets

We perform our experiments on six data sets from UCI machine learning repository [22]. The data sets description and distribution of the classes is shown on the figures and table below:

We had to change the data sets proposed due to the fact that DBSCAN was unable to process large data sets.

3.2. EVALUATION METRICS

Four evaluation metrics were used to assess the validity of the results of this experiments.

1. False alarm rate
2. F-Score (weighted)
3. Jaccard coefficient
4. Adjusted rand Index

Table 1: DBSCAN results on chosen data sets

Data set	#points	#dim	#outliers	outlier %
pima	768	8	268	35
cardio	1831	21	176	9.60
wine	129	13	10	7.70
glass	214	9	9	4.20
breastw	683	9	239	35
shuttle	43500 (36752)	9	2644	7.19

False alarm rate is the ratio of number of incorrectly labelled noise instances that were normal instances in ground truth over total number of noise instances predicted.

The F- measure of a cluster is the harmonic mean of the precision and recall values of a cluster. We took the weighted F-measure values of each cluster as the final $F - score$. [23]

$$F_i = \frac{2}{\frac{1}{prec_i} + \frac{1}{recall_i}} = \frac{2 * prec_i * recall_i}{prec_i + recall_i}$$

$$F = \sum_{i=1}^{k} w_i * F_i$$

where w_i is the weight of the cluster

The Jaccard Coefficient measures the fraction of true positive point pairs, but after ignoring the true negatives. It is defined as follows: [23]

$$Jaccard = \frac{TP}{TP + FN + FP}$$

Before we use these metrics, we converted the ground truth and cluster labels to two classes containing normal and outlier classes. This helped us to focus more on noise prediction results rather than looking at cluster predictions. The adjusted rand index assessment is included but was not use in interpreting the result.

3.3. Empirical Results presented in the form of tables

3.4. Discussion of results

The factor parameter in DBSCAN++ is the number of point that were quarried by the algorithm. l and k parameters in the k-means−− results indicates the outlier parameter and cluster number parameter. FAR means False Alarm Rate, ARS means Adjusted Rand Score and JS means Jaccard Score.

Each data set's result will be described separately and will make conclusions based on the entire results. For each data set, we experimented with various parameters, and we included a range of parameters and their results in the empirical result tables. The density-based algorithms (DBSCAN and DBSCAN++) did not perform well on the *Pima* data set. We only included the parameter and best possible result for the *Pima* data set. In most cases, it either recognized all the points as normal instances or outliers. We believe this was as a result of the high percentage of outliers in the *Pima* data set, which

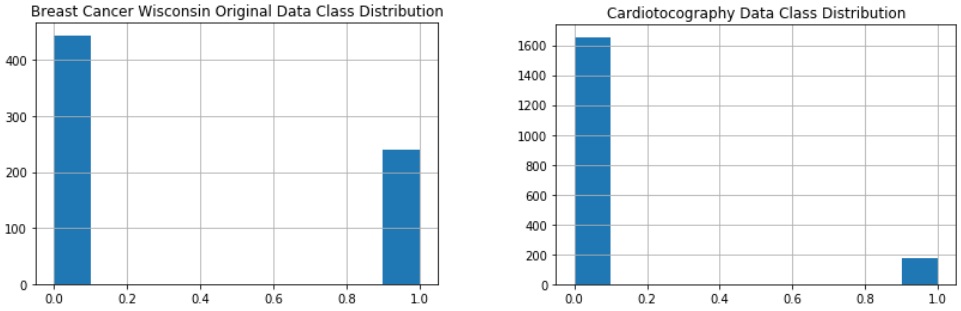

Figure 4: Breast cancer and Cardiotography data sets class distribution

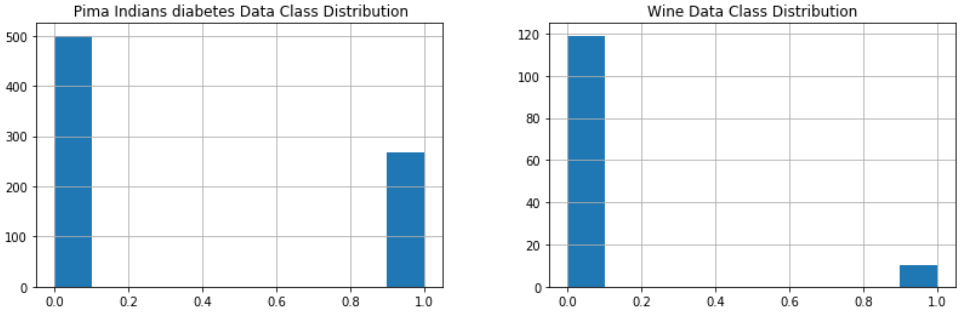

Figure 5: Pima and Wine data set class distribution

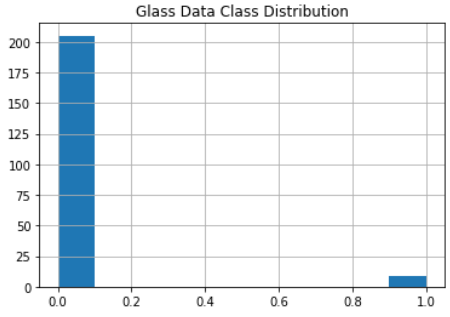

Figure 6: Glass Data Set Class distribution

is 35% as shown in table 1. We could not find the balance between the *minpoints* and *epsilon* values that can distinguish the clusters and the noise points. This is because this data set has different densities points at different levels. When we ran k-means−− on *Pima* data set, we got slightly better results (table 4), and we had a smaller false alarm rate value of 0.4216 as opposed to 0.651 obtained from DBSCAN algorithm. In the case of DBSCAN or DBSCAN++ tests, more than half of the detected noise points are false positive values indicating that *Pima* data set does not consist of a density-based cluster structure. Low F-score of 0.185 in both DBSCAN and DBSCAN++ tests indicates that quality of instances detected as normal, and outlier cluster are low. Jaccard coefficient of 0.349 indicates that false negatives and false positive value pairs are high compared to true positive pairs. When we look at the k-means−− results of *Pima* data set, we have better Jaccard coefficient and f1 score 0.4068 and 0.7057 respectively (table 4). Even though k-means−− results are better than DBSCAN and DBSCAN++ results, the results are not accurate enough. Then we employed statistical techniques such as dimensionality reduction (PCA) on the *Pima* data set before using the above mentioned algorithms, which resulted in better results. We first normalize the data, and then applied PCA on the *Pima* data set. We then ran DBSCAN and DBSCAN++ algorithms on the data set after selecting the best 7 components from the results. For the results refer the table A5. We had to increase the min points parameters to 270 − 290 range to get better results. This is because the data set only contains one major cluster and all the other points are considered outliers. There are 268 outliers in the data set, thus we had to bring the *minpoints* to 270 range to exclude outliers from the result. This resulted in better outputs. We could achieve a false alarm rate of 0.45 at $eps = 0.35$ with $minpts = 270$ and F-score of 0.69 on DBSCAN algorithm. We could achieve similar results for DBSCAN++ on both initial-

Table 2: DBSCAN results on chosen data sets

Data sets	Parameters		Evaluation Measures			
	min pts	epsilon	f1 score	False alarm	Adjusted Rand Index	Jaccard
Pima	20	5	0.1805	0.651	0	0.349
Cardio	10	3	0.88	0.58	0.31	0.26
	10	4	0.88	0.43	0.19	0.14
	10	**5**	**0.88**	**0.15**	**0.17**	**0.12**
Wine	4	25	0.91	0.58	0.46	0.41
	4	30	0.95	0.41	0.664	0.58
	4	35	0.97	0.23	0.83	0.76
	4	38	0.98	0.16	0.88	0.83
	4	**72**	**0.95**	**0.14**	**0.65**	**0.54**
Glass	8	0.9	0.8736	0.8085	0.206	0.1915
	8	1	0.8767	0.8043	0.213	0.1957
	8	1.2	0.86	0.875	0.11	0.11
	8	2	0.9	0.88	0.08	0.08
BreastW	10	2.9	0.9508	0.119	0.8098	0.8745
	10	**4**	**0.9636**	**0.072**	**0.8579**	**0.9027**
	10	5	0.82	0.05	0.429	0.54
	10	5.5	0.74	0.03	0.26	0.36
	20	5	0.9225	0.0483	0.7143	0.7912

Table 3: DBSCAN++ k-center results on data sets

Data sets	Parameters			Evaluation Measures			
	min pts	factor	epsilon	f1 score	False alarm	Adjusted Rand Index	Jaccard
Pima	10	0.2	0.9	0.1805	0.651	0	0.349
	10	**0.35**	**2**	**0.6797**	**0.3396**	**0.1177**	**0.4136**
	20	0.2	5	0.1805	0.651	0	0.349
Cardio	**10**	**0.5**	**5**	**0.8835**	**0.1538**	**0.1788**	**0.1222**
	10	0.5	4	0.88	0.42	0.2	0.15
	10	0.5	3	0.88	0.59	0.3	0.27
Wine	4	0.3	25	0.84	0.72	0.25	0.27
	4	0.3	30	0.9	0.6	0.43	0.4
	4	0.3	35	0.92	0.54	0.51	0.45
	4	0.3	38	0.97	0.28	0.78	0.71
	4	**0.3**	**72**	**0.95**	**0.14**	**0.65**	**0.54**
	10	2	1.5	0.0112	0.9225	0	0.0775
Glass	8	0.5	0.4	0.41	0.94	-0.04	0.05
	8	0.5	0.9	0.87	0.8	0.2	0.19
	8	0.5	1	0.87	0.8	0.21	0.19
	8	0.5	1.2	0.86	0.87	0.11	0.11
	8	0.5	2	0.8885	0.8485	0.1499	0.1351
BreastW	10	0.5	1.9	0.8812	0.2578	0.5723	0.7422
	10	0.5	2.9	0.9509	0.1218	0.8098	0.875
	10	**0.5**	**4**	**0.96**	**0.07**	**0.85**	**0.89**
	10	0.5	5	0.82	0.05	0.42	0.54
	10	0.5	5.5	0.74	0.03	0.26	0.36
	20	1	4	0.9607	0.0794	0.847	0.8958
	20	1	5	0.9634	0.0542	0.8576	0.9008

izations at 0.5 factor values. Then we ran the k-means$--$ on the data set (dimensionally reduced), we got better results compared to DBSCAN results. Refer the table A4. It shows at $k = 1$ and $l = 268$, we get the lowest false alarm rate of 0.45 and highest $F1$-score of 0.68 for this data set. However, on both algorithms there are considerable amount of false positive noise predictions. And the Jaccard coefficients of both algorithms for this data set is low as well, which indicates that there are false

Table 4: k means −− results on data set

Data sets	Parameters				Evaluation Measures			
	k	l	iteration	epsilon	f1 score	False alarm	Adjusted Rand Index	Jaccard
Pima	2	268	10	0.2	0.7057	0.4216	0.1614	0.4068
Cardio	2	176	10	0.05	0.9312	0.358	0.5536	0.4728
	2	176	20	0.05	0.9345	0.3409	0.5734	0.4915
	3	**176**	**20**	**0.05**	**0.9421**	**0.3011**	**0.6201**	**0.5371**
Wine	2	10	10	0.2	0.9845	0.1	0.8753	0.8182
	2	**10**	**10**	**0.05**	**0.9845**	**0.1**	**0.8753**	**0.8182**
Glass	2	9	10	0.05	0.9252	0.8889	0.0658	0.0588
BreastW	**2**	**239**	**10**	**0.2**	**0.9444**	**0.0795**	**0.7879**	**0.8527**
	2	239	10	0.05	0.9239	0.1088	0.716	0.8038
	2	239	10	0.35	0.9444	0.0795	0.7879	0.8527
	3	239	10	0.05	0.9209	0.113	0.706	0.797

Table 5: DBSCAN Result on ANN using Shuttle data set

Data set	Parameters			Evaluation Measures			
	minpts	eps	factor	f-score	False alarm rate	ARS	Jaccard score
Shuttle	10	4.5	1	0.88758301	0.775670841	0.146802475	0.130299252
	10	4.8	1	0.89255751	0.757217848	0.149587292	0.126857143
	10	5	1	0.89574317	0.740279938	0.150095945	0.123035363
	10	5.3	1	0.90068313	0.695989651	0.162924769	0.126344086
	10	5.5	1	0.90195679	0.679245283	0.162242457	0.123463687
	10	5.8	1	0.90298582	0.659681475	0.158728371	0.118332848
	10	6	1	0.90418584	0.628742515	0.156338474	0.11362248
	10	6.8	1	0.90566123	0.566360053	0.153339959	0.107317073
	10	7	1	0.90567793	0.550143266	0.149568631	0.103698811
	10	**9**	**1**	**0.90520426**	**0.504621072**	**0.135841986**	**0.091875214**
	10	**10**	**1**	**0.90441125**	**0.507936508**	**0.126851763**	**0.085517241**
	10	28	1	0.89643437	0.655462185	0.041910848	0.029285714
	10	28.5	1	0.89647958	0.65106383	0.042092565	0.029317125

Table 6: Result of Shuttle data set on $k-$means −−

Data set	Parameters			Evaluation Measures				
	k	outliers	iterations	f1 score	FAR	ARS	JS	time
Shuttle	1	2500	50	0.96890	0.08634	0.72703	0.62157	55
	1	**2644**	**50**	**0.97034**	**0.09082**	**0.74045**	**0.63810**	**45**
	1	2700	50	0.97089	0.09704	0.74599	0.64520	35
	2	2500	50	0.97169	0.07394	0.75167	0.65113	149
	2	**2644**	**50**	**0.97226**	**0.08608**	**0.75763**	**0.65910**	**128**
	2	2700	50	0.96234	0.25512	0.68565	0.58323	56

negative pairs in the result. Slightly higher F-scores indicates that quality of the clustering is much better.

The second data set we experimented on was the Cardio data set, which also performed well on k-means−− algorithm. This data set contains about 9.60% outliers. On DBSCAN, Cardio data set gives about $0.8 - 0.9$ F-score, which means that the quality of clusters is high. Note that we used weighted F-score. False alarm rate of both DBSCAN and DBSCAN++ algorithms were around 0.15, indicating a low false positive noise points in the predicted labels. Jaccard coefficient results of the DBSCAN algorithm produced less-than-ideal results, with 0.12 where DBSCAN++ gave around 0.15. This happened because of high false negative value pairs. We know that we have low number of FP due to lower false alarm rate, so we can conclude that we have low Jaccard coefficient because of false negative pairs, which means DBSCAN could not identify when two points are in different groups in the data set. It should be noted that DBSCAN++ has a higher F-score and a lower false alarm rate just like DBSCAN. Since we took the weighted F-score and there are higher number of normal instances in the Cardio data set, we can conclude that normal point prediction accuracy is high. We can also conclude that true positives of predicting

Table 7: Result of Shuttle data set using Modified DBSCAN on ANN

Data set	Parameters				Evaluation Measures				
	minpts	eps	factor	threshold	f1 score	FAR	ARS	JS	#noise pts
Shuttle	10	4.5	1	0.2	0.61831	0.87032	-0.00768	0.12968	27136
	10	4.8	1	0.2	0.93744	0.50524	0.56876	0.49476	9603
	10	5	1	0.2	0.96127	0.37524	0.70772	0.62476	8210
	10	5.3	1	0.2	0.96998	0.31289	0.76610	0.68711	7797
	10	5.5	1	0.2	0.97280	0.29058	0.78586	0.70942	7660
	10	5.8	1	0.2	0.97900	0.23716	0.83098	0.76284	7392
	10	6	1	0.2	0.98369	0.19243	0.86654	0.80757	6875
	10	6.8	1	0.2	0.98868	0.14016	0.90574	0.85984	6648
	10	7	1	0.2	0.98988	0.12682	0.91537	0.87318	6584
	10	9	1	0.2	0.99274	0.09359	0.93872	0.90641	6444
	10	**10**	**1**	**0.2**	**0.99313**	**0.08834**	**0.94191**	**0.91103**	**6414**
	10	28	1	0.2	0.89643	0.65546	0.04191	0.02929	263
	10	28.5	1	0.2	0.89648	0.65106	0.04209	0.02932	260

normal instances as normal instances is high with this result.

k-means−− result of the Cardio data set has best F-score of 0.94 and 0.30 false alarm rate (lowest of *k*-means−− tests for cardio data set) and Jaccard coefficient of 0.5 (table 4). F-score and Jaccard coefficients are better than the density-based results, although we had higher false alarm rate than the density level results, which indicate that out of the noise points predicted, there were high number of false positives. However, the Jaccard coefficient was high for this test, indicating that false negatives pairs are low in the *k*-means−− result. As a result of these high F-score values, we can conclude that if the quality of two clusters are high, then true positive numbers should be high as well. Then this higher Jaccard coefficient should have come from the low false negative pairs. This indicates that *k*-means−− was good at identifying pairs of points that are in different clusters but it was not good at identifying some normal instances as normal instances.

Our third data set was the Wine data set. Both density-based and representative-based algorithms performed well on this data set. DBSCAN and DBSCAN++ algorithms gave best F-scores around 0.98 and best false alarm rates around 0.14 and best Jaccard coefficients of 0.83 (table 2, table 3 and table A1). Note that the best Jaccard coefficient came from uniform initialization of DBSCAN++. The best value for the DBSCAN++ *k*-center was 0.71. *k*-means−− algorithm gave the best results for this data set in terms of all the assessment metrics. F-score of 0.98, false alarm rate of 0.14 and Jaccard coefficient of 0.81 (table 4).

Glass data set was our fourth data set. On both types of algorithms (representative-based and the density-based), they were able to identify the noise instances as noise instances but there were lot of false positives. Both algorithms types gave more than 0.8 F-score, indicating that the clustering is of high quality. But both types of algorithms had very high false alarm rates, which means algorithms classified normal instances as noise instances. Both result types had low Jaccard coefficients as well, which occurred due to high false positives and false negatives. This occurred due to classifying pairs of normal instances in two different clusters. This shows that both algorithms could not identify the anomalies correctly.

The fifth data set is the BreastW data set, which has around 35% of outliers. Both types of algorithms performed very well on this data set. Both had very low false alarm rates and high F-scores, and Jaccard coefficients, which indicates that algorithms were able to predict the anomalies accurately. BreastW data set has a Gaussian-based and density-based cluster structure, which helped the algorithms to identify the cluster structures more accurately.

However, we could not run our DBSCAN or DBSCAN++ implementations on large data sets because of the KDTree limitations. As a result, we used an approximate nearest neighbour library to query the nearest neighbours. We tested this on Shuttle data set, which has 43500 data points. There are 7 ground truth class labels in the data set. Class label 1 has the highest frequency, all the other classes has lower frequencies compared the class 1. We removed data with class label 4 and considered all other classes except class 1 as outliers. The important thing about this data set is that its outliers are in small clusters. For example, class 2, 3, 5, 6, 7 are outlier classes. If those outlier classes have different densities, such as lower densities, density-based algorithms cannot detect those outliers by tweaking the *minpoint* and $epsilon$ parameters. Please refer table 5. With $eps = 9$, we had the lowest false alarm rate, and then it increases. This is because outliers form small clusters and density-based algorithms cannot find it by tweaking the parameters due to breaking of cluster structure. Out of DBSCAN and DBSCAN++ algorithms, DBSCAN on ANN performed well, this is because we are only querying a part of data points to find the core points. Thus, some core points that are identified in the DBSCAN are no longer identified as a core point, thus we would get a higher false positive noise points, which is why we get high false alarm rate for the uniform and *k*-center initialization (Refer tables A2 and A3). We also ran this on *k*-means−− algorithm, and it gave us excellent results on this data set. Not only it took less time, but the resultant clusters were of higher quality. In table 6, we have very low false alarm rates and high

F-score. We even changed the input k value to the algorithm and checked the results, even if we input a higher cluster value, we still get the correct numbers of outliers, with good results. We even changed the range of outlier numbers input to the parameter and algorithm seem robust even if we slightly increase or decrease the number of outliers.

Next, we modified the DBSCAN on ANN algorithm as mentioned in 2.4 then we ran the algorithm on Shuttle data set. We got extremely good results (refer table 7). We changed the threshold to 0.2 because the Shuttle data set only has 1 class and it takes about 80% of the data. We ran the experiments on a wide range of eps values from 4.5 to 28.5. The results we obtained were better. We got best F-score of 0.99 and False alarm rate of 0.88, and Jaccard score of 0.91. This accuracy is better than the k-means results. The best thing about this is that we only need to know the percentage of outliers in the data set. We do not need the number of clusters in the data to get a better result. However, we ran this on normalized and dimensionality reduced *Pima* data set, hoping to see better results, but we did not obtain better results, but they were very close to DBSCAN results. Thus, we did not include the results in this paper.

One of the interesting observation that was identified in the results is that k-center initialization of DBSCAN takes more time to run than the normal DBSCAN instance on the same parameters, even for a small factor value. This contradicts with the results shown in [19], where they showed that k-center initialization runs faster than normal DBSCAN. However, we did not see this in our results. It seems that speed up gained from running fewer KDTree queries does not compensate the time that it takes to initialize k-center points. We also implemented a slightly better k-center initialization algorithm mentioned in [20] and improved the calculations by vectorizing. Still, time taken to run the DBSCAN++ on k-center is greater than DBSCAN on same parameters. However, DBSCAN++ on uniform initialization ran faster than all the other algorithms.

4. Conclusions and Future Directions

From the results obtained from these experiments, we can conclude that k-means$--$ is a more robust algorithm than DBSCAN or DBSCAN++ algorithms in terms of time and performance, especially when data sets have small outlier clusters with different densities. Density-based algorithms struggle to find the outliers, especially if the small outlier clusters have higher densities than the normal clusters, it becomes challenging to tweak the DBSCAN hyper-parameters. k-means$--$ algorithm seems more robust in this case; however, we need to know the number of clusters and outlier percentage beforehand to get better results. Nonetheless, k-means$--$ has shown to be robust to slight changes in input parameters. Refer tables A4 and 6. The modification of DBSCAN algorithm with approximate nearest neighbour implementation worked very well in terms of time. We could also improve the k-center initialization a little bit more by paralleling the k-center initialization, which can be a good future direction in terms of improving the running time. Although k-means$--$ is robust, we can create synthetic data sets that would not work very well on k-means$--$ by adding non-Gaussian-shaped clusters and adding noise points. The problem with density-based algorithms to find noise points is that it is hard for the density-based algorithms to identify small outlier clusters, but we can change this by modifying the DBSCAN algorithm. We need to add another parameter (say "t") that will act as a threshold for determining a small outlier cluster. At the end of the DBSCAN algorithm, when we go through the connected components, we need to check the number of nodes in these connected components, if the fraction of number of nodes in these connected components is less than this threshold, we can identify these nodes as a outlier cluster. By making this modification, we can overcome this weakness in density-based algorithms. We already made this change and tested on a data set with oultlier clusters which resulted in extremely good results. However, we could not run this modified algorithm on all the data sets because of time limitations, we believe testing this modified algorithm will be a good future direction. The weakness of k-means$--$ is that we need to have an understanding about the cluster structure to get accurate result, but we believe by making this change to DBSCAN we could have a robust algorithm than the k-means$--$. Another proposed change will be to run in polynomial time given that we only have to go through the connected components to find the size of it; if we improve this graph data structure we should be able to this in constant time. These are some good future directions that we can use. We still believe density-based algorithm should be more powerful than representative-based methods, but we need to make some modifications to these algorithms to make it better. And also find how these algorithms can perform against unsupervised learning (like Isolation Forest) [24] and semi-supervised learning (One-Class Support Vector) [25] based outlier detection methods as well. Also, we should look at how k-nearest neighbour-based methods perform against these algorithms. Another future work area would be to find how we can use time series data on density-based algorithm to find the outliers.

Acknowledgments

We thank the reviewers for the positive enlightenment, comments, and suggestions, which have greatly helped us in making improvements to this paper.

References

[1] S. M. Shagari, D. Gabi, N. M. Dankolo & N. N. Gana, "Countermeasure to Structured Query Language Injection Attack for Web Applications using Hybrid Logistic Regression Technique", Journal of the Nigerian Society of Physical Sciences **4** (2022) 832. https://doi.org/10.46481/jnsps.2022.832

[2] C. L. Udeze & I. E. Eteng & A. E. Ibor, "Application of Machine Learning and Resampling Techniques to Credit Card Fraud Detection", Journal of the Nigerian Society of Physical Sciences **4** (2022) 3769. https://doi.org/10.46481/jnsps.2022.769

[3] K. Oshinubi, A. Amakor, O. J. Peter, M. Rachdi & J. Demongeot, "Approach to COVID-19 time series data using deep learning and spectral analysis methods[J]", AIMS Bioengineering **9** (2022) 1.https://www.aimspress.com/article/doi/10.3934/bioeng.2022001.

[4] V. Chandola, A. Banerjee & V. Kumar, "Anomaly Detection: A Survey", ACM computing surveys (CSUR), ACM New York, NY, USA **41** (2009) 1. https://doi.org/10.1145/1541880.1541882

[5] P. O. Odion & M. N. Musa, & S. U. Huaibu, "Age Prediction from Sclera Images using Deep Learning", Journal of the Nigerian Society of Physical Sciences **4** (2022) 787. https://doi.org/10.46481/jnsps.2022.787

[6] Z. He, X. Xu & S. Deng, "Discovering Cluster Based Local Outliers", Pattern Recogn. **24** (2003) 1641

[7] Z. Li, Y. Zhao, N. Botta, C. Ionescu & X. Hu, "COPOD: Copula-Based Outlier Detection.", Pattern Recogn. **24** (2020) 9.

[8] R. J. G. B. Campello, D. Moulavi, A. Zimek J. Sander, "Hierarchical density estimates for data clustering, visualization, and outlier detection", ACM Transactions on Knowledge Discovery from Data (TKDD), ACM New York, NY, USA **10** (2015) 1.

[9] S. Hariri, M. C. Kind & R. J. Brunner, "Extended isolation forest", IEEE Transactions on Knowledge and Data Engineering **44** (2019) 4.

[10] P. Guo, W. Lijuan, S. Jun & F. Dong, "A hybrid unsupervised clustering-based anomaly detection method", Tsinghua Science and Technology **26** (2020) 146.

[11] Y. Zhang, "DBSCAN Clustering Algorithm Based on Big Data Is Applied in Network Information Security Detection", Security and Communication Networks **2022** (2022) 9951609.

[12] G. Du, X. Li, L. Zhang, L. Liu & C. Zhao, "Novel Automated K-means++ Algorithm for Financial Data Sets", Mathematical Problems in Engineering **2021** (2021) 1.

[13] T. Srikanth, B. Philip, J. Jiong & S. Jugdutt, "A comprehensive survey of anomaly detection techniques for high dimensional big data", Journal of Big Data **7** (2020) 1.

[14] W. Wang, X. Hu & Y. Du, "Algorithm optimization and anomaly detection simulation based on extended Jarvis-Patrick clustering and outlier detection", Alexandria Engineering Journal **61** (2022) 2106.

[15] W. Wang, X. Hu & Y. Du, "Algorithm optimization and anomaly detection simulation based on extended Jarvis-Patrick clustering and outlier detection", Alexandria Engineering Journal **61** (2022) 2106.

[16] T. Chandrakala, & S. N. S. Rajini, "An Analysis of Outlier Detection through clustering method", International Journal of Advanced Engineering, Management and Science **6** (2020) 571.

[17] S. Chawla & G. Aristides, "K-means-: A unified approach to clustering and outlier detection", Proceedings of the 2013 SIAM International Conference on Data Mining (SDM) (2013) 189.

[18] J. Han, M. Kamber & J. Pei, *Data Minig: Concepts and Techniques* , Third Edition, pp. 471–476.

[19] J. Jang & H. Jiang, "DBSCAN++: Towards fast and scalable density clustering", Proceedings of Machine Learning Research (PMLR) **97** (2019) 3019.

[20] S. Har-Peled, *Geometric Approximation Algorithms*, American Mathematical Society, 2011.

[21] E. Bernhardsson, *spotify/annoy: v1.17.0* . https://github.com/spotify/annoy

[22] D. Dheeru & G. Casey, "UCI Machine Learning Repository", University of California, Irvine (2017). http://archive.ics.uci.edu/ml

[23] M. J. Zaki& W. Meira, *Data Mining and Machine Learning: Fundamental Concepts and Algorithms*, Cambridge University Press, 2020.

[24] F. T. Liu & K. M. Ting, & Z. H. Zhou, *Isolation forest*, Eighth IEEE International Conference on Data Mining, 2008.

[25] L. M. Manevitz & M. Yousef, "One-class SVMs for document classification",Journal of machine Learning research **2** (2011) 139.

Table A1: DBSCAN++ uniform results on the dataset

Dataset	Parameters			Evaluation Measures			
	min pts	factor	epsilon	f1 score	False alarm	Adjusted Rand Index	Jaccard
Pima	10	0.2	0.9	0.1805	0.651	0	0.349
	10	**0.35**	**2**	**0.6797**	**0.3396**	**0.1177**	**0.4136**
	20	0.2	5	0.1805	0.651	0	0.349
Cardio	10	0.5	3	0.88	0.6	0.3	0.27
	10	0.5	4	0.88	0.46	0.2	0.15
	10	**0.5**	**5**	**0.88**	**0.17**	**0.18**	**0.12**
Wine	4	0.3	25	0.78	0.78	0.14	0.21
	4	0.3	30	0.9	0.6	0.43	0.4
	4	0.3	35	0.89	0.61	0.41	0.38
	4	0.3	38	0.97	0.23	0.83	0.76
	4	**0.3**	**72**	**0.98**	**0.16**	**0.88**	**0.83**
Glass	8	0.5	0.4	0.55	0.93	-0.02	0.06
	8	0.5	0.9	0.83	0.85	0.13	0.15
	8	0.5	1.2	0.87	0.8	0.21	0.19
	8	0.5	2	0.86	0.87	0.11	0.11
BreastW	10	0.5	2.9	0.94	0.12	0.79	0.84
	10	0.5	4	0.96	0.07	0.84	0.89
	10	0.5	5	0.84	0.06	0.49	0.6
	10	**0.5**	**5.5**	**0.77**	**0.05**	**0.33**	**0.44**
	20	1	4	0.9607	0.0794	0.847	0.8958
	20	1	5	0.9634	0.0542	0.8576	0.9008

Table A2: Result of Shuttle data set on DBSCAN++ on ANN with uniform initialization

Dataset	Parameters			Evaluation Measures				
	minpts	eps	factor	f-score	FAR	ARS	JS	#noise recnzd
Shuttle	10	4.5	0.1	0.7798	0.87893	0.05142	0.10694	13234
	10	4.8	0.1	0.7947	0.87076	0.06308	0.11277	12169
	10	5	0.1	0.7961	0.87803	0.05530	0.10488	11582
	10	5.3	0.1	0.8028	0.88439	0.04908	0.09699	10638
	10	**5.5**	**0.1**	**0.8134**	**0.86978**	**0.06712**	**0.10918**	**10154**
	10	**5.8**	**0.1**	**0.8186**	**0.87152**	**0.06558**	**0.10584**	**9673**
	10	6	0.1	0.8188	0.87662	0.05956	0.10065	9395
	10	6.8	0.1	0.8180	0.89827	0.03413	0.08005	8597
	10	7	0.1	0.8261	0.87291	0.06435	0.10194	8553
	10	9	0.1	0.8272	0.89449	0.03868	0.08094	7713
	10	10	0.1	0.8285	0.90223	0.02960	0.07354	7398
	10	28	0.1	0.8226	0.92128	0.00768	0.05852	7255
	10	28.5	0.1	0.8203	0.92319	0.00551	0.05739	7575

Table A3: Result of Shuttle data set on DBSCAN++ on ANN with KCENTER initialization

Dataset	**Parameters**			**Evaluation Measures**				
	minpts	**eps**	**factor**	**f-score**	**FAR**	**RS**	**JS**	**#noise pts rcgzd**
Shuttle	10	4.5	0.1	0.63363	0.92146	0.00109	0.07327	20717
	10	4.8	0.1	0.66854	0.92519	0.00127	0.06839	18291
	10	5	0.1	0.68213	0.93147	-0.00188	0.06170	16989
	10	5.3	0.1	0.70901	0.93335	-0.00383	0.05880	14934
	10	5.5	0.1	0.72200	0.93444	-0.00510	0.05716	13929
	10	5.8	0.1	0.73162	0.93805	-0.00852	0.05329	13025
	10	**6**	**0.1**	**0.73937**	**0.94119**	**-0.01173**	**0.04999**	**12320**
	10	6.8	0.1	0.75174	0.95081	-0.02170	0.04069	11051
	10	7	0.1	0.75317	0.95335	-0.02429	0.03839	10832
	10	9	0.1	0.75998	0.96095	-0.03254	0.03151	10029
	10	10	0.1	0.76010	0.96276	-0.03435	0.02998	9941
	10	28	0.1	0.75842	0.98063	-0.05189	0.01525	9609
	10	28.5	0.1	0.75853	0.98061	-0.05190	0.01526	9603

Table A4: Result of Pima data set on k-means−− after running PCA and selecting 7 Components

Dataset	**Parameters**			**Evaluation Measures**				
	k	**Iterations**	**l (outliers)**	**f1 score**	**FAR**	**ARS**	**JS**	**type**
Pima	1	50	250	0.6559	0.4880	0.0899	0.3282	k-means−−
	1	**50**	**268**	**0.6823**	**0.4552**	**0.1247**	**0.3744**	**k-means−−**
	1	**50**	**275**	**0.6923**	**0.4436**	**0.1393**	**0.3923**	**k-means−−**
	2	50	250	0.6743	0.4600	0.1147	0.3525	k-means−−
	2	**50**	**268**	**0.6979**	**0.4328**	**0.1487**	**0.3958**	**k-means−−**
	2	50	275	0.6534	0.4982	0.0851	0.3407	k-means−−
	3	50	250	0.7163	0.3960	0.1815	0.4114	k-means−−
	3	**50**	**268**	**0.6823**	**0.4552**	**0.1247**	**0.3744**	**k-means−−**
	3	50	275	0.6689	0.4764	0.1053	0.3609	k-means−−

Table A5: Result of DBSCAN and DBSCAN++ algorithms on Pima dataset after running PCA with 7 Components

Dataset	Parameters			Evaluation Measures				
	minpts	eps	factor	f-score	FAR	RS	JS	Type
Pima	270	0.5	1	0.5377	0.5806	0.0068	0.0455	DBSCAN
	270	0.4	1	0.6566	0.4425	0.1001	0.2812	DBSCAN
	270	**0.35**	**1**	**0.6901**	**0.4554**	**0.1342**	**0.4064**	**DBSCAN**
	270	0.3	1	0.1805	0.6510	0.0000	0.3490	DBSCAN
	270	0.2	1	0.1805	0.6510	0.0000	0.3490	DBSCAN
	270	0.1	1	0.1805	0.6510	0.0000	0.3490	DBSCAN
	280	0.5	1	0.5430	0.5455	0.0110	0.0524	DBSCAN
	280	0.4	1	0.6617	0.4350	0.1066	0.2899	DBSCAN
	280	0.35	1	0.6973	0.4532	0.1441	0.4330	DBSCAN
	280	0.3	1	0.1805	0.6510	0.0000	0.3490	DBSCAN
	280	0.2	1	0.1805	0.6510	0.0000	0.3490	DBSCAN
	280	0.1	1	0.1805	0.6510	0.0000	0.3490	DBSCAN
	290	0.5	1	0.5456	0.5294	0.0131	0.0559	DBSCAN
	290	**0.4**	**1**	**0.6684**	**0.4301**	**0.1145**	**0.3046**	**DBSCAN**
	290	0.35	1	0.6821	0.4767	0.1207	0.4469	DBSCAN
	290	0.3	1	0.1805	0.6510	0.0000	0.3490	DBSCAN
	290	0.2	1	0.1805	0.6510	0.0000	0.3490	DBSCAN
	290	0.1	1	0.1805	0.6510	0.0000	0.3490	DBSCAN
	270	0.5	0.5	0.5433	0.5676	0.0095	0.0554	Initialization.UNIFORM
	270	0.4	0.5	0.6613	0.4513	0.1030	0.3006	Initialization.UNIFORM
	270	**0.35**	**0.5**	**0.6937**	**0.4540**	**0.1391**	**0.4185**	**Initialization.UNIFORM**
	270	0.3	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.UNIFORM
	270	0.2	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.UNIFORM
	270	0.1	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.UNIFORM
	280	0.5	0.5	0.5692	0.4348	0.0323	0.0903	Initialization.UNIFORM
	280	0.4	0.5	0.6692	0.4439	0.1126	0.3175	Initialization.UNIFORM
	280	0.35	0.5	0.6859	0.4724	0.1262	0.4487	Initialization.UNIFORM
	280	0.3	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.UNIFORM
	280	0.2	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.UNIFORM
	280	0.1	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.UNIFORM
	290	0.5	0.5	0.5451	0.5641	0.0104	0.0586	Initialization.UNIFORM
	290	**0.4**	**0.5**	**0.6770**	**0.4242**	**0.1250**	**0.3239**	**Initialization.UNIFORM**
	290	0.35	0.5	0.6833	0.4755	0.1226	0.4491	Initialization.UNIFORM
	290	0.3	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.UNIFORM
	290	0.2	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.UNIFORM
	290	0.1	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.UNIFORM
	270	0.5	0.5	0.5458	0.5526	0.0116	0.0588	Initialization.KCENTRE
	270	0.4	0.5	0.6613	0.4513	0.1030	0.3006	Initialization.KCENTRE
	270	**0.35**	**0.5**	**0.6821**	**0.4762**	**0.1207**	**0.4420**	**Initialization.KCENTRE**
	270	0.3	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.KCENTRE
	270	0.2	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.KCENTRE
	270	0.1	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.KCENTRE
	280	0.5	0.5	0.5468	0.5610	0.0113	0.0619	Initialization.KCENTRE
	280	0.4	0.5	0.6618	0.4518	0.1034	0.3025	Initialization.KCENTRE
	280	**0.35**	**0.5**	**0.6183**	**0.5280**	**0.0469**	**0.4359**	**Initialization.KCENTRE**
	280	0.3	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.KCENTRE
	280	0.2	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.KCENTRE
	280	0.1	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.KCENTRE
	290	0.5	0.5	0.5468	0.5610	0.0113	0.0619	Initialization.KCENTRE
	290	**0.4**	**0.5**	**0.6612**	**0.4550**	**0.1021**	**0.3036**	**Initialization.KCENTRE**
	290	0.35	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.KCENTRE
	290	0.3	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.KCENTRE
	290	0.2	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.KCENTRE
	290	0.1	0.5	0.1805	0.6510	0.0000	0.3490	Initialization.KCENTRE