Highlights

SDCOR: Scalable Density-based Clustering for Local Outlier Detection in Massive-Scale Datasets

Sayyed-Ahmad Naghavi-Nozad,Maryam Amir Haeri,Gianluigi Folino

- We introduce a novel scalable density-based clustering approach for local outlier detection in massive data.
- The input data is processed in chunks, hence, there is no need for it to be entirely loaded in memory.
- Our assessments prove that the proposed method has a linear time complexity with a low constant.
- Experiments on real and synthetic data show that our proposed approach is more effective and efficient than traditional density-based methods which require the data to be residing in memory at the first place.

SDCOR: Scalable Density-based Clustering for Local Outlier Detection in Massive-Scale Datasets

Sayyed-Ahmad Naghavi-Nozad[a,*], Maryam Amir Haeri[a] and Gianluigi Folino[b]

[a]Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran
[b]ICAR-CNR, Via P.Bucci 7/C, Univ. della Calabria 87036 Rende (CS), Italy

ARTICLE INFO

Keywords:
Local outlier detection
massive-scale datasets
scalable
density-based clustering
anomaly detection

ABSTRACT

This paper presents a batch-wise density-based clustering approach for local outlier detection in massive-scale datasets. As opposed to well-known traditional algorithms which assume that the data is memory-resident, our proposed method is scalable and processes the data chunk-by-chunk within the confines of a limited memory buffer. A temporary clustering model is built at the first phase, and will be updated due to consecutive memory-loads of points. At the end of scalable clustering, the approximate structure of original clusters is obtained. Subsequently, by another scan of the entire dataset and using a suitable criterion, the proposed algorithm will give each object an outlying score, as we call it the SDCOR (Scalable Density-based Clustering Outlierness Ratio). Our evaluations on real-life and synthetic datasets demonstrate that our proposed method has a low linear time complexity and is more effective and efficient compared to best-known conventional density-based methods which need to observe the whole data at once.

1. Introduction

Outlier detection, which is a noticeable and open line of research [9, 20, 43], is a fundamental issue in data mining. Outliers refer to those rare objects that deviate from the well-defined notions of expected behavior and finding them is sometimes compared with searching for a needle in a haystack, because the rate of their occurrence is much smaller than that of normal objects. Outliers often interrupt the learning procedure from data for most of analytical models and thus, capturing them is very important, because it can enhance model accuracy and reduce computational loads of the algorithm. But outliers are not always annoying and sometimes, they become special interest of the data analyst, such as controlling cellular phones activity to detect fraudulent usage like stolen phone airtime. Outlier detection methods could also be considered as a preprocessing step before applying any other advanced data mining analytics and it has a wide range of applicability in many research areas including intrusion detection, activity monitoring, satellite image analysis, medical condition monitoring etc. [20, 3].

Outliers can be generally divided into two global and local categories. Global outliers are those objects which show significant abnormal behavior in comparison to rest of the data, and thus in some cases they are considered as point anomalies. On the contrary, local outliers only deviate significantly w.r.t. a specific neighborhood of the object [9, 17]. In [7, 11], it is noted that the concept of local outlier is more comprehensive than that of the global, meaning that a global outlier could also be considered as local, but not vice versa. This is the reason that makes finding local outliers much more cumbersome.

*Corresponding author
sa_na33@aut.ac.ir (Sayyed-Ahmad Naghavi-Nozad); haeri@aut.ac.ir (Maryam Amir Haeri); gianluigi.folino@icar.cnr.it (Gianluigi Folino)
ORCID(s):

Advances in data acquisition in recent years have made massive collections of data, which contain valuable information in diverse fields like business, medicine, society, government etc. As a result, the common traditional software methods for processing and management of this massive amount of data will no longer be efficient, because most of these methods assume that the data is memory-resident and their computational complexity for large-scale datasets is really exhausting.

In this paper, we propose a new scalable and density-based clustering method for local outlier detection in massive-scale datasets that do not fit into RAM at once, employing a chunk-by-chunk procedure. Our proposed approach, is in essence a clustering method for very large datasets, which outlier identification comes after that as a side effect, and is inspired by a scaling clustering algorithm for very large databases, named after its authors, BFR [6]. BFR has a strong assumption on the structure of existing clusters, which shall be Gaussian distributed with uncorrelated attributes and more importantly, it is not introducing noise. Its clustering framework is in brief, reading data as successive (preferably random) samples, as each sample fills free space in RAM buffer, and updating the current clustering model over contents of buffer. Based on the updated model, singleton data are classified in three groups; some of them can be discarded with updates to the sufficient statistics (discard set DS), some can be reduced via compression and summarized as sufficient statistics (compression set CS), and some need to be retained in the buffer (retained set RS).

Like BFR, our proposed method, operates within the confines of a limited memory buffer, thus, by assuming that an interface to the database allows the algorithm to load an arbitrary number of requested data points, whether sequentially or randomized, we are forced to load data chunk-by-chunk, so that there is enough space for both loading and processing each chunk at the same time. Our proposed approach is based

on clustering and in every memory-load of points, all of our effort is to not let outliers play any role in forming and updating clusters. After processing each chunk, we have to combine its approximate results with those of previous chunks, in a way that the final approximate result will compete with the exact result of processing the entire data at once, if it would be possible. An algorithm which is capable of handling data in such a gradual way and finally provides an approximate result, from an operational perspective is called a scalable algorithm [41]. Moreover, from an algorithmic point of view, scalability means that algorithm complexity should be nearly linear or sublinear w.r.t. the problem size [39]. In summary, contributions of our proposed method are as follows:

- There is no need to know about the true number of original clusters.
- It can work best with Gaussian clusters with correlated or uncorrelated features, and also works well with convex-shaped clusters from any distribution.
- It has a linear time complexity with a low constant.
- However, it is conducting the density-based clustering and outlier detection in a scalable and chunk-by-chunk manner, not by observing the whole dataset at once, but in the case of detection accuracy, it is significantly competing with best-known conventional density-based methods which need the data to be memory-resident.

The paper is organized as follows: Section 2 discusses some related works in the field of outlier detection. In Section 3, we present the detailed descriptions of the proposed approach. In Section 4, the experimental results and analysis on various real and synthetic datasets are provided. Finally, conclusions are given in Section 5.

2. Related Works

Outlier detection methods can be divided into following six categories [2]: extreme value analysis, probabilistic methods, distance-based methods, information-theoretic methods, clustering-based methods, and density-based methods. In extreme value analysis, we think of the whole population as a unique probability density distribution and consider only those objects that lie at the very ends of it as outliers. These types of methods are useful while trying to find global outliers [2, 8]. In probabilistic methods, we assume that the data were generated from a mixture of different distributions as a generative model, and we use the same data to estimate the parameters of the model. After determining specified parameters, outliers will be those objects with low likelihood of being generated by this model [2]. Schölkopf et al. [36] propose a supervised approach in which, a probabilistic model w.r.t. the input data is provided, so that it can fit normal data in the best possible way. In this manner, the goal is to find the smallest region that contains most of normal objects; data outside that region are presumed to be outliers. This method is, in fact, an extended version of Support Vector Machines (SVM), which has been improved for imbalanced data, and in that, small number of outliers is considered as rare class and the rest of data as normal objects.

In distance-based methods, distances among all objects are computed to detect outliers. An object is assumed to be a distance-based outlier if it has d_0 distance away from at least fraction p_0 of other objects in the dataset [28]. Information-theoretic methods could be considered as almost equivalent to distance-based and other deviation-based models, except that the outlier score is defined by the model size for a fixed deviation, rather than deviation for a fixed model [2]. Wu and Wang [40] propose a single-parameter method for outlier detection in categorical data using a new concept of Holoentropy and by utilizing that, a formal definition of outliers and an optimization model of outlier detection is presented. According to this model, a function for the outlier factor is defined, which is solely based on the object itself not the entire data and could be updated efficiently.

Clustering-based methods use a global analysis to detect crowded regions and outliers will be those objects not belonging to any cluster [2]. A Cluster-Based Local Outlier Factor (CBLOF) in [19], and a Cluster-Based Outlier Factor (CBOF) in [13] are presented, which in both of them, after clustering procedure is carried out, due to a specific criterion, clusters are divided into two large and small groups; and it is assumed that outliers lie in small clusters. At last, the distance of each object to its nearest large cluster is used in different ways to define the outlier score.

In density-based methods, local density of each object is calculated in a specific way and then is utilized to define outlier scores. Given an object, the lower its local density compared to that of its neighbors, the more likely it is that the object is an outlier. Density around the points could be calculated through many ways which most of them are distance-based [7, 2]. For example, Breunig et al. [7] propose a Local Outlier Factor (LOF) that uses the distance values of each object to its nearest-neighbors for calculating local densities. But LOF has a shortage which is the scores obtained through this approach are not globally comparable between all objects in the same dataset or even through various datasets. A Local Outlier Probability (LoOP) in [29] is presented which is basically an enhanced version of LOF. LoOP gives each object a score in the interval [0,1] which is assumed as the probability of the object being an outlier and is widely interpretable among various situations. A Local Distance-based Outlier Factor (LDOF) in [42] is proposed for finding outliers in scattered datasets which uses the relative distance from an object to its neighbors, and an INFLuenced Outlierness (INFLO) score is presented in [23] which utilizes both neighbors and reverse neighbors of an object to estimate its relative density distribution. Moreover, Tang and He [38] propose a local density-based outlier detection approach, in which the local density around each object is approximated with local kernel density estimation (KDE) through nearest-neighbors of it. In this approach, not only the k-nearest-neighbors of an object are accounted but further reverse-nearest-neighbors and shared-nearest-neighbors are

considered for density distribution estimation.

Remark 1. According to the fact that, during scalable clustering, we use the Mahalanobis distance measure [31, 35] to assign each object to a minicluster, and of course, the size of temporary clusters is much smaller than that of original clusters, it would be worth mentioning an important matter here. With respect to [35, 15, 22, 4], in the case of high-dimensional data, classical approaches based on the Mahalanobis distance are usually not applicable. Because, when the cardinality of a cluster is less than or equal to its dimensionality, the sample covariance matrix will become singular and not invertible, hence, the corresponding Mahalanobis distance will no longer be reliable.

Therefore, to overcome such problem, in a preprocessing step, we need to resort to dimensionality reduction approaches. But, due to the serious dependence of some dimensionality reduction methods like PCA [33] to original attributes, and the following high computational load because of the huge volume of the input data, we need to look for alternative methods to determine a basis for data projection.

A simple and computationally inexpensive alternative is the use of a random basis projections [24, 10, 1]. The main characteristic of such methods is that they will approximately preserve pairwise euclidean distances between data points, and also, the dimension of the transformed space is independent of the original dimension, and only depends logarithmically on the number of data points. Finally, after such preprocessing step, we can be optimistic that singularity problem will not happen during clustering procedures. But, in the case of happening, we present a suitable mechanism to handle it.

Remark 2. As stated earlier, our proposed approach is inspired by BFR. But BFR, by default, uses the Kmeans algorithm [16] in almost all of its clustering procedures. And beside this weak spot of Kmeans, which is being seriously depending on foreknowing the true number of original clusters, also, in the case of presence of outliers, Kmeans fails and we need to resort to a density-based clustering approach, like DBSCAN [14][1]. Other density-based clustering algorithms could be utilized as a substitute option too, but definitely with their specific configurations.

However, about DBSCAN, it is strongly parametric, which means without the true values for its parameters, no significant outcomes are anticipated from applying it to the problem. Thus, we are forced to utilize optimization algorithms to find the best values of these parameters. Here, we prefer to use the evolutionary algorithm PSO [27].

Remark 3. Another important difference between proposed method and BFR is on the volume of structural information which they store for clustering procedure. As the proposed method can handle Gaussian clusters with correlated attributes too, while BFR cannot, hence, the covariance matrix is not diagonal anymore and will have many non-zero elements. Therefore, proposed method will consume more space than BFR for building clustering structures.

Since Mahalanobis distance criterion is crucially based on the covariance matrix, hence, this matrix will be literally the most prominent property of each subcluster. But according to high computational expense of computing Mahalanobis distance in high-dimensions, thus, as in [15, 32], we will use properties of principal components in the transformed space. Therefore, the covariance matrix of each minicluster will become diagonal and by transforming each object to the new space of the minicluster, like BFR, we can calculate the Mahalanobis distance without the need to matrix inversion.

According to [30], when the covariance matrix is diagonal, the corresponding Mahalanobis distance becomes the same normalized Euclidean distance. Moreover, we can establish a threshold value for defining the Mahalanobis radius. If the value of this threshold is, e.g. 4, it means that all points on this radius are as far as four standard deviations from the mean, and if we just denote the number of dimensions by p, the size of this Mahalanobis radius equals to $4\sqrt{p}$.

3. Proposed Approach

Our proposed method consists of three major phases. In the first phase, a preliminary random sampling is conducted and by using that, primary information of original clusters and of course, parameters necessary to continue clustering are obtained. In the second phase, the scalable clustering is carried out, as in each memory-load of points, a density-based clustering approach is used to identify dense regions. Discovered clusters in this way, form the temporary clustering model which will sometimes be referred to as miniclusters or subclusters. After loading each chunk of data, according to the points already loaded in memory and those undecided from previous chunks, and by employing Mahalanobis distance measure and the density-based clustering standards, we update the temporary clustering model, which consists of making some changes to existing miniclusters or adding new subclusters. Throughout the whole scalable clustering procedure, our endeavor is to not let outliers participate actively in forming and updating any minicluster, and thus at the end of this phase, there will be some objects in buffer remained undecided. Some of these data are true outliers that have been properly suspended in memory, and some others are inliers, which due to constraints, have failed to play any effective role in forming a subcluster. Ultimately, all of these undecided points will be cleared from buffer and it is only the structural information of the temporary clusters that is maintained in memory. At the end of scalable clustering, once more, we utilize a clustering approach to combine information of miniclusters and obtain the approximate structure of the final or the same original clusters. At last, in the third phase of the proposed approach, w.r.t. the final clustering model gained from the second phase, once again, we process the entire dataset in chunks, to give each object an outlying

[1] Although we will demonstrate that even DBSCAN sometimes fails during scalable clustering, to form regular miniclusters. And hence, we will be forced to use the same Kmeans for fixing the issue.

score, according to the same Mahalanobis distance criterion. Fig. 1 demonstrates the brief software architecture of the proposed approach. Moreover, Table 1 summarizes the major notations used in the paper.

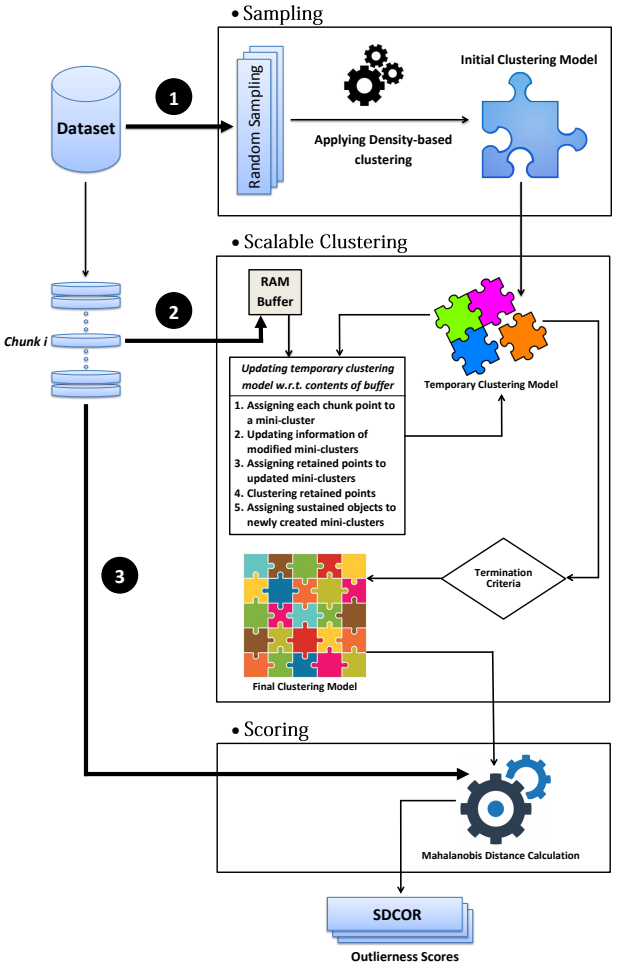


Figure 1: Software architecture of the proposed approach

The framework of proposed approach is presented in Algorithm 1, which consists of three main phases including: 1) Sampling; 2) Scalable Clustering; and 3) Scoring. All of these phases will be described in details in following subsections.

3.1. Sampling

At this phase, we afford to take a random sample of the entire dataset. It seems simple at first, but in reality it can be a complicated issue. Because there is no guarantee that in some databases, records are not ordered by some property, and this makes it almost impossible to carry out an efficient random sampling. This could also make it hard to achieve enough samples from dense areas of each original cluster in the input dataset, and thus the clustering structure obtained out of sampled data may not be similar enough to the original one. Hence, outliers could be misclassified during scalable clustering. In such a situation, when the input data are in a

Table 1
Major Notations

Notation	Description
$[\mathcal{X}]_{n\times p}$	Input dataset $\mathcal{X}$ with n objects and p dimensions
$x \in \mathcal{X}$	An instance in $\mathcal{X}$
$\mathcal{S} \subset \mathcal{X}$	A random sample of $\mathcal{X}$
$\mathbb{X} \subset \mathcal{X}$	A set of points
$\mathcal{Y} \subset \mathcal{X}$	A chunk of data
Ω	A partition of miniclusters points as $\{\mathbb{X}_1, \cdots, \mathbb{X}_{\mathbb{k}}\}$
$n_{\mathfrak{S}}$	Number of objects in $\mathcal{Y}$
$\{\Gamma\}_{7\times\mathcal{L}}$	Information array of temporary clusters with 7 properties for each minicluster
$\{\Gamma_i\}_{7\times1} \in \Gamma$	Information standing for the ith minicluster in $\Gamma, 1 \leq i \leq \mathcal{L}$
X_i	Minicluster points associated with Γ_i, which are removed from buffer
$\mathfrak{R}$	Retained set of objects in RAM buffer
$\mathfrak{R}_i$	The ith minicluster of retained points in buffer, discovered through DBSCAN
$\mathfrak{R}_\zeta$	Retained set of objects in buffer introduced as noise by DBSCAN
γ	List of indices to recently created or updated miniclusters associated with Γ, to be checked on for membership
γ'	Temporary list of indices to recently created or updated miniclusters associated with Γ, to be checked on for membership
m	Current number of objects associated with Γ_i
p'	Current Number of superior components associated with Γ_i
e_i	The ith PC coefficient
λ_i	The ith PC variance
$\mathcal{L}$	Current number of temporary clusters associated with Γ
$\mathbb{k}$	Number of miniclusters which are about to be added to temporary clustering model
$\mathbb{K}$	Kmeans parameter for number of clusters
$\mathbb{K}'$	Number of retained set subclusters discovered through DBSCAN
$\mathbb{K}''$	Number of retained set subdivided subclusters obtained out of Kmeans
$\mathcal{T}$	True number of original clusters in $\mathcal{X}$
$\{f\}_{2\times\mathcal{T}}$	Information array of final clustering model with 2 properties for each ultimate cluster
$[\mathcal{M}]_{\mathcal{L}\times p}$	Means matrix of whole miniclusters
$\mathcal{M}_i$	The ith final cluster, comprising of some temporary means
μ_f	Final mean location of an ultimate cluster
Σ_f	Final covariance structure of an ultimate cluster
$\mathcal{U}$	A set of regenerated points
η	Random sampling rate
Λ	PC total variance ratio for temporary clusters
α	Membership threshold for temporary clusters
β	Pruning threshold for final clusters
ϵ	DBSCAN parameter Epsilon
ψ	DBSCAN parameter MinPts
$\epsilon_{\mathcal{S}}$	DBSCAN parameter ϵ required for clustering $\mathcal{S}$
$\psi_{\mathcal{S}}$	DBSCAN parameter ψ required for clustering $\mathcal{S}$
$\mathbb{C}_\epsilon$	Coefficient of $\epsilon_{\mathcal{S}}$ necessary for clustering $\mathcal{X}$
$\mathbb{C}_\psi$	Coefficient of $\psi_{\mathcal{S}}$ necessary for clustering $\mathcal{X}$
μ_X	Mean location of cluster X
Σ_X	Covariance structure of cluster X
$\mathbb{S}_X$	Scatter matrix of cluster X
$\mathcal{A}_X$	Transformation matrix of cluster X
z	Object x in the space of eigenvectors
μ'_X	Mean location of cluster X in the space of eigenvectors
$MD(x, X)$	Mahalanobis distance of object x from cluster X
$SingCheck(\Sigma_X)$	A function that checks on the singularity of Σ_X, and outputs 1 in the case of being singular and 0 if not
$CohrCheck(X)$	A function that checks on the coherence of input data X, so that whether or not only one dense cluster will be discovered through DBSCAN. It outputs 1 in the case of being coherent, and 0 if not
det_{Σ_X}	Covariance determinant of cluster X
$\vec{\delta}$	Vector of maximum covariance determinant condition for miniclusters discovered through scalable clustering
$\lvert\cdot\rvert$	Cardinality of a set of objects
Φ	The empty set
κ	A low constant near zero

Algorithm 1: Framework of SDCOR

Input : $[\mathcal{X}]_{n\times p}$ - The n by p input dataset $\mathcal{X}$; η - Random sampling rate; Λ - PC total variance ratio; α - Membership threshold; β - Pruning threshold; $\mathbb{C}_\varepsilon$ - Sampling Epsilon coefficient; $\mathbb{C}_\psi$ - Sampling MinPts coefficient

Output : The outlying score for each object in $\mathcal{X}$

Phase 1 — Sampling:

Step 1. Take a random sample $\mathcal{S}$ of $\mathcal{X}$ according to the sampling rate η

Step 2. Employ PSO algorithm to find optimal values for DBSCAN parameters $\varepsilon_{\mathcal{S}}$ and $\psi_{\mathcal{S}}$, required for clustering $\mathcal{S}$

Step 3. Run DBSCAN on $\mathcal{S}$ using the obtained optimal parameters, and reserve the count of discovered miniclusters as $\mathcal{T}$, as the true number of original clusters in data

Step 4. Build the very first array of miniclusters information (temporary clustering model) out of result of step 3, w.r.t. Algorithm 2

Step 5. Reserve the covariance determinant values of initial subclusters as a vector $\vec{\delta} = [\delta_1, \cdots, \delta_{\mathcal{T}}]$, for the maximum covariance determinant condition

Step 6. Clear $\mathcal{S}$ from RAM and maintain the initial temporary clustering model in buffer

Phase 2 — Scalable Clustering:

Prepare input data to be processed chunk by chunk, as each chunk could be fit and processed in RAM buffer at the same time

Step 1. Drag the next available chunk from data into RAM

Step 2. Update the temporary model of clustering over contents of buffer, w.r.t. Algorithm 3

Step 3. If there is any available unprocessed chunk, go to step 1

Step 4. Build the final clustering model, w.r.t. Algorithm 7, using the temporary clustering model obtained out of earlier steps

Phase 3 — Scoring:

According to the final clustering model, for each data point $x \in \mathcal{X}$, use Mahalanobis distance criterion to find the closest cluster, and finally assign x to that cluster and use the criterion value as the object outlierness score

special order, taking a random sample could be completed at the cost of scanning the entire dataset. Here, we assume that the records of our dataset are not in a particular order and taking a total random sample is possible. It should be noted that after gaining the sampled data, the rate at which we do random sampling, and could be e.g. in percentage terms, 0.5%, is stored for later use. We demonstrate this rate here, with η.

After obtaining the sampled data, we conduct DBSCAN algorithm to cluster them, and assume that the number of miniclusters obtained through this, is the same as number of original clusters $\mathcal{T}$ in the main dataset. We reserve $\mathcal{T}$ for later use. Besides, we presume that the location (centroid) and the shape (covariance structure) of such subclusters are so close to original ones. In Section 4, we will show that even by using a small rate of random sampling, the mentioned properties of sampled clusters are similar enough to original ones. The idea behind making these primary subclusters, which are so similar to original clusters in the input dataset in terms of basic characteristics, is that, we intend to determine a Mahalanobis radius, which collapses a specific percentage of objects belonging to each original cluster, and let other subclusters be created around this folding area during successive memory-loads of points, and ultimately, by merging these miniclusters, we will obtain the approximate structure of original clusters.

As we utilize PSO to attain optimal values of parameters $\varepsilon_{\mathcal{S}}$ and $\psi_{\mathcal{S}}$ for DBSCAN, w.r.t. the fact that the density of sampled distribution is much less than that of original data, we cannot use the same parameters for the original distribution, while applying DBSCAN on objects loaded in memory, during scalable clustering. Thus, we have to use a coefficient in the interval [0,1] for each parameter. It is also necessary to mention that ε is much more sensitive than ψ, as with a slight change in the value of ε, we may observe a serious deviation in the clustering result, but this does not apply to ψ. We show the mentioned coefficients with $\mathbb{C}_\varepsilon$ and $\mathbb{C}_\psi$, for $\varepsilon_{\mathcal{S}}$ and $\psi_{\mathcal{S}}$ respectively, and their values could be obtained through the user, but the best values gained out of our experiments are 0.5 and 0.9, for $\mathbb{C}_\varepsilon$ and $\mathbb{C}_\psi$ respectively.

Now, it is time to build the very first clustering model through sampled data. For this purpose, we have to extract some information out of sampled clusters obtained through DBSCAN, and store them in a special array. As stated earlier about the benefit of using properties of principal components for high-dimensional data, we need to find those PCs that have higher contribution to the cluster representation. For this matter, we can sort PCs due to corresponding variances in a descending order, and then choose those top PCs that their share of total variance is at least equal to Λ percent. We call these top PCs, superior components and denote their number as p'. Let x, be an object among total n objects in dataset $[\mathcal{X}]_{n\times p}$, belonging to the temporary cluster $\left[\mathbb{X}_i\right]_{m\times p}$, then the information about this subcluster as $\left\{\Gamma_i\right\}_{7\times 1}$, in the array of temporary clustering model $\{\Gamma\}_{7\times\mathcal{L}}$ is as follows:

1. Mean vector in original space, $\mu_{\mathbb{X}_i} = \frac{1}{m}\sum_{x\in\mathbb{X}_i} x$
2. Scatter matrix in original space, $\mathbb{S}_{\mathbb{X}_i} = \sum_{x\in\mathbb{X}_i}(x - \mu_{\mathbb{X}_i})^t \cdot (x - \mu_{\mathbb{X}_i})$
3. p' superior components, $\left[e_1, \cdots, e_{p'}\right]$, derived from covariance matrix $\sum_{\mathbb{X}_i} = \frac{1}{m-1}\mathbb{S}_{\mathbb{X}_i}$, which form columns of transformation matrix $\mathcal{A}_{\mathbb{X}_i}$
4. Mean vector in transformed space, $\mu'_{\mathbb{X}_i} = \mu_{\mathbb{X}_i}\mathcal{A}_{\mathbb{X}_i}$
5. Square root of top p' PC variances, $\left[\sqrt{\lambda_1}, \cdots, \sqrt{\lambda_{p'}}\right]$
6. Size of minicluster, m
7. Value of p'

Algorithm 2, demonstrates the process of obtaining and adding this information per each minicluster to temporary clustering model. We also use this algorithm while adding information of new discovered miniclusters through scalable clustering to temporary clustering model[2].

According to [21, 25], when a multivariate Gaussian distribution is contaminated with some outliers, then the corresponding covariance determinant is no longer robust and is

[2] It is clear that all those sampled points, which the initial clustering model is built upon them, will be met again during scalable clustering. But as they are totally randomly sampled, hence, one can assert that they will not impair the structure of final clustering model.

Algorithm 2: $[\Gamma]$ = MiniClustMake(Γ, Ω, Λ)

Input : Γ - Current array of miniclusters information; $\Omega = \{\mathbb{X}_1, \cdots, \mathbb{X}_{\Bbbk}\}$ - Partition of miniclusters points; Λ - PC share of total variance

Output : Γ - Updated temporary clustering model

$c \leftarrow \mathcal{L}$
foreach *minicluster* $\mathbb{X}_i, 1 \leq i \leq \Bbbk$ **do**
  Apply PCA on $\mathbb{X}_i$ and obtain its PC coefficients and variances. Then choose p' as the number of those top PC variances, which their share of total variance is at least Λ percent
  $\Gamma\{1, c+i\} \leftarrow$ Mean vector of $\mathbb{X}_i$
  $\Gamma\{2, c+i\} \leftarrow$ Scatter matrix of $\mathbb{X}_i$
  $\Gamma\{3, c+i\} \leftarrow$ Top p' PC coefficients corresponding to top p' PC variances
  $\Gamma\{4, c+i\} \leftarrow$ Transformed mean vector, as $\Gamma\{1, c+i\} \cdot \Gamma\{3, c+i\}$
  $\Gamma\{5, c+i\} \leftarrow$ Square root of top p' PC variances
  $\Gamma\{6, c+i\} \leftarrow$ Number of objects in $\mathbb{X}_i$
  $\Gamma\{7, c+i\} \leftarrow$ Value of p'
end

significantly more than that of the main cluster. Following this contamination, the corresponding Mahalanobis contour lines[3] will also become wider than that of the clean cluster to contain all objects, including abnormal data. So, it makes sense that there is a direct relationship between the value of covariance determinant of a cluster and the wideness of its tolerance ellipses[4]. Moreover, by being contaminated, this volume could increase and become harmful.

Hence, given that during scalable clustering, objects are coming over time and miniclusters are growing gradually, so, it is possible for a minicluster to accept some outliers, in the case being non-convex[5]. Then, the following covariance matrix will no longer be robust, and the corresponding Mahalanobis contour lines will keep getting wider too. This could happen again over time and so many other outliers could be assigned to such contaminated subcluster. Therefore, to impede the creation process of those voluminous non-convex subclusters, which can absorb outliers, we have to put a limit on the covariance determinant of every subcluster, which is discovered through scalable clustering. Here, we follow a heuristic approach and use the covariance determinant of the nearest initial subcluster obtained out of "Sampling" stage, as this limit. Thus, we reserve the covariance determinant values of initial miniclusters for later use.

This problem that, outliers are absorbed to normal clusters and could no longer be detected is called masking effect. It is better to be noted that, we are using the mentioned constraint only when subclusters are created for the first time, not while they are growing over time. The reason is that, while an object is about to be assigned to a minicluster, the other constraint on the Mahalanobis radius is somehow hindering outliers to be accepted as a member. In other words, when the Mahalanobis distance of an outlier is more than the predefined radius threshold, it cannot be assigned to the subcluster, if and only if that threshold is set to a reasonable value. Hence, we do not check on the covariance determinant of subclusters while they are growing.

Here, the first phase of the proposed approach is finished, and we need to clear RAM buffer of any sampled data, and only maintain the very initial information obtained about existing original clusters.

3.2. Scalable Clustering

At this phase, we have to process the entire dataset chunk-by-chunk, as for each chunk there is enough space in memory for both loading and processing it, all at the same time. After loading each chunk, we afford to update the temporary clustering model according to data points, which are currently in RAM from this chunk or retained from other earlier loaded chunks. Finally, after processing the entire chunks, the final clustering model is built out of temporary clustering model. Detailed descriptions of this phase are provided as follows.

3.2.1. Updating Temporary Clustering Model w.r.t. Contents of Buffer

After loading each memory load of points, the temporary clustering model is getting updates, regarding the objects from the currently loaded chunk and other sustained ones from the earlier loaded pieces of data. First of all, the algorithm checks for the membership of each point of the currently loaded chunk to existing mini-clusters in the temporary clustering model[6].

Then, after the possible assignments of current chunk points, regarding the established membership criteria, there are some primary and secondary information of the modified sub-clusters that shall be updated. Employing this fundamental updating procedure, the structure of the altered sub-clusters will change, and thus they might still be capable of absorbing more inliers. Therefore, the algorithm checks again for the probable memberships of sustained points in memory to updated sub-clusters. This updating and assignment checking cycle will keep going till there is no retained point that could be assigned to an updated mini-cluster.

When the membership evaluation of the present chunk and retained objects is carried out, the algorithm tries to cluster the remaining sustained objects in RAM, regarding the density-based clustering criteria which have been founded at the Sampling phase. After new mini-clusters were created out of the last retained points, there is this probability that some sustained inliers in buffer might not be capable of participating actively in forming new sub-clusters, because of the density-based clustering standards, but could be assigned to them considering firstly settled membership measures. Hence, the algorithm goes another time in the cycle of assignment and updating procedure, like what was done in earlier steps.

Algorithm 3, demonstrates the steps needed for updating the temporary clustering model, w.r.t. an already loaded

[3] Since the terms "Mahalanobis contour line" and "tolerance ellipse" are the same in essence, thus, we will use them alternatively in this paper.

[4] We will sometimes refer to the wideness of tolerance ellipses as the spatial volume of the cluster.

[5] This will be clarified with a natural example later.

[6] After this state, the unassigned objects of the lastly and previously loaded chunks, will be considered as retained or sustained objects in buffer.

chunk of data and other undecided objects retained in memory from before. The following subsections will explain the details of this algorithm.

Algorithm 3: $[\Gamma, \Re] = \text{MemoProcess}(\mathcal{Y}, \Gamma, \alpha, \Re, \vec{\delta}, \varepsilon_S, \psi_S, \mathbb{C}_\varepsilon, \mathbb{C}_\psi, \Lambda)$

Input : $\mathcal{Y}$ - A chunk of data; Γ - Current array of miniclusters information; α - Membership threshold; $\Re$ - Retained set; $\vec{\delta}$ - Covariance determinant threshold; ε_S - Sampling Epsilon; ψ_S - Sampling MinPts; $\mathbb{C}_\varepsilon$ - Sampling Epsilon coefficient; $\mathbb{C}_\psi$ - Sampling MinPts coefficient; Λ - PC share of total variance

Output : Γ - Updated temporary clustering model; $\Re$ - Modified retained set

/* Trying to assign each datum of chunk to a minicluster */
$\gamma \leftarrow \{1, \cdots, \mathcal{L}\}$
$[\Gamma, \gamma, \Re] = \text{MiniClustUpdate}(\mathcal{Y}, \Gamma, \gamma, \alpha, \Re)$
/* Checking out retained set */
if $|\Re| \neq 0$ **then**
  /* Checking on retained set membership for recently updated miniclusters */
  $[\Gamma, \Re] = \text{RetSetMemb}(\Re, \Gamma, \gamma, \alpha)$
  /* Clustering retained set */
  if $|\Re| \neq 0$ **then**
    $l \leftarrow \mathcal{L}$
    $[\Gamma, \Re] = \text{RetSetClust}(\Re, \Gamma, \vec{\delta}, \varepsilon_S, \psi_S, \mathbb{C}_\varepsilon, \mathbb{C}_\psi, \Lambda)$
    $\gamma \leftarrow \{l+1, \cdots, \mathcal{L}\}$
    /* Checking on retained set membership for recently created miniclusters */
    if $|\Re| \neq 0$ **then**
      $[\Gamma, \Re] = \text{RetSetMemb}(\Re, \Gamma, \gamma, \alpha)$
    end
  end
end

3.2.1.1. *Trying to Assign Each Datum of Chunk to a Minicluster*

After loading each chunk of data in buffer, we need to use properties of PCs for each minicluster and transform each datum to the new space of that minicluster, and then, like BFR, calculate the Mahalanobis distance using the mean vector and square root of variances, but in the space of eigenvectors. That is,

$$MD(x, X_i) = \sum_{j=1}^{p'} \left(\frac{z_j - \mu'_j}{\sqrt{\lambda_j}}\right)^2 \tag{1}$$

Where $MD(x, X_i)$ is the Mahalanobis distance of object x from minicluster X_i; $z = x \cdot \mathcal{A}_{X_i}$ is the object in the eigenvector space of the minicluster, and z_j is its jth component; μ'_j and λ_j are jth components of the mean vector and variance vector in the space of eigenvectors; and finally, p' is the number of superior components associated with the minicluster. As stated earlier, in this style, the amount of computations is much less than when we use matrix inversion to calculate the distance. Moreover, w.r.t. [30], the accepted Mahalanobis radius in the eigenvector space of the relevant subcluster will be the product of membership threshold, times square root of number of dimensions in the transformed space, as $\alpha \cdot \sqrt{p'}$.

For each data point, w.r.t. (1), we need to find the closest minicluster and check whether or not it falls in the accepted Mahalanobis threshold of that minicluster, and if it does, some information connected to corresponding subcluster shall be updated.

Algorithm 4: $[\Gamma, \gamma', \Re] = \text{MiniClustUpdate}(\mathbb{X}, \Gamma, \gamma, \alpha, \Re)$

Input : $\mathbb{X}$ - A set of points; Γ - Current array of miniclusters information; γ - List of indices to recently created or updated miniclusters associated with Γ, to be checked on for membership; α - Membership threshold; $\Re$ - Retaine set

Output : Γ - Updated temporary clustering model; $\gamma' \subseteq \gamma$ - Modified list of indices to recently updated miniclusters; $\Re$ - Modified retained set

/* Updating primary information of subclusters */
foreach $x \in \mathbb{X}$ **do**
  $b \leftarrow \text{argmin}_{i \in \gamma}\, MD(x, X_i)$
  if $MD(x, X_b) \leq \alpha \cdot \sqrt{\Gamma_b\{7\}}$ **then**
    $\Gamma_b\{2\} \leftarrow \Gamma_b\{2\} + x'x$
    $\Gamma_b\{6\} \leftarrow \Gamma_b\{6\} + 1$
    Remove x from RAM buffer
  else
    $\Re \leftarrow \Re \cup x$
  end
end
/* Updating secondary information of subclusters */
$\gamma' \leftarrow \Phi$
foreach $X_i, i \in \gamma$ **do**
  if X_i *has accepted any new members* **then**
    Obtain its updated covariance matrix $\sum_{X_i}$, through normalizing its updated scatter matrix, w.r.t. current size of minicluster as $\left(\frac{1}{\Gamma_i\{6\}-1}\right) \cdot \Gamma_i\{2\}$
    Apply PCA on $\sum_{X_i}$ to acquire its eigenvalues and eigenvectors, and then, update Γ_i as follows:
      $\Gamma_i\{7\} \leftarrow$ Value of p' as the updated number of superior components
      $\Gamma_i\{3\} \leftarrow$ Updated superior coefficients
      $\Gamma_i\{4\} \leftarrow$ Updated transformed mean vector, as $\Gamma_i\{1\} \cdot \Gamma_i\{3\}$
      $\Gamma_i\{5\} \leftarrow$ Square root of updated superior variances
    $\gamma' \leftarrow \gamma' \cup i$
  end
end

3.2.1.2. *Updating Primary and Secondary Information of Temporary Clusters*

Related information to a subcluster which need to be updated after objects assignment, are twofold; primary and secondary. Primary information comprises of scatter matrix of the subcluster and its cardinality, which should be updated after each individual assignment. For updating the scatter matrix, the outer product of the belonged data point with itself is added to current scatter matrix, and for cardinality, the number of objects assigned to the subcluster is increased by one. Each object, after joining a subcluster is removed from buffer, otherwise will be retained to be decided on later.

After checking on membership of all points and updating primary information of subclusters, for each minicluster which has accepted any new members, its PC properties, which are considered as its secondary information, shall be updated too. For this purpose, due to size of the minicluster, we normalize its scatter matrix in an unbiased manner to acquire its covariance matrix. Then, by applying PCA on this

Algorithm 5: $[\Gamma, \Re] = \text{RetSetMemb}(\Re, \Gamma, \gamma, \alpha)$

Input : $\Re$ - Retained set; Γ - Current array of miniclusters information; γ - List of indices to recently created or updated miniclusters associated with Γ, to be checked on for membership; α - Membership threshold

Output : Γ - Updated temporary clustering model; $\Re$ - Modified retained set

```
if |γ| ≠ 0 then
    while true do
        [Γ, γ, ℜ] = MiniClustUpdate(ℜ, Γ, γ, α, Φ)
        if |γ| ≡ 0 then
            break
        end
    end
end
```

matrix, we update the transformation matrix, mean vector in the space of eigenvectors and superior PC variances of the minicluster. Algorithm 4, demonstrates the required steps for finding the closest subcluster, due to relevant limitations, and updating its information.

3.2.1.3. Trying to Assign Retained Objects in Buffer to Newly Updated Miniclusters After updating secondary information of each subcluster, the corresponding tolerance ellipses will rotate a little around the centroid, and in other words, their accepted Mahalanobis neighborhood is modified. Hence, w.r.t. Algorithm 5, it would be necessary to check on objects retained in buffer, whether they can belong to a modified minicluster, and if it is so, the corresponding minicluster information need to be updated, w.r.t. Algorithm 4. But, this is not the end. By keeping up this cycle of membership checking and minicluster updating, more and more objects could be assigned to miniclusters and then be discarded. In this iterative manner, after each iteration, memory contents should be evaluated using only updated subclusters from the last iteration, and thus the list of updated miniclusters will be shrinking over time till it becomes an empty list. This means that there is not any other sustained object in buffer, which falls in the accepted Mahalanobis threshold of any of the updated miniclusters, or every retained object has been eventually assigned to an updated minicluster. Here, by this procedure, it seems that tolerance ellipses of miniclusters are sweeping inliers through the cycle of assignment and updating.

Fig. 2 demonstrates a situation in which, after updating the core information of a subcluster, its Mahalanobis neighborhood is modified and, some objects which were not able to belong to this subcluster, now are capable of being assigned to it. Black circle points and black dashed line tolerance ellipse respectively represent a subcluster and its Mahalanobis neighborhood. Red circle points represent objects assigned to the subcluster during last memory process, as they reside in the accepted neighborhood of it. Red dashed line represents the updated tolerance ellipse of the updated subcluster. And finally, blue triangle points represent objects which could be assigned to the updated subcluster, if they lie in the updated Mahalanobis radius of it.

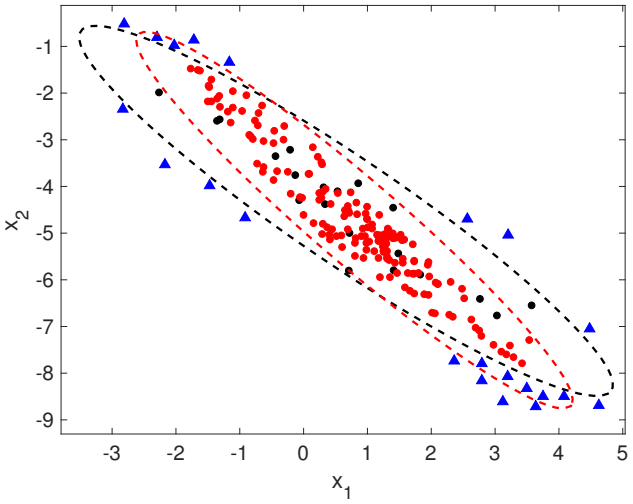


Figure 2: Trying to assign retained objects in buffer to newly updated subclusters

3.2.1.4. Clustering Retained Objects in Buffer Here, after checking on the membership of each present datum in memory, we afford to cluster retained data in RAM buffer, using again DBSCAN algorithm. However, as emphasized earlier at "Sampling" stage, according to the significant difference in the density of sampled and original data, we have to use predefined coefficients for the obtained optimal parameters out of sampled data.

Furthermore, as it was described earlier, it is possible that some miniclusters could be discovered during scalable clustering by DBSCAN, which are suffering from singularity problem. Thus, for handling such situation, there are some ways. One is to use the pseudoinverse of covariance structure, but it is not totally accurate. The better way is to disregard such minicluster and let its points still be in memory, to be resolved later. Therefore, for every discovered subcluster, we shall check on its covariance matrix, whether or not it is singular. And in the case of singularity, we disregard that subcluster.

Now, w.r.t. this prementioned matter that, we have to put a limit on the boundaries of miniclusters which are being created during scalable clustering, we are going to demonstrate with an intuitive example that if a minicluster with a non-convex shape is formed, how outliers could be absorbed to such irregular minicluster and cause serious damage to the final clustering results.

Fig. 3a illustrates the structure of an original cluster represented with red dots, with a newly discovered non-convex subcluster shown with blue dots, and a black square and a black dashed line as its centroid and accepted tolerance ellipse respectively. The irregular minicluster is formed around the initial minicluster, which its centroid and accepted Ma-

[7] Regarding the three-sigma rule of thumb, Mahalanobis radii equal to 1 and 2, cover roughly 68 and 95 percent of total objects in a Gaussian distribution, respectively. For convex-shaped clusters of other distributions, the amount of coverage might vary, but for non-covex-shaped clusters, it could contain objects not belonging to the distribution. Here, in all subfigures, the presented radius is equal to 1.5.

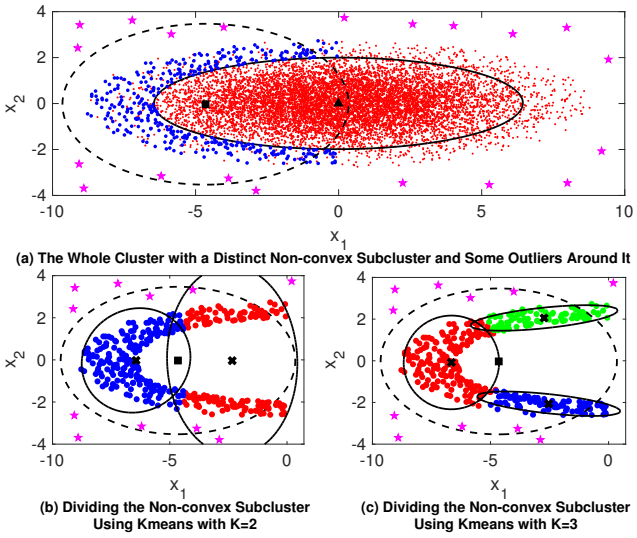


Figure 3: Breaking a non-convex minicluster with very wide tolerance ellipses into smaller pieces by Kmeans algorithm[7]

halanobis radius are denoted as a black triangle and a black solid line respectively. There are also some local outliers around the original cluster, which are illustrated with magenta pentagons[8].

As it is evident, the irregular minicluster can absorb some local outliers, as its tolerance ellipse is covering a remarkable space out of the containment area by the original cluster. Moreover, the covariance determinant value for irregular minicluster is equal to 15.15, which is almost twice that of the initial minicluster, which is equal to 7.88. Thus, for fixing this concern, as it was stated earlier, by considering the proportion between the covariance determinant of an arbitrary cluster and its spatial volume, we decide to divide the irregular minicluster to smaller coherent pieces, with smaller covariance determinants, and also, more limited Mahalanobis radii as well. Therefore, we heuristically set the threshold value for the covariance determinant of any newly discovered subcluster or subdivided subcluster, as that of the nearest initial minicluster[9].

For the division process of an irregular subcluster, we prefer to use Kmeans algorithm. But Kmeans can cause some incoherent subdivided subclusters in such cases[10], as shown in Fig. 3b. In Fig. 3b, two smaller miniclusters, produced as Kmeans result, are represented in different colors, with covariance determinants of 1.63 and 7.71 for the coherent blue and incoherent red miniclusters respectively. The related centroids and tolerance ellipses are denoted as black crosses and black solid lines respectively. It is clear that even incoherent subdivided subclusters, with a covariance determinant less than or equal to the predefined threshold though, could be hazardous as non-convex subclusters with a high value of covariance determinant, for their tolerance ellipses could get out of the scope of the main cluster, and suck outliers in[11].

An alternative for this is to use hierarchical clustering algorithms, with a much higher computational load than Kmeans. But, we follow another solution and utilize the same Kmeans, just with this difference that for every subdivided subcluster resulted out of Kmeans, we apply DBSCAN again and verify its cohesion.

Finally, we increase the value of $\mathbb{K}$ for Kmeans, from 2 till a value for which[12], three conditions are met. One is not being singular, the second is having a covariance determinant less than or equal to δ_i, and at last, it has to be coherent.

Fig. 3c illustrates a situation in which, three smaller miniclusters, are represented as Kmeans output in different colors, and centroids and tolerance ellipses shown as in Fig. 3b. The covariance determinant values are equal to 0.12, 1.02 and 0.10 for the green, red and blue subdivided subclusters respectively. As it is obvious, all subdivided subclusters are coherent and not singular, with much smaller determinants than the threshold, and much tighter spatial volumes as such. Ultimately, after attaining acceptable subclusters, it is time to update the temporary clustering model w.r.t. them, due to Algorithm 2. Algorithm 6 shows all the steps required to cluster data points retained in memory buffer.

3.2.1.5. Trying to Assign Retained Objects in Buffer to Newly Created Miniclusters After checking on retained objects in buffer in the case of being capable of forming a new minicluster, it would be essential to examine remaining retained objects once more, w.r.t. Algorithm 5, whether or not they could be assigned to recently created miniclusters, in the same cycle of membership checking and minicluster updating, like what we did in subsection 3.2.1.3. The reason for this concern is that, due to limitations related to the density-based clustering algorithm which has been used, such objects may not have been capable of being an active member of one of the newly created subclusters, even though they lie in the accepted Mahalanobis radius of that subcluster. Hence, it becomes essential to check on the assignment of these latter retained objects to those latter temporary clusters, which have been discovered through the latter process of memory contents.

Fig. 4 demonstrates an intuitive example of such situation, in which, some objects, according to density restrictions cannot be assigned to a cluster, in consideration of they lie in the accepted Mahalanobis radius of that cluster. Objects

[8] Here, for challenging our method, we are taking outliers so close to the original cluster. But in reality, it is not usually like that and outliers have a significant distance from every normal cluster in data.

[9] This threshold is denoted as $\delta_i, 1 \leq i \leq \mathcal{T}$, for any subcluster discovered near the ith initial minicluster, through scalable clustering. The proximity measure is as the Mahalanobis distance of new discovered subcluster mean from initial miniclusters. We assume any subcluster with a covariance determinant more than such threshold, as a candidate for a non-convex subcluster, whose spatial volume could cover some significant space out of the scope of the related original cluster.

[10] Because Kmeans focuses solely on finding the best locations for the means, not considering the cohesion of the output clusters.

[11] However, the divided minicluster could be significantly smaller in size and determinant, but because of lack of coherency, some PC variances could be very larger than others. Therefore, tolerance ellipses will be more stretched in those PCs, and thus harmful.

[12] Here, the upper bound for $\mathbb{K}$ is $\lfloor |\mathfrak{R}_i| / (p+1) \rfloor$, to avoid singularity problem for every subdivided subcluster of retained objects. $|\mathfrak{R}_i|$ stands for cardinality of the ith minicluster of retained points.

Algorithm 6: $[\Gamma, \mathfrak{R}] = \text{RetSetClust}(\mathfrak{R}, \Gamma, \vec{\delta}, \varepsilon_{\mathcal{S}}, \psi_{\mathcal{S}}, \mathbb{C}_{\varepsilon}, \mathbb{C}_{\psi}, \Lambda)$

Input : $\mathfrak{R}$ - Retained set; Γ - Current array of miniclusters information; $\vec{\delta}$ - Covariance determinant threshold; $\varepsilon_{\mathcal{S}}$ - Sampling Epsilon; $\psi_{\mathcal{S}}$ - Sampling MinPts; $\mathbb{C}_{\varepsilon}$ - Sampling Epsilon coefficient; $\mathbb{C}_{\psi}$ - Sampling MinPts coefficient; Λ - PC share of total variance

Output : Γ - Updated temporary clustering model; $\mathfrak{R}$ - Modified retained set

Apply DBSCAN algorithm to cluster $\mathfrak{R}$ w.r.t. two parameters $\mathbb{C}_{\varepsilon} \cdot \varepsilon_{\mathcal{S}}$ and $\mathbb{C}_{\psi} \cdot \psi_{\mathcal{S}}$ for Epsilon and MinPts respectively. Consider the result of such clustering as $\{\mathfrak{R}_1, \cdots, \mathfrak{R}_{\mathbb{K}'}\} \cup \mathfrak{R}_{\zeta}$
/* Adding information of newly discovered miniclusters to temporary clustering model */
foreach $\mathfrak{R}_i, 1 \le i \le \mathbb{K}'$ **do**
  if $SingCheck\left(\Sigma_{\mathfrak{R}_i}\right) \equiv 1$ **then** /* Singularity check */
    $\mathfrak{R}_{\zeta} \leftarrow \mathfrak{R}_{\zeta} \cup \mathfrak{R}_i$
    continue
  end
  $b \leftarrow \text{argmin}_{h \in \{1,\cdots,\mathcal{T}\}} MD(\mu_{\mathfrak{R}_i}, X_h)$
  if $det_{\Sigma_{\mathfrak{R}_i}} > \vec{\delta}(b)$ **then** /* Irregular minicluster */
    Apply Kmeans with number of clusters $2 \le \mathbb{K}'' \le \lfloor |\mathfrak{R}_i| / (p+1) \rfloor$ on $\mathfrak{R}_i$. Find the minimum value for $\mathbb{K}''$ as by which, for every subdivided subcluster $\mathfrak{R}_{i,j}, 1 \le j \le \mathbb{K}''$, we have $SingCheck\left(\Sigma_{\mathfrak{R}_{i,j}}\right) \equiv 0, CohrCheck\left(\mathfrak{R}_{i,j}\right) \equiv 1$ and $det_{\Sigma_{\mathfrak{R}_{i,j}}} \le \vec{\delta}(b)$
    if *such* $\mathbb{K}''$ *is not found* **then**
      $\mathfrak{R}_{\zeta} \leftarrow \mathfrak{R}_{\zeta} \cup \mathfrak{R}_i$
      continue
    end
    $[\Gamma] = \text{MiniClustMake}(\Gamma, \{\mathfrak{R}_{i,1}, \cdots, \mathfrak{R}_{i,\mathbb{K}''}\}, \Lambda)$
    Remove $\{\mathfrak{R}_{i,1}, \cdots, \mathfrak{R}_{i,\mathbb{K}''}\}$ from RAM buffer
  else /* Regular minicluster */
    $[\Gamma] = \text{MiniClustMake}(\Gamma, \mathfrak{R}_i, \Lambda)$
    Remove $\mathfrak{R}_i$ from RAM buffer
  end
end
/* Setting unresolved points as retained set */
$\mathfrak{R} \leftarrow \mathfrak{R}_{\zeta}$

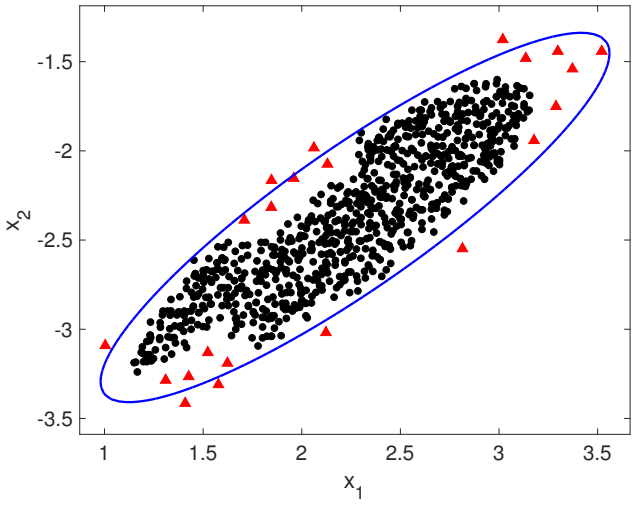


Figure 4: Trying to assign retained objects in buffer to newly created subclusters

that have had the competence to form a cluster are shown with black solid circles, and those which are not a part of the cluster, but reside in its accepted Mahalanobis radius, which is represented by a blue solid line, are denoted as red triangles.

Thus, if retained objects could belong to a newly created subcluster, the corresponding information of that subcluster will be modified w.r.t. Algorithm 4. And if not, such data will be still retained in buffer for further process. But if it was the last chunk which had been processed, all these retained objects will be marked as temporary outliers. But, all of these temporary outliers are not true outlying points. As stated earlier, some of them are normal objects which have not found the competence of forming a minicluster or being assigned to one, due to applied restrictions. However, at last, all of these true and untrue anomalies which are maintained in buffer will be discarded and, this is only the temporary clustering model which is remained after all.

3.2.2. *Building Final Clustering Model*

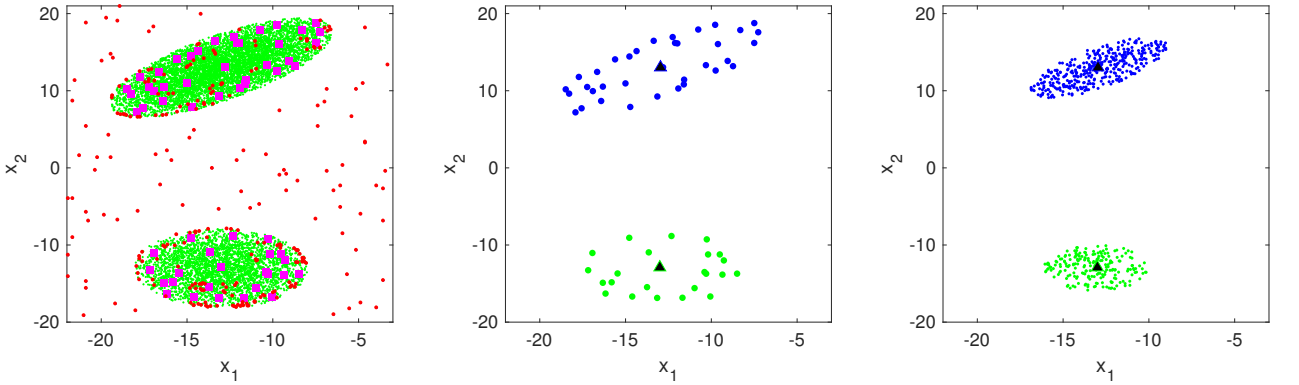
Here, at the end of scalable clustering, it is time to construct the final clustering model or the same approximate structure of $\mathcal{T}$ original clusters, w.r.t. the temporary clustering model, with $\mathcal{L}$ miniclusters. Hence, we follow Algorithm 7, as at the first step, Kmeans algorithm is carried out to cluster the centroids of temporary clusters[13].

After clustering the temporary means, we shall merge the information of associated miniclusters in each of such clusters to obtain final clusters. Here, we presume that the core information of each final cluster only consists of a centroid μ_f, and a covariance matrix Σ_f. Hence, w.r.t. Algorithm 7, if a final cluster contains only one temporary cluster, then the final centroid and the final covariance matrix will be the same as for the temporary cluster. Otherwise, in the case of containing more than one temporary cluster, we utilize sizes and centroids of the associated miniclusters, to obtain the final mean.

However, attaining final covariance matrix is a bit more complex. Hence, for each of the associated miniclusters and w.r.t. its centroid and covariance structure, we afford to regenerate a specific amount of fresh data points with Gaussian distribution. We define the regeneration size of each minicluster equal to product of sampling rate, which was used at the "Sampling" stage, times cardinality of that minicluster, as $\eta \cdot |X_j|$. And this is necessary for saving free space in memory, while regenerating approximate structure of an original cluster. We consider all regenerated objects of all subclusters belonging to a final cluster, as an unique and coherent cluster and afford to obtain the final covariance structure out of it.

But before using the covariance matrix of such regenerated cluster, we need to eliminate the effect of some generated outliers, which could be created unavoidably during the re-

[13] Since, the original clusters are convex and also, the miniclusters are developed among their space, thus, using Kmeans here seems reasonable. Therefore, there is no need to utilize a density-based clustering method, for the case of non-convex clusters.



(a) After Processing the Last Chunk (b) Temporary Means and Final Means (c) Regenerated Objects and Final Means

Figure 5: Proposed method appearance at the last steps of scalable clustering

Algorithm 7: $[_F] = \text{FinalClustBuild}(\Gamma, \eta, \beta)$

Input : Γ - Current array of miniclusters information; η - Random sampling rate; β - Pruning threshold

Output : $_F$ - Final clustering model

Consider mean vectors of whole temporary clusters X_j's, $1 \le j \le \mathcal{L}$, as a matrix $[\mathcal{M}]_{\mathcal{L}\times p}$, and apply Kmeans with $\mathbb{K} = \mathcal{T}$ to cluster them. Assume the result of such clustering as $\{\mathcal{M}_1, \cdots, \mathcal{M}_\mathcal{T}\}$

$_F \leftarrow \Phi$

foreach $\mathcal{M}_i, 1 \le i \le \mathcal{T}$ **do**

  if $|\mathcal{M}_i| \equiv 1$ **then** /* Isolated minicluster */

    Use the same mean location and covariance structure of the isolated subcluster, as for those of the final cluster

  else /* Group of miniclusters */

    /* Calculating final mean location */

    $\mu_f \leftarrow \frac{\sum_{j:\, x_j \in \mathcal{M}_i} [\Gamma_j\{6\} \cdot \Gamma_j\{1\}]}{\sum_{j:\, x_j \in \mathcal{M}_i} \Gamma_j\{6\}}$

    /* Calculating final covariance structure */

    $\mathcal{U} \leftarrow \Phi$

    foreach $X_j \in \mathcal{M}_i$ **do**

      Regenerate $\eta \cdot \Gamma_j\{6\}$ number of points, with Gaussian distribution, due to $\Gamma_j\{1\}$ as the temporary mean and $\left(\frac{1}{\Gamma_j\{6\}-1}\right) \cdot \Gamma_j\{2\}$ as the temporary covariance matrix

      Add these regenerated points to $\mathcal{U}$

    end

    Calculate Mahalanobis distance of points in $\mathcal{U}$ based upon sample mean $\mu_\mathcal{U}$, and sample covariance matrix $\Sigma_\mathcal{U}$

    Prune $\mathcal{U}$ by discarding those points with Mahalanobis distance more than $\beta \cdot \sqrt{p}$ and recalculate $\Sigma_\mathcal{U}$

    $\Sigma_f \leftarrow \Sigma_\mathcal{U}$

    Remove $\mathcal{U}$ from buffer

    /* Adding information to final clustering model */

    $_F\{1, i\} \leftarrow \mu_f$

    $_F\{2, i\} \leftarrow \Sigma_f$

  end

end

generation process, and can potentially prejudice the final accuracy outcomes. For this purpose, we need to prune this transient final cluster, according to the Mahalanobis threshold β, obtained through the user. Thus, regenerated objects having a Mahalanobis distance more than $\beta \cdot \sqrt{p}$, from regenerated cluster, regarding its unpruned centroid and covariance matrix, will be obviated. Now, it is just required to compute the covariance matrix of such pruned regenerated and transient final cluster, to obtain the ultimate covariance matrix. The final regenerated cluster will be discarded after achieving rough properties of the original cluster. We conduct this procedure in sequence for every final cluster which consists of more than one temporary cluster.

Fig. 5a demonstrates a dataset consisting of two dense Gaussian clusters with some local outliers around them. This figure is in fact, a sketch of what the proposed method looks like at the final steps of scalable clustering and before building the final clustering model. The green dots are normal objects belonged to a minicluster. The red dots are temporary outliers, and the magenta square points represent the temporary centroids. Fig. 5b demonstrates both temporary means and final means, represented by solid circles and triangles respectively, with a different color for each final cluster. Fig. 5c colorfully demonstrates pruned regenerated data points for every final cluster beside the final means, denoted as dots and triangles respectively.

Here, after obtaining the final clustering model, the second phase of proposed approach is finished. In the following subsection, the third phase named "Scoring" is presented, and we will describe how to give each object, a score of outlierness, w.r.t. the final clustering model obtained out of the scalable clustering.

3.3. Scoring

At this phase, w.r.t. the final clustering model which was obtained through scalable clustering, we give each data point an outlying rank. Therefore, like phase 2, once more, we need to process the whole dataset in chunks, and use the same Mahalanobis distance criterion to find the closest final cluster to each object, and assign this least distance to the object as its outlying score. The higher the distance, the more likely it is that the object is an outlier. Such least Mahalanobis distance is also referred to as the local Mahalanobis distance [2]. Besides, from the clustering perspective, each object will be assigned to a cluster with least Mahalanobis distance from

it. Here, we name such score obtained out of our proposed approach, SDCOR, which stands for "Scalable Density-based Clustering Outlierness Ratio"[14].

3.4. Algorithm Complexity

Here, at first, we analyze the time complexity of the proposed approach. For two first phases, "Sampling" and "Scalable Clustering", the most expensive operations are applying DBSCAN on objects residing in memory after loading each chunk, and applying PCA on each minicluster to obtain and update its secondary information. Let $n_{\mathfrak{C}}$ be the number of data points in a chunk, regarding our non-reported experiments and according to this concern that w.r.t. the three-sigma rule of thumb, in every memory-load of points, the majority of these points lie in the accepted Mahalanobis neighborhood of current temporary clusters and are being assigned to them (and this will escalate over time by the increasing number of subclusters, which are being created during each memory process), and also by utilizing an indexing structure for k-NN queries, the time complexity of DBSCAN algorithm will be $O\left(\kappa n_{\mathfrak{C}} \log\left(n_{\mathfrak{C}}\right)\right)$, which in that, κ is a low constant almost zero. Applying PCA on miniclusters is $O\left(min\left(p^3, n_{\mathfrak{C}}^3\right)\right)$ [25], as p stands for dimensionality of the input dataset. But according to our strong assumption that $p < n_{\mathfrak{C}}$, thus, applying PCA will be $O\left(p^3\right)$. Hence, the two first phases of the algorithm will totally take $O\left(max\left(\kappa n_{\mathfrak{C}} \log\left(n_{\mathfrak{C}}\right), p^3\right)\right)$. The last phase of the algorithm, w.r.t. this concern that only consists of calculating the Mahalanobis distance of any of total n objects in the input dataset to $\mathcal{T}$ final clusters, is $O(n\mathcal{T})$, and regarding that $\mathcal{T} \ll n$, hence, the time complexity of this phase will be $O(n)$. The overall time complexity is thus at most $O\left(max\left(\kappa n_{\mathfrak{C}} \log\left(n_{\mathfrak{C}}\right), p^3\right) + n\right)$. But, it is evident that both p and $n_{\mathfrak{C}}$ values are trivial w.r.t. n, and therefore, we can state that the time complexity of our algorithm is linear.

Analysis of the algorithm space complexity is twofold. First, we consider stored information in memory for clustering models, and second, the amount of space required for processing resident data in RAM. With respect to the fact that the most voluminous parts of the temporary and final clustering models are the scatter and covariance matrices respectively, and regarding the truth that $\mathcal{L} \gg \mathcal{T}$, thus, the space complexity of the first part will be $O\left(\mathcal{L}p^2\right)$. For the second part, according to this matter that in each memory-load of points, the most expensive operations belong to clustering algorithms DBSCAN and Kmeans, and regarding linear space complexity of these methods, hence, the overall space complexity will be $O\left(n_{\mathfrak{C}} + \mathcal{L}p^2\right)$.

4. Experiments

In this section, we conduct effectiveness and efficiency tests to analyze the performance of the proposed method. The following experiments were implemented with MATLAB 9, and run on a laptop with Intel Core i5 processor (clocked at 2.5 GHz) and 6 G memory. To test effectiveness, we compare our method with competing methods on synthetic and real-life datasets. For efficiency test, we conduct evaluations on synthetic datasets for compared methods, to show how final accuracy results varies with the increasing number of outliers. Moreover, there are some other tests on the scalability of proposed algorithm and also, about the effect of random sampling rate on the final detection accuracy. Finally, for reproducibility, we publish our code on GitHub[15].

4.1. Compared Methods and Experiment Outline

For our experiments, we implement and compare our algorithm with some other mainstream methods for local outlier detection in datasets with numerical attributes. These representative methods include LOF and LoOP from density-based approaches. Various experimental results are reported in this section. For evaluating the efficacy of the proposed method, we compare it with other mentioned competing methods on diverse real-life datasets. Furthermore, we conduct experiments on some synthetic datasets to illustrate the effectiveness and stability of the proposed method for large-scale datasets. In sequel, to evaluate the efficiency, synthetic datasets with diverse numbers of outliers are employed to demonstrate the behavior of detection outcomes of compared methods, in complex conditions. In addition to effectiveness and efficiency tests, we will also provide the diagram of time consumption per increasing number of objects, to prove scalability of our approach. Moreover, the diagram of variation of the covariance determinant of sampled data of a unique cluster per sampling rate is presented, to illustrate that even with very low rates of random sampling, it is still possible to anticipate significant accuracy results from our proposed method.

4.2. Effectiveness Test

Here, evaluation results of experiments on various real and synthetic datasets with a diversity of size and dimensionality are presented, to illustrate the efficacy and stability of the proposed approach.

4.2.1. Test on Real Datasets

A few number of public and large-scale real benchmark datasets, all of them from UCI [12], and preprocessed and labeled by ODDS [34], are used in our experiments, representing a various range of domains in science and humanities. In all of our experiments, the Area Under the Curve (AUC) (curve of detection rate and false alarm rate) [9, 20] is used to evaluate the detection performance of compared algorithms. The AUC and runtime results[16] of different methods along

[14] Due to high computational loads associated with calculating Mahalanobis distance in high-dimensions, one can still gain benefit of using properties of principal components for computing outlying scores, like what we did during scalable clustering.

[15] https://github.com/sana33/Researches/tree/master/SDCOR

[16] For proposed method, as we know the true structural characteristics of all real and synthetic data, we compute the accurate anomaly score, based on Mahalanobis distance criterion, for each object and report the following optimal AUC next to the attained result by SDCOR. Moreover, we do not account the required time for finding optimal parameters of DBSCAN, as a part of total runtime.

Table 2
AUC and Runtime Results of Tested Algorithms on Real and Synthetic Datasets

	Data Specifications				AUC			Time(Secs)		
	Dataset	#n	#p	#o (%o)	LOF	LoOP	SDCOR (Optimal)	LOF	LoOP	SDCOR
Real Datasets	Shuttle	49,097	9	3,511 (7.15%)	0.602	0.553	**0.973** (0.994)	303.92	354.10	**1.14**
	Smtp (KDDCUP99)	95,156	3	30 (0.03%)	0.874	**0.900**	0.778 (0.815)	1,206.39	1,055.63	**1.47**
	ForestCover	286,048	10	2,747 (0.96%)	0.598	0.550	**0.943** (0.950)	9,005.15	11,907.55	**2.92**
	Http (KDDCUP99)	567,498	3	2,211 (0.38%)	~	~	**0.994** (0.999)	>16h	>16h	**4.96**
real data results average					0.691	0.668	**0.922** (0.939)			
Synth. Datasets	Data1	500,000	30	5,000 (1.00%)	~	~	**1.000** (1.000)	>22h	>22h	**9.54**
	Data2	1,000,000	40	10,000 (1.00%)	~	~	**1.000** (1.000)	>4d	>4d	**26.64**
	Data3	1,500,000	50	15,000 (1.00%)	~	~	**1.000** (1.000)	>1w	>1w	**55.77**
	Data4	2,000,000	60	20,000 (1.00%)	~	~	**1.000** (1.000)	>2w	>2w	**101.34**

with the characteristics of all test datasets, such as the numbers of objects (#n), attributes (#p) and outliers (#o), are summarized in the left part of Table 2. Also, for outliers in each dataset, their share of total objects in corresponding dataset is reported in percentage terms. For some datasets, there is no result for LOF and LoOP, because these methods have a high computational complexity on large-scale datasets and cannot finish the process in reasonable time. The **bold-faced** AUC and runtime indicate the best method for a particular dataset. For competing density-based methods, the parameters are set as suggested, i.e. $MinPtsLB = 10$, $MinPtsUB = 50$ in LOF and $k = 30$, $\lambda = 3$ in LoOP. For proposed method, we prefer to process each dataset in 10 chunks.

Moreover, as long as we are trying to not let outliers participate actively in forming and updating miniclusters, thus, two of input parameters of proposed algorithm will be more influential than others. One of them is random sampling rate η, which by that, parameter $\vec{\delta}$, the boundary on the volume of miniclusters which are being created during scalable clustering for the first time ever, is established. The second influential input parameter is membership threshold α, which is again for restraining the volume of miniclusters, but while growing over time, during scalable clustering.

For η, it should not be set too low, as by which, singularity problem might happen during "Sampling" phase, or even some original clusters may not take initial form, for the lack of enough data points[17].

Furthermore, for α, it should not be established very low too, as by that, the number of subclusters which are being created during scalable clustering will become too large, which leads to much higher computational loads. And besides, it should not be set so high, hence, the risk of outliers getting joined to normal subclusters will escalate over time, which increases the "False Negative" error. Here, in all experiments, α is set to 2.

Now, let us take a look at the comparison between our proposed method and the competitors. The AUC results in Table 2 reveal that our proposed approach is more effective than LOF and LoOP. The table demonstrates that SDCOR outperforms these methods on almost 90 percent of all datasets. Even in only one case that SDCOR does not produce a better result than the best reported one, we observe that it is close enough to the best result. Furthermore, it is obvious that the attained results by the proposed approach are almost the same as the optimal ones. The average row of the AUC value also indicates that SDCOR performs much better overall than all other methods. More importantly, SDCOR is effective on the largest dataset $Http$. Moreover, Table 2 shows that in terms of execution time, there is a huge difference between SDCOR and other competing methods. This is due to the fact that in SDCOR, it is not required to compute pairwaise distances of total objects in a dataset, as in other density-based methods, LOF and LoOP.

As stated earlier at the beginning of this paper, the strong assumption of SDCOR is on the structure of the existing clusters in the input dataset, which should have Gaussian distribution. In practice though, w.r.t. [37], quite a lot of real world data are Gaussian distributed — thanks to the Central Limit Theorem. Furthermore, w.r.t. [26, 18, 5], even when original variables are not Normal, employing properties of PCs for detecting outliers is possible and the corresponding results will be reliable. Since, given that PCs are linear functions of p random variables, an appeal to the Central Limit Theorem may justify approximate Normality for the PCs, even when the original variables are not Normal. Following this issue, it is possible to set up more formal tests for outliers based on PCs, assuming that the PCs are normally distributed. Moreover, the use of Mahalanobis distance criterion for outlier detection is only viable for convex-shaped clusters. Otherwise, outliers could be assigned to an irregular (density-based) cluster under masking effect and thus, will be misclassified.

4.2.2. Test on Synthetic Datasets

We also compare the effectiveness of different methods on synthetic datasets in a relatively ideal setting, since these datasets are following strong assumptions of our algorithm on the structure of existing clusters, and also, the generated outliers are usually more distinctive than those in real data,

[17] Hence, one can state that there is a straight relationship between the random sampling rate, and the true number of original clusters in data. In other words, for datasets with a high frequency and variety of clusters, we are forced to take higher ratios of random sampling, to avoid both problems of singularity and misclustering, in the "Sampling" stage. Here, in all efficacy experiments, regarding our preknowledge about real and synthetic data, we have set η to 0.5%.

and the outliers "truth" can be used to verify whether an outlier algorithm is capable of finding them. Four experiments are reported at the bottom part of Table 2, where each dataset consists of 6 Gaussian clusters, and outliers take up 1 percent of its volume.

Here, we are about to describe the process of generating synthetic datasets in details. For each dataset with p dimensions, we shall create arbitrary Gaussian clusters, which for any of them we need to create an arbitrary mean vector and an approved arbitrary covariance matrix. For the mean vector, we create its elements randomly in the space, as it would be far enough away from other means, to hinder possible unwanted overlappings between multidimensional clusters. For the covariance matrix, at first, we create a matrix $[\mathbb{A}]_{p\times p}$, whose elements are uniformly distributed in the interval [0,1]. Then, we randomly select half of the elements in $\mathbb{A}$ and make them negative. At last, the corresponding covariance matrix is obtained in the form of $\left[\sum\right]_{p\times p} = \mathbb{A}^T\mathbb{A}$. Now, w.r.t. to the mean vector (location) of the cluster and its covariance matrix (shape of the cluster), we can generate arbitrary numbers of data points from p-variate Gaussian distribution. Moreover, to eliminate marginal noisy objects of each cluster, once again, we can get the benefit of Mahalanobis distance criterion and purge objects outside of the Mahalanobis radius, e.g., 1.

For injecting local outliers to each cluster, first, we consider a hypercube covering the boundaries of the corresponding cluster in each dimension, with the same centroid as of the cluster, and having a specific amount of vacant space around it. Then, we randomly generate records in this space and accept them as local outliers if they fall in the accepted Mahalanobis distance interval of the cluster, as it could be $\left[4\sqrt{p}, 10\sqrt{p}\right]$. Due to the fact that the volume of the hypercube increases so rapidly by p for high-dimensions, we need to extend the accepted interval for generated points to save the time consumed for generation. Hence, some of these generated outliers will be more general outliers than local, and this is inevitable. Although as it was mentioned earlier, the concept of local outliers is covering that of global ones, hence, each global outlier could be presumed as a local outlier, but not vice versa.

The results in Table 2 verify that synthetic datasets are in general too easy for SDCOR, as it always achieves perfect results, since, in such datasets, normal objects are totally in very dense areas and outliers are in very sparse zones and far enough away from normal clusters. For LOF and LoOP, as stated before, because of their high computational load, they cannot finish very large datasets in reasonable time, thus, no results for synthetic datasets are provided.

4.3. Efficiency Test

To measure the final accuracy result with increasing number of outliers, we follow the same procedure to generate synthetic datasets, as was described in details in the last subsection. Hence, for this test, we assume the percentage of outliers in a dataset is increased from 50 to 150 percent with the step length of 10 percent. The numbers of normal objects and attributes for any of these datasets are constant values 20,000 and 2 respectively. Also, the intrinsic manifold of normal objects in all these datasets is the same, which consists of 4 Gaussian clusters. It is worth noting that certainly, in a real data, not by any means, number of outliers is this huge, and we do this just to demonstrate the robustness of our proposed approach.

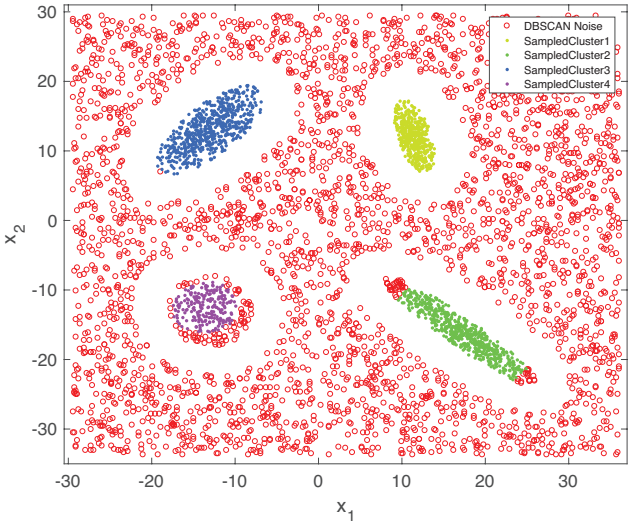


Figure 6: Result of DBSCAN with optimal parameters on high noisy sampled data

Our efficiency experiments indicate that SDCOR is totally noise tolerant, as by the increasing percentage of outliers, always perfect AUC results are achieved out of that. However, for LOF and LoOP, they are completely misclassifying outliers in all complex conditions. The reason for this is that in SDCOR, the basis for forming miniclusters, during scalable clustering, is the fulfilment of DBSCAN requirements, which are obtained out of "Sampling" stage. As all normal objects follow Gaussian distribution and outliers are injected using a continuous uniform distribution, hence, the local density of normal points is much higher than that of outliers. Therefore, the acquired optimal parameters for density-based clustering are obtained proportional to the dense regions containing only sampled inliers. For this reason, in each memory process, the probability of a subcluster being formed by outliers is very less than that of normal objects.

Fig. 6 shows DBSCAN result on sampled data of the test dataset with 150 percent injected outliers. Four discovered Gaussian clusters and noises are represented with dots in different colors, and red empty circles, respectively. As it is evident, even at this highly noisy situation, outliers cannot satisfy DBSCAN constraints on forming a minicluster.

Here, it is worth remarking that w.r.t. our non-reported evaluations on other efficiency tests, when the numbers of normal objects and attributes are increasing (like what we did in the "percentage of outliers" test), under this guarantee that the singularity will not happen during clustering procedures, one can still anticipate significant detection results out of our proposed approach. But for LOF and LoOP, w.r.t. the

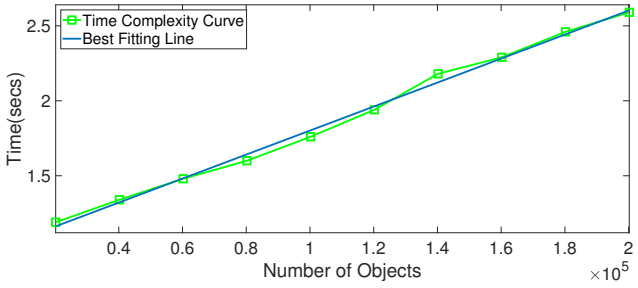


Figure 7: Result of scalability test

corresponding high computational load, it would be a tedious task to obtain accuracy results out of them for very large datasets employed in our experiments.

4.4. Scalability Test

For proving the scalability of proposed method, we measure the time consumption with increasing number of objects. For this case, first, like before, we generate a synthetic dataset with 200,000 normal objects having 10 attributes, containing 4 Gaussian clusters. Then, we conduct random sampling with sampling rates of 10 to 100 percent with step length of 10 percent, and for each resulting dataset, we inject 200 outliers into it. The reason for this kind of generation is that, we intend to conduct time complexity experiments w.r.t. datasets having very similar basic characteristics, namely the location (centroid) and the shape (covariance matrix) of each cluster. Hence, now, it will be fair enough to compute the consumed time for running the algorithm on any of these datasets and make a deduction out of the resulting diagram[18].

As Fig. 7 illustrates, the run time of SDCOR is almost a linear function of the number of objects, and this is a confirmation of what we discussed in subsection 3.4 on algorithm complexity. Due to corresponding diagram, for the dataset with minimum size equal to 20,200, the run time is 1.19 seconds, but when number of objects goes to its maximum value equal to 200,200, which is almost ten times the minimum value and has the significant difference of 180,000, the processing time increases by almost 2.2 times, to only 2.59 seconds. It shows that SDCOR has a low constant in its runtime complexity, and hence, is scalable.

4.5. Sampling Rate Test

Here, we examine the variation of covariance determinant of sampled data of a unique cluster per various random sampling rates. Therefore, first of all, we create an arbitrary Gaussian cluster with 10,000 objects and 2 attributes. Then, we start sampling with the sampling rate of 0.5 percent and proceed to 100 percent with the step length of 0.5 percent, and then, for each of these resulting sampled clusters, we calculate the corresponding covariance determinant of the cluster.

As Fig. 8a shows, with increase in random sampling rate, the corresponding covariance determinant is approaching that of the original cluster. However, as it is evident, even covariance determinants associated with very low sampling rates are adequately close to that of the main cluster. In example, for the lowest sampling rate equal to 0.5 percent, the corresponding covariance determinant is approximately 300, which is sufficiently near to that of the original cluster, which is roughly 220.

Now, if we plot the tolerance ellipses for both the main cluster and the sampled cluster with the sampling rate of 0.5 percent, we observe that they are so similar to each other. Fig. 8b illustrates such situation in which, objects belonging to original cluster and those belonging to the sampled one are shown with blue dots and red squares respectively. Moreover, tolerance ellipses of the main and the sampled clusters are shown in red and black respectively.

5. Conclusion

In this paper, we have provided a new scalable density-based clustering approach for local outlier detection in massive data, which processes the input data in chunks. First of all, by obtaining a random sample of the entire dataset and applying a density-based clustering algorithm to it, we built the initial temporary clustering model, which contains the rough information of original clusters in data, and then, by consecutive memory-loads of points, we updated the temporary clustering model. Ultimately, after processing the whole chunks, the final clustering model was acquired, which w.r.t. that and conducting another scan of the entire dataset, we gave each object an outlying score.

Our thorough evaluation on both real-world and synthetic datasets demonstrated the appealing performance and effectiveness of SDCOR, comparing with different mainstream density-based outlier algorithms, which need the data to be memory-resident. Moreover, the efficiency outcomes confirm the robustness of proposed method comparing to other methods, in very noisy conditions. Furthermore, experiments on algorithm complexity, verifies that SDCOR has a linear time complexity with a low constant. Besides, the results of sampling rate test show that, even with very low rates of random sampling, one can still anticipate significant outcomes out of our proposed approach. For the future work, we would like to enhance our proposed approach in a way that can deal with density-based non-convex clusters in the input data too.

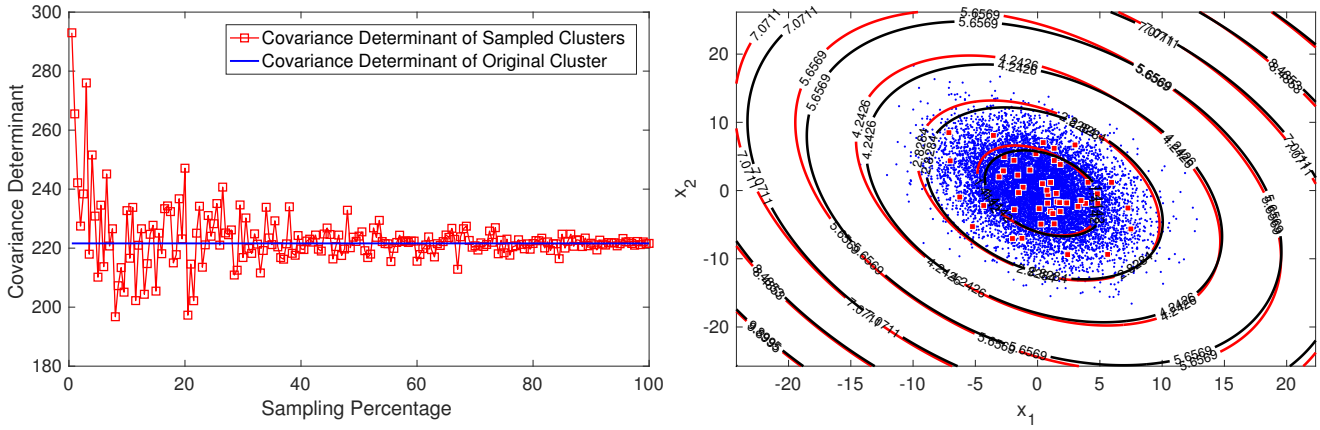
Acknowledgment

Special thanks to Dr. Victoria J. Hodge (University of York, UK), for her clever advice on the title of this paper, and Dr. Mohammad-Mehdi Ebadzadeh (Tehran Polytechnic, Iran), for his valuable comments about some of the algorithms. Also, we appreciate Dr. Ali-Mohammad Saghiri (Tehran Polytechnic, Iran) for his generous review of the paper before submission.

References

[1] Achlioptas, D., 2001. Database-friendly random projections, in: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART sympo-

[18] In all experiments, the parameters are set as suggested in effectiveness test. Moreover, in all cases, perfect detection result is achieved.



(a) Covariance Determinant Variation per Sampling Rate

(b) Tolerance Ellipses for both Original and Sampled Clusters

Figure 8: Covariance determinant and tolerance ellipses variation per random sampling rate

sium on Principles of database systems, ACM. pp. 274–281.

[2] Aggarwal, C.C., 2015. Data mining: the textbook. Springer.

[3] Agyemang, M., Barker, K., Alhajj, R., 2006. A comprehensive survey of numeric and symbolic outlier mining techniques. Intelligent Data Analysis 10, 521–538.

[4] Ayyıldız, E., Purutçuoglu, V., Wit, E., 2012. A short note on resolving singularity problems in covariance matrices. International Journal of Statistics and Probability 1, 113–118.

[5] Barnett, V., Lewis, T., 1974. Outliers in statistical data. Wiley.

[6] Bradley, P.S., Fayyad, U.M., Reina, C., et al., 1998. Scaling clustering algorithms to large databases., in: KDD, pp. 9–15.

[7] Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J., 2000. Lof: identifying density-based local outliers, in: ACM sigmod record, ACM. pp. 93–104.

[8] Cabras, S., Morales, J., 2007. Extreme value analysis within a parametric outlier detection framework. Applied Stochastic Models in Business and Industry 23, 157–164.

[9] Chandola, V., Banerjee, A., Kumar, V., 2009. Anomaly detection: A survey. ACM computing surveys (CSUR) 41, 15.

[10] Dasgupta, S., Gupta, A., 1999. An elementary proof of the johnson-lindenstrauss lemma. International Computer Science Institute, Technical Report 22, 1–5.

[11] De Vries, T., Chawla, S., Houle, M.E., 2010. Finding local anomalies in very high dimensional space, in: 2010 IEEE International Conference on Data Mining, IEEE. pp. 128–137.

[12] Dua, D., Graff, C., 2017. UCI machine learning repository. URL: http://archive.ics.uci.edu/ml.

[13] Duan, L., Xu, L., Liu, Y., Lee, J., 2009. Cluster-based outlier detection. Annals of Operations Research 168, 151–168.

[14] Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al., 1996. A density-based algorithm for discovering clusters in large spatial databases with noise., in: Kdd, pp. 226–231.

[15] Filzmoser, P., Maronna, R., Werner, M., 2008. Outlier identification in high dimensions. Computational Statistics & Data Analysis 52, 1694–1711.

[16] Forgey, E., 1965. Cluster analysis of multivariate data: Efficiency vs. interpretability of classification. Biometrics 21, 768–769.

[17] Han, J., Pei, J., Kamber, M., 2011. Data mining: concepts and techniques. Elsevier.

[18] Hawkins, D.M., 1980. Identification of outliers. volume 11. Springer.

[19] He, Z., Xu, X., Deng, S., 2003. Discovering cluster-based local outliers. Pattern Recognition Letters 24, 1641–1650.

[20] Hodge, V., Austin, J., 2004. A survey of outlier detection methodologies. Artificial intelligence review 22, 85–126.

[21] Hubert, M., Debruyne, M., 2010. Minimum covariance determinant. Wiley interdisciplinary reviews: Computational statistics 2, 36–43.

[22] Hubert, M., Rousseeuw, P.J., Vanden Branden, K., 2005. Robpca: a new approach to robust principal component analysis. Technometrics 47, 64–79.

[23] Jin, W., Tung, A.K., Han, J., Wang, W., 2006. Ranking outliers using symmetric neighborhood relationship, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer. pp. 577–593.

[24] Johnson, W.B., Lindenstrauss, J., 1984. Extensions of lipschitz mappings into a hilbert space. Contemporary mathematics 26, 1.

[25] Johnstone, I.M., Lu, A.Y., 2009. Sparse principal components analysis. arXiv preprint arXiv:0901.4392 .

[26] Jolliffe, I., 2011. Principal component analysis. Springer.

[27] Kennedy, J., 2010. Particle swarm optimization. Encyclopedia of machine learning , 760–766.

[28] Knox, E.M., Ng, R.T., 1998. Algorithms for mining distancebased outliers in large datasets, in: Proceedings of the international conference on very large data bases, Citeseer. pp. 392–403.

[29] Kriegel, H.P., Kröger, P., Schubert, E., Zimek, A., 2009. Loop: local outlier probabilities, in: Proceedings of the 18th ACM conference on Information and knowledge management, ACM. pp. 1649–1652.

[30] Leskovec, J., Rajaraman, A., Ullman, J.D., 2014. Mining of massive datasets. Cambridge university press.

[31] Mahalanobis, P.C., 1936. On the generalized distance in statistics, National Institute of Science of India.

[32] Maronna, R.A., Zamar, R.H., 2002. Robust estimates of location and dispersion for high-dimensional datasets. Technometrics 44, 307–317.

[33] Pearson, K., 1901. Liii. on lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 2, 559–572.

[34] Rayana, S., 2016. ODDS library. URL: http://odds.cs.stonybrook.edu.

[35] Ro, K., Zou, C., Wang, Z., Yin, G., 2015. Outlier detection for high-dimensional data. Biometrika 102, 589–599.

[36] Schölkopf, B., Platt, J.C., Shawe-Taylor, J., Smola, A.J., Williamson, R.C., 2001. Estimating the support of a high-dimensional distribution. Neural computation 13, 1443–1471.

[37] Shlens, J., 2014. A tutorial on principal component analysis. arXiv preprint arXiv:1404.1100 .

[38] Tang, B., He, H., 2017. A local density-based approach for outlier detection. Neurocomputing 241, 171–180.

[39] Teng, S.H., et al., 2016. Scalable algorithms for data and network analysis. Foundations and Trends® in Theoretical Computer Science 12, 1–274.

[40] Wu, S., Wang, S., 2011. Information-theoretic outlier detection for large-scale categorical data. IEEE transactions on knowledge and data engineering 25, 589–602.

[41] Yin, J., Ho, Q., Xing, E.P., 2013. A scalable approach to probabilistic latent space inference of large-scale networks, in: Advances in neural information processing systems, pp. 422–430.

[42] Zhang, K., Hutter, M., Jin, H., 2009. A new local distance-based outlier detection approach for scattered real-world data, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer. pp. 813–822.

[43] Zimek, A., Schubert, E., Kriegel, H.P., 2012. A survey on unsupervised outlier detection in high-dimensional numerical data. Statistical Analysis and Data Mining: The ASA Data Science Journal 5, 363–387.