ICIC Express Letters
Part B: Applications
Volume **11**, Number **6**, June **2020**

ICIC International ©2020 ISSN 2185-2766
pp. **615–622**

EMPIRICAL STUDIES ON APPLYING DENSITY-BASED CLUSTERING TO STREAM DATA

Yijin Kim[1], Jung-Eun Park[1], Jonghem Youn[2] and Junho Shim[1,*]

[1]Department of Computer Science
Sookmyung Women's University
100 Cheongpa-ro 47-gil, Yongsan-gu, Seoul 04310, Korea
{ yijin3018; je.park }@sookmyung.ac.kr; *Corresponding author: jshim@sookmyung.ac.kr

[2]Voost Inc.
Seoul 06232, Korea
jonghm@voostlab.com

Received December 2019; accepted March 2020

Abstract. *Density-based clustering has advantages over partition-based clustering, such as K-means, in that it does not need to specify the number of clusters (k) and can generate clusters of arbitrary shape. However, density-based clustering requires hyper-parameters such as proximity distance and the minimum number of proximity data that are suitable for data characteristics, and this greatly influences the clustering performance. In this paper, we present a density-based clustering algorithm for stream data, which exploits coresets in the sliding window model. We provide an experimental analysis of these hyper-parameters on the performance of the algorithm.*
Keywords: Data stream, Density-based clustering, Sliding window

1. **Introduction.** As a huge amount of data are generated and many industry sectors increasingly use and apply them in real time, clustering of stream data is becoming more critical. Processing stream data is essential in the area of generating and processing big data in real time, such as in the fields of transportation vehicles, and industrial equipment, where new dynamic data are continuously generated [1]. For example, self-driving vehicles determine their inside and outside situations in real time through various sensors, such as a camera, GPS, and radar [2]. Data stream clustering is an essential step in making a data summary for such large-scale real-time data analysis. However, various algorithms have been suggested, because it is difficult to apply a very complex clustering algorithm directly to a large number of data.

Previous studies have proposed various data stream clustering methods for real-time data processing [3,4]. Most studies have proposed methods based on partition-based clustering. Partition-based clustering, such as K-means [5] and K-medoids [6], has advantages in that it is straightforward to apply it to large-scale data, because the algorithm is simple. It is also relatively easy to interpret the cluster results because it generates only k clusters. However, applying these traditional partition-based clustering methods to data streams has the following disadvantages. 1) Performance deteriorates when choosing an inappropriate k value. 2) Because it is based on the Euclidean distance, noncircular clusters cannot be generated. 3) There is a large variation in performance depending on the distance measure for high-dimensional data.

Density-based data clustering [7,8] finds clusters with geometric shapes by connecting clusters according to density. It has the advantages of not having to specify the number of clusters and being able to reduce the decrease in the clustering performance. Most algorithms in density-based clustering use a damped window model [17,18]. In a damped

DOI: 10.24507/icicelb.11.06.615

window model, the data received once are not removed from the window afterwards; there is inefficiency in that the whole data must be accessed even though the influence of the obsolete data is reduced. Only a few such as [11,19] employ a sliding-window model but yet utilize partition-based clustering algorithms[1].

In this work, we propose an efficient density-based data stream clustering algorithm that utilizes a sliding window. The algorithm applies a well-known density-based spatial clustering of applications with noise (DBSCAN) [8] to data streams and additionally combines locality-sensitive hashing (LSH) [9,11]. We also present the importance of appropriate parameter setting according to the dataset, because the parameter configuration of the density-based clustering method influences the performance of the algorithm.

The structure of this work is as follows. Section 2 presents the related work. Section 3 presents the proposed algorithm. Section 4 reports experiments conducted to find optimal parameters, and the clustering performances are analyzed by comparing the proposed method with traditional methods. Section 5 concludes and provides the future work.

2. **Problem Statement and Preliminaries.**

2.1. **Partition-based clustering vs. density-based clustering.**

The K-means clustering [5] generally divides data into k groups by calculating their proximity to each other according to the Euclidean distance. The k groups have their own center points, and each of the data belongs to the cluster of the closest center point among various center points. Despite its simplicity, there is an issue that k value should be required as a hyper-parameter, and it affects the clustering performance greatly. Moreover, the initial start cluster is randomly determined, so it may not always provide reliable clustering. A clustering algorithm presented in our previous work [11] is also based on K-means and is shown in Figure 1(c). To compensate for the weakness of K-means, K-medoids [12] uses the medoid that is the object at the very center of a cluster. After specifying k medoids, a cluster is built by calculating similarities. It may be stronger in noises and outliers, but it has high computational cost.

Unlike partition-based clustering, density-based clustering is divided into a cluster as a high-density region and a noise as a low-density region, and clusters of arbitrary shape can be found. It uses *eps* (radius or epsilon), which is the distance from the center point of a cluster, and *minPts*, which is the minimum number of points within the radius that can form the cluster. DBSCAN [8], the most commonly used method of density-based clustering, has the advantage of not specifying the number of clusters, unlike K-means. However, it has the disadvantage of specifying *eps* and *minPts* in advance, and its performance varies greatly depending on these two parameters.

2.2. **CSCS: Coreset-based clustering with sliding windows.**

The sliding window [10] means continuously processing data of a limited size to handle large data streams. It removes the data expired over time and adds new tuples to the window. Clustering on it targets these data, and updating it should also update the clustering results. This characteristic enables the sliding window in data stream clustering to reflect the recent elements of the streams [12]. Generally, the size of the sliding window is referred to as a "range", and the update unit of the window is referred to as a "slide". Figure 1(a) shows the process of sliding window.

The coreset-based clustering with sliding windows (CSCS) [11], a two-level clustering method for stream data, generates coresets to group using LSH in the sliding window in Level 1 and generates clusters in Level 2 by applying K-means with coresets generated in Level 1. To reduce the number of generated buckets, it iterates the process until the specified number of clusters is reached. In addition, if updating with new data is

[1]Readers who are interested in different window models on stream data clustering may refer to [14].

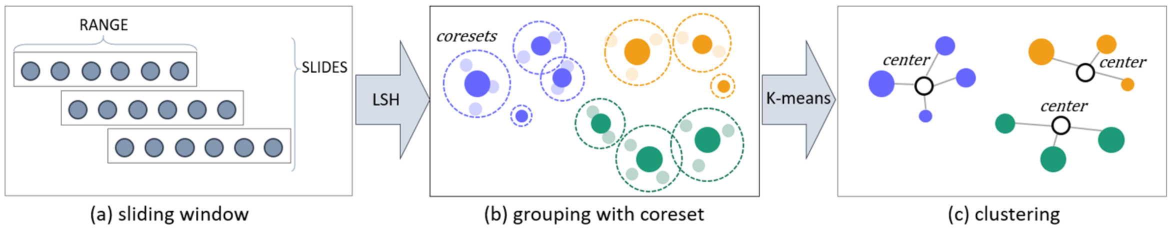

FIGURE 1. Two-step clustering with sliding window

needed, it determines whether to add to existing clusters or to generate new clusters using Kullback-Leibler divergence [13]. This algorithm is illustrated in Figure 1.

3. **Proposed Algorithm.** The proposed algorithm is density-based clustering using coresets and is based on the DBSCAN [8] algorithm for the sliding window. It is also based on the two-level coreset method of CSCS [11], which generates coresets based on LSH prior to clustering, as seen in Figure 1(b). LSH (locality sensitive hashing) can hash similar input values into the same buckets by probability. Adjacent buckets in the hash table may be merged to reduce the size of the hash table. Level 2 performs a density-based clustering with coresets generated in Level 1. It determines neighbors and corrects clusters by comparing the distance between the center point of each coreset with the *eps* value. The proposed algorithm modifies the original DBSCAN in terms of process of finding neighbors.

Algorithm 1 is a pseudocode that shows this process. Unlike DBSCAN, we define GF, if it exists or overlaps within the epsilon-weighted distance, as a neighbor. *Algorithm 1* finds all neighbors within the epsilon distance for the coreset point *cp*. If neighbors are greater than the number of *minPts*, a new cluster is created and assigned for *cp*. The cluster is expanded by iterating this process, which is *Algorithm 2*. If all *cp*s are visited while expanding clusters, the algorithm completes. The algorithm works in place of K-means in the process from (b) to (c) in Figure 1. We call this proposed algorithm "Den-CS".

Algorithm 1. Den-CS

```
Input: coreset $cp$, $\varepsilon$, $m$
Output: cluster $C$
1:  function Den-CS (coreset point $cp$, eps $\varepsilon$, minPts $m$)
2:    for each $cp \in$ CP do
3:      if $cp$ not visited then
4:        neighbor N←GFs with distance $\varepsilon$ from $cp \cup$ overlapped GFs of $cp$
5:        if N >= $m$ then
6:          create new cluster $c$
7:          $expandCluster(cp, N, c, \varepsilon, m)$
8:        end if
9:      end if
10:   end for
11:   return $C$
12: end function
```

4. **Experiment.**

4.1. **Environment.** Experiments were conducted on Ubuntu 18.04.2 LTS with Intel Core i7-4790 CPU 16.00-GB RAM based on java version "1.8.0_191".

Algorithm 2. ExpandCluster

```
1:  function expandCluster(cp, N, c, $\varepsilon$, m)
2:     for each $q \in$ N do
3:        if $q$ not visited then
4:           mark $q$ as visited
5:           neighbor N2←GFs within distance from $cp$ ∪ overlapped GFs of $cp$
6:           if N2 >= $m$ then
7:              add $q$ as N
8:           end if
9:           if $q$ not assigned to existing clusters
             then
10:             add $q$ as cluster $c$
11:          end if
12:       end if
13:    end for
14: end function
```

The three datasets used in the experiments are covtype, knews, and knewsft. Covtype[2] contains tree observation data as cartographic variables in four areas in the Roosevelt National Forest of Colorado. This dataset has seven categories of 54 dimensions with a size of 72 MB. The knews dataset has 17 categories consisting of a set of news documents in Korean. A knewsft dataset was built by using the FastText library and converting this dataset [14].

We evaluated the efficiency of the clustering algorithm by measuring both the total processing time and the quality of the clusters. Note that depending on the characteristic of dataset or the application domain of clustering algorithm, there may be different measures applied to evaluating the clustering quality. Among various measures, we use *purity* in that all three datasets contain documents that are classified into categories [14,15]. It is also used as a density-based clustering quality measure in other works such as [19]. In our experiment, we calculate $purity(C, G)$ by $\frac{1}{N}\sum_{i=1}^{k}\max_j |c_i, g_j|$, where C and G are a set of clusters c_i, and a set of categories g_j, respectively.

4.2. **Experimental analysis.** The experiments were conducted by changing the number of *minPts* or the *eps* value for three datasets to find their optimal parameters. In order to choose proper candidate parameter values for their optimal ones, we followed a heuristic method proposed in [16]. The method can be summarized as follows. First, the parameters proposed in [11] were set as the initial seeds to search for candidate parameters; *minPts* was set to 3 and *eps* to 380, 0.0189, and 0.205 for the covtype, knews, and knewsft, respectively. Experiments were conducted to measure the execution time of the DBSCAN algorithm while changing the numbers to an arbitrary range of numbers to find optimal values. The experiments were performed on each dataset while changing *eps* to 2, 5, 10, 0.5, 0.1, 0.05, and 0.01 times the reference value and *minPts* to 3, 5, 10, and 20. We set candidate parameters to make the algorithm run in less processing time than the average processing time of DBSCAN. Then, we make pairs of (*minPts*, *eps*) with candidate parameters. After making pairs of them, we conduct the experiment with DBSCAN and Den-CS algorithms to calculate the *purity* for each candidate parameter pair. From these results of the experiment, the optimal parameters are selected as a pair with the highest *purity* and the shortest processing time for each dataset. Finally, applying those optimal parameters to both DBSCAN and Den-CS algorithms, we compare the processing time and show the difference of performance of both algorithms.

[2]https://archive.ics.uci.edu/ml/datasets/covertype

Figure 2 shows the measurement results of the execution time of the DBSCAN algorithm for each dataset while fixing *minPts* to 3 and changing *eps*. In the meanwhile, Figure 3 shows the measurement results of the execution time of the DBSCAN algorithm for each dataset while fixing *eps* to the reference value for each dataset and changing *minPts*. Figures 2 and 3 also show the average execution time of the DBCSAN for each dataset in case of the sliding range of 5K, the largest range in our experiment. As summarized earlier, the parameters of data that are below the average are then selected as candidate parameters. For example, in case of the covtype dataset in Figure 2(a), 380, 190, 3.8, and 19 are selected as candidate *eps* values for *minPts* of 3. In case of the knews dataset in Figure 3(b) where *eps* value is fixed, only *minPts* of 3 is selected in that other *minPts* values do not make the algorithm run less than the average processing time.

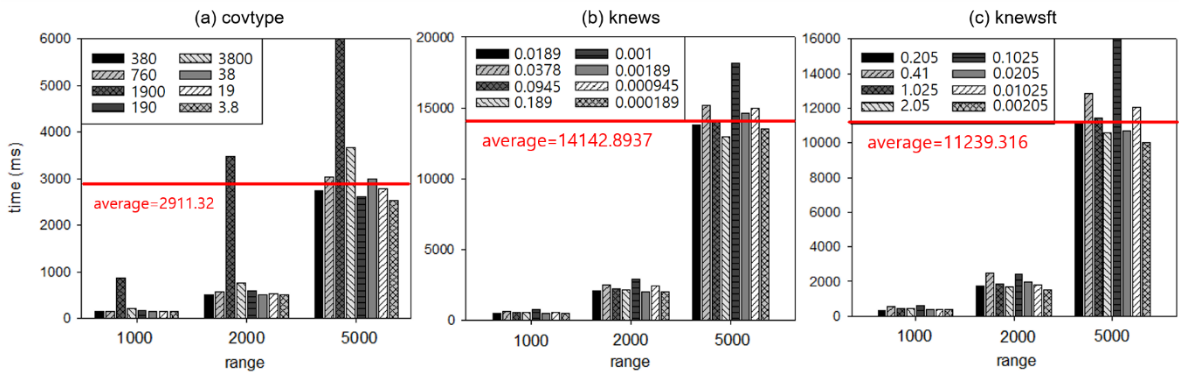

FIGURE 2. Processing time of DBSCAN by various *eps*

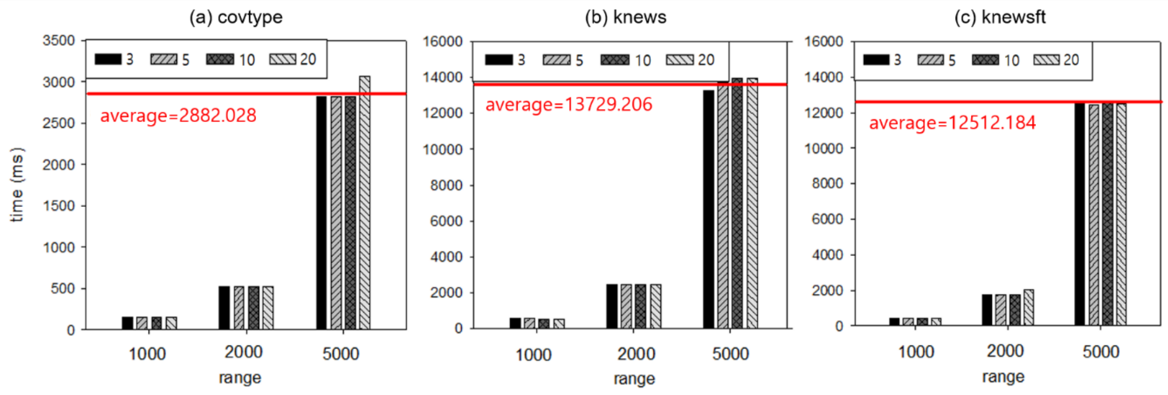

FIGURE 3. Processing time of DBSCAN by various *minPts*

In Figures 2 and 3, especially in Figure 3 with variable *minPts*, we see that the execution time does not vary significantly when the range is small. However, as the range becomes larger, both experiments show that the difference of the execution time becomes also greater. It is expected that the larger the range, the greater the performance difference when inappropriate parameter values are set. We expect more of this tendency to happen when processing very large data sizes, such as actual data. Therefore, it is very important to set the appropriate values of *eps* and *minPts* for each of the data to use them in practice.

Once we derived candidate parameters for the pair of (*minPts*, *eps*), we then evaluated the *purity* of DBSCAN and Den-CS for each dataset through candidate parameters. The results of each *purity* evaluation are shown in Figures 4 and 5. In Figure 4, we evaluated the *purity* as applying those pairs to DBSCAN algorithm for each dataset and selected the pairs with the highest *purity*. For example in Figure 4(b), (3, 0.0189) was selected

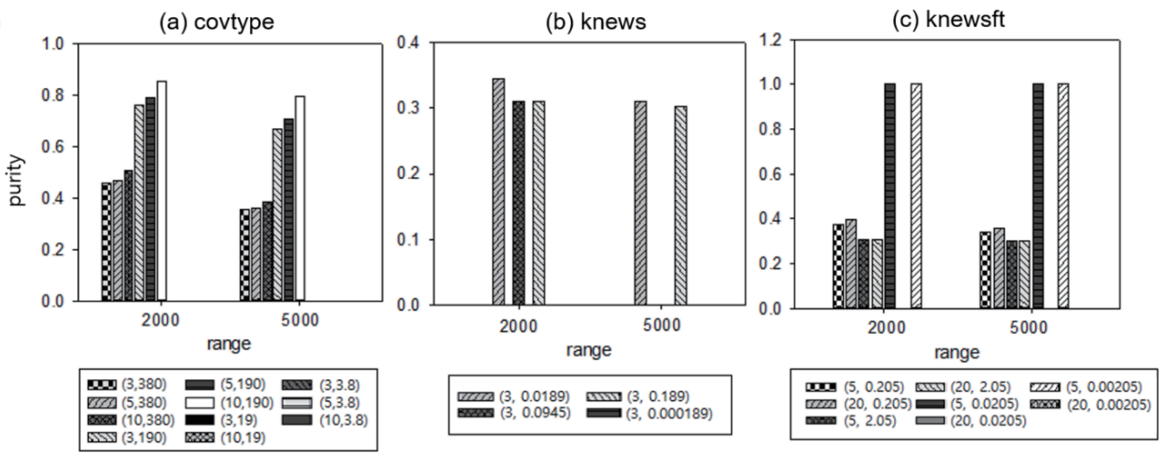

FIGURE 4. *Purity* of DBSCAN with candidate parameter pairs

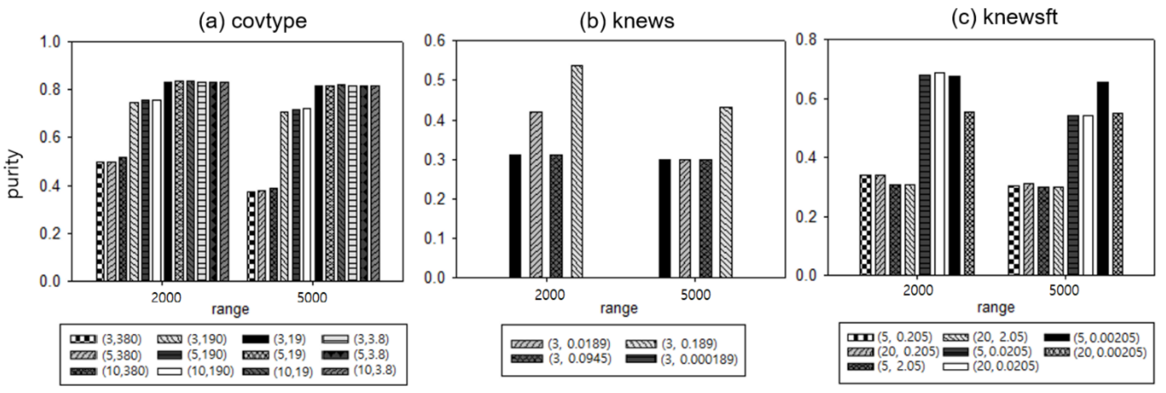

FIGURE 5. *Purity* of Den-CS with candidate parameter pairs

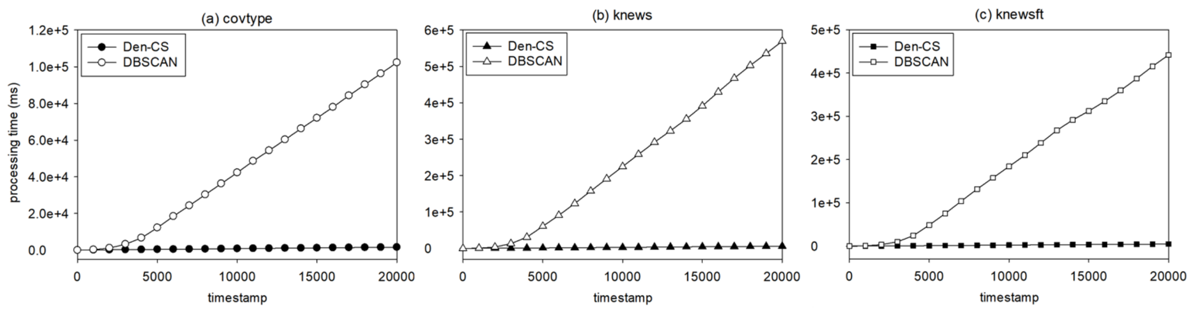

FIGURE 6. Comparison of total processing time of DBSCAN and Den-CS for each of the data

for the knews dataset. We applied those selected pairs to Den-CS algorithm for each dataset, and Figure 5 shows the results. As shown in Figure 5, the selected value pairs with highest *purity* value make the Den-CS run well in general, although some may not always do. We then found the optimal parameter pairs ($minPts$, eps) for the covtype, knews, and knewsft datasets as (10, 190), (3, 0.0189), and (5, 0.00205), respectively. After this, to evaluate the performance of the traditional algorithm DBSCAN, and the proposed Den-CS algorithm, experiments were conducted to compare the execution times of these two algorithms with optimal parameters for each dataset found in the above experiments. Figure 6 shows the results of the experiments.

As seen in Figure 6, Den-CS achieved much better performance than DBSCAN in terms of algorithm execution time. For example in Figure 6(c), the processing time of Den-CS is about one hundred times faster than DBSCAN. Through the experiments, for data that are larger than the experimental datasets, we can see that the proposed Den-CS algorithm with the optimal parameters proposed in this work has more advantages in terms of its execution time.

5. Conclusions.

In this work, we present a density-based clustering algorithm to resolve the problems of partition-based algorithms that might arise in data stream clustering. The proposed Den-CS algorithm is based on the sliding window model, and it exploits the LSH and coresets for summarizing the stream data. The density-based algorithm requires such parameters as the proximity distance *eps* and the minimum number of proximity data *minPts* to make it suitable for data characteristics, and it is critical to set the appropriate parameter values. Through the experimental evaluations, we demonstrate the importance of setting the appropriate parameter values for the algorithm, and also that the clustering of the proposed algorithm may outperform the state-of-the-art algorithm. However, our experiments are limited in terms of the variety and the size of the datasets. We plan to perform more experiments for more various types of larger stream data. In addition, we may also need to find out better ways of building coresets in Level 1. In the future, we will investigate which hash functions are more appropriate for stream data and density-based clustering.

Acknowledgment. This research was supported in part by the National Research Foundation of Korea (NRF), funded by the Ministry of Science, ICT and Future Planning, through the Basic Science Research Program under Grant 2017R1E1A1A03070004. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] M. Zhang, T. Wo, T. Xie, X. Lin and Y. Liu, CarStream: An industrial system of big data processing for Internet-of-vehicles, *Proc. of VLDB Endowment*, vol.10, no.12, pp.1766-1777, 2017.

[2] S. Amini, I. Gerostathopoulos and C. Prehofer, Big data analytics architecture for real-time traffic control, *2017 the 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems*, Naples, pp.710-715, 2017.

[3] S. Guha, A. Meyerson, N. Mishra, R. Motwani and L. O'Callaghan, Clustering data streams: Theory and practice, *IEEE Trans. Knowledge and Data Engineering*, vol.15, no.3, pp.515-528, 2003.

[4] P. P. Rodrigues, J. Gama and J. Pedroso, Hierarchical clustering of time-series data streams, *IEEE Trans. Knowledge and Data Engineering*, vol.20, no.5, pp.615-627, 2008.

[5] S. Har-Peled and S. Mazumdar, On coresets for k-means and k-median clustering, *Proc. of ACM Symp. Theory Computing*, pp.291-300, 2004.

[6] T. Velmurugan and T. Santhanam, Computational complexity between K-means and K-medoids clustering algorithms for normal and uniform distributions of data points, *Journal of Computer Science*, vol.6, no.3, pp.363-368, 2010.

[7] E. Schubert, A. Koos, T. Emrich, A. Züfle, K. A. Schmid and A. Zimek, A framework for clustering uncertain data, *Proc. of VLDB Endowment*, vol.8, no.12, pp.1976-1979, 2015.

[8] M. Ester, H. Kriegel, J. Sander and X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, *Proc. of the 2nd International Conference on Knowledge Discovery and Data Mining*, pp.226-231, 1996.

[9] J. Wang, H. T. Shen, J. Song and J. Ji, Hashing for similarity search: A survey, *CoRR*, 2014.

[10] V. Braverman, H. Lang, K. Levin and M. Monemizadeh, Clustering problems on sliding windows, *Proc. of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp.1374-1390, 2016.

[11] J. Youn, J. Shim and S.-g. Lee, Efficient data stream clustering with sliding windows based on locality-sensitive hashing, *IEEE Access*, vol.6, pp.63757-63776, 2018.

[12] S. Shah and M. Singh, Comparison of a time efficient modified K-mean algorithm with K-mean and K-medoid algorithm, *2012 International Conference on Communication Systems and Network Technologies*, Rajkot, India, pp.435-437, 2012.

[13] S. Kullback and R. A. Leibler, On information and sufficiency, *Ann. Math. Statistics*, vol.22, no.1, pp.79-86, 1951.

[14] J. Youn, *A Scalable Clustering Algorithm for High-Dimensional Data Streams over Sliding Windows*, Ph.D. Thesis, Dept. Comput. Sci., Seoul Nat. Univ., Seoul, Korea, 2017.

[15] M. Deepa, P. Revathy and P. Student, Validation of document clustering based on purity and entropy measures, *International Journal of Advanced Research in Computer and Communication Engineering*, vol.1, no.3, 2012.

[16] D. Lee and J. Shim, Impact parameter analysis of subspace clustering, *International Journal of Distributed Sensor Networks*, 2015.

[17] P. Kranen, I. Assent, C. Baldauf and T. Seidl, The clusTree: Indexing micro-clusters for anytime stream mining, *Knowledge and Information Systems*, vol.29, pp.249-272, 2011.

[18] L. Wan, W. K. Ng, X. H. Dang, P. S. Yu and K. Zhang, Density-based clustering of data streams at multiple resolutions, *ACM Trans. Knowledge Discovery from Data*, vol.3, pp.14:1-14:28, 2009.

[19] A. Zhou, F. Cao, W. Qian and C. Jin, Tracking clusters in evolving data streams over sliding windows, *Knowledge and Information Systems*, vol.15, pp.181-214, 2008.