Decomposition/Aggregation K-means for Big Data

Alexander Krassovitskiy[1](✉), Nenad Mladenovic[1,2], and Rustam Mussabayev[1](✉)

[1] Institute of Information and Computational Technologies, Pushkin Str. 125, Almaty, Kazakhstan
akrassovitskiy@gmail.com, rmusab@gmail.com
[2] Khalifa University, Abu Dhabi 41009, United Arab Emirates
nenad.mladenovic@ku.ac.ae

Abstract. Well-known and widely applied k-means clustering heuristic is used for solving Minimum Sum-of-Square Clustering problem. In solving large size problems, there are two major drawbacks of this technique: (i) since it has to process the large input dataset, it has heavy computational costs and (ii) it has a tendency to converge to one of the local minima of poor quality. In order to reduce the computational complexity, we propose a clustering technique that works on subsets of the entire dataset in a stream like fashion. Using different heuristics the algorithm transforms the Big Data into Small Data, clusters it and uses obtained centroids to initialize the original Big Data. It is especially sensitive for Big Data as the better initialization gives the faster convergence. This approach allows effective parallelization. The proposed technique evaluates dynamically parameters of clusters from sequential data portions (windows) by aggregating corresponding criteria estimates. With fixed clustering time our approach makes progress through a number of partial solutions and aggregates them in a better one. This is done in comparing to a single solution which can be obtained by regular k-means-type clustering on the whole dataset in the same time limits. Promising results are reported on instances from the literature and synthetically generated data with several millions of entities.

Keywords: k-means · Parallel · Clustering · Big Data · MSSC · Dataset · Decomposition · Aggregation

1 Introduction

Recently, clustering methods have attracted much attention as effective tools in theoretical and applied problems of machine learning that allows to detect patterns in raw/poorly structured data. Another motivation is the need to process

This research is conducted within the framework of the grant num. BR05236839 "Development of information technologies and systems for stimulation of personality's sustainable development as one of the bases of development of digital Kazakhstan".

large data sets to obtain a natural grouping of data. Therefore, one of the main aspects for clustering methods is their scalability [9,15].

There are a number of studies aimed at improving clustering quality using methods with high cost and time complexity [15–17,19]. This type of methods usually has a significant drawback: it is practically intractable to cluster medium and large data sets (approximately 10^5–10^7 objects or more). These methods cannot work with huge databases, because the computational complexity in time/space (memory) growths (polynomially) very rapidly. Therefore, it has a sense to look for algorithms with reasonable trade-offs between effective scalability and the quality of clustering [7,8,10,11].

One of the known methods of data clustering is the k-means algorithm, which is widely used due to its simplicity and good characteristics [18,20]. A number of algorithms and technologies have improved this method by clustering input data objects in portions. There are algorithms that use the data stream clustering or decomposition approach, for example, mini-batch k-means [3,8,13]. It has a weighted version of k-means algorithm with many applications [10].

Meta-heuristics can be of great help when the exact solution is difficult or expensive in terms of used computation time and space. Some heuristics for k-means that accelerate calculations have been developed and implemented: part of the heuristics is devoted to accelerating the convergence of the method, another discards redundant or insignificant intermediate calculations. The following meta-heuristics have shown their effectiveness in clustering big data:

– deletion at each iteration of data patterns that are unlikely to change their membership in a particular cluster, as in [11,12];
– using the triangle inequality in [14];
– combinations of various techniques [1,4].

For many machine learning algorithms, processing of big data is problematic and severely limits functionality usage. Our approach is directed to make an advantage out of this drawback, i.e., the more data is given, the better estimates can be obtained. The k-means is one of the fastest algorithms, so we use it as the underlying basis in our approach. In this paper, we use the k-means++ modification to build an algorithmic meta-heuristic that uses some subsets from the entire dataset at each step. We note that ++ version of the k-means has a special initialization of centroids [6].

Formally, given a set of objects $X = \{x_1, ..., x_N\}$ in Euclidean space to be clustered and a set of corresponding weights $\{w_1, ..., w_N\}$, for $w_l \in \boldsymbol{R}_+$, $l \in 1, ..., N$. Then $\{C_1, ..., C_k\}$ is a partition of X in k clusters if it satisfies (i) $C_i \neq \emptyset$, (ii) $C_i \cap C_j = \emptyset, i \neq j$, $i, j = 1, 2, ..., k$, and (iii) $\bigcup C_i = X$. Then, minimum sum-of-squared clustering problem is defined as following:

$$MSSD = \min_{C_1,...,C_k} \sum_{l=1}^{N} w_l \min_{j=1,...,k} ||x_l - c_j||^2,$$

where centroids $c_j = \sum_{l \in arg(C_j)} w_l x_l / \sum_{l \in arg(C_j)} w_l$. Correspondingly, SSD criteria gives an estimate on a particular clustering partition:

$$SSD(C_1, ..., C_k) = \sum_{l=1}^{N} w_l \min_{j=1,...,k} ||x_l - c_j||^2.$$

In case the weights of objects are not specified, then $w_l = 1$ for $l \in 1, ..., N$.

Decomposing the dataset can be technically realized by a stream like methods. Streaming to process a window may be considered as searching dependencies between the obtained essential information and the one gathered previously by the computational model. The principal goal of the study is to investigate methods of dataset decomposition in a stream-like fashion for computing k-means centroid initialization of clustering that produces convincing results regarding MSSD criteria (*Minimum* Sum-of-Squared Distance) [5]. Shortly, methods for finding close to optimal k-means initialization, while having fast computational speed.

The idea of merging clusters obtained by partial clusterings is known in the literature: there are formal approximations that guarantee certain estimates on performance, and quality [9]. Known clustering algorithm STREAM with k-median l1-metric in [13] weights each center by the number of points assigned to it. The stream clustering usually assumes processing the input data in sequential order. Unlike this we use decomposition that may use essential the parallelization of the clustering of the input dataset portions. In our algorithm we add additional heuristic SSD estimates to the weighting. This algorithm can be used in cases the dataset is replenished dynamically, on fly, and possibly in real time. The clustering of additional portions clarifies the clustering structure.

The goal of this work is to create a decomposition method for the k-means algorithm on large-scale datasets to initialize centroids in order to obtain qualitative results with respect to the MSSD criteria. In other words, we use the method of finding the initialization of k-means so that it is close to optimal while having a high calculation speed. Different types of meta-heuristics are used in the task of clustering k-means by processing the obtained data in a secondary (high-level) clustering procedure. Another goal of this work is to study the influence of meta-parameters to the algorithm behaviour with glance to the time and the SSD (Sum-of-Squared Distance) criterion minimization. Another purpose of our research is to define the bounds of such algorithm efficiency and its behaviour regarding meta-parameters.

2 Algorithm

The idea of the algorithm is to make a partition of the dataset into smaller portions, then find the corresponding clustering structure and replace them by single centroid points. On this stage we obtain compact representation of initial dataset that preserves its most essential structural information. Then aggregate and clusterize these centroids in different possible ways getting new heuristic for generalized centroids. Shortly, transform the Big Data into Small Data, cluster them and use obtained centroids to initialize the original Big Data.

More formally, by this approach, first, we decompose the entire dataset entries shuffled randomly on subsets of fixed size (taking either all, or representative portion of elements). Next step is to do k-means clustering of some of these subsets (batches/windows). We use the term 'window' (along the term 'batch' from the literature) to stress that the data subsets are taken in sizes proportional to the entire dataset.

(Meta-) parameters of the algorithm:

- k is the number of required clusters.
- N is the number of objects in the entire data set.
- d is the window size (number of objects in one window). The sizes of the windows are chosen in proportion to the entire dataset. E.g., taking 5 wins decomposition means taking the size of the windows equal to $[N/5]$.
- n is the number of windows used for independent initialization of k-means during Phase 1, see next section.
- $m(\geq n)$ is the total number of windows used for the clustering. The union of m windows may or may not cover the entire dataset.

By using SSD estimates on their corresponding clusterings we make heuristics for better initialization of the algorithm on the entire dataset.

We considered the following two modes for the windows (wins) generation: 1. Segmentation of the entire data set on windows, then a random permutation of objects in the data set is created. The data set is segmented into successive windows of size d. We refer to this as uniform window decomposition mode. 2. For each window, d random objects are selected from the entire set. By repeating this, the required number of windows is generated (objects may be picked repeatedly in different wins). We refer to this mode as random window generation mode.

In order to simplify description we distinguish centroids according to algorithmic steps at which they appear:

- centroids that results from k-means++ on separate windows and used for subsequent initializations we call *local* centroids;
- set of *generalized* centroids is obtained by gathering (uniting) resulted local centroids from clusterings on windows;
- *basis* centroids are obtained by k-means++ clusterings of the set of generalized centroids (considered as a small dataset of higher level representation of windows);
- *final* centroids are obtained by computing k-means on the entire datasest, initialized by basis centroids.

2.1 Phase 1: Aggregation of Centroids

An independent application of k-means++ algorithm on a fixed number n of windows in order to obtain local centroids with following aggregation to generalized set of centroids. The scheme for the algorithms is shown in Fig. 1. This centroids are considered as higher level representation of clustered windows. Each object of generalized set of centroids is assigned to the weight corresponding to the normalized SSD value for the window in which it is calculated as centroid. The weight of i-th object is calculated as follows:

$$w_i = 1 - (SSD_i - SSD_{min})/(SSD_{max} - SSD_{min}), \tag{1}$$

where SSD_i is the SSD value for such window from which the i-th centroid is taken as an object. Then, using k-means, the new dataset of generalized centroids is divided into k-clusters, taking into account the weights w_i of the objects. In the case of degeneration, k-means is reinitialized. The resulting (basis) centroids are used for:

1. initialization of k-means on the Input Dataset in order to obtain final centroids;
2. evaluation of the SSD on the Input Dataset;
3. initialization Phase 2 of the algorithm described in the following sections.

Alternatively, during processing subsequent windows $n+1$, $n+2$, ..., m we have considered the following options in Phase 2: parallel option, straightforward option, and sequential option.

2.2 Phase 2: Parallel Option

The centroids obtained in the previous Phase 1 are used to initialize k-means on each subsequent window $n+1$, $n+2$, ..., m. The resulting (local) centroids and SSD estimates are stored if there is no centroid degeneration. The stopping condition is the specified limit either on the computation time or on the number of windows being processed. Similar to Phase 1 we do the clustering on the generalized set of centroids. Subsequent use of its results is similar to clauses 1.1 and 1.2 of Phase 1. Both Phase 1. and parallel option of the Phase 2 are unified in Fig. 1.

2.3 Phase 2: Straightforward Option

An alternative heuristic of splitting the entire dataset and an alternative way of choosing centroids for the clustering initialization is used (see Fig. 2).

The idea of this heuristics is to evaluate and to use the best centroids regarding SSD criteria for initialization of k-means on the subsequent window. While each window is clustered the best obtained centroids are accumulated to process them for final clustering, like in Phase 1.

Algorithm Sketch:

1. Make dataset decomposition on subsets $win_0, win_1, ..., win_l, ...$ of equal size.
2. Obtain list of initial centroids $cent_0$ either by k-means++ on the first window win_0 or by Phase 1. Assign $AC \leftarrow [cent_0]$, $c \leftarrow cent_0$, $BestSSD \leftarrow SSD_0$, $l = 1$.

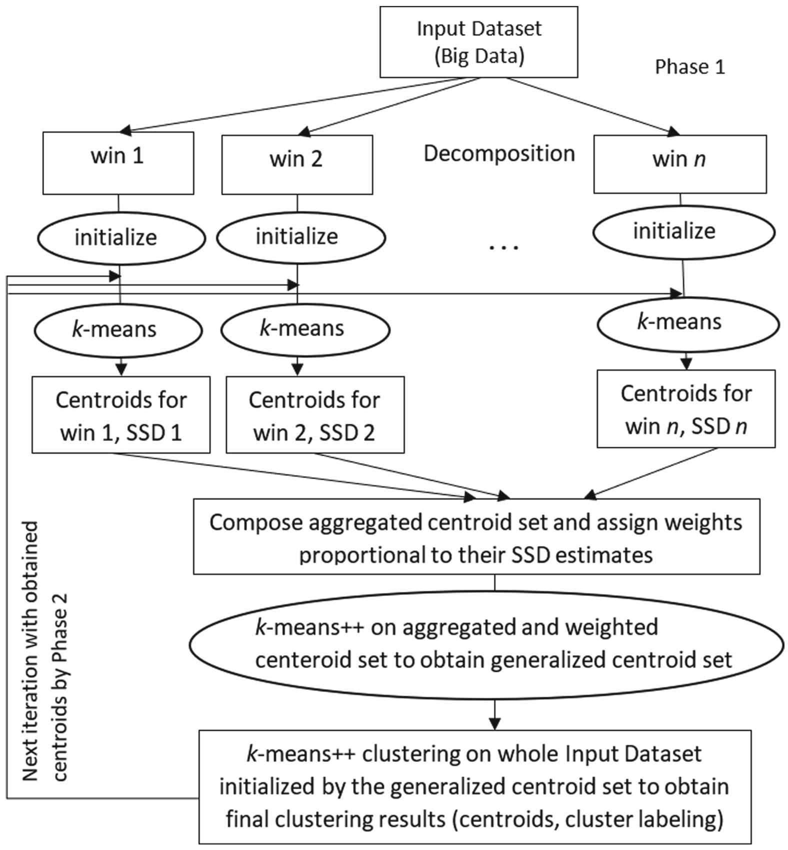
Input Dataset (Big Data)
Phase 1
win 1
win 2
Decomposition
win *n*
initialize
initialize
...
initialize
k-means
k-means
k-means
Centroids for win 1, SSD 1
Centroids for win 2, SSD 2
Centroids for win *n*, SSD *n*
Compose aggregated centroid set and assign weights proportional to their SSD estimates
k-means++ on aggregated and weighted centeroid set to obtain generalized centroid set
k-means++ clustering on whole Input Dataset initialized by the generalized centroid set to obtain final clustering results (centroids, cluster labeling)
Next iteration with obtained centroids by Phase 2

Fig. 1. Scheme for the decomposition/aggregation clustering method. In Phase 1 k-means++ initialization is performed independently on windows $1, ..., n$. Resulted final centroids are used for initialization during Phase 2 on windows $n+1, ..., m$.

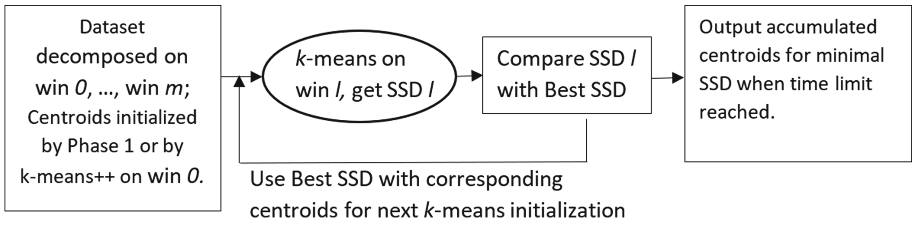
Dataset decomposed on win *0*, ..., win *m*; Centroids initialized by Phase 1 or by k-means++ on win *0*.
k-means on win *l*, get SSD *l*
Compare SSD *l* with Best SSD
Output accumulated centroids for minimal SSD when time limit reached.
Use Best SSD with corresponding centroids for next *k*-means initialization

Fig. 2. Direct subsequent use of the best obtained SSD

3. (Start iteration) Use centroids c to initialize k-means with the next window win_l. Calculate $cent_l$ and SSD_l.

4. If degeneracy in the clustering is presented (i.e., the number of obtained non-trivial clusters less then k, $|cent_l| < k$) then withdraw win_l and continue from step 3 for the next $l \leftarrow l + 1$.
5. If its clustering SSD is within the previously obtained or best SSD then $AC \leftarrow AC \cup cent_l$.
6. If its clustering SSD gives better score then mark it as the best and use for the following initializations, i.e., $BestSSD \leftarrow SSD_l$, $c \leftarrow cent_l$.
7. Repeat from step 3 until all windows have been processed or, time bounding condition is satisfied.
8. The accumulated centroids AC are considered as elements for additional clustering, while their SSD values are used to calculate corresponding weights like in Phase 1.
9. The obtained centroids AC are used for the clustering (like in the previous part) and its final SSD value has been compared with SSD of k-means++ on the entire dataset.

2.4 Phase 2: Sequential Option

The following is the sequential version of the algorithm. It is schematically represented in Fig. 3.

Algorithm Sketch:

1. $l = 1$, $init \leftarrow$ centroids from Phase 1, m is the fixed parameter
2. k-means clustering on the window $m + l$ with initialization $init$.
3. If there is no degeneration during clustering then memorize the resulting (local) centroids and the corresponding SSD values.
4. In order to obtain new centroids, we carry out clustering with weights on the united set of centroids (similarly to Phase 1).
5. If the time limit has not been exhausted then $init \leftarrow$ centroids obtained in step 4, $l = l + 1$ and go to step 2, otherwise step 6.
6. Subsequent usage of obtained centroids is similar to clauses 1 and 2 of Phase 1.

3 Computational Experiments

In this section we show the testing results of our algorithm from Sect. 2.4 on three datasets. We only present the results of computation by the sequential version described in Phase 2, with the initial centroids precomputed (sequentially) according to Phase 1. We do not include parallel version of Phase 2 as it distinguishes in the way windows are clustered and it requires additionally efforts in order to compare computational times (taking into account parallelism). The straightforward case can be seen as a particular case of the window aggregation.

Table 1 summarizes clustering estimates of used datasets. Results of computations on various meta-parameter sets from Table 2 are compared regarding SSD/time estimates to the ones obtained by k-means++ and summarized in Table 3. Each line of Table 3 corresponds to unique meta-parameter's set and

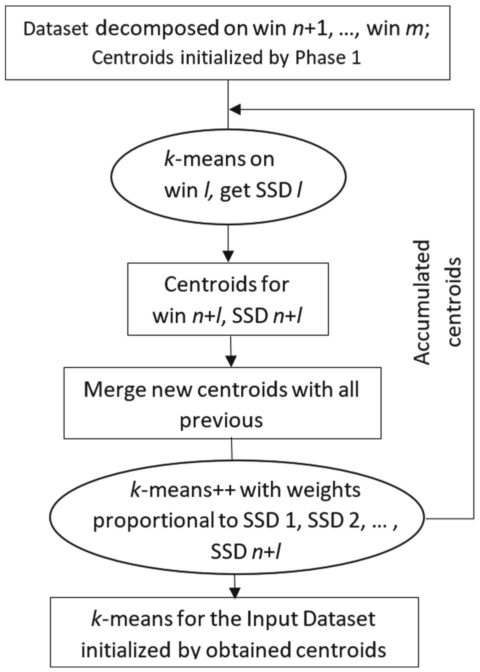
Dataset decomposed on win n+1, ..., win m;
Centroids initialized by Phase 1
k-means on win l, get SSD l
Centroids for win n+l, SSD n+l
Merge new centroids with all previous
k-means++ with weights proportional to SSD 1, SSD 2, ... , SSD n+l
k-means for the Input Dataset initialized by obtained centroids
Accumulated centroids

Fig. 3. Sequential aggregation (accumulation) of the heuristically optimal SSD and centroids

includes two SSD estimates, average time per clustering an proportion between computation time of proposed decomposition algorithm and k-means++. The first SSD estimate is obtained as following: we cluster corresponding entire dataset by k-means++ with corresponding parameters and consider obtained SSD criteria as a baseline value. Then, we do independent clusterings by our algorithms on ranges of experiments to calculate basis centroids and compare whether obtained SSD (on the basis centroids) values improves baseline values. The rates are presented for the cases our algorithm finds better solution. We present it in order to show what approximation our algorithm gives if the entire dataset have not been involved. We note that in order to obtain basis centroids we only need to process separate windows. The second SSD estimate is calculated in the same way with addition of one more step. Specifically, k-means is processed on the entire dataset while initialized by the basis centroids. Comparing these two columns of SSD estimates in Table 3 on various parameters and datasets allows us to consider obtained basis centroids as reasonable approximation to MSSD on the entire dataset.

Datasets Description:
We used three datasets DS1, DS2, DS3 of real numbers.

Table 1. SSDs and computation times for datasets DS1, DS2, DS3. The k-means++ is performed with default parameters from the programming library sklearn [1], i.e., 10 separate initializations are executed, the result of the best is presented.

Dataset	Num	SSD	Time (sec)	SSD
	Clusters	k-means++	k-means++	Ours
DS1	5	3.0625×10^8	389.69	3.0622×10^8
DS1	10	2.7270×10^8	617.70	2.7209×10^8
DS1	20	2.2960×10^8	836.44	2.2740×10^8
DS1	30	1.9482×10^8	677.52	1.9449×10^8
DS2	5	1.5947×10^8	118.23	1.5947×10^8
DS2	10	1.1111×10^8	198.92	1.1106×10^8
DS2	20	7.5041×10^7	627.57	7.4883×10^7
DS2	30	7.3088×10^7	3312.62	7.3080×10^7
DS3	5	1.5734×10^6	312.77	1.5734×10^6
DS3	10	1.2657×10^6	776.62	1.2657×10^6
DS3	20	1.0151×10^6	1279.21	1.0151×10^6
DS3	30	9.1251×10^5	2785.85	9.1173×10^5

Table 2. Meta-parameters of experiments from Table 3. Window sizes are taken in the ranges $(N/20, N/25, ..., N/100)$, $(N/100, N/110, ..., N/250)$ and $(N/10, N/20, ..., N/150)$. 'Allow repeats in windows' refers to the mode how windows are generated. Random window generation mode allows repeats in data objects (TRUE), while uniform window decomposition does not allow it (FALSE).

Param id	Window sizes ($\times N$)	Allow repeats in windows	Number of clusters	Time limit (sec)
1	1/20, ..., 1/100; step 1/5	FALSE	5	30
2	1/20, ..., 1/100; step 1/5	FALSE	10	30
3	1/20, ..., 1/100; step 1/5	FALSE	20	30
4	1/20, ..., 1/100; step 1/5	FALSE	30	30
5	1/100, ..., 1/250; step 1/10	FALSE	5	45
6	1/100, ..., 1/250; step 1/10	FALSE	10	45
7	1/100, ..., 1/250; step 1/10	FALSE	20	45
8	1/100, ..., 1/250; step 1/10	FALSE	30	45
9	1/10, ..., 1/150; step 1/10	TRUE	5	60
10	1/10, ..., 1/150; step 1/10	TRUE	10	60
11	1/10, ..., 1/150; step 1/10	TRUE	20	60
12	1/10, ..., 1/150; step 1/10	TRUE	30	60

Table 3. Experiments on different parameter sets from Table 2. SSD criteria and computation times are presented. Two SSD estimates are considered: 1. the rates our algorithm improves* computed SSD in regards to k-means++ (baseline), where centroids are resulted from aggregation step, i.e., the criteria is estimated on basis centroids; 2. the rate our algorithm improves** the baseline with final centroids, i.e., after additional step with k-means initialized by basis centroids. The improvement rates are given regarding k-means++ on windows from Table 1. Average times per clustering procedure are given.

Dataset	Param id	Improves SSD rate %		Avg. time in seconds	$\frac{time(ours)}{time(kmeans{+}{+})}$
		*direct	**global step		
DS1	1	11.8	52.9	42.0	0.108
DS1	2	58.8	100.0	54.8	0.089
DS1	3	41.2	88.2	56.1	0.067
DS1	4	0.0	2 3.5	54.9	0.081
DS2	1	0.0	58.8	21.2	0.179
DS2	2	11.8	64.7	29.0	0.146
DS2	3	76.5	100.0	45.2	0.072
DS2	4	0.0	5.9	458.2	0.138
DS3	1	0.0	17.6	28.7	0.092
DS3	2	0.0	29.4	56.9	0.073
DS3	3	0.0	23.5	69.8	0.055
DS3	4	5.9	41.2	196.3	0.070
DS1	5	0.0	68.8	39.7	0.102
DS1	6	0.0	87.5	56.0	0.091
DS1	7	0.0	93.8	61.0	0.073
DS1	8	0.0	68.8	49.2	0.073
DS2	5	0.0	43.8	21.2	0.179
DS2	6	0.0	68.8	26.1	0.131
DS2	7	87.5	93.8	39.9	0.064
DS2	8	0.0	25.0	445.2	0.134
DS3	5	0.0	6.2	27.9	0.089
DS3	6	0.0	37.5	60.8	0.078
DS3	7	0.0	25.0	53.8	0.042
DS3	8	0.0	18.8	200.4	0.072
DS1	9	6.7	66.7	53.7	0.138
DS1	10	26.7	93.3	76.8	0.124
DS1	11	20.0	100.0	82.0	0.098
DS1	12	0.0	26.7	72.1	0.106
DS2	9	0.0	46.7	26.9	0.227
DS2	10	0.0	66.7	35.7	0.179
DS2	11	80.0	86.7	55.8	0.089
DS2	12	0.0	6.7	519.2	0.157
DS3	9	0.0	6.7	39.4	0.126
DS3	10	0.0	26.7	72.1	0.093
DS3	11	0.0	26.7	85.6	0.067
DS3	12	6.7	60.0	229.6	0.082

- DS1 contains 4×10^6 objects and number of attributes (features) is 25. The structure of the data: 50 synthetic blobs having Gaussian distribution, each having the same number of elements and the same standard deviation value. There is no overlaps in separate blobs.
- DS2 contains 4×10^6 objects and number of attributes (features) is 20. The structure of the data: 20 synthetic blobs having Gaussian distribution, each blob has variable number of objects (from 10^4 to 40×10^4) and variable standard deviation values (distributed randomly in the range from 0.5 to 1.5).
- DS3 is SUSY dataset from open UCI database [2]. The number of attributes is 18 and the number of objects is 5×10^6. In our study we do not take into account the true labelling provided by the database, i.e., the given predictions for two known classes. The purpose of using such dataset is to search for internal structure in the data. This dataset is preprocessed by normalization prior the clustering.

4 Conclusions

In this approach we show that it is possible to achieve better results in the meaning of SSD criteria by applying iteratively the clustering procedure on subsets of the dataset. Obtained centroids are processed again by (meta-) clustering, resulting to the final solution.

Some observations:

- One promising result is that centroids calculated by shown method on large datasets provide reasonable good quality SSD values even without clustering on the whole dataset. Step 6 in Part 2.2 and step 9 in Part 2.3 in many cases may be omitted giving essential advantage in computational speed.
- It is observed there is no sense in splitting the dataset for a huge number of windows as the number of degenerated clusters growths as well.
- Slight improvement is detected on normalized data and small number of clusters.
- Our experiments mostly support the idea that quality and precision of clustering results are highly dependant on the dataset-size and its internal data structure, while it does not strongly depend on the clustering window/batch size, as far as the majority of windows represents the clustering structure of the entire dataset.

References

1. Comparing different clustering algorithms on toy datasets. http://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html
2. Online clustering data sets uci. https://archive.ics.uci.edu/ml/datasets.html
3. Comparison of the k-means and minibatch-kmeans clustering algorithms, January 2020. https://scikit-learn.org/stable/auto_examples/cluster/plot_mini_batch_kmeans.html

4. Abbas, O.A.: Comparisons between data clustering algorithms. Int. Arab J. Inf. Technol. **5**(3), 320–325 (2008). http://iajit.org/index.php?option=com_content&task=blogcategory&id=58&Itemid=281
5. Alguwaizani, A., Hansen, P., Mladenovic, N., Ngai, E.: Variable neighborhood search for harmonic means clustering. Appl. Math. Model. **35**, 2688–2694 (2011). https://doi.org/10.1016/j.apm.2010.11.032
6. Arthur, D., Vassilvitskii, S.: K-means++: the advantages of careful seeding. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (2007)
7. Bagirov, A.M.: Modified global k-means algorithm for minimum sum-of-squares clustering problems. Pattern Recogn. **41**(10), 3192–3199 (2008). https://doi.org/10.1016/j.patcog.2008.04.004. http://www.sciencedirect.com/science/article/pii/S0031320308001362
8. Bahmani, B., Moseley, B., Vattani, A., Kumar, R., Vassilvitskii, S.: Scalable k-means++. Proc. VLDB Endow. **5** (2012). https://doi.org/10.14778/2180912.2180915
9. HajKacem, M.A.B., N'Cir, C.-E.B., Essoussi, N.: Overview of scalable partitional methods for big data clustering. In: Nasraoui, O., Ben N'Cir, C.-E. (eds.) Clustering Methods for Big Data Analytics. USL, pp. 1–23. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-97864-2_1
10. Capo, M., Pérez, A., Lozano, J.: An efficient approximation to the k-means clustering for massive data. Knowl.-Based Syst. **117** (2016). https://doi.org/10.1016/j.knosys.2016.06.031
11. Chiang, M.C., Tsai, C.W., Yang, C.S.: A time-efficient pattern reduction algorithm for k-means clustering. Inf. Sci. **181**, 716–731 (2011). https://doi.org/10.1016/j.ins.2010.10.008
12. Cui, X., Zhu, P., Yang, X., Li, K., Ji, C.: Optimized big data K-means clustering using MapReduce. J. Supercomput. **70**(3), 1249–1259 (2014). https://doi.org/10.1007/s11227-014-1225-7
13. Guha, S., Mishra, N., Motwani, R.: Clustering data streams. In: Annual Symposium on Foundations of Computer Science - Proceedings, pp. 169–186, October 2000. https://doi.org/10.1007/978-0-387-30164-8_127
14. Heneghan, C.: A method for initialising the k-means clustering algorithm using kd-trees. Pattern Recogn. Lett. **28**, 965–973 (2007). https://doi.org/10.1016/j.patrec.2007.01.001
15. Karlsson, C. (ed.): Handbook of Research on Cluster Theory. Edward Elgar Publishing, Cheltenham (2010)
16. Krassovitskiy, A., Mussabayev, R.: Energy-based centroid identification and cluster propagation with noise detection. In: Nguyen, N.T., Pimenidis, E., Khan, Z., Trawiński, B. (eds.) ICCCI 2018. LNCS (LNAI), vol. 11055, pp. 523–533. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98443-8_48
17. Mladenovic, N., Hansen, P., Brimberg, J.: Sequential clustering with radius and split criteria. Cent. Eur. J. Oper. Res. **21**, 95–115 (2013). https://doi.org/10.1007/s10100-012-0258-3
18. Sculley, D.: Web-scale k-means clustering. In: Proceedings of the 19th International Conference on World Wide Web, WWW 2010, pp. 1177–1178. Association for Computing Machinery, January 2010. https://doi.org/10.1145/1772690.1772862
19. Shah, S.A., Koltun, V.: Robust continuous clustering. Proc. Natl. Acad. Sci. **114**(37), 9814–9819 (2017)
20. Wu, X., et al.: Top 10 algorithms in data mining. Knowl. Inf. Syst. **14**, 1–37 (2007). https://doi.org/10.1007/s10115-007-0114-2