SOCEN-CLUSTER : A New Rapid Clustering Technique

Farouk Baccari and Mounir Sayadi

SICISI Unit, ESSTT, University of Tunis, 5 Av Taha Hussien, 1008, Tunisia

Emails: farouk.baccari@topnet.tn, mounir.sayadi@esstt.rnu.tn

Abstract— In this paper, we present a novel clustering technique for unindexed, randomized, multidimensional, datasets. The main advantage of the proposed technique is the time and space complexity that were reduced to linear cardinality dependency. The algorithmic implementation shown in this paper uses some heuristics to enhance the overall execution time and space required making them fully scalable. This particularity makes it easier for ASICS / FPGA architects to implement such a technique in a constrained environment.

Keywords—component; clustering; classification; linear time complexity; hierarchical clustering; incremental clustering; algorithmic hurisites; scalable clustering technique;

I. INTRODUCTION

Clustering around centers is a meaningful computer learning task when confronted to non special / unindexed data. Unindexed data features, by nature, give a qualitative description of a phenomenon and, therefore, neither continuity nor special shapes of clusters are meaningful. Instead, data features are intuitively classified according to some prototypes – modeled as centers – tainted by a dissimilarity tolerance – modeled as radius. In such a modeling, clusters are assumed to be of hyper-spherical shapes when spread in special representation.

II. SOCEN-CLUSTER: SOLVED CENTROIDS CLUSTERING

A. General overview

The main idea about SOCEN-CLUSTER technique is bringing the clustering problem to a basic classification one by solving in advance the clusters centers and radius. This can be quickly done with the proposed technique.

The algorithm achieves linear time complexity relatively to dataset cardinality and dimensionality. The algorithm is also scalable in terms of resources and can fit any industrial constraints without affecting clustering quality.

The algorithm integrates a native noise filter and tow classification operating modes:

1-The first and basic operating mode tries to cluster all the data (including noise) according to their closeness to the clusters centers already detected.

2-The second operating mode eliminates all noise from data and outputs the filtered result as is.

The first operating mode gives better results when applied to image segmentation. The clustering result is, visually, more “meaningful”.

B. Global Parameters

The algorithm only requires few user defined parameters:

The initial clusters radius: this is a user defined parameter that gives an interpretation of the data plasticity/closeness. This parameter should be linearly proportional to $\sqrt{d}$ where d is the dataset dimensionality. The overall performance of the algorithm is tightly dependent on this parameter.

Further work on automatically initializing the clusters radius could lead to a completely unsupervised clustering technique.

Resources constraints: The resources constraints are the resources available for computing distance from a given set of cluster centers. This constraint is expressed in terms of maximum allowed – here denoted as M – clusters per loop/iteration.

The amount of resources needed is given by the following:

$$\begin{aligned} \text{Computing Memory} \\ &= 2 * M * d * \text{sizeof(Signgle Data point)} \\ &+ \text{Constant} \end{aligned}$$

So space-complexity scales linearly with dimensionality and at most scales constantly with data cardinality.

Maximum number of iterations: the SOCEN-CLUSTER algorithm is an iterative one. In order to keep tight constraints on the running time, the number of iterations should already be specified, so, even if the clustering task is not completely achieved, the algorithm returns. SOCEN-CLUSTER outputs the right clusters centers in a linear among of time even if convergence is not reached.

Radius regression ratio: when dealing with dance crossed over clusters, the algorithm could diverge by merging all clusters in a single one. To overcome this problem, we’ve incorporated a regression ratio when computing the merged cluster radius as above:

$$\begin{aligned} \text{ActualMergedRadius} \\ &= \text{RegressionRatio} \times \text{ThoricalMergedRaduis} \end{aligned}$$

This regression ratio is between 0 and 1. The default value is 0.8.

Cross-over thresholding ratio: To prevent the algorithm from over-clustering, we threshold the clusters distance $\Delta(<., <., <.)$ proportionally to centers distance $\text{dist}(<., <., <.)$ according a cross-over thresholding ratio denoted here as “ δ ”. The merging condition would look like:

$$\begin{aligned} \text{if } (\Delta(\text{Cluster1}, \text{Cluster2}) \\ < -\delta \cdot \text{dist}(\text{Cluster1}_{\text{Center}}, \text{Cluster2}_{\text{Center}})) \end{aligned}$$

C. The SOCEN Algorithm

As presented earlier, the SOCEN algorithm models clusters as hyper-spheres defined by a center and a radius. The main idea is searching for crossed over clusters and merging them, one couple at a time.

The above figure illustrates the initial and final state of the clustering process (2-D representation):

Figure 1. SOCEN Clustering process

To detect cross over clusters, we compute the “*clusters distance*”, denoted as Δ , between a pair of clusters that we define as following:

$$\begin{aligned} \Delta(\text{Cluster1}, \text{Cluster2}) &= \text{dist}(\text{Cluster1}_{\text{center}}, \text{Cluster2}_{\text{center}}) \\ &\quad - \text{Cluster1}_{\text{Radius}} - \text{Cluster2}_{\text{radius}} \end{aligned}$$

Figure 2. Clustering initial parameters

If cluster distance is negative, the pair of clusters is crossed. When this distance is less than its associated threshold, (see Cross-over thresholding ratio paragraph) merging can occur.

Figure 3. Clusters merging

For every merging step, the algorithm computes the new cluster by determining its new cardinality, new center and new radius:

$$\text{NewCluster}_{\text{card}} = \text{OldClust1}_{\text{card}} + \text{OldClust2}_{\text{card}}$$

$$\text{NewCluster}_{\text{Cntr}} = \frac{\text{OldClust1}_{\text{card}} \times \text{OldClust1}_{\text{Cntr}} + \text{OldClust2}_{\text{card}} \times \text{OldClust2}_{\text{Cntr}}}{\text{OldClust1}_{\text{card}} + \text{OldClust2}_{\text{card}}}$$

$$\begin{aligned} \text{NewCluster}_{\text{Rad}} &= \text{RegRatio} \\ &\times \frac{\text{OldClust1}_{\text{Rad}} + \text{OldClust2}_{\text{Rad}} + \text{dist}(\text{OldClust1}_{\text{Cntr}}, \text{OldClust2}_{\text{Cntr}})}{2} \end{aligned}$$

Figure 4. Clustering resulting parameters

The algorithm proceeds in an agglomerative way to cluster all data. Every data point is initiated as singleton cluster. The clustering is done in hierarchical way:

Initialize Input clusters

Loop until no merging occurs or until max iteration number is reached

Prepare empty output clusters container

For every Input cluster,

Compute distance from existing output clusters and find the minimal distance

```

    If minimal distance is negative (this means that there is
    crossing over)
        Merge with the associated output cluster
    Else
        Directly Insert Input cluster as an output one
    End if
End for
Assign all output clusters as input ones
End loop

```

This is a classical agglomerative clustering algorithm with super linear time complexity. The most time/space consuming step recorded is computing distances from all output clusters step.

It will take $O\left(\frac{n(n-1)}{2}\right)$ to state that no clusters are detected which is quadratic time complexity.

D. Complexity improvement heuristic

To limit the time/space complexity we improve the algorithm by limiting the number of output clusters so that the distance computing stage is constrained to a limited amount of space/time. This will bring time complexity to $O(n \times Limit)$ – which is linear – and Space complexity to $O(Limit)$ – which is at most constant.

When the maximal/limit number of output clusters is reached, the algorithm classes every unclustered element in a standby class and reinserts them on the next clustering loop. The algorithm would look like:

```

Initialize Input clusters
Loop until no merging occurs or until max iteration number is
reached
    Prepare empty output clusters container
    Prepare empty standby clusters container
    For every Input cluster,
        Compute distance from existing output clusters and find the
        minimal distance
        If minimal distance is negative (this means that there is
        crossing over)
            Merge with the associated output cluster
        Else if the maximum number of output cluster is reached
            Insert input cluster in the standby container
        Else
            Directly Insert Input cluster as an output one
    End if
End for
Concatenate standby class and output cluster
Assign output clusters as input ones
End loop

```

When the dataset is initially sorted, the clustering process is slow. This is explained by the fact that output limitation prevents the algorithm from summarizing the whole dataset classes during the clustering process.

To deal with this delay we randomly address input clusters in order to summarize as many classes as possible before reaching the output limitation number.

```

Initialize Input clusters
Loop until no merging occurs or until max iteration number is
reached

```

```

    Prepare empty output clusters container
    Prepare empty standby clusters container
    Permute input clusters
    For every Input cluster,
        Compute distance from existing output clusters and find the
        minimal distance
        If minimal distance is negative (this means that there is
        crossing over)
            Merge with the associated output cluster
        Else if the maximum number of output cluster is not
        reached
            Directly Insert Input cluster as an output one
        Else
            Insert input cluster in the standby container
    End if
End for
Concatenate standby class and output cluster
Assign output clusters as input ones
End loop

```

After testing the above algorithm several times on different datasets we noticed that standby clusters are heterogeneous from output ones. For this reason, it is obvious that clustering them separately would lead to more merged clusters. So we modified the algorithm to reprocess stand by clusters separately in the same loop until no standby clusters are recorded. This will help distributing clustering possibly but also fasten the global process. The pseudo-code would look like:

```

Initialize Input clusters
Loop until no merging occur or until max iteration number is
reached
    Prepare empty output clusters container
    Loop until no input clusters remains
    Prepare empty standby clusters container
    Permute input clusters
    For every Input cluster,
        Compute distance from New output clusters and find the
        minimal distance
        If minimal distance is negative (this means that there is
        crossing over)
            Merge with the associated New output cluster
        Else if the maximum number of New output cluster is
        reached
            Insert input cluster in the standby container
        Else
            Directly Insert Input cluster as New output one
    End if
End for
Assign standby clusters as input ones
End loop
Assign all Output clusters as Input ones
End loop

```

E. Filtering the resulting clusters

After solving all the desired clusters the result should be filtered according to the following criterions:

All the unclustered singletons are eliminated as considered noise

The remaining clusters are filtered according to the following criterion:

- The mean cardinality of clusters is computed, we note it as $Mean_Card$
- Only clusters verifying the above thresholding criterion are considered:

$$Cluster_Descriptor.Cardinality > \alpha * Mean_Card$$

Where α is the cardinality thresholding ratio (the default value is 0.1)

N.B.: This filtering process is a basic thresholding and can be improved in a future work.

The filtering process is considered strait forward and no pseudo-code is presented for this stage.

Although, we would like to emphasize the fact that the time complexity for this process is $O(n)$ and so does the amount of space required. This preserves the time and space linearity through the whole process.

F. Classifying the Dataset

The classification process is also strait forward. The whole dataset and the detected clusters are indexed with integer IDs. Tow classification ways are available:

Nearest Cluster classification: this trivial technique compares every single data point to all existing cluster centers and classes it within the nearest one. The result is stored in a classification matrix directly related to the dataset index and labeled with the detect clusters IDs. This method leads to a classification of the whole data including the filtered noise.

Noise filtering classification: this technique scans every single data point and checks if it fits in one of the detected clusters by comparing its distance to the center with the associated radius. If the data point does not fit in any cluster, it is considered as noise and labeled with ID zero.

Time and space complexity for both classification ways is $O(d*N*K)$ where K is the number of detected clusters. This also preserves the overall time and space linearity within the whole process.

III. EXPERIMENTAL EVALUATION

In order to get more concrete evaluation of the above technique we compared it with two of the most known clustering techniques that show linear time complexity: BIRCH and KMEANS.

To do so, we have used the 3 datasets that were earlier presented in BIRCH paper:

TABLE I. DATASETS SPECIFICATIONS

DS	Card	Dim	Nb clusters	disposition	Rand
DS1	100000	2	100	Grid	Yes
DS2	100000	2	100	Sine	Yes
DS3	100000	2	100	Random	Yes

Downloader from <http://cs.joensuu.fi/sipu/datasets/>

To maintain tighter conditions than for BIRCH evaluation, we run the tests on a laptop with 2 GHz dual core processor and 3GB RAM. To code the algorithm, we used MATLAB interpreter which performs, at least, 10 times slower than a compiled code (C/C++ or FORTRAN).

Below, we present the clustering results for DS1:

Figure 5. DS1 plot

Figure 6. SOCEN clustering result for DS1

TABLE II. EXPERIMENTAL RESULTS FOR DS1

DS	Init rad	Reg ratio	Ress Constraint	Cross thresh: δ	Run time	Filtering ratio : α	Nb found clusters
DS1	$4\sqrt{2}$	0,8	100	0,2	8,8401	0,2	100

It took SOCEN-CLUSTER less than 10 seconds to find 100 appropriate clusters in a 100000 elements dataset, using nothing but a conventional laptop.

Clustering results for DS2:

Figure 7. DS2 plot

Figure 8. SOCEN clustering result for DS2

TABLE III. EXPERIMENTAL RESULTS FOR DS1

DS	Init rad	Reg ratio	Ress Constraint	Cross thresh: δ	Run time	Filtering ratio : α	Nb found clusters
DS2	$4\sqrt{2}$	0,8	100	0,2	8,0589	0,2	100

Clustering results for DS3:

Figure 9. DS2 plot

Figure 10. SOCEN clustering result for DS2

TABLE IV. EXPERIMENTAL RESULTS FOR DS1

DS	Init rad	Reg ratio	Ress Constraint	Cross thresh: δ	Run time	Filtering ratio : α	Nb found clusters
DS3	$2\sqrt{2}$	0,8	100	0,2	9,0279	0,2	68

A. Run time scaling with dataset cardinality for a fixed number of cluster

To evaluate the run time according to cardinality we generated a dataset of 10 clusters of different sizes as above:

We vary the dataset cardinality and record the running time. The table above summarizes the benchmark results:

TABLE V. LINEARITY BENCHMARK RESULTS

Card	dim	Nb clusters	Running time	Optimal detection
5000	4	10	0,98612	Yes
10000	4	10	1,8709	Yes
15000	4	10	2,8737	Yes
20000	4	10	3,9435	Yes
25000	4	10	5,0999	Yes
30000	4	10	6,446	Yes
35000	4	10	7,5222	Yes
40000	4	10	8,9222	Yes
45000	4	10	10,2412	Yes
50000	4	10	11,8572	Yes
55000	4	10	12,7372	Yes
60000	4	10	13,8763	Yes
65000	4	10	15,3438	Yes
70000	4	10	16,5034	Yes
75000	4	10	17,4244	Yes
80000	4	10	18,6974	Yes
85000	4	10	20,362	Yes
90000	4	10	21,3302	Yes
95000	4	10	22,3637	Yes
100000	4	10	24,1959	Yes

As demonstrated by the above graphic, the running time is empirically scaling linearly with the dataset cardinality.

IV. CONCLUSION

In this paper, we have proposed for the first time the SOCEN CLUSTER Technique which is an incremental agglomerative clustering technique able to classify randomized, unindexed multidimensional datasets. We have shown how much the use of SOCEN CLUSTER can give robust results when dealing with huge datasets. The algorithm was optimized by mean of certain heuristics to show linear time complexity and scalable resources consumption. Noise filtering and meaningful

clustering is demonstrated through examples taken from one of the most popular clustering techniques, the well-known BIRCH.

Further work can address applications of this technique in image segmentation. Multidimensional data management can be helpful when dealing with color images.

- [1] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An efficient data clustering method for very large databases," in Proc. ACM SIGMOD Conf. Management of Data, 1996, pp. 103–114.
- [2] SPATH H. 1980. Cluster Analysis Algorithms. Ellis Horwood, Chichester, England.
- [3] JAIN, A. and DUBES, R. 1988. Algorithms for Clustering Data. Prentice-Hall, Englewood Cliffs, NJ.
- [4] KAUFMAN, L. and ROUSSEEUW, P. 1990. Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley and Sons, New York, NY.
- [5] DUBES, R.C. 1993. Cluster Analysis and Related Issues. In Chen, C.H., Pau, L.F., and Wang, P.S. (Eds.) Handbook of Pattern Recognition and Computer Vision, 3-32, World Scientific Publishing Co., River Edge, NJ.47
- [6] EVERITT, B. 1993. Cluster Analysis (3rd ed.). Edward Arnold, London, UK.
- [7] MIRKIN, B. 1996. Mathematic Classification and Clustering. Kluwer Academic Publishers.
- [8] JAIN, A.K, MURTY, M.N., and FLYNN P.J. 1999. Data clustering: a review. ACM Computing Surveys, 31, 3, 264-323.
- [9] FASULO, D. 1999. An analysis of recent work on clustering algorithms. Technical Report UW-CSE01 -03-02, University of Washington.
- [10] KOLATCH, E. 2001. Clustering Algorithms for Spatial Databases: A Survey. PDF is available on the Web.
- [11] HAN, J., KAMBER, M., and TUNG, A. K. H. 2001. Spatial clustering methods in data mining: A survey. In Miller, H. and Han, J. (Eds.) Geographic Data Mining and Knowledge Discovery, Taylor and Francis.
- [12] JAIN, A.K. and FLYNN, P.J. 1966. Image segmentation using clustering. In Advances in Image Understanding: A Festschrift for Azriel Rosenfeld, IEEE Press, 65-83.