Proceedings of the 2014 Federated Conference on Computer Science and Information Systems pp. 695–699
DOI: 10.15439/2014F258
ACSIS, Vol. 2

An Optimized Version of the K-Means Clustering Algorithm

Cosmin Marian Poteraș
University of Craiova
Faculty of Automation, Computers and Electronics
Blvd. Decebal nr. 107,
Craiova, Romania
Email: cpoteras@software.ucv.ro

Marian Cristian Mihăescu
University of Craiova
Faculty of Automation, Computers and Electronics
Blvd. Decebal nr. 107,
Craiova, Romania
Email: mihaescu@software.ucv.ro

Mihai Mocanu
University of Craiova
Faculty of Automation, Computers and Electronics
Blvd. Decebal nr. 107,
Craiova, Romania
Email: mmocanu@software.ucv.ro

***Abstract*—This paper introduces an optimized version of the standard K-Means algorithm. The optimization refers to the running time and it comes from the observation that after a certain number of iterations, only a small part of the data elements change their cluster, so there is no need to re-distribute all data elements. Therefore the implementation proposed in this paper puts an edge between those data elements which won't change their cluster during the next iteration and those who might change it, reducing significantly the workload in case of very big data sets. The prototype implementation showed up to 70% reduction of the running time.**

I. Introduction

THE MORE data volumes continue to grow in complexity and diversity, the harder it is to structure and manipulate them. Finding data with similar characteristics and labeling it accordingly has become one of the greatest challenges in nowadays data analyzing applications. Grouping data based on common characteristics is what we call clustering. Clustering algorithms fall into the unsupervised classification techniques category. They classify a set of objects into a subset of clusters based on similarities between them. Differences between clusters have to be obvious and clearly expressed.

Clustering can be applied to a wide range of domains like: marketing [1] (market analysis and recommendations, methodological weaknesses), medicine [2] (medical image segmentation), e-business [3] (comments analysis on news portal) or e-learning [4][5] (prediction of students' academic performance).

There is no secret recipe for choosing the best clustering algorithm. The choice should be based on experimental studies and data description possibly mixed with some human intuition unless there is no obvious mathematical model.

Some of the problems raised by clustering algorithms which worth investing research efforts are: scalability, handling heterogeneous data, execution time complexity when dealing with very large data sets, multi-dimensional data, etc.

The standard K-Means algorithm represents one of the most popular unsupervised exclusive clustering algorithms. It has been successfully applied to medical image segmentation as shown in [6] where the authors propose an algorithm for the segmentation of three-dimensional (3-D) image data based on a combination of adaptive K-Means clustering and knowledgebased morphological operations.

K-Means is based on the minimization of the average squared Euclidean distance between the data items and the cluster's center (called centroid). The results of the algorithm are influenced by the initial centroids. Different initial configurations might lead to different final clusters. The cluster's center is defined as the mean of the items in a cluster.

This paper focuses on the execution of the K-Means algorithm, namely it tries to improve the running time when dealing with high volumes of data. The standard implementation of K-Means consists of successive iterations. Each iteration requires visiting the entire data set in order to assign data objects to their corresponding cluster. At the end of each iteration, new centroids are being computed so that the next iteration will employ the new centroids. After a certain number of such iterations, the centroids will keep the same and the algorithm stops.

The optimization proposed by this paper relies on the observation that after performing a number of iterations, just a small part of the data set might change the cluster it belongs to. Our implementation traces a border between that part of the data set which could possibly switch to another cluster and the data that will hold the cluster it belongs to, during the next iteration. As K-Means algorithm's execution advances, the centroids come closer to their final position. The more iterations are performed, the less the centroids deviate from their current position, resulting in less data objects to be checked against. Similar to the classical implementation, the final clusters are sensitive to the initial configuration (initial centroids).

The rest of the paper is structured as follows: section II presents previous results in speeding up the K-Means algorithm, section III describes the proposed optimization for the K-Means algorithm, section IV experimentally evaluates the potential of the proposed optimization, while section V concludes the paper and presents our future research intentions.

978-83-60810-58-3/$25.00

Algorithm 1 Standard K-Means algorithm

1. Choose k data objects representing the cluster centroids;
2. Assign each data object of the entire data set to the cluster having the closest centroid
3. Compute new centroid for each cluster, by averaging the data objects belonging to the cluster
4. If at least one of the centroids has changed, go to step 2, otherwise go to step 5
5. Output the clusters.

II. Related Work

Researches have shown special interest for speeding up the K-Means algorithm, by either reducing the computation complexity or by adopting K-Means implementations for parallel and distributed platforms.

In [7] the authors propose an efficient implementation of the Lloyd's (K-Means) algorithm called the filtering algorithm which employs kd-trees for storing the data elements.

In [8], the authors propose an algorithm which reduces the computations for determining the closest centroid of a data element, by making use of the observation that if a data element gets closer to the centroid defined at the previous iteration, it won't switch the cluster it belongs to.

Other strategies [9][10][11][12][13] focused on parallelizing the K-Means algorithm and take advantage of powerful parallel and distributed environments, addressing issues specific to those environments, like data availability, synchronization, etc. and adapting the K-Means algorithm to different distributed architectures (client-server, peer-to-peer, etc). The results were satisfactory for very big data sets.

The highly-parallel GPUs haven't been ignored either. Papers [14][15] propose parallel implementations of K-Means to be run on GPUs.

III. Optimized K-Means Method

Before proceeding with our optimized K-Means, let us examine first the standard K-Means algorithm. It consists of repetitive steps, as presented in algorithm 1.

Let us have a look at the algorithm and try to identify what step causes the most computations. Obviously in case of very large data sets, step number 2 would require the biggest time frame in the algorithm's execution. The bigger the data set, the wider the time frame of step 2's execution as it visits each data object and performs some computations on it. The question that arises here is: do we need to visit the entire data space? Figure 1 illustrates the centroids' evolution in a standard K-Means execution.

The data objects are represented by 2D points. The algorithm starts with centroids A1, B1 and C1, which change their position with each iteration, successively to A_i, B_i and C_i, where i=1..6 until they no longer change. If we take a closer look, we can easily see that as the execution progresses, the centroids get very close to their final position. Actually, it happens very often that after only few iterations, the centroids undergo their trip to the very close neighborhood of their final position. This observation leads us to the conclusion that most of the data objects belonging to a cluster whose centroid slightly moves, should not be affected by the move; they will remain part of the same cluster during the next iteration. The less the centroid moves, the less points get affected by the move.

Being able to determine which of the data objects could be affected by a move, could lead us to a very important improvement on step number 2 as we no longer need to visit the entire data set, but just a small list of data objects (let us call that the 'border' list). Before deciding which data objects should be placed into the 'border' list we need to establish the criteria that need to be fulfilled by a data element so that it can be considered a 'border' element. Let us consider Figure 2 which assumes the iteration i is to be computed.

Let point P be part of cluster C. All other points have been omitted on purpose for the ease of presentation. Point P is part of cluster C as the distance from P to C (d_{PC})is less than the distance to A (d_{PA}) and less than the distance to B (d_{PB}). We want to know, how far away is point P from jumping to another cluster. That would obviously be:

$$e_P = min(d_{PA}\text{-}d_{PC},\ d_{PB} - d_{PC}) \quad (1)$$

We've labeled as e_P the distance from P to the closest edge. We can say that point P is e_P-away from switching the cluster.

At the end of iteration i, centroids need to be updated based on the new clusters' configuration. Let us assume that centroid A moved to A', centroid B moved to B' and centroid C moved to C'.

In the context shown in figure 2, the worst case scenario for point$P$ would be: point C got farther away from P by $|CC'|$ while point A got closer by $|AA'|$ and point B got closer by $|BB'|$. What would be the condition for P to stay in cluster C? Obviously that would be

$$e_P > |CC'| + |AA'| \quad (2)$$

and

$$e_P > |CC'| + |BB'| \quad (3).$$

To simplify a little the algorithm and reduce the computations, we can blend conditions (2) and (3):

$$e_P > 2 * max(|AA'|, |BB'|, |CC'|). \quad (4)$$

That being said, we've just found a way of determining whether a point is part of the 'border' list or not.

But, we're still not ok because checking the inequality for each of our data elements at each iteration gets us back to where we started. To avoid such computations, we can map all of our data elements into wider intervals for the value of e, as shown in algorithm 2:

The algorithm 2 groups points with close values of e so that instead of visiting each data element and checking against their close-to-the-edge distance, we can do that that for the entire group. This compromise is the key of the entire optimization. The *WIDTH* constant has a big influence on the optimization. If the value of *WIDTH* is to small, then the

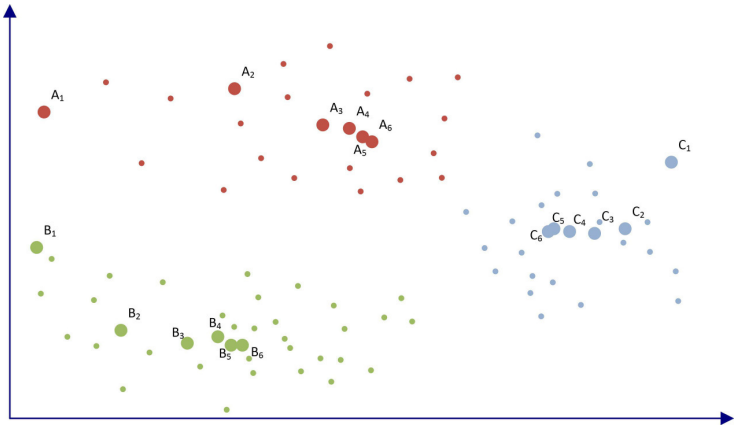

Fig. 1. Example of centroids evolution

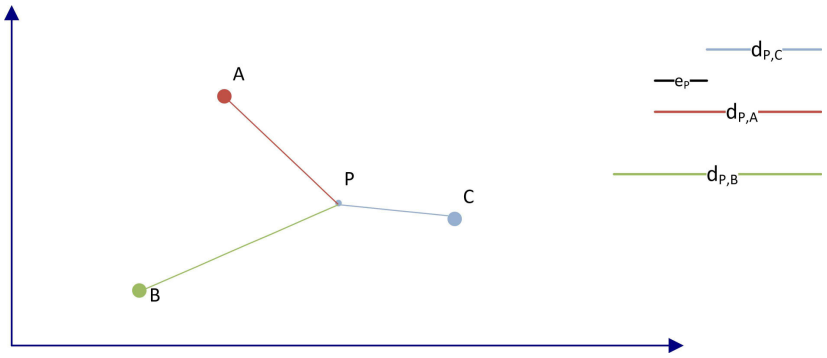

Fig. 2. Edge condition for data elements

Algorithm 2 Optimized K-Means algorithm

1. Define constant *WIDTH*
2. Define intervals $I_i = [i * WIDTH, (i+1) * WIDTH)$ and tag them with value $i * WIDTH$
3. Mark the entire data set to be visited
4. For each point to be visited
5. Compute $e = min(d_{PC_l} - d_{PC_w})$ where C_w is the center of the winner (closest) cluster and C_l, $l=1..k$, $l \neq w$ stands for all other centroids
6. Map all points with $i * WIDTH < e < (i+1) * WIDTH$ to interval $i * WIDTH$ where i is a positive integer
7. Compute new centroids C_j, where $j=1..k$ and their maximum deviation $D = max(|C_jC_j'|)$
8. Update I_i's tag by subtracting $2 * D$ (points owned by this interval got closer to the edge by $2 * D$)
9. Pick up all points inside intervals whose tag is less or equal to 0, and go to *4* to revisit them

number of intervals will increase and the workload involved by checking and updating the intervals at each iteration can increase significantly. If the value of *WIDTH* is to big, then the number of points for each interval increases, and even though the number of intervals is reduced, the performance loss is obvious when a big interval is marked for re-visiting.

One can easily observe that the quality of the final clusters is not affected by the proposed optimization. At each iteration, the composition of clusters is exactly the same as if we would have run the standard K-Means.

IV. Prototype Evaluation

To validate the potential of our K-Means optimization, we've implemented a prototype that runs on data sets composed of 2D points having their coordinates greater than 0 and less than 1. There have been randomly generated data sets of different sizes ranging from 100,000 points to 5,000,000 points. The random numbers generator used for generating the points coordinates, made use of an uniform distribution. The data sets were split into 4, 8 or 12 clusters. The points belonging to each cluster have been grouped into 0.1 wide intervals (*WIDTH = 0.1*). The running time of the optimized algorithm was compared to the running time of the standard K-Means processing of the same data sets in exactly the same conditions (same centroids, same execution environment). Experiments were conducted on a machine consisting of an Intel i7-4700MQ CPU, 8 GB RAM memory. Many runs were carried out for each use case, and the running times were averaged. Let us have a look at the results.

Table I presents the running time for both optimized and standard versions of the K-Means algorithm where the data

TABLE I
OPTIMIZED K-MEANS VS. STANDARD K-MEANS RUNNING TIMES - 4 CLUSTERS

Data set size	Running Time Standard K-Means (ms)	Running Time Optimized K-Means (ms)	Improved by(%)
100,000	6300	1967	68.77
250,000	14382	4528	68.51
500,000	40321	11402	71.72
750,000	55088	15943	71.05
1,000,000	73957	21436	71.01
2,000,000	140339	42962	69.38
5,000,000	420516	116630	72.26

TABLE II
OPTIMIZED K-MEANS VS. STANDARD K-MEANS RUNNING TIMES - 8 CLUSTERS

Data set size	Running Time Standard K-Means (ms)	Running Time Optimized K-Means (ms)	Improved by(%)
100,000	75664	28418	62.44
250,000	148472	57485	61.28
500,000	629777	224277	64.38
750,000	1004100	359230	64.22
1,000,000	1096291	396923	63.79
2,000,000	2798319	1006918	64.01
5,000,000	9685205	3355996	65.34

sets were divided into 4 clusters.

The running time has been reduced by up to 72.26%

Table II, presents the running time for both optimized and standard versions of the K-Means algorithm where the data sets were divided into 8 clusters. The running time has been reduced by up to 64.48%, which is less than the improvement shown in case of only 4 clusters. That can be explained by the fact that the more clusters we use, the higher the chances are for a bigger maximum centroid deviation (max($|C_iC_i|$), where $i = 1..K$), which decrease the chances of fulfilling inequality (4) causing more points to become part of the 'border' area.

Table III, presents the running time for both optimized and standard versions of the K-Means algorithm where the data sets were divided into 12 clusters. The best improvement we have got here rises up to 53.42% which again, is less than the improvement we have got for 4 and 8 clusters. These results confirm that a higher number of clusters results in wider 'border' areas, reducing the computational gain.

V. CONCLUSIONS AND FUTURE WORK

The paper introduces an optimized version of the K-Means algorithm. The optimization refers to the running time. Optimization comes from the considerable reduction of the data space that is re-visited at each loop.

The algorithm defines a 'border' area made of those points that are close enough to the edge of their cluster so that the next centroids move could cause them to switch clusters.

A prototype implementation of a domain specific data set has been evaluated. The implementation assumes the data set is made of 2D points with their coordinates between 0 and 1. The data set has been generated using a uniform distribution generator.

TABLE III
OPTIMIZED K-MEANS VS. STANDARD K-MEANS RUNNING TIMES - 12 CLUSTERS

Data set size	Running Time Standard K-Means (ms)	Running Time Optimized K-Means (ms)	Improved by(%)
100,000	53879	27388	49.16
250,000	186923	90140	51.77
500,000	323584	158888	50.89
750,000	681331	317328	53.42
1,000,000	809675	377522	53.37
2,000,000	1650657	776873	52.93
5,000,000	4835146	2324173	51.93

Running times for 4, 8 and 12 centroids have been compared to the running times of the standard K-Means algorithm, showing a reduction ranging from 49.16% to 72.26 %. At this stage we can not confirm that the improvement shown by the prototype will be held in all real-world use cases, but the results are certainly encouraging.

Our future research will focus on the domain-independent implementation and evaluation of the algorithm. The algorithm's scalability as well as data sensitivity (form and distribution) are to be analyzed with the purpose of concluding upon what would be the best and the worst environments (data and configuration) for the algorithm.

A natural question would be if the algorithm can be improved. One can easily note that the grouping intervals' width might be a point of vulnerability for the performance gain. The lower the width, the more intervals are to be checked; the higher the width, the more points are to be checked when their interval's distance to the edge goes bellow 0. A tradeoff has to be made here, therefore we will also focus on designing an auto-calibration algorithm for the interval width.

Implementations for parallel and distributed environments, as well as integration with existing frameworks (Hadoop, Mahout) are also on our goals list as they could lead our way towards big data sets.

REFERENCES

[1] Dolnicar, S, Using cluster analysis for market segmentation - typical misconceptions, established methodological weaknesses and some recommendations for improvement, Australasian Journal of Market Research, 2003, 11(2), 5-12.

[2] Ng, H.P., Ong, S.H.; Foong, K.W.C.; Goh, P.S.; Nowinsky, W.L. - Medical Image Segmentation Using K-Means Clustering and Improved Watershed Algorithm, 7th IEEE Southwest Symposium on Image Analysis and Interpretation, March 26-28, 2006, Denver, Colorado, pages 61-66

[3] Hongwei Xie, Li Zhang ; Jingyu Sun ; Xueli Yu - Application of K-means Clustering Algorithms in News Comments - The International Conference on E-Business and E-Government, May 2010, Guangzhou, China, pages 451-454

[4] kK Oyelade, O. J, Oladipupo, O. O, Obagbuwa, I. C - Application of K-Means Clustering algorithm for prediction of Students' Academic Performance, (IJCSIS) International Journal of Computer Science and Information Security, Vol. 7, No. 1, 2010, pages 292–295

[5] Burdescu, D.D.; Mihaescu, M.C., "Enhancing the Assessment Environment within a Learning Management Systems," EUROCON, 2007. The International Conference on "Computer as a Tool", vol., no., pp.2438,2443, 9-12 Sept. 2007

[6] Chang Wen Chen, Jiebo Luo, Kevin J. Parker - Image Segmentation via Adaptive K-Mean Clustering and Knowledge-Based Morphological Operations with Biomedical Applications, IEEE Transactions on Image Processing, VOL. 7, NO. 12, DECEMBER 1998, pages 1673 - 1683

[7] T. Kanungo, D.M. Mount, K.D. Piatko, N.S. Netanyahu, R. Silverman, A. Y. Wu - An efficient k-means clustering algorithm: analysis and implementation, Pattern Analysis and Machine Intelligence, IEEE Transactions on (Volume:24, Issue: 7), pages 881-892, July 2002, ISSN 0162-8828

[8] Fahim A.M., Salem A.M., Torkey F.A., Ramadan M.A. - An Efficient Enhanced K-means Clustering Algorithm Journal of Zhejiang University SCIENCE A, ISSN 1009-3095 (Print); ISSN 1862-1775 (Online), pages 1626 - 1633, 2006 7(10)

[9] Souptik Datta, Chris Giannella, Hillol Kargupta - K-Means Clustering Over a Large, Dynamic Network, Proceedings of the Sixth SIAM International Conference on Data Mining, April 20-22, 2006, Bethesda, MD, USA. SIAM 2006 ISBN 978-0-89871-611-5, pages 153 - 164

[10] Yufang Zhang, Zhongyang Xiong, Jiali Mao, Ling Ou - The Study of Parallel K-Means Algorithm, Proceedings of the 6th World Congress on Intelligent Control and Automation, June 21 - 23, 2006, Dalian, China, pages 5868 - 5871

[11] Jing Zhang, Gongqing Wu, Xuegang Hu, Shiying Li, Shuilong Hao - A Parallel K-means Clustering Algorithm with MPI, 4th Internation Symposium on Parallel Architectures, Algorithms and Programming, ISBN 978-0-7695-4575-2, pages 60-64, 2011

[12] Fazilah Othman, Rosni Abdullah, Nur'Aini Abdul Rashid, and Rosalina Abdul Salam - Parallel K-Means Clustering Algorithm on DNA Dataset, Parallel and Distributed Computing: Applications and Technologies, Lecture Notes in Computer Science Volume 3320, 2005, pp 248-251

[13] Jitendra Kumar, Richard T. Mills, Forrest M. Hoffman, William W. Hargrove - Parallel k-Means Clustering for Quantitative Ecoregion Delineation Using Large Data Sets, Proceedings of the International Conference on Computational Science, ICCS 2011, Procedia Computer Science 4 (2011) 1602-1611

[14] Reza Farivar, Daniel Rebolledo, Ellick Chan, Roy Campbell - A Parallel Implementation of K-Means Clustering on GPUs, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2008, Las Vegas, Nevada, USA, July 14-17, 2008, 2 Volumes. CSREA Press 2008 ISBN 1-60132-084-1, pages 340-345

[15] Mario Zechner, Michael Granitzer - Accelerating K-Means on the Graphics Processor via CUDA, The First International Conference on Intensive Applications and Services INTENSIVE 2009, 20-25 April, Valencia, Spain, pages 7-15, ISBN 978-1-4244-3683-5