Comparative Study of Apache Spark MLlib Clustering Algorithms

Sasan Harifi(✉), Ebrahim Byagowi, and Madjid Khalilian

Department of Computer Engineering, Karaj Branch, Islamic Azad University, Karaj, Iran
{s.harifi,ebrahim.byagowi,khalilian}@kiau.ac.ir

Abstract. Clustering of big data has received much attention recently. Analytics algorithms on big datasets require tremendous computational capabilities. Apache Spark is a popular open- source platform for large-scale data processing that is well-suited for iterative machine learning tasks. This paper presents an overview of Apache Spark Machine Learning Library (Spark.MLlib) algorithms. The clustering methods consist of Gaussian Mixture Model (GMM), Power-Iteration Clustering method, Latent Dirichlet Allocation (LDA), and k-means are completely described. In this paper, three benchmark datasets include Forest Cover Type, KDD Cup 99 and Internet Advertisements used for experiments. The same algorithms that can be compared with each other, compared. For a better understanding of the results of the experiments, the algorithms are described with suitable tables and graphs.

Keywords: Clustering · k-means · Bisecting k-means · Spark MLlib · Big data · KDD cup 99 · Cover type · Train time · Cohesion

1 Introduction

Clustering is an unsupervised learning method which tries to find some distributions and patterns in unlabeled datasets. Usually, those points in the same cluster should have more similarity than other points in other clusters [1].

Considered that clustering pursues the arrangement of a family of items into homogeneous groups, taking into account inherent quantitative and qualitative information about them. However, the practical implementation of some clustering techniques requires certain mathematical assumptions that sometimes are difficult, if not impossible, to be checked. Some of these assumptions are quite often simply hidden in the interpreter's mind [2].

Also clustering items according to some notion of similarity is a major primitive in machine learning. Correlation clustering serves as a basic means to achieve this goal: given a similarity measure between items, the goal is to group similar items together and dissimilar items apart. In contrast to other clustering approaches, the number of clusters is not determined a priori, and good solutions aim to balance the tension between grouping all items together versus isolating them [3].

Y. Tan et al. (Eds.): DMBD 2017, LNCS 10387, pp. 61–73, 2017.
DOI: 10.1007/978-3-319-61845-6_7

Clustering has been used in many areas such as machine learning, pattern recognition, image processing, marketing and customer analysis, agriculture, security and crime detection, information retrieval, and bioinformatics [1].

Clustering algorithms can be categorized into partitioning methods, hierarchical methods, density-based methods, grid-based methods, and model-based methods. Recently, quantum clustering, spectral clustering, and synchronization clustering have been presented and gained some attention.

In this paper MLlib clustering algorithms are described. Rest of the paper is structured as follows: Sect. 2 consists of related works, Sect. 3 describes Spark MLlib clustering algorithms. Section 4 includes comparison by experimental setup. Section 5 represents conclusions.

2 Related Works

In this section, we provide a brief description of the related works. Daniel Gómez et al. [2] introduced a hierarchical clustering algorithm in networks based upon a first divisive stage to break the graph and a second linking stage which is used to join nodes. They show that this algorithm is very flexible as well as quite competitive in relation with a set previous algorithms.

Madjid Khalilian et al. [4] proposed method Divide-and-Conquer Stream and compared it with Stream and incremental online Con-Stream for efficiency and accuracy in clustering results in data stream clustering. Stream utilizes Divide-and-Conquer method to overcome difficulties in data stream clustering and should be distinguished from the proposed method. Divide-and-Conquer Stream uses Divide-and-Conquer method based on length of vector as it is described later whereas Stream divides data by using sampling. In another study [5], he evaluate different aspects of existing obstacles in data stream clustering.

Renxia Wan et al. [6] extended Fuzzy C-Means and proposed a weighted fuzzy algorithm for clustering data stream. The algorithm tries to fuzzily cluster data stream. Their Experimental results on both standard datasets KDD-CUP'99 and synthetic datasets show its superiority over the traditional FCM algorithms.

Jingdong Wang et al. [7] proposed a novel approximate k-means algorithm to greatly reduce the computational complexity in the assignment step. Their approach is motivated by the observation that most active points changing their cluster assignments at each iteration are located on or near cluster boundaries.

Francesco Finazzi et al. [8] considered two approaches for clustering of time series. The first is a novel approach based on a modification of classic state-space modelling while the second is based on functional clustering. For the latter, both k-means and complete-linkage hierarchical clustering algorithms are adopted. The two approaches are compared using a simulation study. For more details see [8].

Matthias Brust et al. [9] proposed a 3-D clustering algorithm for autonomous positioning (virtual forces based clustering algorithm) of aerial drone networks based on virtual forces. These virtual forces induce interactions among drones and structure the system topology. According to their statements, the advantages of their approach are

that virtual forces enable drones to self-organize the positioning process and virtual forces based clustering algorithm can be implemented entirely localized.

Celal Ozturk et al. [10] improved searching mechanism of the discrete binary artificial bee colony algorithm by the efficient genetic selection and they tested its performance on the dynamic clustering problem, in which the number of clusters is determined automatically. Moreover, they demonstrated the superiority of their proposed algorithm by comparing it with the discrete binary artificial bee colony, binary particle swarm optimization (BPSO), genetic algorithm (GA), Fuzzy C-means (FCM) and k-means algorithms on benchmark problems.

Shifei Ding et al. [11] reviewed the development and trend of data stream clustering and analyzes typical data stream clustering algorithms proposed in recent years, such as Birch algorithm, Local Search algorithm, Stream algorithm and CluStream algorithm. They also summarized the latest research achievements in this field and introduced some new strategies to deal with outliers and noise data.

Yan et al. [12] proposed a multitask clustering framework for activity of daily living analysis from visual data gathered from wearable cameras. Their intuition is that, even if the data are not annotated, it is possible to exploit the fact that the tasks of recognizing everyday activities of multiple individuals are related, since typically people perform the same actions in similar environments. For more details see [12].

3 Spark.MLlib Clustering Algorithms

Apache Spark is a cluster computing platform that is used for general purposes and designed to be fast [13, 14].

On the speed side, Spark expands MapReduce model to support more types of computing like interactive queries and stream processing. Speed is very important in processing large datasets, as it means the difference between exploring data interactively and waiting minutes or hours. One of the main features that Spark proposes for speed, is the ability to compute in memory. But this system is more efficient than the Hadoop MapReduce to run complex applications on disk, as well.

On the generality side, Spark is designed to cover a wide range of workloads that were previously required separate distributed systems, including batch applications, algorithms, iterative, interactive query and streaming. By supporting these workloads in the same engine, Spark makes easy and inexpensive, the combination of different types of processing are often required in production data analysis pipelines. Spark offers APIs in Java, Scala and Python. Spark components are shown in Fig. 1.

MLlib package includes common machine learning functionality that is the focus of this paper. MLlib includes several types of machine learning algorithms such as classification, regression, clustering and collaborative filtering and also includes model evaluation and data import. In the following MLlib clustering algorithms are described.

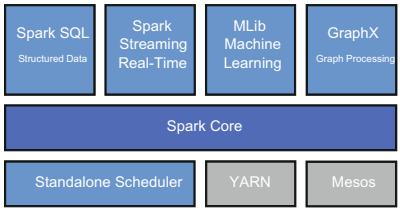

Fig. 1. Spark components [13].

3.1 Gaussian Mixture Model

Model-based clustering is assumed the data comes from a source with several subpopulations. Each subpopulations are modeled individually and the entire population is a mixture of these sub-populations. The final model is a finite mixture model [15]. When data are multivariate continuous observations, the component parameterized density is usually a multidimensional Gaussian density [15].

According to the model-based perspective, each cluster can be mathematically provided by a parametric distribution. All datasets can be modeled by a mixture of these distributions [16]. The model widely used is a mixture of Gaussians:

$$P(x|\Theta) = \sum_{i=1}^{K} \alpha_i p_i(x|\theta_i). \tag{1}$$

where each p_i is a Gaussian density function parameterized by θ_i and $\Theta = (\alpha_1, \ldots, \alpha_K, \theta_1, \ldots, \theta_K)$ such that $\sum_{i=1}^{K} \alpha_i = 1$. Here it is assumed k component densities mixed together with k mixing coefficients α_i. In Eq. (1) $X = (x_1, \ldots, x_m)$ is a set of data points. It is expected to find Θ such that $p(X|\Theta)$ is a maximum. This is known as the Maximum Likelihood (ML) estimate for Θ. In order to estimate Θ, it is typical to introduce the log-likelihood function defined as follows:

$$\mathcal{L}(\Theta) = \log P(X|\Theta) = \log \prod_{i=1}^{m} P(x_i|\Theta) = \sum_{i=1}^{m} \log\left(\sum_{j=1}^{K} \alpha_j p_j(x_i|\theta_j)\right). \tag{2}$$

3.2 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is a generative probabilistic model of a corpus. The basic idea is that documents are represented as random mixtures over latent topics, where each topic is characterized by a distribution over words [18]. More generally, LDA helps to explain the similarity of data by grouping features of this data into unobserved sets. A mixture of these sets then constitutes the observable data [19]. LDA was first introduced by Blei et al. [18]. The modeling process of LDA can be described as finding a mixture of topics for each resource, i.e., $P(z|d)$, with each topic described by terms following another probability distribution [19], i.e., $P(t|z)$. This can have such formula as:

$$P(t_i|d) = \sum_{j=1}^{z} P(t_i|z_i = j)P(z_i = j|d). \tag{3}$$

where $P(t_i|d)$ is the probability of the i^{th} term for a given document d and z_i is the latent topic. $P(t_i|z_i = j)$ is the probability of t_i within topic j. $P(z_i = j|d)$ is the probability of picking a term from topic j in the document. The number of latent topics Z has to be defined in advance and allows to adjust the degree of specialization of the latent topics.

3.3 Power Iteration Clustering

For power iteration, Consider a set of data points: $\{x_1, x_2, \ldots, x_n\}$, where x is a d-dimensional vector, and some notion of similarity, for example:

$$s(x_i, x_j) = \exp\left(-\frac{\left\|x_i - x_j\right\|_2^2}{2\sigma^2}\right). \tag{4}$$

where σ is a scaling parameter that controls the kernel width. An affinity matrix A can built with $a_{ij} = s(x_i, x_j)$ if $i \neq j$ and $a_{ij} = 0$ if $i = j$. The degree matrix associated with A, denoted by D, is a diagonal matrix with the diagonal entries equal to the row sums of A, i.e., $D_{ii} = \sum_j A_{ij}$. A normalized random-walk Laplacian matrix L is defined as $L = \Delta - D^{-1}A$ [26], where Δ is the identity matrix. The intrinsic clustering structure is often revealed by representing the data in the basis composed of the smallest eigenvectors of L. The very smallest eigenvector is a constant vector that doesn't have discriminative power. In another matrix $W = D^{-1}A$ defining, the largest eigenvector is the smallest eigenvector of L. A well-known method for computing the largest eigenvector of a matrix is Power Iteration (PI), which randomly initializes an N-dimensional vector v^0 and iteratively updates the vector by multiplying it with W,

$$v^t = \gamma W v^{t-1}, \quad t = 1, 2, \ldots \tag{5}$$

where γ is a normalizing constant to keep v^t numerically stable.

An interesting property of the largest eigenvector of W discovered by Lin and Cohen [21]. They called their algorithm Power Iteration Clustering (PIC), which its details is in [21]. PIC is computationally efficient since it only involves iterative matrix–vector multiplications and clustering of the one dimensional embedding of the original data, which is relatively easy to do. However, similar to other spectral clustering algorithms, a bottleneck for PIC when applied to large datasets lies in the calculation and storage of big matrices [22].

The *spark.mllib* includes an implementation of PIC using GraphX as its backend. It takes a resilient distributed datasets of (*srcId*, *dstId*, *similarity*) tuples and outputs a model with the clustering assignments. The similarities must be nonnegative. PIC assumes that the similarity measure is symmetric. A pair (*srcId*, *dstId*) regardless of the

ordering should appear at most once in the input data. If a pair is missing from input, their similarity is treated as zero.

3.4 k-means

k-means clustering is a method commonly used to automatically partition a dataset into k groups [23]. The number of clusters k is assumed to be fixed in k-means clustering. Let the k prototypes $\left(w_1, w_2, \ldots, w_k\right)$ be initialized to one of the n input patterns $\left(i_1, i_2, \ldots, i_n\right)$. Therefore, $w_j = i_l, j \in \{1, 2, \ldots, k\}, l \in \{1, 2, \ldots, n\}$. C_j is the j^{th} cluster whose value is a disjoint subset of input patterns. The quality of the clustering is determined by the following error function [24]:

$$E = \sum_{j=1}^{k} \sum_{i_l \in C_j} \left| i_l - w_j \right|^2. \tag{6}$$

The number of iterations required can vary in a wide range from a few to several thousand depending on the number of patterns, number of clusters, and the input data distribution. Thus, a direct implementation of the k-means method can be computationally very intensive. This is especially true for typical data mining applications with large number of pattern vectors.

k-means||. The *spark.mllib* implementation includes a parallelized variant of the k-means++ method called k-means|| [17]. A parallel version of the k-means++ initialization algorithm is obtained and empirically demonstrate its practical effectiveness. The main idea is that instead of sampling a single point in each pass of the k-means++ algorithm, $O(k)$ points in each round is sampled and repeat the process for approximately $O(\log n)$ rounds [25]. At the end of the algorithm, $O(k \log n)$ points are left form a solution that is within a constant factor away from the optimum. These $O(k \log n)$ points into k initial centers for the Lloyd's iteration are clustered again. This initialization algorithm, which is called k-means||, is quite simple and lends itself to easy parallel implementations. However, the analysis of the algorithm turns out to be highly non-trivial, requiring new insights, and is quite different from the analysis of k-means++.

Bisecting k-means. Bisecting k-means can often be much faster than regular k-means, but it will generally produce a different clustering [17]. Bisecting k-means is a kind of hierarchical clustering. Hierarchical clustering is one of the most commonly used method of cluster analysis which seeks to build a hierarchy of clusters. Also the bisecting k-means has a time complexity which is linear in the number of documents. If the number of clusters is large and if refinement is not used, then bisecting k-means is even more efficient than the regular k-means algorithm.

Streaming k-means. When data arrive in a stream, we may want to estimate clusters dynamically, updating them as new data arrive. The *spark.mllib* also provides support for streaming k-means clustering, with parameters to control the decay or forgetfulness of the estimates. The algorithm uses a generalization of the mini-batch k-means update

rule. For each batch of data, all points to their nearest cluster are assigned, compute new cluster centers, then update each cluster.

4 Comparison by Experimental Setup

In this section at first the system configuration is introduced and then datasets used in this paper are described. Then the experiments along with their tables are explained and finally the graph of each experiment is shown.

4.1 Configuration

All evaluation experiments have been run on a Core™ processor Intel® CPU i7-4930MX 3.00 GHz with 32 GB RAM and GNU/Linux Ubuntu 16.04 operating system. Implementations have been run on Spark 1.6.1-Scala 2.10.5 for coding.

4.2 Data Sets

We use three benchmark datasets to evaluate the large scale clustering performance:

Forest Cover Type. The Forest Cover Type dataset [27] is composed of 581,012 data points from the US Geological Survey (USGS) and the US Forest Service (USFS). Each data point is represented by a vector of 54 dimensions and assigned to one of 7 classes, each class representing a Forest Cover Type. This is a challenging dataset for any clustering algorithm as it contains ten continuous features, and 44 binary features (four wilderness types and 40 soil types) [28].

KDD Cup 99. Since 1999, KDD'99 has been the most wildly used dataset for the evaluation of anomaly detection methods. This dataset is built based on the data captured in DARPA'98 IDS evaluation program. DARPA'98 is about four gigabytes of compressed raw (binary) tcpdump data of seven weeks of network traffic, which can be processed into about five million connection records, each with about 100 bytes. The two weeks of test data have around two million connection records. KDD training dataset consists of approximately 4,900,000 single connection vectors each of which contains 41 features and is labeled as either normal or an attack, with exactly one specific attack type [28].

Internet Advertisements. The Internet Advertisements dataset is available through the UCI Machine Learning Repository [30]. The 3279 instances of this dataset represent image advertisements and the rest do not. There are missing value in approximately 20 percent of the instances. The proposed task is to determine which instances contain advertisements based on 1557 other attributes related to image dimensions, phrases in the URL of the documents or the images, and text occurring in or near the image's anchor tag in the documents. The first three attributes encode the image's geometry. The binary local feature indicates whether the image URL points to a server in the same internet

domain as the document URL. The remaining features are based on phrases in various parts of the documents [29].

4.3 Experiments

In this subsection, the experiments and their tables are explained. Please note that in all experiments, the unit of time is second and time refers to training time[1]. Also, unit of cohesion is Davies Bouldin Cohesion [20].

Table 1. Running k-means on KDD Cup 99 with/without Max Iterations

K Value	**A**. k-means on KDD Cup 99				**B**. k-means on KDD Cup 99 (Max Iterations)			
	1. KDD Cup 99 (10%)		**2**. KDD Cup 99 (100%)		**1**. KDD Cup 99 (10%)		**2**. KDD Cup 99 (100%)	
	Cohesion	TT(S)	Cohesion	TT(S)	Cohesion	TT(S)	Cohesion	TT(S)
10	19.59062076	2.707	20.46709771	13.452	19.21086	2.472	20.15667	12.907
20	17.60970038	2.553	15.83935119	19.674	18.08498	3.773	19.62121	29.413
30	13.75523234	2.855	14.70325715	22.295	17.55708	5.778	19.4199	42.413
40	14.38443687	3.085	17.22224174	23.761	18.82096	3.213	18.88906	26.718
50	13.09798986	4.047	15.13062897	23.970	18.63252	3.701	19.83298	24.724
60	12.68632938	3.572	14.56952279	25.982	21.96627	3.250	19.8327	37.368
70	11.07109838	3.275	14.75204737	35.910	22.90242	4.655	19.00778	22.229
80	10.80793379	3.887	13.5979721	29.095	17.91366	3.836	22.73483	27.774
90	9.862917001	4.027	12.2244106	32.993	21.45907	6.883	19.41999	47.379
100	8.850165571	4.767	10.57837409	40.724	17.62087	3.485	19.18537	35.456

Experiment 1. In the first experiment k-means algorithm is applied on 10 percent KDD Cup 99 dataset and its iteration is considered 10. The results are shown in Table 1(A-1). As can be seen in the Table 1(A-1), if the lowest value (10) considered for k, In terms of time the best clustering time is achieved, but the cohesion of clusters are not suitable. If the value of k is increased, the cohesion is increased too, and therefore more time is spent on clustering. So the best time of clustering is when k value is low, and most coherent clusters are achieved when k value is high.

Experiment 2. In this experiment, experiment 1 is repeated with 100 percent data of KDD Cup 99 dataset. The expected results are achieved. The obtained time and cohesion in experiment 1 are repeated with higher scale in experiment 2. Table 1(A-2) shows k-means experiment with 100 percent of KDD Cup 99 data. As can be seen in Table 1(A-2), if k value is considered the highest value (100), the cohesion becomes very appropriate and the time becomes very inappropriate. The time is approximately 40 s.

Experiment 3. In this experiment, the previous experiments are done with max iterations. The results of experiment 3 are shown in Table 1(B). As can be seen in the Table 1(B-1), with max iterations the time is improved generally and clusters becomes more coherent as well. As can be seen in the Table 1(B-2), k-means algorithm run with

[1] TT(S) in all Tables describes Training Time (Second).

max iterations on full KDD Cup 99 dataset. If the iterations become more, the more coherent clusters with lower time of running are achieved.

Experiment 4. In this experiment, k-means algorithm is applied on the Forest Cover Type dataset. The iteration is considered 10 in this experiment. The results are shown in Table 2(C-1).

Table 2. Running k-means and Bisecting k-means on Forest Cover Type and Ads Dataset

K Value	**C**. Forest Cover Type dataset				**D**. Internet Advertisements dataset			
	1. k-means		**2**. Bisecting k-means		**1**. k-means		**2**. Bisecting k-means	
	Cohesion	TT(S)	Cohesion	TT(S)	Cohesion	TT(S)	Cohesion	TT(S)
10	43.0750461	4.034	44.0392431	11.906	1300.92123	1.340	1328.03257	2.107
20	32.5306901	9.409	38.6940653	17.051	1232.96979	1.725	1263.92542	1.927
30	27.413065	9.927	33.0507343	20.645	1153.17953	2.470	1213.8558	1.954
40	19.6669817	9.025	29.1864917	17.280	1066.70068	2.978	1141.27995	2.353
50	20.8943908	9.829	26.4425548	32.512	1035.76626	4.043	1051.90256	2.414
60	12.5754266	11.644	21.0348994	29.417	987.513667	4.500	1006.61273	2.994
70	14.8703284	13.782	20.0067657	25.880	873.877784	5.043	934.134936	3.113
80	10.0004226	15.651	18.7004353	26.935	816.97583	5.738	826.726687	3.107
90	9.5745834	17.078	17.4730561	28.264	784.340603	6.267	803.771187	3.787
100	10.9289989	16.593	17.0222951	26.964	727.975713	11.114	756.455351	3.847

Experiment 5. In this experiment, the Bisecting k-means algorithm is applied on the Forest Cover Type dataset and the iterations considered 10, the same as experiment 4. The results are shown in Table 2(C-2). Experiments 4 and 5 show that k-means algorithm, has better performance both in terms of time and in terms of cohesion in comparison with Bisecting k-means algorithm on the Forest Cover Type dataset.

Experiment 6. In this experiment k-means algorithm is applied on the Internet Advertisements dataset. The iteration is also considered 10 like the previous experiments. The results are shown in Table 2(D-1).

Experiment 7. In this experiment, Bisecting k-means algorithm is applied on the Internet Advertisements dataset and the iteration is considered 10 again and results are shown in Table 2(D-2). Experiments 6 and 7 show that in Internet Advertisements dataset, Bisecting k-means algorithm has better performance in terms of time. Both k-means and Bisecting k-means algorithms are approximately similar in terms of cohesion.

4.4 Comparisons

In this subsection for a better understanding of the results of the experiments, separately and based on dataset, time and cohesion, the algorithms are compared. Figures 2, 3, 4 and 5 show graphs related to each comparison.

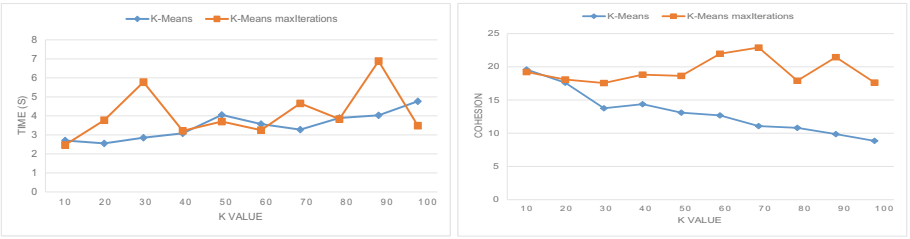

Fig. 2. Left graph: Comparison of k-means and k-means maxIterations training time on KDD Cup (10%). Right graph: Comparison of k-means and k-means maxIterations cohesion on KDD Cup (10%).

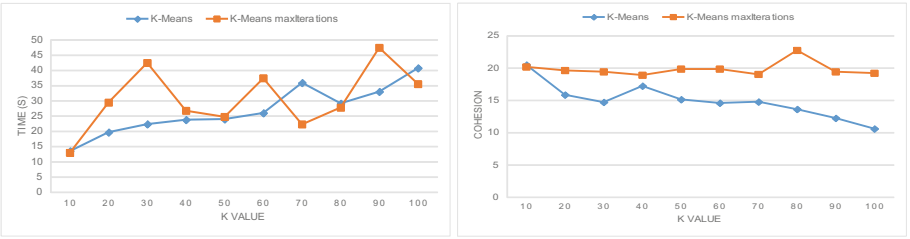

Fig. 3. Left graph: Comparison of k-means and k-means maxIterations training time on KDD Cup (100%). Right graph: Comparison of k-means and k-means maxIterations cohesion on KDD Cup (100%).

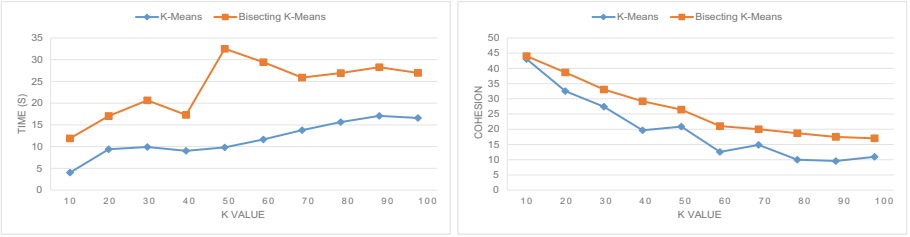

Fig. 4. Left graph: Comparison of k-means and Bisecting k-means training time on FCT dataset. Right graph: Comparison of k-means and Bisecting k-means cohesion on FCT dataset.

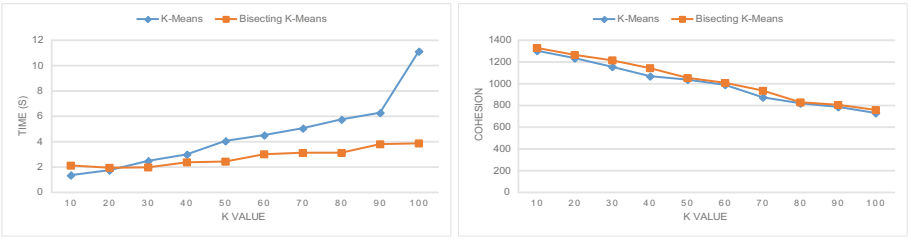

Fig. 5. Left graph: Comparison of k-means and Bisecting k-means training time on Ads dataset. Right graph: Comparison of k-means and Bisecting k-means cohesion on Ads dataset.

5 Conclusions

In this paper, at first MLlib library clustering algorithms are described in details. Being brief and usefulness of descriptions can help the researchers for future works. Then comparison of two algorithms, Bisecting k-means and k-means on three datasets, KDD Cup 99, forest cover type and was Internet Advertisements is done. Algorithms that compared with each other, were the same. In the experiments, Power-Iteration algorithm was not compared because it uses graph as an input. Low speed of GMM algorithm on the selected datasets was the reason of ignoring it in comparison with other algorithms. For example, for GMM algorithm in conditions similar to experiment 1, if the value of k equals 10, the amount of cohesion is 27.813 and train time is 973.058 s. So, this algorithm is non-optimal and slow and is not suitable for using in big data. LDA algorithm is also using with documents. The results of experiments and the comparisons show that depending on the circumstances, type and dimension of data, each algorithm can be best in clustering. Graphs of time and cohesion are applied in this paper to find a better view of the results of the experiments.

Future works can be the comparison of other MLlib library clustering algorithms. Even other parts of spark such as GraphX, Spark Streaming and Spark SQL can be compared. Being brief and usefulness of descriptions of other parts also can help to the quality of studies and reducing the time of research.

References

1. Chen, X.: A new clustering algorithm based on near neighbor influence. Expert Syst. Appl. **42**, 7746–7758 (2015)
2. Gómez, D., Zarrazola, E., Yáñez, J., Montero, J.: A Divide-and-Link algorithm for hierarchical clustering in networks. Inf. Sci. **316**, 308–328 (2015)
3. Pan, X., Papailiopoulos, D., Oymak, S., Recht, B., Ramchan-dran, K., I. Jordan, M.: Parallel correlation clustering on big graphs. In: Advances in Neural Information Processing Systems, pp. 82–90 (2015)
4. Khalilian, M., Mustapha, N., Sulaiman, N.: Data stream clustering by divide and conquer approach based on vector model. J. Big Data **3**, 1 (2016)

5. Khalilian, M., Mustapha, N., Sulaiman, N., Mamat, A.: Different aspects of data stream clustering. In: Elleithy, K., Sobh, T. (eds.) Innovations and Advances in Computer. Information, Systems Sciences, and Engineering, pp. 1181–1191. Springer, New York (2013). doi:10.1007/978-1-4614-3535-8_97
6. Wan, R., Yan, X., Su, X.: A weighted fuzzy clustering algorithm for data stream. In: 2008 ISECS International Colloquium on Computing, Communication, Control, and Management, pp. 360–364. IEEE (2008)
7. Wang, J., Wang, J., Ke, Q., Zeng, G., Li, S.: Fast approximate k-means via cluster closures. In: Multimedia Data Mining and Analytics, pp. 373–395. Springer International Publishing (2015)
8. Finazzi, F., Haggarty, R., Miller, C., Scott, M., Fassò, A.: A comparison of clustering approaches for the study of the temporal coherence of multiple time series. Stochast. Environ. Res. Risk Assess. **29**, 463–475 (2014)
9. Brust, M.R., Turgut, D.: VBCA: a virtual forces clustering algorithm for autonomous aerial drone systems. In: 2016 Annual IEEE Systems Conference (SysCon), pp. 1–6. IEEE (2016)
10. Ozturk, C., Hancer, E., Karaboga, D.: Dynamic clustering with improved binary artificial bee colony algorithm. Appl. Soft Comput. **28**, 69–80 (2015)
11. Ding, S., Wu, F., Qian, J., Jia, H., Jin, F.: Research on data stream clustering algorithms. Artif. Intell. Rev. **43**, 593–600 (2015)
12. Yan, Y., Ricci, E., Liu, G., Sebe, N.: Egocentric daily activity recognition via multitask clustering. IEEE Trans. Image Process. **24**, 2984–2995 (2015)
13. Karau, H., Konwinski, A., Wendell, P., Zaharia, M.: Learning Spark: Lightning-Fast Big Data Analysis. O'Reilly Media, Inc., (2015)
14. Meng, X., Bradley, J., Yuvaz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J.: Mllib: machine learning in apache spark. JMLR **17**(34), 1–7 (2016)
15. Maugis, C., Celeux, G., Martin-Magniette, M.: Variable selection for clustering with gaussian mixture models. Biometrics **65**, 701–709 (2009)
16. He, X., Cai, D., Shao, Y., Bao, H., Han, J.: Laplacian regularized gaussian mixture model for data clustering. IEEE Trans. Knowl. Data Eng. **23**, 1406–1418 (2011)
17. Clustering - RDD-based API - Spark 2.1.0 Documentation. http://spark.apache.org/docs/latest/mllib-clustering.html
18. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. J. Mach. Learn. Res. **3**, 993–1022 (2003)
19. Krestel, R., Fankhauser, P., Nejdl, W.: Latent dirichlet allocation for tag recommendation. In: Proceedings of the Third ACM Conference on Recommender Systems, pp. 61–68. ACM (2009)
20. Davies, D., Bouldin, D.: A cluster separation measure. IEEE Trans. Pattern Anal. Mach. Intell. **PAMI-1**, 224–227 (1979)
21. Lin, F., Cohen, W.: Power iteration clustering. In: Proceedings of the 27th International Conference on Machine Learning (ICML 2010), pp. 655–662 (2010)
22. Yan, W., Brahmakshatriya, U., Xue, Y., Gilder, M., Wise, B.: p-PIC: parallel power iteration clustering for big data. J. Parallel Distrib. Comput. **73**, 352–359 (2013)
23. Wagstaff, K., Cardie, C., Rogers, S., Schrödl, S.: Constrained k-means clustering with background knowledge. In: ICML, pp. 577–584 (2001)
24. Alsabti, K., Ranka, S., Singh, V.: An efficient k-means clustering algorithm. Electrical Engineering and Computer Science (1997)
25. Bahmani, B., Moseley, B., Vattani, A., Kumar, R., Vassil-vitskii, S.: Scalable k-means++. Proc. VLDB Endowment **5**, 622–633 (2012)
26. Meila, M., Shi, J.: A random walks view of spectral segmentation (2001)

27. Blackard, J., Dean, D.: Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. Comput. Electron. Agric. **24**, 131–151 (1999)
28. Kumar, D., Bezdek, J., Palaniswami, M., Rajasegarar, S., Leckie, C., Havens, T.: A hybrid approach to clustering in big data. IEEE Trans. Cybern. **46**, 2372–2385 (2016)
29. Alvarez, S.A., Kawato, T., Ruiz, C.: Mining over loosely coupled data sources using neural experts. In: International Workshop on Multimedia Data Mining. In Conjunction with the Ninth ACM SIGKDD International Conference on Knowledge Dis-cover and Data Mining (2003)
30. Lichman, M.: UCI Machine Learning Repository. University of California, School of Information and Computer Science, Irvine, CA (2013). http://archive.ics.uci.edu/ml