A-BIRCH: Automatic Threshold Estimation for the BIRCH Clustering Algorithm

Conference Paper *in* Advances in Intelligent Systems and Computing · October 2017

DOI: 10.1007/978-3-319-47898-2_18

6 authors, including:

Boris Lorbeer
Technische Universität Berlin
6 PUBLICATIONS **47** CITATIONS
SEE PROFILE

Ana Kosareva
Technische Universität Berlin
2 PUBLICATIONS **21** CITATIONS
SEE PROFILE

Bersant Deva
Technische Universität Berlin
16 PUBLICATIONS **106** CITATIONS
SEE PROFILE

Peter Ruppel
CODE University of Applied Sciences
23 PUBLICATIONS **184** CITATIONS
SEE PROFILE

A-BIRCH: Automatic Threshold Estimation for the BIRCH Clustering Algorithm

Boris Lorbeer, Ana Kosareva, Bersant Deva, Dženan Softić, Peter Ruppel, Axel Küpper

Abstract Clustering algorithms are recently regaining attention with the availability of large datasets and the rise of parallelized computing architectures. However, most clustering algorithms do not scale well with increasing dataset sizes and require proper parametrization for correct results. In this paper we present A-BIRCH, an approach for automatic threshold estimation for the BIRCH clustering algorithm using Gap Statistic. This approach renders the global clustering step of BIRCH unnecessary and does not require knowledge on the expected number of clusters beforehand. This is achieved by analyzing a small representative subset of the data to extract attributes such as the cluster radius and the minimal cluster distance. These attributes are then used to compute a threshold that results, with high probability, in the correct clustering of elements. For the analysis of the representative subset we parallelized Gap Statistic to improve performance and ensure scalability.

1 Introduction

Clustering is an unsupervised learning method that groups a set of given data points into well separated subsets. Two prominent examples of clustering algorithms are k-means [9] and the EM algorithm [4]. This paper addresses two issues with clustering: (1) clustering algorithms usually do not scale well and (2) most algorithms require the number of clusters (cluster count) as input. The first issue is becoming more and more important. For applications that need to cluster, e.g., millions of documents, huge image or video databases, or terabytes of IoT sensor data, scalability

Boris Lorbeer, Ana Kosareva, Bersant Deva, Peter Ruppel, Axel Küpper
Service-centric Networking, Telekom Innovation Laboratories, Technische Universität Berlin
e-mail: {lorbeer|ana.kosareva|bersant.deva|peter.ruppel|axel.kuepper}@tu-berlin.de

Dženan Softić
Service-centric Networking, Telekom Innovation Laboratories, Technische Universität Berlin
e-mail: softic.dzenan@gmail.com

is essential. The second issue severely reduces the applicability of clustering in situations where the cluster count is very difficult to predict, such as data exploration, feature engineering, and document clustering.

An important clustering method is BIRCH [17], which is one of the fastest clustering algorithms available. It outperforms most of the other clustering algorithms by up to two orders of magnitude. Thus, BIRCH already solves the first issue mentioned above. However, to achieve sufficient clustering quality, BIRCH requires the cluster count as input, therefore failing to solve the second issue. This paper describes a method to use BIRCH without having to provide the cluster count, yet preserving cluster quality and speed. We achieve this by omitting the global clustering step and carefully choosing the *threshold* parameter of BIRCH. Following an idea by Bach and Jordan [7], we propose to learn this parameter from representative data. Our approach aims at datasets drawn from two-dimensional isotropic Gaussian distributions which are typical when dealing with, for example, geospatial data.

2 Related Work

Clustering algorithms usually do not scale well, because often they have a complexity of $O(N^2)$ or $O(NM)$, where N is the number of data points and M is the cluster count. Scalability is typically achieved by parallelization of the algorithm in compute clusters, e.g., Mahout's k-means clustering [11] or Spark's distributed versions of k-means, EM clustering, and power iteration [10]. Other parallelization attempts use the GPU. This has been done for k-means [16], EM clustering [8], and others. The bottleneck here is the relatively slow connection between host and device memory if the data does not fit into device memory.

The second issue we are concerned with is the identification of the cluster count. A standard approach is to use one of the clustering algorithms that require the cluster count to be input as a parameter, then run it for each count k inside an interval of likely values. Then, the "elbow method" [14] is used to determine the optimal number k. For probabilistic models, one can apply information criteria like AIC [1] or BIC [13] to rate the different clustering results, see, for example, [18]. But all those methods increase the computation time considerably, especially if there is not enough prior information to keep the range of possible cluster counts small. Some clustering algorithms find the number of clusters directly, without being required to run the algorithm for all possible counts. Two of the more well-known examples are DBSCAN [5] and Gap Statistic [15]. There are also some attempts to improve the clustering quality of BIRCH by changing the algorithm itself, e.g. with non-constant thresholds [6], with two different global thresholds [2], or by using DBSCAN on each CF level to reduce noise [3]. However, while sometimes improving the quality, those approaches slow BIRCH down and still require the cluster count as input.

3 BIRCH

We shortly describe BIRCH, mainly to fix notations. For details, see [17]. BIRCH requires three parameters: the branching factor Br, the threshold T, and the cluster count k. While the data points are entered into BIRCH, a height-balanced tree, the *CF tree*, of hierarchical clusters is built. Each node contains the most important information of the belonging cluster, the *cluster features* (CF). From those, the cluster center $C = 1/n \sum_i^n x_i$, where $\{x_i\}_{i=1}^n$ are the elements of the cluster, and cluster radius $R = \sqrt{1/n \sum_i^n x_i^2}$ can be computed for each cluster. Every new point starts at the root and recursively walks down the tree entering the subcluster with the nearest center.

When adding a point at the leaves, a new cluster is created if the cluster radius R increases beyond the threshold T, otherwise the point is added to the nearest cluster. If the creation of a new cluster leads to more than Br child nodes of the parent, the parent is split. To ensure that the tree stays balanced, the nodes above might need to be split recursively. Once all points are submitted to BIRCH, the centers of the leaf clusters are, in the *global clustering* phase, entered into k-means with the cluster count k. This last step improves the cluster quality by merging neighboring clusters. In this paper, we will refer to the BIRCH algorithm as *full-BIRCH* and to BIRCH without its global clustering phase as *tree-BIRCH*.

Tree-BIRCH is very fast. It clusters 100,000 points into 1000 clusters in 4 seconds, on a 2,9 GHz Intel Core i7, using scikit-learn [12]. The k-means implementation of the same library needs over two minutes to complete the same task on the same architecture. Furthermore, tree-BIRCH doesn't require the cluster count as input, which in full-BIRCH is only needed for the global clustering phase. However, tree-BIRCH usually suffers from bad clustering quality. Therefore, this paper focuses on improving the clustering quality of tree-BIRCH.

4 Concept

In the following, we present a method that automatically chooses an optimal threshold parameter for tree-BIRCH. First, note that the CF-tree depends on the order in which the data is entered. If the points of a single cluster are entered in the order of increasing distance from the center, tree-BIRCH is more likely to return just one cluster than if the first two points are from opposite sides of the cluster. In the latter case, the single cluster is likely to split, a situation we will refer to as *cluster splitting* (Figure 1A). Next, consider two neighboring clusters. If the first two points are from opposite clusters but still near each other, they could be collected into the same cluster, given the threshold is large enough, which we refer to as *cluster combining* (Figure 1B). *Cluster combining* often co-occurs with *cluster splitting*. To reduce *splitting* of a single cluster, the threshold parameter of tree-BIRCH has to be increased, whereas a decreased threshold parameter reduces *cluster combining*. Datasets with a large ratio of cluster distance (the distance between the cluster cen-

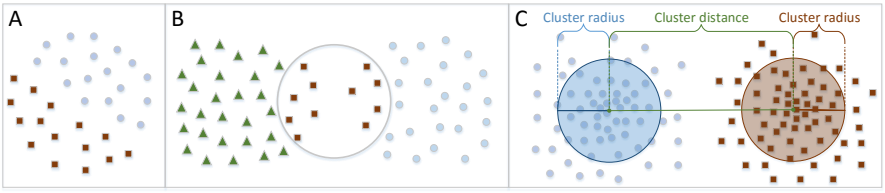

Fig. 1: *Cluster splitting* (A), *cluster combining* where the combining cluster is circled (B), depiction of cluster radius and cluster distance (C). Different forms and colors of the observations correspond to different clusters they belong to.

ters) to cluster radius are less likely to produce such errors (Figure 1C). In the next section, we derive a condition for the error probability to be less than one percent. Also, a formula for an appropriate threshold is provided. Note that there is, in fact, another source of splitting. This happens if a cluster overlaps with two regions belonging to two non-leaf nodes in the CF-tree. Therefore, we choose the branching factor Br to be larger than the highest possible cluster count.

5 Automatic Estimation of the BIRCH Threshold

In this paper it is presumed that the clusters are samples from two-dimensional isotropic Gaussian distributions of roughly the same variance. To find conditions for tree-BIRCH to work well, we first need to determine the common radius R of the clusters and the minimum cluster distance D_{min}. It is presumed that there exists a small but representative subset of the data that has the same cluster radius and minimum cluster distance D_{min} as the full dataset. On this small dataset, Gap Statistic is applied to obtain the cluster count k. This k in turn is given to k-means to produce a clustering of the subset, which finally yields the two values R and D_{min}.

For the determination of R and D_{min} one could also use any other clustering algorithm that finds the cluster centers and radii without requiring the cluster count k as an input. However, Gap Statistic is chosen here, due to its high precision. Our approach is heuristic. In each case, tree-BIRCH is run often enough to deduce estimates of *cluster splitting* and *combining* probabilities with sufficiently small 95% confidence intervals (with 10,000 repetitions of each tree-BIRCH clustering, the confidence interval of our error estimate at 0.01 is roughly 0.01 ± 0.002).

Avoiding Cluster Splitting

We create many clusters containing the same number of elements n by sampling from a single isotropic two dimensional Gaussian probability density function. The units are chosen such that the radius R of this cluster will be one. Then, tree-BIRCH is applied with the same threshold T to each of those datasets. After each iteration,

we determine whether tree-BIRCH returns the correct number of clusters, namely one. From this, we assess if the error probability estimate for tree-BIRCH is less than 0.01, or one percent. We also investigate the impact of varying the number of elements n in the cluster on the resulting error probability. Finally, we repeat all the above for several thresholds T. For this heuristic analysis, we use the python library scikit-learn and its implementation of BIRCH.

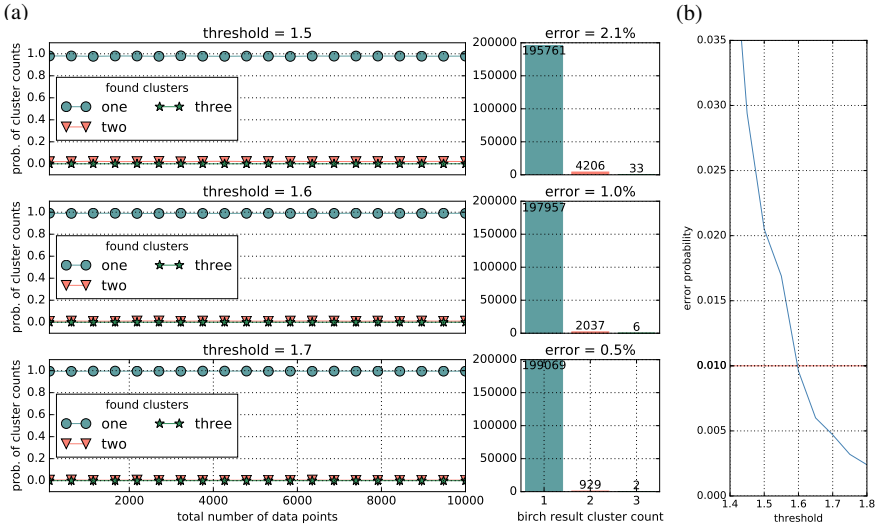

Fig. 2: (a) For each pair (n,T) of total number n of objects, running from 100 to 10,000, and threshold $T \in \{1.5, 1.6, 1.7\}$, we sampled n elements from a Gaussian of radius $R = 1$, and applied tree-BIRCH 10,000 times to compute the probabilities for each of the cluster counts 1, 2, and 3. Every count different from 1 is an error. (b) For each threshold we sample 500 points from a single Gaussian of radius $R = 1$, apply tree-BIRCH and record how often it returns the right number of clusters. This is repeated 10,000 times for each T to obtain an estimate for the error probability.

According to the results presented in Figure 2a, there is no indication that the number of objects in the cluster impacts the error probability. However the error probability is clearly affected by the threshold parameter; the error drops below one percent when the threshold value is greater than or equal to $1.6 \cdot R$.

Avoiding Cluster Combining

We use a mixture of two Gaussians with a cluster distance $D = 6$, both with radius $R = 1$. Again, the error probabilities are not dependent on the total number of data points, as shown in Figure 3a. This figure pertains only to the cluster distance 6.0. To understand the situation for different cluster distances, consider Figure 3b. Here, we see the dependence of the error probability on the threshold for several different cluster distances. For small thresholds ($T < 1.7$) we witness *cluster splitting* which

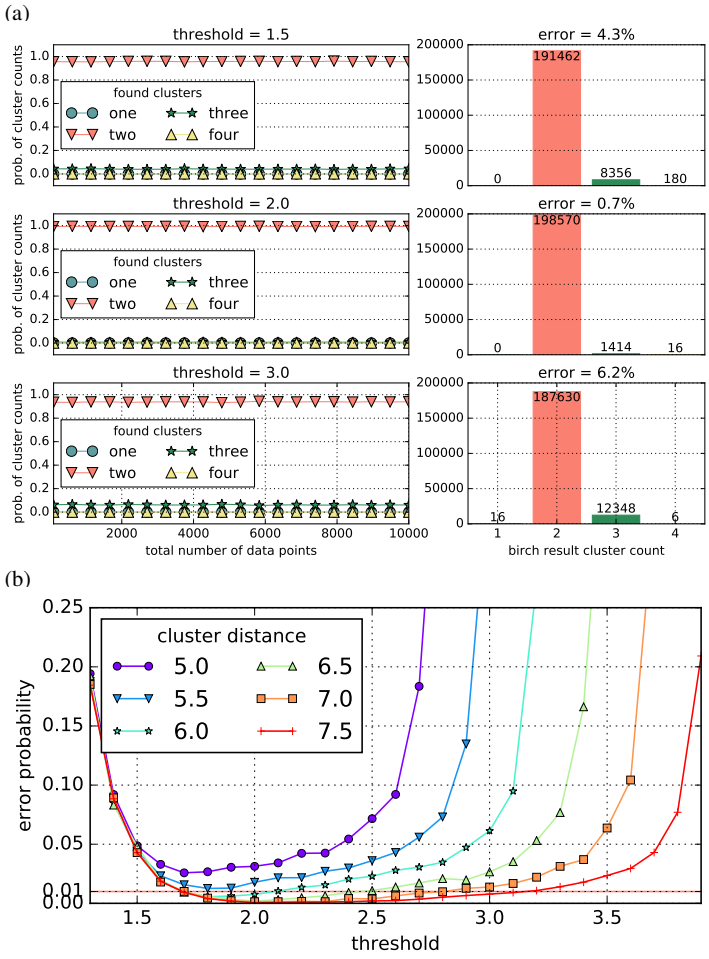

Fig. 3: (a) For each pair (n,T) of total number n of objects, running from 100 to 10,000, and threshold $T \in \{1.5, 2.0, 3.0\}$, we sampled $10,000$ times n elements from a mixture of two Gaussians of distance 6.0, and each time applied tree-BIRCH to compute the probabilities for each of the cluster counts 1, 2, 3, and 4. Every count different from 2 is an error. (b) For each pair (D,T) of cluster distance D and threshold T, we sampled $10,000$ times 500 elements from a mixture of two Gaussians of radius $R = 1$ and distance D. Each time we applied tree-BIRCH with the threshold set to T and computed the error probabilities.

results in higher cluster counts and a higher error. This can also be deduced from Figure 3a, where for the small threshold $T = 1.5$ we see many cluster counts of three and four. With $T = 2.0$ less splitting occurs and the error probability decreases. If the threshold continues growing ($T = 3.0$), *cluster combining* occurs more frequently, which increases the cluster counts of three and therefore increases the error probability. The fact that the graphs in Figure 3b are dropping below one percent later than in Figure 2b is due to *cluster combining*, that was not possible with just one cluster. For $D \geq 6.0$ there are thresholds where the error probability drops below one percent. The minimum is located near 1.9. From this observation it can be inferred, that, if $D_{min} \geq 6.0 \cdot R$, a choice of

$$T = 1.9 \cdot R \tag{1}$$

would ensure that for each pair of neighboring clusters in the dataset, the error probability would be less than one percent.

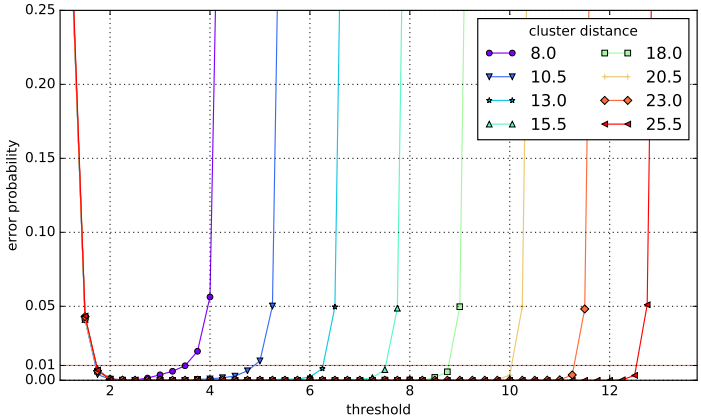

Fig. 4: For each pair (D,T) of cluster distance D and threshold T, we sampled $10,000$ times 500 elements from a mixture of two Gaussians of radius $R = 1$ and distance D. Each time we applied tree-BIRCH to compute the error probabilities.

Of course, if D_{min} is clearly larger than $6.0 \cdot R$, it would be beneficial to increase the threshold beyond (1). While the lower bound on the threshold is nearly the same for all cluster distances, the upper bound increases linearly, roughly with half the increase of the distance (Figure 4). This is intuitively clear since two times the threshold should fit comfortably between two clusters to avoid *cluster combining*. To place the threshold in the middle between lower and upper bound, we choose $\frac{1}{4}$ as the ratio of ΔD_{min} and ΔT. We then fit an intercept of 0.7, which yields the following expression for choosing the threshold (in arbitrary units), provided $D_{min} \geq 6.0 \cdot R$:

$$T = \frac{1}{4} D_{min} + 0.7 \cdot R. \tag{2}$$

A-BIRCH with parallel Gap Statistic

We want to build a fast clustering algorithm in order to enable scalability. While tree-BIRCH is very fast, Gap Statistic is not. Therefore, we developed a parallel version of Gap Statistic. Recall that Gap Statistic runs k-means for each cluster count $k \in \{1, \ldots, k_{max}\}$ not only on the dataset itself, but also on many Monte Carlo simulations (the R and MATLAB implementations choose 100 simulations as default value). Therefore, we parallelized the loop over the Monte Carlo reference simulations. The distribution of work and collection of the results are performed by Apache Spark. The proposed approach and the results from Section 5 are summarized in Algorithm 1.

Algorithm 1: A-BIRCH: Automatic threshold for the BIRCH algorithm

Data: N 2-dimensional data points $\{X_i\}$, k_{max}, number of Monte Carlo simulations B, distance metric D, branching factor Br
Result: CF-tree
begin
  $k^* \leftarrow$ parallel Gap Statistic(subsample($\{X_i\}$), k_{max}, B)
  labels $\leftarrow$ k-means(subsample($\{X_i\}$), k^*)
  compute the common radius R and the minimal D_{min} from the clustered data
  if $D_{min} < 6 \cdot R$ **then**
    Warning: the clusters are too close - BIRCH result might be inaccurate
  $T \leftarrow \frac{1}{4} D_{min} + 0.7 \cdot R$
  CF-tree $\leftarrow$ tree-BIRCH($\{X_i\}$, distance metric D, branching factor Br, T)

6 Evaluation

We evaluated the accuracy of A-BIRCH with the threshold estimation as stated in Equation (2). Figure 5 shows that A-BIRCH performs correctly with different sizes of D_{min} and different numbers of clusters. The evaluation datasets contain samples from two-dimensional isotropic Gaussian distributions with equal variance and a $D_{min} \geq 6.0 \cdot R$, which fulfills the previously defined requirement.

In an additional step, we evaluated the scalability of A-BIRCH. While tree-BIRCH is considered scalable with an increasing number of elements and clusters, Gap Statistic is the bottleneck as described in 5. We have tested the parallelized implementation of Gap Statistic on an Apache Spark cluster on Microsoft Azure. Two cluster configurations have been evaluated, each with two master nodes and with four and eight workers, respectively, each of which running on `Standard_D3` virtual machines. A `Standard_D3` virtual machine currently provides 4 CPU cores, 14GB of memory, running the Linux operating system. The parallelization has been implemented using the Spark Python API (PySpark). The computation of Gap Statistic was run on a dataset with 10 clusters, each consisting of 1000 two-dimensional data points. The computation times for varying numbers B of reference datasets and maximal number of clusters k_{max} are shown in Table 1.

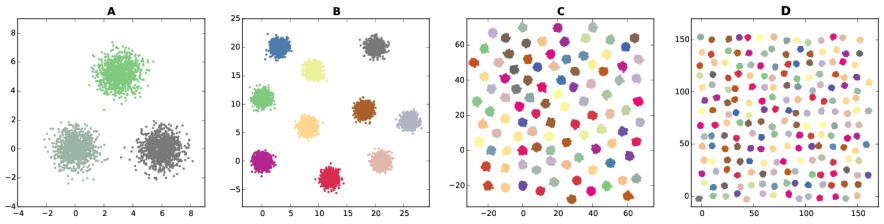

Fig. 5: The datasets A, B, C and D contain 3, 10, 100 and 200 clusters, respectively. Each cluster consists of 1000 elements, the radius of the clusters is $R = 1$, and the D_{min} is in all cases larger than 6: in A - 6.001, in B - 7.225, in C - 6.025, in D - 6.410.

	sequential	Spark: 4 workers	Spark: 8 workers
$B = 100, k_{max} = 20$	1775 s	349 s	197 s
$B = 100, k_{max} = 40$	7114 s	1425 s	795 s
$B = 500, k_{max} = 20$	8803 s	1470 s	725 s
$B = 500, k_{max} = 40$	35242 s	5953 s	2909 s

Table 1: Speedup of Gap Statistic by parallelization on Spark

The results show that the parallelized implementation of Gap Statistic with Spark is scalable as the computation times decrease linearly with an increasing number of worker nodes. Although the Gap Statistic phase is considered computationally expensive, it increases the correctness of BIRCH significantly and does not require any prior knowledge on the dataset.

7 Conclusion

In this paper we introduced A-BIRCH: a parameter-free variant of BIRCH. Choosing the correct parameters for clustering algorithms is often difficult as it requires information about the dataset, which is often not available. This is also true for BIRCH, which requires the cluster count k as well as a threshold T in order to compute the clusters correctly. For this reason, we removed the global clustering phase, thus rendering the cluster count parameter k unnecessary, and proposed a method that automatically estimates the threshold T, which is achieved using Gap Statistic to determine cluster properties. The evaluation proved the applicability of our approach in a very robust manner for two-dimensional isotropic Gaussian distributions with roughly the same variance, regardless of the number of clusters or it's elements. In future work, we plan to use other methods such as DBSCAN to either verify the found cluster properties or to decrease the computational complexity.

References

1. Akaike, H.: Information Theory and an Extension of the Maximum Likelihood Principle, pp. 199–213. Springer New York (1998)
2. Burbeck, K., Nadjm-Tehrani, S.: Adaptive real-time anomaly detection with incremental clustering. Information Security Technical Report **12**(1), pp. 56 – 67 (2007)
3. Dash, M., Liu, H., Xu, X.: '$1 + 1 > 2$': Merging distance and density based clustering. In: Database Systems for Advanced Applications, 2001. Proceedings. Seventh International Conference on, pp. 32–39 (2001)
4. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. Journal of the Royal Statistical Society. Series B (Methodological) **39**(1), pp. 1–38 (1977)
5. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: E. Simoudis, J. Han, U.M. Fayyad (eds.) Second International Conference on Knowledge Discovery and Data Mining, pp. 226–231. AAAI Press (1996)
6. Ismael, N., Alzaalan, M., Ashour, W.: Improved multi threshold birch clustering algorithm. International Journal of Artificial Intelligence and Applications for Smart Devices **2**(1), pp. 1–10 (2014)
7. Jordan, M.I., Bach, F.R., Bach, F.R.: Learning spectral clustering. In: Advances in Neural Information Processing Systems 16. MIT Press (2003)
8. Kumar, N.S.L.P., Satoor, S., Buck, I.: Fast parallel expectation maximization for gaussian mixture models on gpus using cuda. In: 11th IEEE International Conference on High Performance Computing and Communications, pp. 103–109 (2009)
9. Macqueen, J.B.: Some Methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Math, Statistics, and Probability, vol. 1, pp. 281–297. University of California Press (1967)
10. Meng, X., Bradley, J.K., Yavuz, B., Sparks, E.R., Venkataraman, S., Liu, D., Freeman, J., Tsai, D.B., Amde, M., Owen, S., Xin, D., Xin, R., Franklin, M.J., Zadeh, R., Zaharia, M., Talwalkar, A.: MLlib: machine learning in apache spark. CoRR (2015)
11. Owen, S., Anil, R., Dunning, T., Friedman, E.: Mahout in Action. Manning Publications Co. (2011)
12. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research **12**, pp. 2825–2830 (2011)
13. Schwarz, G.: Estimating the dimension of a model. The Annals of Statistics **6**(2), pp. 461–464 (1978)
14. Sugar, C., of Statistics, S.U.D.: Techniques for clustering and classification with applications to medical problems. Stanford University (1998)
15. Tibshirani, R., Walther, G., Hastie, T.: Estimating the number of clusters in a data set via the gap statistic. Journal of the Royal Statistical Society: Series B (Statistical Methodology) **63**(2), pp. 411–423 (2001)
16. Zechner, M., Granitzer, M.: Accelerating k-means on the graphics processor via cuda. In: First International Conference on Intensive Applications and Services, pp. 7–15 (2009)
17. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: a new data clustering algorithm and its applications. Data Mining and Knowledge Discovery **1**(2), pp. 141–182 (1997)
18. Zhou, B., Hansen, J.: Unsupervised audio stream segmentation and clustering via the bayesian information criterion. In: Proceedings of ISCLP-2000: International Conference of Spoken Language Processing, pp. 714–717 (2000)