Indexes to find the optimal number of clusters in a hierarchical clustering

José David Martín-Fernández, José María Luna-Romera, Beatriz Pontes, and José C. Riquelme-Santos[1]

University of Seville, Sevilla 41012, Spain,
jfernandez94@us.es,
WWW home page: http://grupo.us.es/minerva/

Abstract. Clustering analysis is one of the most commonly used techniques for uncovering patterns in data mining. Most clustering methods require establishing the number of clusters beforehand. However, due to the size of the data currently used, predicting that value is at a high computational cost task in most cases. In this article, we present a clustering technique that avoids this requirement, using hierarchical clustering. There are many examples of this procedure in the literature, most of them focusing on the dissociative or descending subtype, while in this article we cover the agglomerative or ascending subtype. Being more expensive in computational and temporal cost, it nevertheless allows us to obtain very valuable information, regarding elements membership to clusters and their groupings, that is to say, their dendrogram. Finally, several sets of data have been used, varying their dimensionality. For each of them, we provide the calculations of internal validation indexes to test the algorithm developed, studying which of them provides better results to obtain the best possible clustering.

Keywords: Machine Learning, Hierarchical Clustering, Internal Validation Indexes

1 Introduction

In recent years, the size of the information available for various types of studies has grown considerably. Areas like medicine [1], social networks [2], energy [3] or electronic consumption [4] are just a few examples of this, with an increasing amount of data. This information needs to be processed to get some useful knowledge.

Among the different possible solutions to data analysis we focus on Machine Learning techniques, allowing us to extract the main features and a model covering the main information in a dataset. One of the most used model is called clustering, which determines the number of instances of a certain grouping within the data under study. Within the existing grouping variants, hierarchical clustering provides us with very interesting additional information. We can see the evolution of the clusters in each step of the algorithm, thus studying the grouping of X elements within the data. There exists two subtypes within hierarchical

clustering: dissociative or descending, starting from a group with all the elements, and ending in a cluster for each instance in the dataset; or agglomerative or ascending, starting with as many clusters as exists elements in the dataset and ending with a single agglomerative cluster with all of them.

Nowadays, there exists several frameworks to work with Machine Learning techniques to obtain knowledge. One of the most known is Apache Hadoop [5], that is built around the programming model based on the Google paradigm MapReduce [6]. Moreover, one of the most widely used open source projects is Apache Spark [7]. In the Google paradigm, it is read and written from the hard disk on many occasions, which reduces produces a detriment in the speed of data processing. Spark, the number of write/read cycles on the disk, so that intermediate calculations are logically and quickly stored in RAM. To do this, Spark uses a data structure called *"Resilient Distributed Datasets"* (RDD), that are specially designed to parallelize cache calculations with high data volume. In addition, this system contains the scalable library for Machine Learning (*MLlib*), with a series of such as algorithms classification, regression, recommendation systems and clustering techniques, will be of great help to achieve our goal [8].

The purpose of this article is to present a new agglomerative clustering technique implemented in Apache Spark. We have tested our algorithm using diverse datasets, that were created by means of a random database generator. Furthermore, we have applied different internal clustering validation indexes (CVIs) [9] in order to test our clustering results and compare the CVIs performance between the agglomerative hierarchical clustering implemented (AHC) and that provided by Spark as a dissociative modality, Bisecting K-Means (BK-Means) [10].

The rest of the document is organized as follows. Section 2 reviews different types of clustering techniques, as well as the CVIs used during our experimentation. Section 3 describes the algorithm and its implementation. Section 4 presents the experiments carried out and, finally, Section 5 summarizes the main conclusions of this work.

2 Related Work

In this section are reviewed the main grouping methods, as well as the internal validation indexes that have been used in our experimentation.

2.1 Clustering methods

There are several types of clustering algorithms, which could be classified into the following categories depending on the method we use [11]:

- *Grouping by partitions*: Given a set of n elements, the partition method builds K groups, where each partition represents a cluster and $K \leq n$. It's based on the principle of distance between the individuals, so given an initial K, a first solution could be obtained. Then the process consists in iterating

over the dataset, moving objects between groups and trying to improve the previous solution.
- *Density-based methods*: In this approach it is possible to obtain a clustering whose groups are made up of high-density areas and separated from each other by low-density areas.
- *Grid-based methods*: It consist in dividing the elements into a finite cell space which is part of a grid structure. It is applied independently to the size of the data, and the difference is given by the number of cells in each dimension of the generated space.
- *Hierarchical clustering*: It groups data to form a set, or to separate some already existing sets to give origin to other two. Thus, the distance is minimized or the similarity between them is maximized. It is possible to choose different measures to quantify both distance and similarity in this type of grouping.

Within the family of hierarchical algorithms there are two versions or strategies that can be used:

- *Dissociative or descending*: It starts with a cluster that includes all the objects, from which successive divisions are made, forming smaller groups until as many groups as there are elements in the dataset are obtained.
- *Agglomerative or ascending*: It works the opposite way to the descending version. It starts with as many clusters as there are elements in our dataset. At each step, more and more clusters of instances are formed until you end up with a single cluster made up of all available data.

From the first group of algorithms, we can find numerous examples in the literature [12,13,14]. However, in this work we present a version of the agglomerative option. The main problem of this implementation lies in the computation time needed when treating with large amount of data. Hence, there are few examples in the literature of implementations of this type of strategies [15,16].

2.2 Validation indexes

The validation of the results obtained by clustering algorithms is a fundamental part of the clustering process. CVI have been typically used to evaluate the partition obtained. Most popular CVIs are *Dunn* [17] and *Silhouette* [18]. Furthermore, indexes presented in [9], have been used in this work, being some of them a simplification of *Dunn* and *Silhouette*.

Dunn and Silhouette. We describe in the following the two most used validation indexes in the literature, which have been implemented in this work for our experiments:

- *Dunn*: Measure widely applied in literature, but open to the possibility of choosing between several variants for calculation. The index is defined by:

$$Dunn = \frac{Min(Inter\text{-}cluster)}{Max(Intra\text{-}cluster)} \tag{1}$$

,where *Inter-cluster* is computed as the distance between all the points of a certain cluster M to all the points of a cluster N, and divided by the product of the number of elements in both clusters. *Intra-cluster* represents the distance between all the elements that are part of a cluster, divided by the number of instances within that set.

– *Silhouette*: Silhouette distance is calculated for each point i of a cluster:

$$Silhouette_i = \frac{(b_i - a_i)}{Max(b_i, a_i)} \tag{2}$$

,where b_i is the shortest distance between point i to the rest of points of any cluster in which i is not a part of; and a_i is the average distance between point i and the rest of the points of the clusters to which it belongs. Silhouette value is in the interval $-1 \leq Silhouette_i \leq 1$, being its optimal value equals to 1.

Other indexes. As aforementioned, we have used three other validation indexes from [9] in order to check and compare the goodness of our clustering algorithm. *Davis-Bouldin* is included [19], which uses data object quantities and features inherent to the dataset to set the compactness and separation of the clusters; *BD-Silhouette* is a simplification of the traditional *Silhouette* index based on using intra-cluster and inter-cluster distances to the centroid of each cluster, rather than every element within them; and *BD-Dunn*, which also simplifies the calculations of the internal validation index *Dunn* is used the centroid of each group of data.

3 Our proposal

In this section we present our approach for AHC. Our technique starts from as many clusters as instances and, in an ascending way, groups them until it reaches a single cluster. In the next, we show the pseudocode of our strategy is shown in Algorithm 1:

The algorithm receives as parameters: a RDD of objects of type *"Distance"* [20], (class created internally to represent the distance between any two elements of the database); the number of clusters to be obtained; the strategy for computing the distances; and the total number of instances in the dataset.

Line 10 refers to the the calculation of the Cartesian product, which is necessary to find the distances between the points or clusters of each iteration with respect to the other elements of the RDD of objects of type *"Distance"*. In addition, the step performed on line 11 is configurable according to the designated strategy for calculating the distance between elements in the database, being the implemented options *"min"*, *"max"* and *"avg"*. They refer to the minimum, maximum or mean distance between the distances of the remaining points from each of the points that make up the pair found during line 2 of the algorithm, respectively. In the literature, each of these strategies establishes a different hierarchical clustering typology. Being "minimum or simple link grouping" (*simple*

Algorithm 1 Agglomerative Hierarchical Clustering (AHC)

Input: RDD with objects *Distance*, number of clusters, strategy for the distance between elements and number of elements of the DataSet.

Output: Hierarchical clustering model.

1: **for** $a \leftarrow 0$ to $(elementsDataSetNumber - numClusters)$ **do**
2: Find the next cluster as the pair of elements with the shortest distance between them within the RDD of *Distance* (they can be two DataSet points; a DataSet object and a cluster; or two clusters).
3: Save the elements of the pair that make up the new cluster.
4: Update the hierarchical clustering model with the cluster found on line 2 and its elements.
5: **if** a < (elementsDataSetNumber - numClusters - 1) **then**
6: Delete from the RDD of *Distance* the match found on line 2.
7: Search the RDD for *Distance* for all the relationships between the first point or cluster found on line 2 and the rest of the elements.
8: Search the RDD for *Distance* for all the relationships between the second point or cluster found on line 2 and the rest of the elements.
9: Delete from the RDD of *Distance* all items found on lines 7 and 8.
10: Calculate the Cartesian product from the elements found in lines 7 and 8.
11: Add to the RDD of *Distance* the distances from the cluster found on line 2 to the other elements calculated on line 10.
12: **end if**
13: Every 5 iterations make a backup copy of the RDD of *Distance*.
14: **end for**

linkage), "maximum or complete link grouping" (*complete linkage*) and "average or average link grouping" (*average linkage*), respectively [21].

3.1 Implementation

For the creation of this hierarchical grouping, several variants can be made depending on the basis for storing the information, using *"Resilient Distributed Dataset"* (RDD) or *"DataFrames"*. Both objects are provided by Apache Spark. In addition, some functions from MLlib were used, which allows us to delegate some calculations of our algorithm.

Following Spark recommendations, *collect()* and *coalesce()* [22] methods were used in order to accelerate the process. With the *collect()* method, it is possible to obtain data stored in memory during previous calculations, so that it is not necessary to wait for the executions *lazy* of the Spark framework. Through the use of the *coalesce()* method, it is possible to reduce considerably the partitions in which the data are parallelized during the execution of our algorithm. Spark divides the stored data into four times the number of working nodes being used.

Spark offers other alternatives to solve this issue, such as using the *count()* method, which count the number of elements of a given RDD, thus forcing the system to use the data stored in memory. Finally, this alternative was replaced by the former one in our implementation due to the results in terms of execution time between both during the performance tests of the algorithm.

In addition, through experimentation, *checkpoints* each several iterations helps to achieve better performance of the algorithm. Therefore, it is necessary to remember the data that are within the RDD of the distances between all elements. After several performance tests, 5 iterations were inferred as the optimum number for these backups. As for the *coalesce()* method, tests were carried out with several multiples of 4, following Spark's recommendations that establish to use them by multiplying by the number of CPUs used during the execution. The best configuration was to use 8 partitions to distribute the data since the equipment where the tests have been performed has 4 CPUs (it is described in Section 4.1). Following Spark's recommendations, it is one of the configurations that usually work best.

4 Experimentation

In this section we present the experimental setup and results obtained using our AHC approach and the comparison with respect to the use of the dissociative clustering algorithm BK-Means.

4.1 Working environment and Datasets

Our goal is to check the goodness of the different grouping obtained by our algorithm, using several datasets and evaluating the results by means of multiple CVIs. The experiments were executed in: IntelliJ IDEA development environment; the Apache Spark framework using the Scala language; and the Machine Learning library provided by the MLlib framework; a computer with an Intel Core i7-7700HQ CPU with 4 cores of 2.8 GHz, 16 GB of RAM, an SSD of 256 GB and a HDD of 1TB.

A total of 60 datasets have been used in our experimentation, which were generated by using the database generator in [9]. This tool allowed us to configure the desired number of clusters, dimensions, and the number of points for each cluster.

For the experimentation, three different configurations for the number of clusters (K) have been used : 3, 5 and 7; 20 different configurations for the dimensionality of the data: from 1 to 20; and 100 points for each of cluster. In order to achieve the 20 different dimensions expressed above, a dataset has been taken as the basis for each different K with 20 total dimensions, from which the different dimensions from 1 to 20 have been selected.

4.2 Experimental results

In order to study which CVI offers the best results using our hierarchical algorithm as the basis for clustering, we must first define how to measure this goodness. In our experiments, the modality *avg* distance of the hierarchical clustering algorithm has been chosen, explained in Section 3. As for the configuration of the BK-Means algorithm, all the default values for parameters have been set.

Since the number of clusters (K) for each of the datasets is known beforehand, we would check in how many datasets each of the indexes matches K. After executing the algorithm for each of the datasets, different cluster numbers on the resulting model for each CVI has been tested. Specifically, K values from 3 to 9 have been used ($3 < K < 9$). This interval has been chosen in order to guarantee that all K values can be found, since, the minimum would be 3 and the maximum would be 7 in the data used during the experimentation.

The results have been grouped according to two criteria: by the number of clusters, and by the number of dimensions in each of the databases. Following the first of the criteria, each index could obtain a maximum of 20 hits in each of the numbers of clusters. Whereas for the second criterion, each index could obtain a maximum of 3 hits for each of the dimensions available in our databases. In this sense, each of the indexes will have two different hits: one grouping all the cluster numbers for each dimension, being able to obtain a maximum of 20; and another grouping all dimensions for each cluster number, being able to obtain a maximum of 3. The results can be summarized in the following Tables 1, 2 and 3:

Index	AHC (K3)	AHC (K5)	AHC (K7)	BK-Means (K3)	BK-Means (K5)	BK-Means (K7)	AHC (Total)	BK-Means (Total)
Silhouette	**20**	18	14	**20**	18	7	**52**	**45**
Dunn	19	18	14	17	18	7	51	42
Silhouette-BD	17	15	9	13	14	3	41	30
Dunn-BD	19	18	13	17	18	7	50	42
Davis-Bouldin	**20**	18	14	18	18	7	**52**	43

Table 1. Summary of hits of each index grouped by the number of clusters in each dataset.

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Total
Silhouette	1	1	2	2	2	2	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**14**
Dunn	1	1	1	2	2	2	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**14**
Silhouette-BD	0	0	1	2	1	2	**3**	2	**3**	2	**3**	**3**	2	2	2	2	2	**3**	**3**	**3**	7
Dunn-BD	0	1	2	2	2	2	**3**	**3**	2	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	13
Davis-Bouldin	1	1	2	2	2	2	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**14**

Table 2. Summary of hits of each index grouped by the number of dimension in each dataset in AHC execution.

Calculating the total success percentage of each of the CVIs studied would be as simple as dividing the "Total" columns of the previous tables by the maximum number of clusters and dimensions of those groupings, 60 and 20, respectively. Figure 1 summarizes the global results for each CVI, showing the percentage of total hits, and grouping the tests carried out by the number of clusters, and by the number of dimensions for each dataset.

The best indexes taking into account the grouping of databases by the number of clusters have been *Silhouette* and *Davis-Bouldin* in the case of using our

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Total
Silhouette	1	1	2	2	2	2	**3**	2	**3**	**3**	**3**	**3**	**3**	2	**3**	2	2	2	2	2	**7**
Dunn	0	0	1	2	2	2	**3**	2	**3**	**3**	**3**	**3**	**3**	2	**3**	2	2	2	2	2	**7**
Silhouette-BD	0	0	0	1	0	0	1	2	2	**3**	**3**	2	2	2	2	2	2	2	2	2	2
Dunn-BD	0	0	1	2	2	2	**3**	2	**3**	**3**	**3**	**3**	**3**	2	**3**	2	2	2	2	2	**7**
Davis-Bouldin	1	0	1	2	2	2	**3**	2	**3**	**3**	**3**	**3**	**3**	2	**3**	2	2	2	2	2	**7**

Table 3. Summary of hits of each index grouped by the number of dimension in each dataset in BK-Means execution.

AHC algorithm, both obtaining a success rate of 87%. Studying the case of the BK-Means algorithm it can be observed how the best index has been the only *Silhouette*, with a 75% success rate. If we study the percentages by the number of dimensions, we find the same previous winners plus *Dunn* in the case of using our AHC algorithm, all obtaining a 70% success rate on the number of dimensions. However, in the case of using the BK-Means algorithm, the *Dunn-BD* index would have to be added to the previous winners, all with a 35% success rate. A much smaller percentage than in the case of the AHC algorithm. So it can be concluded that the best indexes to validate our hierarchical clustering algorithm are both *Silhouette* and *Davis-Bouldin*, as they are the ones that are most accurate in all the conditions studied in the case of using our AHC algorithm and the only *Silhouette* in the case of the BK-Means algorithm.

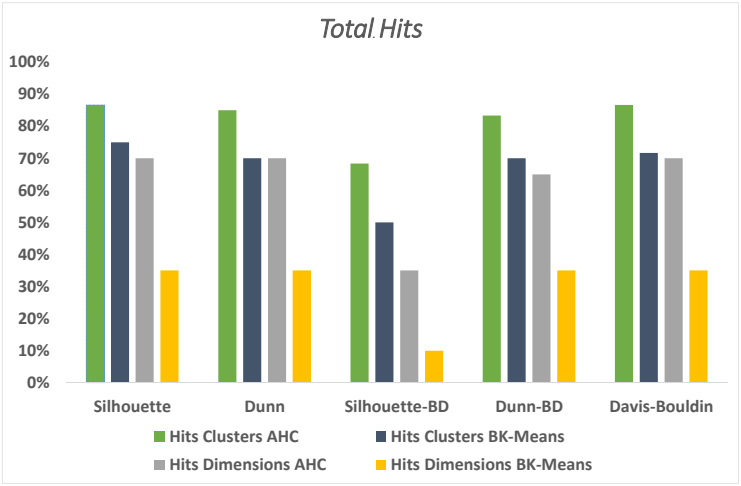

Fig. 1. Percentage of total hits for each of the indexes.

We have found that the greater the number of clusters, by the more difficult the finding the optimum number of clusters. Specifically, the worst results were obtained for $K = 7$. On the other hand, from the point of view of dimensions, all

indexes have had problems when the databases had a low number of dimensions. More specifically from 1 to 6 in the most cases, concluding that the greater the number of dimensions, the better all the indexes studied in this article behave with our hierarchical clustering algorithm.

With respect to computation time, the following Table compare the different algorithms used 4:

Hierarchical Type	Clustering (300p)	Clustering (500p)	Clustering (700p)
BK-Means	**1.95s**	**1.86s**	**1.68s**
AHC	43.12s	105.94s	156.85s

Table 4. Average computation time for each type of hierarchical clustering.

As it can be seen, the executions have been grouped according to the number of points of each dataset studied with 300, 500 and 700 points. In all the variants studied, the BK-Means algorithm has obtained better computation times than the proposed AHC algorithm, as expected.

5 Conclusions

This article presents a new approach for AHC, together with a performance comparison involving several clustering validation indexes. On the one hand, the effectiveness of the generated model by this clustering algorithm has been verified. On the other hand, we have established *Silhouette* and *Davis-Bouldin* as the best validation indexes for our algorithm. In addition, better results than the BK-Means algorithm developed by Apache Spark have been achieved in all indexes.

Moreover, all the indexes under study have problems in finding the optimum number of clusters when the number of clusters is high, and the dimensions of the points are reduced. Therefore, with the developed algorithm in environments may be used a high number of dimensions, since knowing a priori the optimal number of clusters in a database is not an easy task.

For future work, the main objetive is to improve the implementation of this hierarchical clustering algorithm; by increasing the number of instances that can be introduced, and also trying to reduce the computation time with respect to the dissociative version (BK-Means).

All the code generated, as well as the used databases during the study can be found at the following link: https://github.com/Joseda13/LinkageClustering.

References

1. H. M. Krumholz, “Big Data And New Knowledge In Medicine: The Thinking, Training, And Tools Needed For A Learning Health System,” *Health Affairs*, vol. 33, no. 7, pp. 1163–1170, 7 2014.

2. Z. Su and Q. X. et al., "Big data in mobile social networks: a qoe-oriented framework," *IEEE Network*, vol. 30, no. 1, pp. 52–57, January 2016.
3. R. Pérez-Chacón and J. M. L. et al., "Big data analytics for discovering electricity consumption patterns in smart cities," *Energies*, vol. 11, no. 3, 2018.
4. H. Guo and Z. L. et al., "Big Earth Data: a new challenge and opportunity for Digital Earths development," *International Journal of Digital Earth*, vol. 10, no. 1, pp. 1–12, 1 2017.
5. J. Dean and S. Ghemawat, "MapReduce," *Communications of the ACM*, vol. 51, no. 1, p. 107, 1 2008.
6. S. Ghemawat and H. G. et al., "The Google file system," in *Proceedings of the nineteenth ACM symposium on Operating systems principles - SOSP '03*, vol. 37, no. 5. New York, New York, USA: ACM Press, 2003, p. 29.
7. A. Spark, "Lightning-Fast C. C." [Online]. Available: https://spark.apache.org/
8. A. Spark, "Clustering Documentation," https://spark.apache.org/docs/2.2.0/ml-clustering.html, 2019.
9. J. M. Luna-Romera and J. G. et al., "An approach to validity indices for clustering techniques in Big Data," *Progress in Artificial Intelligence*, pp. 1–14, 10 2017.
10. A. Spark, "Clustering - Bisecting k-means." [Online]. Available: https://spark.apache.org/docs/2.2.0/mllib-clustering.html#bisecting-k-means
11. A. Fahad and N. A. et al., "A Survey of Clustering Algorithms for Big Data: Taxonomy and Empirical Analysis," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 3, pp. 267–279, 9 2014.
12. A. Sharma and Y. L. et al., "Divisive hierarchical maximum likelihood clustering," *BMC Bioinformatics*, vol. 18, no. S16, p. 546, 12 2017.
13. E. Kim and W. O. et al., "Divisive Hierarchical Clustering towards Identifying Clinically Significant Pre-Diabetes Subpopulations." *AMIA ... Annual Symposium proceedings. AMIA Symposium*, vol. 2014, pp. 1815–24, 2014.
14. A. K. Patnaik and P. K. B. et al., "Divisive Analysis (DIANA) of hierarchical clustering and GPS data for level of service criteria of urban streets," *Alexandria Engineering Journal*, vol. 55, no. 1, pp. 407–418, 3 2016.
15. Y. Loewenstein and E. Portugaly, "Efficient algorithms for accurate hierarchical clustering of huge datasets: tackling the entire protein space," *Bioinformatics*, vol. 24, no. 13, pp. i41–i49, 7 2008.
16. I. Uchiyama, "Hierarchical clustering algorithm for comprehensive orthologous-domain classification in multiple genomes," *Nucleic Acids Research*, vol. 34, no. 2, pp. 647–658, 1 2006.
17. J. C. Dunn, "Well-Separated Clusters and Optimal Fuzzy Partitions," *Journal of Cybernetics*, vol. 4, no. 1, pp. 95–104, 1 1974.
18. P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 11 1987.
19. D. L. Davies and D. W. Bouldin, "A Cluster Separation Measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 2, pp. 224–227, 4 1979.
20. J. D. Martín-Fernández and J. M. Luna-Romera, "Distance Class," 2018. [Online]. Available: https://github.com/Joseda13/linkage/blob/master/src/main/scala/es/us/linkage/Distance.scala
21. T. Hastie and R. T. et al., "Springer Series in Statistics The Elements of Statistical Learning The Elements of Statistical Learning."
22. A. Spark, "ScalaDoc - RDD." [Online]. Available: https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.rdd.RDD