CLUS: Parallel Subspace Clustering Algorithm on Spark

Bo Zhu(✉), Alexandru Mara, and Alberto Mozo

Universidad Politécnica de Madrid, Madrid, Spain
bozhumatias@ict-ontic.eu, alexandru.mara@alumnos.upm.es, a.mozo@upm.es

Abstract. Subspace clustering techniques were proposed to discover hidden clusters that only exist in certain subsets of the full feature spaces. However, the time complexity of such algorithms is at most exponential with respect to the dimensionality of the dataset. In addition, datasets are generally too large to fit in a single machine under the current big data scenarios. The extremely high computational complexity, which results in poor scalability with respect to both size and dimensionality of these datasets, give us strong motivations to propose a parallelized subspace clustering algorithm able to handle large high dimensional data. To the best of our knowledge, there are no other parallel subspace clustering algorithms that run on top of new generation big data distributed platforms such as MapReduce and Spark. In this paper we introduce CLUS: a novel parallel solution of subspace clustering based on SUBCLU algorithm. CLUS uses a new dynamic data partitioning method specifically designed to continuously optimize the varying size and content of required data for each iteration in order to fully take advantage of Spark's in-memory primitives. This method minimizes communication cost between nodes, maximizes their CPU usage, and balances the load among them. Consequently the execution time is significantly reduced. Finally, we conduct several experiments with a series of real and synthetic datasets to demonstrate the scalability, accuracy and the nearly linear speedup with respect to number of nodes of the implementation.

Keywords: Subspace · Parallel · Clustering · Spark · Big data

1 Introduction

Clustering is one of the main techniques for unsupervised knowledge discovering out of unlabeled datasets. This technique uses the notion of similarity to group data points in entities known as clusters. Traditional clustering algorithms such as partitioning based approaches (e.g. K-Means[1]), density based approaches (e.g. DBSCAN[2]) and hierarchical approaches (e.g. DIANA[3]), take the full

The research leading to these results has been developed within the ONTIC project,which has received funding from the European Union's Seventh Framework Programme (FP7/2007-2011) under grant agreement no. 619633.

T. Morzy et al. (Eds): ADBIS 2015, CCIS 539, pp. 175–185, 2015.
DOI: 10.1007/978-3-319-23201-0_20

feature space into consideration. However, as current datasets become larger and higher-dimensional, these algorithms fail to uncover meaningful clusters due to the existence of irrelevant features and the curse of dimensionality[4].

In many application domains, such as sensor networks, bioinformatics, and network traffic, objects are normally described by hundreds of features. Since data collection and storage become cheaper and more convenient, bigger datasets are generated without an analysis of relevance. Consequently, a number of techniques were intensively studied to address the clustering task for high dimensional datasets. Dimensionality reduction techniques like Principle Component Analysis[5] and feature selection techniques like mRMR feature selection algorithm[6], generate an optimal subset of features containing the most relevant information. Given the fact that clusters can be found in different subsets of features, such techniques fail to detect locally relevant features for each cluster[4].

A special family of algorithms, which derived from the frequent pattern mining field[7], rapidly constituted a novel field named subspace clustering. These algorithms aimed to find clusters hidden in subsets of the original feature space and therefore avoid the curse of dimensionality. Nevertheless, up to our knowledge none of these techniques scales well with respect to the size of datasets. They are generally under the assumption that the whole dataset can fit in a single machine. SUBCLU[8], an Apriori based subspaces clustering algorithm uses a traditional DBSCAN implementation to cluster the data in each subspace resulting in an inefficient and non-scalable solution. In this context, parallelization comes out as a natural solution to improve the efficiency and scalability of existing subspace clustering algorithms. However, during our research we have detected a lack of scalable and parallel subspace clustering approaches for high dimensional data on top of current Big Data distributed platforms. Recently, MapReduce[9] paradigm jointly with the Hadoop framework[10] have become the most popular Big Data distributed framework. Nevertheless, the limitations of MapReduce and its lack of suitability for iterative Machine Learning algorithms have motivated researchers to propose different alternatives, being Spark [11] one of the most representative. Therefore, designing scalable subspace clustering algorithms on top of Spark is an interesting challenge.

Contributions. In this paper, we present CLUS, a novel parallel algorithm on top of Spark based on SUBCLU that overcomes the latter's limitations using a dynamic data partitioning method. CLUS reduces the dimensionality and time complexity of SUBCLU by parallelizing the clustering tasks across different nodes. CLUS also eliminates the dataset size limitation of the centralized algorithm to the available RAM in one machine by distributing it across nodes and spilling it to disk when needed.

To develop CLUS, specific Spark primitives were used as they have the potential to provide higher performance if employed in an appropriate manner. Unlike MapReduce, Spark gives more flexibility allowing users to manage memory, disk and network usage in order to obtain more efficient algorithms. Moreover, instead of writing intermediate results to disk like MapReduce, Spark intends to main-

tain them in memory. Due to this in-memory computing Spark outperforms by an order of magnitude other Big Data platforms [24].

In summary, the main contributions of CLUS algorithm are:

1. A dynamic data partitioning method is carefully designed using Spark's specific operations in order to induce data locality. This optimization step reduces the cost of communications by avoiding unnecessary slow data shuffling, which is common in most MapReduce and Spark applications.
2. CLUS avoids replications of the whole dataset in each node in order to process it in parallel. Based on Spark's specific operations, CLUS can be deployed on a cluster of commodity machines and efficiently process huge datasets.
3. I/O cost is minimized by using indexing so as to access and move to specific nodes only the necessary data in each iteration. This results in faster and more efficient data management and better use of available RAM.
4. We report experimental results on real and synthetic datasets to show the scalability and accuracy of our algorithm. The results show a dramatic decrease in CLUS execution time w.r.t. centralized SUBCLU.

The reminder of this paper is organized as follows: Section 2 presents an overview of the related work. Section 3 gives a brief introduction to SUBCLU algorithm and presents the implementation of CLUS. In Section 4 we show experimental results on various datasets. Finally, Section 5 outlines our conclusions and future work.

2 Related Work

2.1 Subspace Clustering

Many research works have tried to address the subspace clustering task in the last two decades. There are some excellent surveys[4,12,13] that conducted either theoretical or experimental comparisons among different subspace clustering algorithms. These approaches were categorized into several classes considering different algorithmic aspects. In [12] a number of classical algorithms were divided into bottom-up and top-down groups based on the applied search method.

The bottom-up group first generates 1-D histogram information for each dimension, and then tries to perform a bottom-up traversal on the subspace lattice. Since the time complexity of naive traversal on the lattice is exponential w.r.t. the dimensionality of the dataset, most of these approaches conduct a pruning step for selecting candidates. This filtering procedure is based on the anti-monotonicity property.

Anti-Monotonicity Property: if no cluster exists in subspace S_k, then there is no cluster in any higher dimensional subspaces S_{k+1} either. i.e.

$$\exists S_k, C_{S_k} = \varnothing \Rightarrow \forall S_{k+1} \supset S_k, C_{S_{k+1}} = \varnothing.$$

By starting from one dimension and adding another dimension each time, bottom-up approaches tend to work efficiently in relatively small subspaces. Consequently, they generally show better scalability when uncovering hidden subspace clusters in lower dimensions. However, the performance decreases dramatically with the size of candidate subspaces containing clusters[12]. Examples of bottom-up approaches are CLIQUE [14], ENCLUS[15] etc.

In contrast with bottom-up approaches, top-down methods start from the equally-weighted full feature space and generate an approximation of the set of clusters. After the initialization, updates of the weight for each dimension in each cluster and the regeneration of clusters are iteratively conducted. Finally, a refinement of the clustering result is carried out to achieve a better quality of clusters. Since multiple iterations of clustering process are conducted in the full feature space, sampling techniques are generally used to increase efficiency by reducing the accuracy of the results. Clusters generated by this kind of approaches are non-overlapping and of similar dimensionality due to the mandatory input parameters. Algorithms such as PROCLUS[16] and ORCLUS[17] are typical examples of top-down approaches.

In some recent summary research works[4,13] subspace clustering algorithms are classified into three paradigms with regards to the underlying cluster definition and parametrization. Grid-based approaches, e.g. SCHISM[18], MaxnCluster[19], try to find sets of grid bins which contain more data than a density threshold for different subspaces. Density-based approaches, e.g. SUBCLU[8], INSCY[20], search for dense regions separated by sparse regions by calculating the distance of relevant dimensions. Clustering-oriented approaches, e.g. STATPC[21], Gamer[22], assume global properties of the whole cluster set similar to those of top-down approaches[12]. As Emmanuel Müller summarized in [4], "Depending on the underlying clustering model, algorithms always have to tackle the trade-off between quality and runtime. Typically, high quality results have to be paid with high runtime." Our effort in this paper is to increase efficiency by means of parallelization while maintaining high quality clustering results.

2.2 Parallel Subspace Clustering

During a thorough survey of the state-of-the-art, we detected a lack of parallel subspace clustering algorithms despite of the potential performance improvements that could be obtained from parallelization. Up to our knowledge, only two parallel implementations have been proposed by now. They used specific architectures like Message Passing Interface (MPI) and Parallel Random Access Machine (PRAM). Compared with the novel Spark framework, these models suffer from a non-negligibly high communication cost and a complex failure recovery mechanism. Furthermore, they neither scale as well as Spark nor provide as much flexibility to programmers.

In [23] the grid based parallel MAFIA algorithm was proposed as an extension of CLIQUE. MAFIA partitions each dimension into several discrete adaptive bins with distinct densities. In order to run the algorithm in parallel, the original

dataset was randomly partitioned into several parts and read into different nodes. Data parallelization based on a shared-nothing architecture, which assembles a naive version of MapReduce, can bring a significant reduction in execution times. However, the generated partitions can be highly skewed and greatly affect the quality of the clustering results.

The other parallel algorithm was based on the Locally Adaptive Clustering (LAC) algorithm. LAC was proposed in [25] as a top-down approach by assigning values to a weight vector based on the relevance of dimensions within the corresponding subspace cluster. Parallel LAC proposed in [26], transforms subspace clustering task to the problem of finding K centroids of clusters in an N-dimensional space. Given K x N processors that share a global memory, PLAC managed to distribute the whole dataset into a grid of K centroids and N nodes for each centroid. In the experiments, they used one machine as the global shared memory, assuming that the whole dataset could fit in a single node. This architectural design severely limited the scalability of PLAC, as the maximum size of the dataset shown in [26] was no larger than 10,000 points.

3 CLUS: Parallel Subspace Clustering Algorithm on Spark

3.1 SUBCLU Algorithm

SUBCLU follows a bottom-up, greedy strategy intending to detect all density-based clusters in all subspaces by using DBSCAN algorithm. First, a clustering process is performed over each dimension to generate the set of 1-Dim subspaces containing clusters. Then, SUBCLU recursively generates the set of candidates for the (K+1)-Dim by combining pairs of K-Dim candidates with clusters sharing K-1 dimensions. The Anti Monotonicity property is used to prune irrelevant candidates. To increase the efficiency of the subsequent clustering process, the K-Dim subspace with minimum amount of clustered data is chosen as the "best subspace" for running DBSCAN. The recurrence terminates when no more clusters are detected. The algorithm takes the same two parameters: epsilon and minpts as DBSCAN. A more detailed description of SUBCLU, the process of generating candidates as well as experimental results can be found in [8].

3.2 CLUS Algorithm

CLUS is a parallel algorithm on top of the Spark that overcomes the severe limitations of SUBCLU. While SUBCLU sequentially runs DBSCAN clustering for each subspace and has to wait for the termination of all the instances before generating the candidate set for next dimensionality, CLUS is able to execute multiple DBSCANs in parallel. The candidate generation and pruning steps are also performed in a distributed manner without iterating over the dataset. CLUS takes advantage of specific Spark primitives such as ReduceByKey, AggregateByKey etc. to induce data locality and improve the overall performance.

It should be noted that Spark is able to autonomously place the data in the nodes that require it by means of shuffle and repartition operations. However, these primitives imply data movements resulting in slow disk and RAM I/O operations and even slower network I/O operations. Moreover, Spark's attempts to assure data locality using the previously mentioned autonomous mechanisms might result in inefficiencies as they could further trigger other unnecessary shuffles. These dispensable time consuming operations are a very common problem in most MapReduce based platforms and should be avoided as much as possible. The data management strategy to achieve this has to be carefully considered in order to provide the precise partitions at any moment that will achieve load balance in each node and boost the performance.

The main challenges of CLUS are to assure that: 1) each node has the data required to run subspace clustering; 2) partial data replication might be necessary to assure simultaneous subspace clustering; 3) the minimum number of shuffles and repartitions are performed; 4) nodes are well balanced and only specific data is moved across them; 5) there are no idle nodes; 4) the information is always stored and accessed using efficient (key,value) structures.

```
Input: data matrix M ∈ℝ^{m×n}

1.  Compute CRV RDD({Column, Row, Value}) aplying transformations to M
2.  RDD DBS1 := Assign one feature per node, compute DBSCAN and remove features with no clusters
3.  maxSubs := DBS1.count()
4.  if maxSubs > 1
5.      currentSubs := 0
6.      while (currentSubs < maxSubs-1) and (not stop)
7.          currentSubs := currentSubs + 1
9.          candidates := Generate (K+1)-D candidates
8.          if currentSubs != 1 and candToPrune != 0
13.             candidates := candidates.substractByKey(candToPrune)
14.         numCandidates := candidates.count()
15.         if numCandidates != 0
16.             Get real data => candidates.flatMap().leftOuterJoin(CRV).rreduceByKey(CandidatePerNode)
17.             Compute DBSCAN for each candidate
18.             candToPrune := candidates with no cluster
19.             if currentSubs < 2
20.                 stop := True
21.             else
22.                 stop := True
```

Fig. 1. Pseudocode of CLUS algorithm

CLUS pseudocode is shown in Figure 1. The algorithm follows a similar concept to SUBCLU, but designed to reduce data reallocation and improve the efficiency of parallel density-based clustering tasks. CLUS manages the input information by columns/features so that different columns are stored in different machines as (key,value) pairs. The algorithm starts by running parallel DBSCANs on each of these features. The set of dimensions with clustered points are further used to generate (K+1)-Dim candidates. These candidates are generated by adding to each K-Dim subspace a disjoint dimension containing at least one cluster. The resulting (K+1)-Dim subspaces might require the same column to be present in different machines (e.g. to compute the subspaces f1-f2 and f1-f3 at the same time), so an efficient flatMap-leftOuterJoin-reduceByKey schema is

used for data replication. This mechanism allows CLUS to simultaneously execute independent DBSCANs on each subspace. The density based clustering is performed again on each of these higher dimensional subspaces, but using only the points that were not marked as noise in lower subspaces. A pruning step is conducted over the resulting set of (K+1)-Dim subspaces in order to remove the overlapping candidates (e.g. subspace f1-f2 plus subspace f3 generates the same candidate as f2-f3 plus f1). In addition, based on the Anti-Monotonicity property all the (K+1)-Dim subspaces containing a K-Dim subspace without any cluster are also removed. To this end, the subspaces eliminated because of their lack of clusters are used to generate (K+1)-Dim subspaces. These subspaces to be removed are substracted from the pool of valid (K+1)-Dim subspaces of the next iteration. CLUS ends when no higher subspace can be generated.

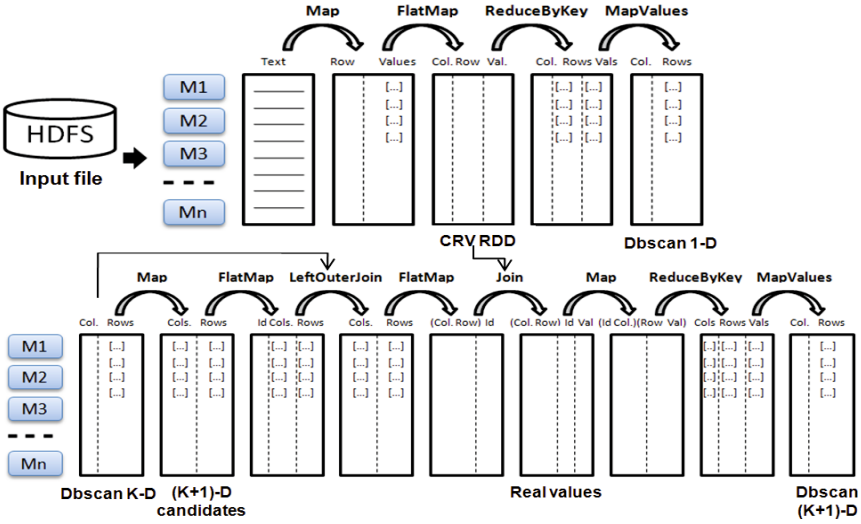

Fig. 2. Workflow of CLUS

The data workflow of CLUS is illustrated in Figure 2. For the sake of clarity some operations have been omitted, thus showing only the most important steps. The top part of the figure shows how clustering is performed on each dimension and how the work is distributed across Spark nodes. As mentioned, data is obtained form a text file stored in the HDFS and reorganized by columns and rows using a (key,value) associative data structure. Each real value has a column and row index and they constitute an RDD named CRV. This CRV RDD is stored across nodes so that each of them has one column. DBSCAN is performed in a distributed fashion and a new RDD with the indexes of the clustered points is obtained. This RDD is further used to generate the (K+1)-Dim candidates. The bottom part of Figure 2 shows the process of generating the set of candidates from prior K-Dim subspaces. By managing only the indexes of clustered points memory and network usage are optimized. The real data values are only accessed when DBSCAN needs to be performed trough a join operation

with the CRV RDD. The special structures of CRV and the index RDDs allows each node to access the precise information it requires for each subspace analysis and assures concurrent access to these data. The keys of the index RDD constitute the set of subspaces with clusters and therefore are used to generate new candidates. For each subspaces dimensionality a single data shuffle and repartition is strictly required. The results generated in each iteration of the algorithm are stored in a specific RDD through a union operation and wrote to HDFS at the and of the execution. This provides a clear and easy way to manage outputs.

4 Preliminary Experiments

In order to evaluate the performance and accuracy of CLUS a series of experiments have been conducted. To this end, both synthetic and real datasets have been considered. The synthetic datasets used, containing 5 and 10 relevant features and which were originally presented in a previous research work [27], were obtained from the webpage of the Ludwig Maximilian University of Munich. The real dataset tested contained aggregated network traffic flows obtained from the core network of an ISP at the ONTIC project. The experiments reported here were executed on a single Core(TM) i-7 and 16GB of RAM machine and a commodity cluster of 10 nodes. The cluster consists of Core(TM) 2 Quad CPUs with 4GB of RAM. Spark was configured to use one executor per core (8 in the single machine and 40 in the cluster) with 1GB of memory each. The machines were running CentOS operating system, Spark version 1.2.2 and Hadoop 2.0.

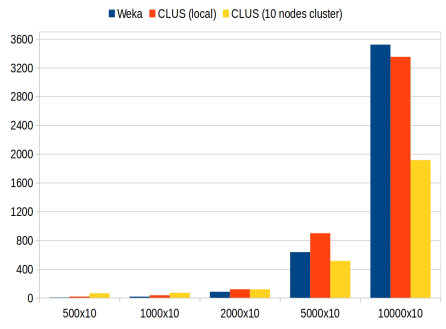

Fig. 3. SUBCLU and CLUS execution time w.r.t. number of nodes

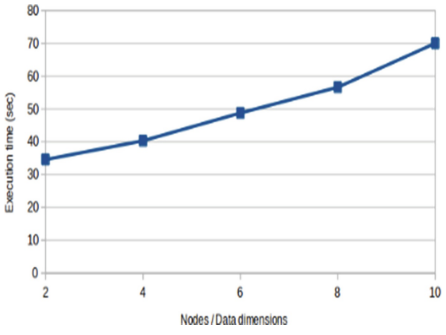

Fig. 4. Execution time with one node per dimension

4.1 Scalability

First we have evaluated the scalability of the algorithm comparing it to a centralized version of SUBCLU available in the OpenSubspace v3.31 project [4]. OpenSubspace extends with subspace clustering algorithms the well known Weka ML library. Figure 3 shows the execution times of SUBCLU and CLUS on a single machine and the cluster for different dataset sizes.

During the initial tests, Weka, without any additional requirement for cluster evaluation or visualization ran out of memory with 1GB of Java heap. CLUS running on a single node and with the same dataset used less than 3,5 MB. The total amount of heap for Weka was further increased to 8GB in order to be able to run all the experiments.

Additional experiments show, as expected, that CLUS's execution time grows with a quadratic factor w.r.t. dataset size and number of features on a constant number of cluster nodes (Figure 5 and Figure 6). However, as the number of machines in the cluster increases, CLUS achieves a nearly linear speedup by taking full advantage of the data partitioning strategy and simultaneous clustering executions. With a number of machines equal to the dimensions of the dataset a nearly linear speedup is achieved, as shown in Figure 4.

We have also tested CLUS with real network traffic data of up to 10000 flows and 10 features obtaining interesting insights. The algorithm discovered clusters in up to 8 dimensions and others in lower ones aggregating flows of the same type. The results were consistent in time and dimensionality with the ones using synthetic data.

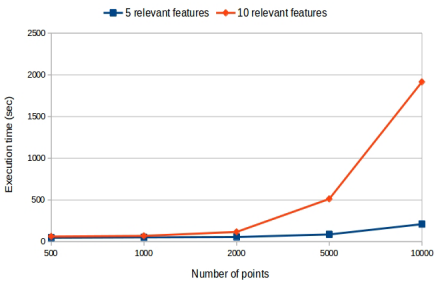

Fig. 5. Execution time w.r.t. dataset size.

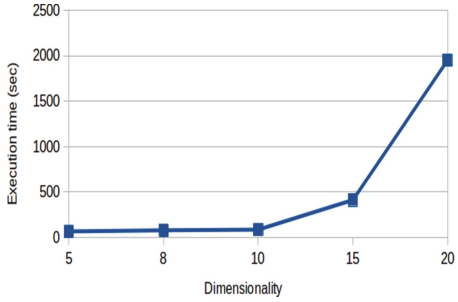

Fig. 6. Execution time w.r.t. dimensionality

4.2 Accuracy

In order to evaluate the accuracy of CLUS the algorithm was compared to the Weka implementation of SUBCLU providing the same clusters. Nevertheless, Weka implementation misclassified some points, so a Java version of the algorithm was implemented from the original paper [8]. Results showed a perfect match between this version of SUBCLU and CLUS on different datasets.

5 Conclusions and Future Work

In this paper we present CLUS: a novel scalable and parallel subspace clustering algorithm on top of Spark. CLUS is highly inspired by the well-known SUBCLU algorithm. Relying on specific Spark primitives, CLUS is able to execute multiple DBSCAN tasks in parallel achieving a significant speedup. In addition,

unnecessary data shuffle is avoided by using a carefully designed data partitioning strategy to induce data locality. We carried out some preliminary tests on a modest cluster showing promising results with respect to scalability. The obtained results show that CLUS efficiently finds correct subspace clusters. In the future, we plan to extend our experiments with respect to scalability both on the number of dimensions and the size of datasets.

References

1. Forgy, E.W.: Cluster analysis of multivariate data: efficiency versus interpretability of classifications. Biometrics **21**, 768–769 (1965)
2. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise, pp. 226–231. AAAI Press (1996)
3. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data. John Wiley & Sons (1990)
4. Müller, E., Günnemann, S., Assent, I., Seidl, T.: Evaluating clustering in subspace projections of high dimensional data. In: Proc. VLDB, vol. 2(1) (2009)
5. Pearson, K.: On Lines and Planes of Closest Fit to Systems of Points in Space. Philosophical Magazine **2**(11), 559–572 (1901)
6. Peng, H., Long, F., Ding, C.: Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. Pattern Analysis and Machine Intelligence **27**(8), 1226–1238 (2005)
7. Zimek, A., Assent, I., Vreeken, J.: Frequent pattern mining algorithms for data clustering. In: Frequent Pattering Mining, chapter 16, pp. 403–423. Springer International Publishing (2014)
8. Kailing, K., Kriegel, H.P., Kröger, P.: Density-connected subspace clustering for high-dimensional data. In: Proc. SIAM, pp. 246–257 (2004)
9. Dean, J., Ghemawat, S.: MapReduce: simplified data In Proc. on large clusters. Communications of the ACM **51**(1), 107–113 (2008)
10. Shvachko, K., et al.: The hadoop distributed file system. In: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). IEEE (2010)
11. Zaharia, M., et al.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proc. USENIX (2012)
12. Parsons, L., Haque, E., Liu, H.: Subspace clustering for high dimensional data: a review. ACM SIGKDD Explorations Newsletter **6**(1), 90–105 (2004)
13. Sim, K., Gopalkrishnan, V., Zimek, A., Cong, G.: A survey on enhanced subspace clustering. Data Mining and Knowledge Discovery **26**(2) (2013)
14. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic subspace clustering of high dimensional data for data mining applications. In: Proc. ACM SIGMOD, pp. 94–105 (1998)
15. Cheng, C., Fu, A., Zhang, Y.: Entropy-based subspace clustering for mining numerical data. In: Proc. SIGKDD, pp. 84–93 (1999)
16. Aggarwal, C.C., Wolf, J.L., Yu, P.S., Procopiuc, C., Park, J.S.: Fast algorithms for projected clustering. In: Proc. ACM SIGMOD, pp. 61–72 (1999)
17. Aggarwal, C.C., Yu, P.S.: Finding generalized projected clusters in high dimensional spaces. In: Proc. ACM SIGMOD, pp. 70–81 (2000)
18. Sequeira, K., Zaki, M.: SCHISM: a new approach for interesting subspace mining. In: Proc. ICDM, pp. 186–193 (2004)

19. Liu, G., Sim, K., Li, J., Wong, L.: Efficient mining of distance-based subspace clusters. Statistical Analysis and Data Mining **2**(5–6), 427–444 (2010)
20. Assent, I., Krieger, R., Müller, E., Seidl, T.: INSCY: indexing subspace clusters with in-process-removal of redundancy. In: Proc. ICDM, pp. 719–724 (2008)
21. Moise, G., Sander, J.: Finding non-redundant, statistically significant regions in high dimensional data: a novel approach to projected and subspace clustering. In: Proc. SIGKDD, pp. 533–541 (2008)
22. Gunnemann, S., Farber, I., Boden, B., Seidl, T.: Subspace clustering meets dense subgraph mining: a synthesis of two paradigms. In: Proc. ICDM (2010)
23. Goil, S., Nagesh, H., Choudhary, A.: MAFIA: efficient and scalable subspace clustering for very large data sets. In: Proc. SIGKDD (1999)
24. Spark. https://spark.apache.org/
25. Domenoconi, C., Papadopoulos, D., Gunopulos, D., Ma, S.: Subspace clustering of high dimensional data. In: Proc. SIAM (2004)
26. Nazerzadeh, H., Ghodsi, M., Sadjadian, S.: Parallel subspace clustering. In: Proc. the 10th Annual Conference of Computer Society of Iran (2005)
27. Achtert, E., Kriegel, H.-P., Zimek, A.: ELKI: a software system for evaluation of subspace clustering algorithms. In: Ludäscher, B., Mamoulis, N. (eds.) SSDBM 2008. LNCS, vol. 5069, pp. 580–585. Springer, Heidelberg (2008)