

ASSIGNMENTS

Density-Connected Subspace Clustering for High-Dimensional Data

Aditya Rajesh Patil
dataHacks
aditya18@iitg.ac.in
180101004

Kartikeya Saxena
dataHacks
kartikey18a@iitg.ac.in
180101034

Bhasker Goel
dataHacks
bhasker18@iitg.ac.in
180101015

V Anirudh
dataHacks
anirudh18@iitg.ac.in
180101084

1 REVIEW OF THE ALGORITHM

1.1 What bottleneck this clustering algorithm attacks?

The algorithm SUBCLU attacks the following bottlenecks -

- (1) Most real life datasets are characterized by high-dimensional and sparse dataspace which in turn make it very difficult to find clusters in the original data space. (The Curse Of Dimensionality)

The Algorithm attacks this by finding interesting clusters in subspaces which would have been hidden/impossible to detect in the original data space.

- (2) Earlier clustering algorithms were primarily based on Grid-based approaches (E.g CLIQUE) which made it difficult for them to detect clusters that were shaped or positioned arbitrarily on the grid.

By not relying on any Grid-based approach and using DBSCAN, SUBCLU is perfectly capable of detecting arbitrary shaped and positioned clusters in the subspace.

- (3) Subspace clustering algorithms often check for clusters in all Subspaces (the number of which is exponential). SUBCLU however,

- i) Using Monotonicity of Density-Connectivity and
- ii) Pruning irrelevant subspaces avoids doing this.

- (4) Algorithms (E.g - Dimensionality Reduction or Projected Clustering) more or less, find one final clustering of the Data. But it is perfectly possible that different clustering is possible based on different attributes of data. SUBCLU allows objects to be clustered varying in different subspaces.

- (5) Dimensionality Reduction Clustering Algorithms create new features from existing features to force-eradicate the Curse of Dimensionality. But in doing this, the features lose their intuitive meaning and the clustering becomes very hard to

interpret. SUBCLU, even the considering subspaces, keeps the attributes the same as the original dataset.

1.2 What is the overall idea of the algorithm? Do include a flowchart or a figure that explain the whole algorithm.

The algorithm is designed to detect the density- connected sets in all subspaces of high dimensional data in a bottom up, greedy fashion. First, the clusters present in 1-dimension are identified applying DBSCAN as a subroutine to each 1-dimensional subspace. Then k-dimensional clusters are used to generate k+1-dimensional clusters by using DBSCAN recursively and the parameters for DBSCAN are predetermined. The algorithm is built on two properties of clusters.

- (1) If two points are not in a cluster in a particular dimension, then they will not be in the same cluster in any higher dimension (Monotonicity of Density-Connected Sets).
- (2) A cluster in a lower dimension might not be a cluster in higher dimension. It might lose a few points from the cluster of lower dimension, but never gain other points.

From the above two properties of clusters the following idea can be concluded:

Pruning Irrelevant Subspaces: Given two subspaces S and T, such that T is a subset of S. If T has no cluster present in it, then S will also have no clusters and computation on S can be ignored.

A naive implementation of SUBCLU would result in running DBSCAN 2^d times, that is in all possible subspaces of a d dimension space. But using Pruning of Irrelevant Subspaces reduces the computational overhead of the algorithm. This leads to an efficient implementation of SUBCLU.

SUBCLU uses inverted files data structure to process range queries encountered in the algorithm.

Flowcharts explaining the whole algorithm is shown in Figure 1 and Figure 2.

1.3 How is the evaluation of the algorithm carried out?

The performance evaluation of the algorithm was carried out with (minPts = 8, epsilon = 2 as DBSCAN parameters) using synthetic data sets and real world Gene Expression Data of CDC15 mutant [SSZ+98]. Synthetic data sets of varying size and structure were

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference'17, July 2017, Washington, DC, USA

© 2021 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

<https://doi.org/10.1145/nnnnnnnn.nnnnnnn>

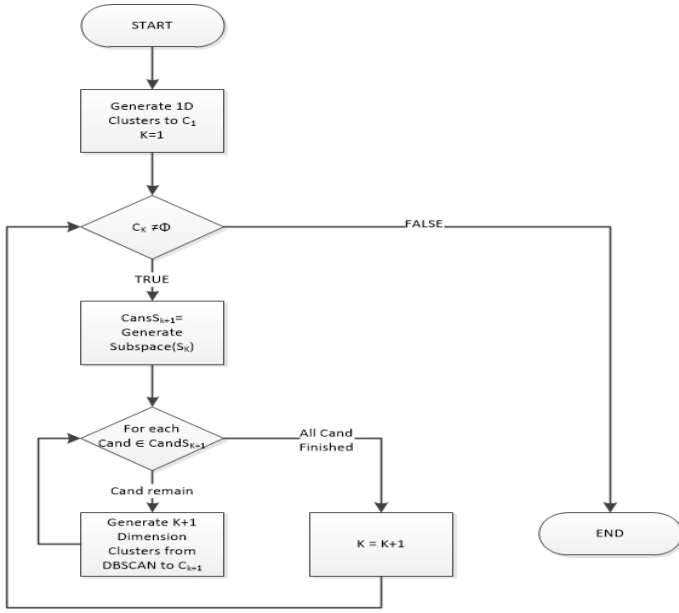


Figure 1: SUBCLU Algorithm

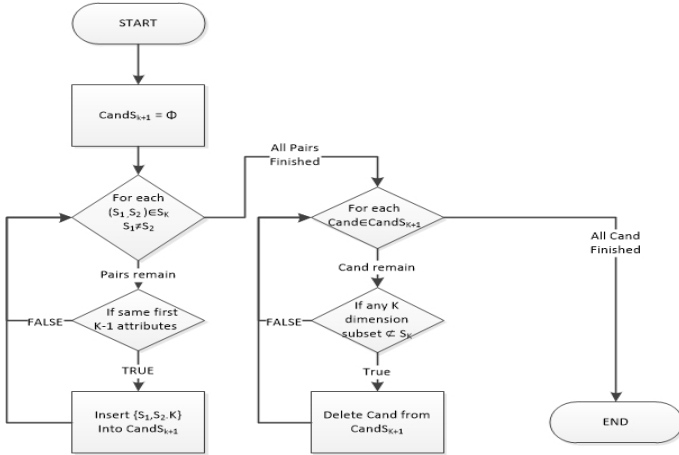


Figure 2: GenerateSubspace Subroutine

created with varying dimensionality of subspace clusters and feature space. It was found that the runtime SUBCLU grows with at least a quadratic factor against data set size, dimensionality of the data set and dimensionality of subspace clusters.

For Accuracy evaluation, the results on the synthetic datasets were compared to an efficient implementation of CLIQUE (CLustering In QUest) Algorithm and it was shown that SUBCLU outperforms the latter.

1.4 Are able to find out the code and datasets for the experiments in the paper? If not, have you contacted the authors?

The Paper provides the Pseudocode for what the authors did, but not the actual code. One of the Datasets used was real world Gene Expression Data [SSZ+98][1] which we were able to find.

The others they synthetically generated to match the application at hand. Although they provided a vague description of how they generated the data, we decided to contact the authors for the Original datasets they generated and used. We have also requested them for the code they prepared.

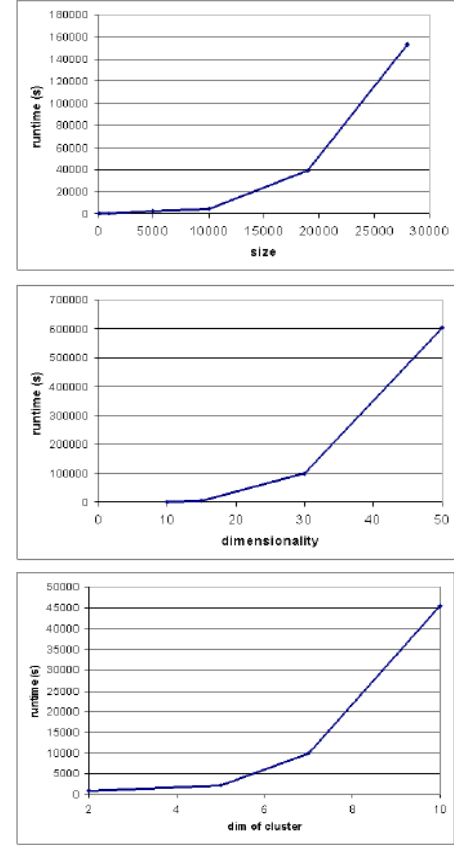


Figure 3: Performance Evaluation

Data set	d	dim. of subspace cluster	N	# generated clusters	true clusters found by	
					SUBCLU	CLIQUE
DS01	10	4	18999	1	1	1
DS02	10	4	27704	1	1	1
DS03	15	5,5	3802	3	3	1
DS04	15	3,5,7	4325	3	2	1
DS05	15	5,5,5	4057	3	3	1
DS06	15	4,4,6,7,10	2671	6	5	2

Figure 4: Accuracy Evaluation

1.5 Could you find any real-world applications of this algorithm? Does any ML/Data analysis package include this algorithm?

Subspace Clustering could potentially be used in the following real-world scenarios:

- (1) **Information Integration Systems:** These systems are motivated by the fact that future needs will be satisfied by autonomous, heterogeneous distributed information sources accessible through the internet. Such a system often maintains coverage statistics for each source based on a log of

previous user queries. With such information, it can rank the sources for a given query for faster retrieval. The subspace clustering algorithm can be applied to the query list with the queries being instances and the sources corresponding to dimensions of the dataset. The result is a rapid grouping of queries where a group represents queries coming from the same set of sources. Conceptually, each group can be considered a query class where the classes are generated in a one-step process using subspace clustering.

- (2) **Web Text Mining:** A fundamental problem with organizing web sources is that web pages are not machine readable. In addition, semantic heterogeneity is a major challenge. Recently, there has been strong interest in developing ontologies to serve as a semantic, conceptual, hierarchical model representing a domain or a web page. If the web pages are presented in the form of a document-term matrix where the instances correspond to the pages and the features correspond to the keywords in the page, the result of subspace clustering will be the identification of a set of keywords (subspaces) for a given group of pages. These keyword sets can be considered to be the main concepts connecting the corresponding groups. Ultimately, the clusters would represent a domain and their corresponding subspaces would indicate the key concepts of the domain.
- (3) **DNA Microarray Analysis:** DNA microarray datasets provide information on the expression levels of thousands of genes under hundreds of conditions. Currently, microarray data must be preprocessed to reduce the number of attributes before meaningful clusters can be uncovered. In addition, individual gene products have many different roles under different circumstances. Subspace clustering is a promising technique that extends the power of traditional feature selection by searching for unique subspaces for each cluster.[2]

We found an implementation of SUBCLU in R in its “subspace” library. The SUBCLU algorithm uses the DBSCAN algorithm to actually cluster the dataset. “mlpack” library of C++ also includes an implementation of DBSCAN accelerated with dual-tree range search technique. “scikit-learn” includes a Python implementation of DBSCAN for arbitrary Minkowski Metrics, which can be accelerated using k-d trees and ball trees but which uses worst-case quadratic memory.[3][4]

2 RELATED WORK

2.1 Is there an existing incremental version of the selected algorithm? If yes, details of the existing incremental algorithm. You have to think how will you compete with the existing incremental algorithm.

No, We did not find any incremental version of SUBCLU. But we still found an incremental version of DBSCAN which is a major subroutine in SUBCLU. The incremental DBSCAN algorithm first checks the ϵ neighbourhood of the newly added point, say p to find the core points which are in the neighbourhood of the core points which were non-core earlier.

- (1) If no new non-core points are found, p is classified as noise.
- (2) If all the new core points were noise earlier, a new cluster is created using p and these new core points.
- (3) If all the new core points were border points of the same cluster, p is absorbed into that cluster
- (4) but if these new core points belonged to different clusters earlier, these clusters are coalesced into a single cluster.

Similarly when a point is removed say p , we check its ϵ neighbourhood to find core points which are in the neighbourhood of the non-core points which were core points earlier.

- (1) If no such points are found, p is simply removed.
- (2) If such points are found and they are density reachable from each other, then some points in the neighbourhood of p will lose their membership and will be classified as noise.
- (3) But if these points are not directly density reachable from each other after removing p , the cluster may split so check must be performed.

We did not find any existing incremental version of SUBCLU but we can incorporate the incremental DBSCAN into SUBCLU to do subspace clustering incrementally.[5][6].

2.2 Are there any interesting variants proposed for the selected algorithm? If yes, details of a few such variants. Knowing such variants will help you to design the incremental version in such a way that it will work on the variants as well.

Yes, there is an interesting variant of SUBCLU algorithm. SUBCLU algorithm is observed to have two bottlenecks. SUBCLU suffers from divergence of density and multi density clusters. Due to these bottlenecks, having a constant value of ϵ throughout the algorithm can lead to merging of two different clusters and inclusion of outliers in the clusters. An algorithm has been suggested to overcome these bottlenecks by dynamically computing ϵ for each cluster in the implementation of SUBCLU algorithm[7].

Another interesting variant of subclu is the combination of subclu and a restricting algorithm. Subclu generates clusters for all subspaces. Many of these can be unnecessary and require human effort to go through all the clustering. To tackle this problem a modification to generate subspaces routine in subclu has been suggested.

User provides the input of interesting attributes to the algorithm and this information is used to prune subspaces which do not have all the interesting attributes present in them. This reduces the number of subspaces with clustering and improves the quality of the output[8].

3 CODE ARCHITECTURE

3.1 Is there existing trust-able implementation available of the selected algorithm? If yes, explain the code architecture along with the data structures and class hierarchy. If no, propose your implementation plan with code architecture, data structures, and class hierarchy.

Yes there existed a trust-able implementation available of the selected algorithm inside ELKI (an open source (AGPLv3) data mining software written in Java). The code mentions the authors of the research paper in its source code. The original paper is not very clear on which clusters to return, as any subspace cluster must be part of a lower-dimensional projected cluster, so these results would be highly redundant. In this implementation, they only include points in clusters that are not already part of sub-clusters.

CODE ARCHITECTURE:

- **Data Structures:**

- (1) **List and ArrayList:** The implementation uses a variety of List and ArrayList to store clusters in a subspace, valid subspaces etc.
- (2) **TreeMap:** The implementation uses the TreeMap (Red-black tree data structure) to efficiently map the subspace to their respective clusters.
- (3) **BitsUtil:** The implementation uses BitsUtil (Bitset data structure) to implement a subspace.
- (4) **HashSet:** The implementation uses HashSet to store DBIDs inside a cluster
- (5) **RangeSearcher:** The implementation uses MTreeVariants to do range query searches on the dataset

- **Class Hierarchy:**

SUBCLU implements the SubspaceClusteringAlgorithm Interface on the SubspaceModel class. SubspaceClusteringAlgorithm is an extension of ClusteringAlgorithm interface which in turn is an extension of Algorithm interface. The SubspaceModel class encapsulates the Subspace class which stores a subspace in bitset format. SUBCLU takes assistance from the DBSCAN class to runDBSCAN on the valid subspaces. SUBCLU class stores the following parameters:

- (1) protected DimensionSelectingSubspaceDistance<V> distance
- (2) protected double epsilon
- (3) protected int minpts
- (4) protected int mindim

SUBCLU runs on the generic NumberVector interface that defines the methods that should be implemented by any Object that is an element of a real vector space. The entire class hierarchy can be found in the given documentation[9].

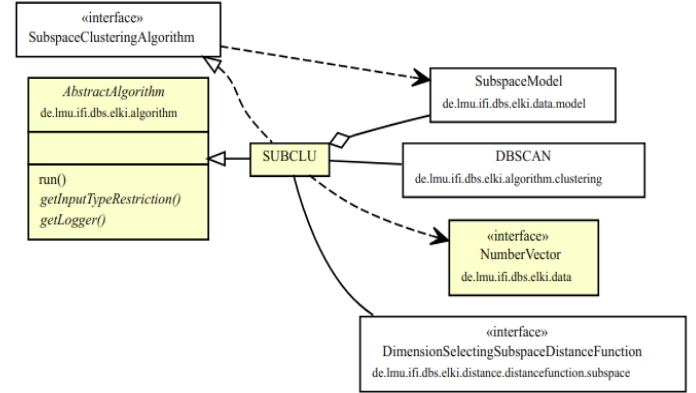


Figure 5: SUBCLU Class Hierarchy

The algorithm starts by call to run(Relation<V> relation) which runs various subroutines for assistance:

- (1) **private List<Cluster<Model>> runDBSCAN (Relation<V> relation, DBIDs ids, Subspace subspace):** Runs the DBSCAN algorithm on the specified partition of the database in the given subspace.
- (2) **private List<Subspace> generateSubspaceCandidates (List<Subspace> subspaces):** Generates d+1 dimensional candidate subspaces from d dimensional subspace.
- (3) **private boolean checkLower(Subspace candidate, List<Subspace> subspaces):** Checks for every d+1 dimensional candidate subspaces its each d dimensional subspace has a cluster.
- (4) **private Subspace bestSubspace(List<Subspace> subspaces, Subspace candidate, TreeMap<Subspace, List<Cluster<Model>>> clusterMap):** Selects the bestSubspace of d dimension with minimal number of objects in the cluster for better runtime performance.

4 C++ IMPLEMENTATION

4.1 Module Organization

We have organized our codebase taking inspiration from the trustable ELKI SUBCLU implementation in java.[9]

- (1) BitsUtil
 - static vector<int> orVectors(vector<int>& a, vector<int>& b);
 - static int intersection(vector<int>& a, vector<int>& b);
- (2) Subspace
 - vector<int> dimensions;
 - int dimensionality;
 - Subspace(int dimension);
 - Subspace(vector<int>& dimensions);
 - bool isSubspace(Subspace& subspace);
 - Subspace join(Subspace& other);
 - bool hasDimension(int i);
- (3) Cluster
 - string name;
 - set<int> ids;
 - bool noise;
 - Subspace subspace;
 - vector<double> mean;
 - Cluster(string name, set<int> &ids, bool noise, Subspace &subspace, vector<double> &mean);
 - int size();
- (4) Relation
- (5) ReadInput
 - ifstream inputFile;
 - ReadInput(string file);
 - ~ReadInput();
 - Relation<double> read();
- (6) DBSCAN
 - Relation<double> m_points;
 - double m_eps;
 - uint m_minPts;
 - vector<int> m_clusterIDs;
 - uint m_numPoints;
 - Subspace m_subspace;
 - map<vector<double>, int> m_ids;
 - int expandCluster(int, uint);
 - vector<int> rangeQuery(vector<double>);
 - double dist(vector<double>, vector<double>);
 - vector<double> getMean(vector<int> &v);
 - vector<Cluster> getClusters();
- (7) SUBCLU
 - double epsilon;
 - int minPts;
 - int minDim;
 - Relation<double> dataBase;
 - map<Subspace, vector<Cluster>> Clustering;
 - map<vector<double>, int> dbids;
 - map<Subspace, vector<Cluster>> run();
 - vector<Cluster> runDBSCAN(Subspace &currSubspace, set<int> &ids);
 - vector<Subspace> generateSubspaceCandidates(vector<Subspace> &subspaces);
 - Subspace bestSubspace(vector<Subspace> &subspaces, Subspace &candidate);
 - bool checkLower(Subspace &candidate, vector<Subspace> &subspaces);

4.2 Class Hierarchy

BitsUtil and Relation are the basic classes. Subspace is built using BitsUtil, in turn Cluster is built using Subspace Class. DBSCAN utilizes Cluster and Subspace Classes. Finally SUBCLU utilizes all ReadInput, Cluster, Subspace and DBSCAN classes. Figure 6 provides a better understanding of the hierarchy.

4.3 Class Description

(1) Subspace

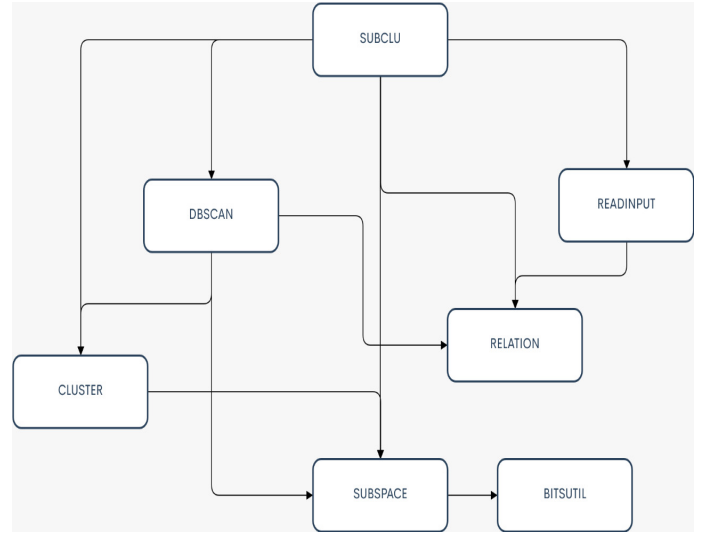


Figure 6: Hierarchy

- (a) Subspace is implemented using a vector to store the dimensions and variable to store the number of dimensions.
 - (b) A value 1 in i^{th} position in vector denotes that i^{th} dimension is present and the value of missing dimensions is 0.
- #### (2) Cluster
- (a) A cluster is represented using a name, IDs of points in the cluster and subspace of the cluster.
 - (b) IDs of points are stored in a set data structure. ID of a point denotes it's position (index) in the dataset.
 - (c) Cluster also maintains a variable noise to distinguish all the noise from the clusters.
- #### (3) Relation
- (a) Each data point in the Relation is represented as a vector of attributes.
 - (b) All the data points are structured in a vector to form the dataset (Relation)
- #### (4) DBSCAN
- (a) DBSCAN takes data, subspace, epsilon and minimum points as inputs. Then returns a vector of clusters.
 - (b) Our implementation of DBSCAN has quadratic time complexity. That is, for processing n points of d dimension time taken is in $O(n^2d)$.
- #### (5) SUBCLU
- (a) SUBCLU takes file name, epsilon, minimum points and minimum dimension as input. Then returns map of subspace and corresponding clusters.
 - (b) SUBCLU maintains a map of each point to it's ID and also a map of each subspace to it's clusters. Map here is implemented using red-black tree and is present in stl library of c++.

REFERENCES

- [1] Gene Expression Dataset,
<https://www.molbiolcell.org/doi/suppl/10.1091/mbc.9.12.3273>
- [2] Example reference,
https://www.kdd.org/exploration_files/parsons.pdf

- [3] Density-based spatial clustering of applications with noise(DBSCAN),
<https://en.wikipedia.org/wiki/DBSCAN>
- [4] SUBCLU R data analysis package,
<https://rdrr.io/cran/subspace/man/SubClu.html>
- [5] Enhanced incremental DBSCAN,
<https://www.sciencedirect.com/science/article/pii/S1110016815001489>
- [6] Incremental DBSCAN,
https://www.researchgate.net/publication/281556454_Incremental_DBSCAN_for_Green_Computing
- [7] SUBCLU variant 1,
<http://j.mecs-press.net/ijitcs/ijitcs-v9-n6/IJITCS-V9-N6-4.pdf>
- [8] SUBCLU variant 2,
https://www.scielo.sa.cr/scielo.php?pid=S0379-39822018000300074&script=sci_arttext&tlng=en
- [9] ELKI SUBCLU documentation,
<https://elki-project.github.io/releases/current/doc/de/lmu/ifi/dbs/elki/algorithm/clustering/subspace/SUBCLU.html>