POLITECNICO DI TORINO

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING

Masters Degree Course in Computer Engineering

Masters degree Thesis

Density-Based Algorithm Based On Spark

Supervisors

prof. GARZA PAOLO

prof. CERQUITELLI TANIA

Candidate:

Biru, Besufekad Assefa

Mat. 202500

July 2018

Dedicated to my late brother
Shimeles Assefa Biru

ABSTRACT

Clustering is one of an important data mining issues especially in big data analysis where large volume of data can be analyzed. It is a technique for finding and grouping similar objects in a data set which objects in the same group or cluster are similar and the ones from different clusters are not similar.

In this thesis paper, we implemented clustering algorithm which identifies dense clusters using association rule mining in subspace of maximum dimensionality and generates clusters irrespective of the order of the input on the top of big data frameworks which satisfies special data mining requirements such as the ability to find clusters in high dimensional data, user understandability of the results, non-presumption of any data distribution, insensitiveness to the order of input records and scalability.

In the next chapters we will show that this algorithm will efficiently discover accurate clusters in high dimensional data sets through experiments.

ACKNOWLEDGEMENTS

Thank you to almighty GOD.

I would like to express my gratitude and extend my sincere thanks to Prof. PAOLO GARZA for agreeing to be my thesis advisor. He has been always extremely focused in his approach, very much accessible for a friendly discussion and spent ample amount to time in understanding my thought process. He taught me how to systematically approach and tackle a research problem.

Thanks mom, family, friends and academic staffs at POLYTECHNICO DI TORINO for making my studies a success.

TABLE OF CONTENTS

CHAPTER ONE

1. INTRODUCTION

This chapter states general topic and gives some background, motivation behind developing this algorithm, objectives & scope, main contribution, and a general description of adopted methods and approaches. The chapter also outlines the general structure of the thesis.

1.1. GENERAL INTRODUCTION

Nowadays technologies are able to store and process huge bulk of data. This kind of data is called big data. Big data requires algorithms, techniques, analytics and new architecture for extracting hidden knowledge, values. It is a large and complex data sets collection which is difficult to process using traditional data processing tools. Big data can be characterized by three V's which are volume of data, variety of data, and velocity (analysis of streaming data). Data become big when their volume, velocity, or variety exceed the abilities of IT systems to store, analyses, and process them. Now widely used as by adding another additional two V's. Which are Veracity that is data uncertainty and Value is information exploited from the data provided

Big data are not just about lots of data, they are actually a new concept providing an opportunity to find a new insight into the existing data. [1]

Mining in big data involves exploring and analyzing large amounts of data to find patterns. It is a process of discovering and exploiting hidden information in data and converting the information more comprehensibly for further applications.

This field involves many tasks for various requirements. A few common tasks are introduced as follows briefly:

Classification identifies categories for new objects by analyzing the existing categories of known data.

Regression analysis is a statistical method for estimating a model that optimally fits data and investigating relationships between variables

Anomaly detection tries to find outliers, which do not have normal patterns compared with other objects.

Clustering is one of the core tasks in data mining because it is the foundation for many other data mining tasks, such as classification, association rule mining and anomaly detection. It is also commonly used in many domains such as marketing, biology, medicine, World Wide Web, pattern recognition, image analysis, bioinformatics, data compression, and computer graphics, machine learning and information retrieval.

Clustering is a technique for finding and grouping set of similar objects in a data set which objects in the same group or cluster are similar and the ones from different clusters are not similar.

Existing Clustering techniques can be broadly classified into two categories [2]

- ✓ Partitional Clustering
- ✓ Hierarchical Clustering

A *partitional clustering* divides a data set into a single partition. The most well-known and commonly used partitioning methods are k-means and k-medoids are (a) the k-means algorithm, where each cluster is represented by the mean value of the objects in the cluster, and (b) k-medoids algorithm, where each cluster is represented by one of the objects located near the center of the cluster. These heuristic clustering methods work well for finding spherical-shaped clusters in small to medium-sized databases. Discovering clusters with complex shapes and very large data sets partitioning based techniques are to be extended.

Whereas *Hierarchical clustering* divides a data set into a sequence of nested partitions, there are two types of hierarchical clustering: Bottom up starts with the points as individual clusters and, at each step, merges the closest pair of clusters.

Top down starts with one cluster and in each step it split into a cluster until only single clusters of individual points remain.

This method suffers from the fact that once a merge or split is done, it can never be undone.

The rigidity behavior is good if it lead to smaller computation costs which will not worry about a combinatorial number of different choices. But, this technique can't adjust wrong decisions.
Aside from two main categories of clustering's the other methods are mainly focused on particular problems. These methods include
[2]

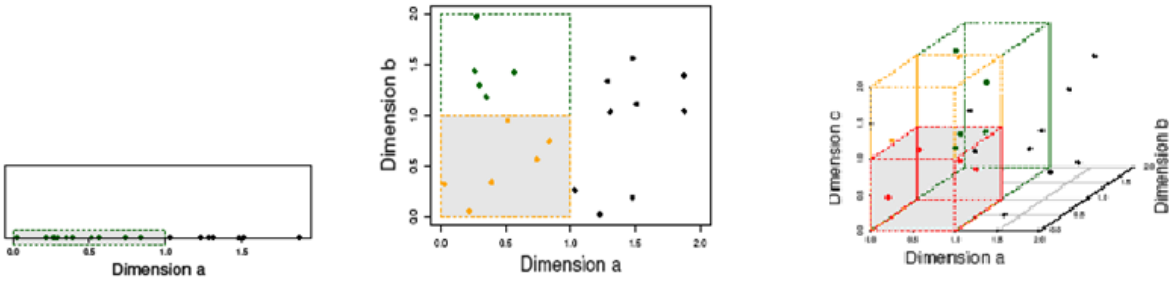
- ✓ Density-based Clustering
- ✓ Grid-based Clustering and
- ✓ others

Density-Based Clustering: This kind of algorithms groups objects based on their density objective functions. It is defined as the number of objects in a specific neighborhood of a data objects. Here a given cluster continues to grow as the number of objects in the neighborhood increases in some parameter. So this approach is considered to be different from the idea in partitional algorithms which uses iterative relocation of points given in a certain number of clusters.

Grid-Based Clustering: This algorithm mainly focuses on spatial data which models the geometric structure of objects in space, their relationships, properties and operations. Its objective is to quantize the data set into a number of cells and then work with objects belonging to these cells. These algorithms don't relocate points but rather build several hierarchical levels of groups of objects. Here merging of grids and clusters doesn't count on a distance measure but it is marked by a predefined parameter so they are closer to hierarchical algorithms.

They consider all dimensions of a data set as a unity for seeking possible clusters which is mainly meaningful for low-dimensional data. However, as soon as the number of dimensions increases, these clustering methods may face some problems.
One problem is that clusters may not exist in the entire space, but in some projections of the dimensions. This often happens in high-dimensional data sets.
Since many dimensions are often irrelevant in high-dimensional data, the irrelevant dimensions can confuse existing clustering algorithms with hidden clusters in noise

Another problem when applying these clustering methods to high-dimensional data arises from the "curse of dimensionality. The phenomenon of the "curse of dimensionality" shows that objects are getting increasingly sparse and dissimilar as the dimensionality increases; meanwhile, the distance between two objects converges and consequently many algorithms may work inefficiently.



(a) 11 Objects in One Unit Bin (b) 6 Objects in One Unit Bin (c) 4 Objects in One Unit Bin

Fig 1.1 Examples of Curse of dimensionality [3]

Unlike the above clustering methods, which search clusters only in the entire space of a data set, **subspace clustering** methods focus on seeking clusters in particular projections of dimensions (subspaces). A subspace cluster can be discovered in arbitrary subspaces instead of the whole space. In other words, only the significant subspaces are found with clusters by subspace clustering algorithms

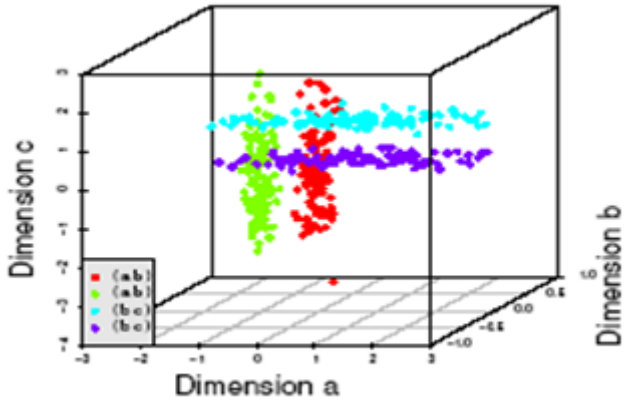


Fig 1.2 Examples of Subspace clustering for three dimensional dataset

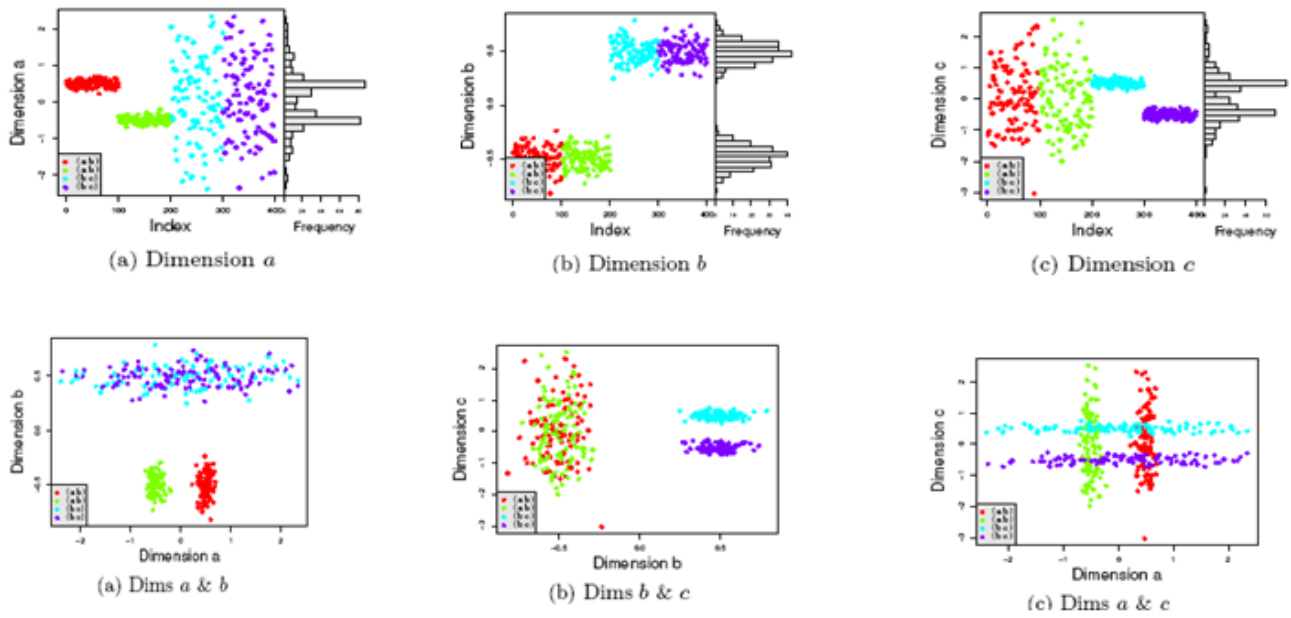


Fig 1.3 View in every dimentsion for the fig 1.2

Association rule mining

This is the data mining process of finding the rules that may mange associations and causal objects between sets of items. [4] And also it is a method for exploring interesting relations between objects.

In a given transaction with multiple items, it tries to find the rules that govern how such items are often bought together and why.

As an example, peanut butter and jelly are bought together sometimes because a many people like to make sandwiches.

And also, diapers and beer are bought together because, as it turns out, that dads are often tasked to do the shopping while the moms are left with the baby.

The main applications of association rule mining:

- Basket data analysis - is analyzing the association of purchased products in single purchase or in single basket.
- Catalog design – In this application type items are often much related or complements, which means the selection of items in a business catalog are often designed to complement each other so that buying one item will lead to buying of another.
- Cross marketing – this type works with other businesses that complement your own, but not competitors. Example, vehicle dealerships and manufacturers have cross marketing campaigns with gas and oil companies for apparent reasons.

Association rule mining objective is the extraction of frequent correlations or pattern from data sets which is the focus of our thesis.

1.2. MOTIVATION

This section of introduction will answer the need of our clustering algorithm. So data mining algorithms as a whole places the following special requirement on clustering techniques such as

Treatment of high dimensionality effectively

When dimensionality grows high clustering techniques encounters challenges because most clustering algorithms are designed for clustering low-dimensional data. The proposed algorithm should deal with data set of high dimensionality

Result Usability and Interpretability

Clustering results should be understandable, comprehensible, and usable by users. That is, it needs to be tied in with specific applications and semantic interpretations. It's important to study how an application goal may influence the selection of clustering features and clustering methods [2].

Results obtained must be understandable and usable to the maximum knowledge about the input parameters can be obtained.

Scalability

Lots of clustering algorithms works well on small datasets containing several hundred data objects, Clustering on only a sample of a given large data set may lead to biased results. Thus, highly scalable clustering algorithms are needed [2]

So the Resulted data must be scalable otherwise we may get the wrong result.

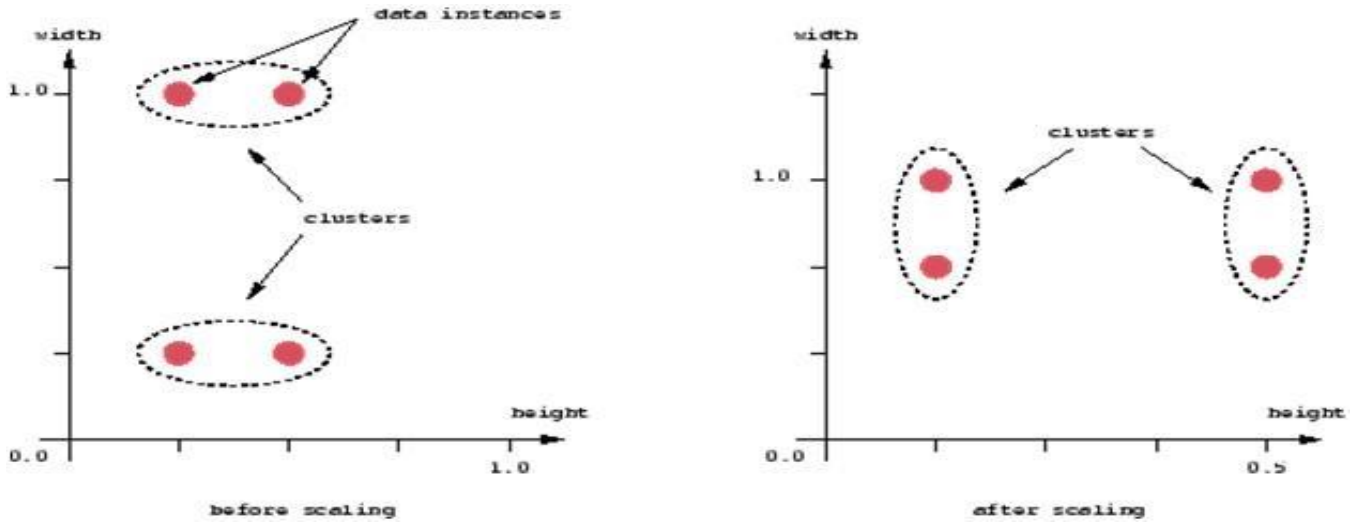


Fig 1.4 Example where scalability may leads to wrong result [5]

1.3. OBJECTIVE AND SCOPE

The main objective of our study is to develop clique subspace clustering algorithm by using association rule mining on top of big data frameworks to identify better clusters in subspace of high dimensional data which satisfies the requirement such as scalability, insensitive to order of input and effectively treat highest dimensionality, and our scope is limited to the following:

1- Identification of subspaces that contain clusters
2- Identification clusters from a given data set based on density of objects

And this algorithm only works for numerical dataset.

1.4. MAIN CONTRIBUTION

In this thesis work we implemented subspace clustering algorithm (clique) by using association rule mining on the top of spark framework as a result we get better cluster in subspace of higher dimensional data based on the density of objects in terms of execution time and clustering accuracy.

1.5. THESIS OUTLINE

This paper is organized in different chapters. The first (this chapter) presents an introduction. The rest of the thesis is organized as follows. In chapter two we will discuss existing clustering algorithms and Subspace clustering with its problem and varieties.

In chapter three we will discuss the techniques behind our main algorithm we implemented; density-based algorithms based on spark.

In Chapter Four Its Implementation which will discuss tools and technologies used to implement and its Pseudocode, Chapter Five experimental observations will be discussed. In the last chapter (chapter 6) we will present Conclusion of our study and future work.

CHAPTER TWO

2. RELATED WORK

In this chapter we will see brief overview of existing well known clustering algorithms and our main subspace clustering categories with their pros and cons. and main problem statement for our study.

2.1. EXISTING CLUSTERING METHODS

The existing clustering algorithms can be classified broadly as partition-based, hierarchy-based clustering, density based methods and others. We introduce some of the well known clustering algorithms from all categories respectively

CLARANS (Ng and Han'94).

It is type of partitional clustering algorithm which draws sample of neighbors dynamically.

The process of clustering is like searching a graph on which every vertex is possible solution, that is, a set of k-medoids. If the local optimum is found, it starts with new randomly selected node in search for a new local optimum

Its advantages over the previous similar algorithms are more efficient and scalable than both PAM and CLARA and to further improve focus on techniques and spatial access structures (Ester et al.'95) [6]

***BIRCH* (Zhang et al., 1996),**

It is type of hierarchical clustering algorithm which is designed for clustering a large amount of numeric data by integrating hierarchical clustering (at the initial micro clustering stage) and other clustering methods such as iterative partitioning (at the later macro clustering stage) which addresses the two difficulties in agglomerative clustering methods:

1- Scalability and
2- Inability to undo what was done in the previous step.

It uses CF-tree clustering features to represent a cluster hierarchy and notions of clustering feature to summarize a cluster, These structures help the clustering method achieve good speed and scalability for databases which are large or even streaming types, and makes effective incremental and dynamic clustering of incoming objects [2].
Its advantage over the previous similar algorithms is each clustering decision is made without scanning all data points and currently existing clusters.
In this technique every data point are not equally important, which uses full memory to derive the best potential sub-clusters although minimizing costs associated to I/O and exploits the observed data space usually not uniformly occupied.
It is also an incremental method that does not require the whole data set in advance.

Its disadvantage over the previous similar algorithms is [7]

- What was done previously is not undone in this algorithm
- O (n2 log n) time complexity.
- It suffers with one or more of the following based on the type of distance matrix chosen for merging different algorithms:
    - Sensitivity to noise and outliers
    - Breaking large clusters
    - Difficulty handling different sized clusters and convex shapes

DBSCAN (Ester et al., 1996)

It is a type of density based clustering algorithm that finds clusters by the density of an object o, which can be measured by the number of objects close to o. It finds core objects, that is, objects that have dense neighborhoods. It connects core objects and their neighborhoods to form dense regions as clusters. It quantifies the neighborhoods with A user-specified parameter $\varepsilon > 0$ is used to specify the radius of a neighborhood we consider for every object.
The ε -neighborhood of an object o is the space within a radius ε centered at o. [2]

Algorithm: DBSCAN: a density-based clustering algorithm.

Input:

- D: a data set containing n objects,
- ϵ: the radius parameter, and
- *MinPts*: the neighborhood density threshold.

Output: A set of density-based clusters.

Method:

```
(1)  mark all objects as unvisited;
(2)  do
(3)      randomly select an unvisited object p;
(4)      mark p as visited;
(5)      if the ε-neighborhood of p has at least MinPts objects
(6)          create a new cluster C, and add p to C;
(7)          let N be the set of objects in the ε-neighborhood of p;
(8)          for each point p' in N
(9)              if p' is unvisited
(10)                 mark p' as visited;
(11)                 if the ε-neighborhood of p' has at least MinPts points,
                     add those points to N;
(12)             if p' is not yet a member of any cluster, add p' to C;
(13)         end for
(14)         output C;
(15)     else mark p as noise;
(16) until no object is unvisited;
```

Fig 2.1 DBSCAN algorithm [2]

Its advantages over the previous similar algorithms are [8]

- It does not require one to specify the number of clusters in the data a priori, as opposed to k-mean
- It can find arbitrarily shaped clusters. It can even find a cluster completely surrounded by (but not connected to) a different cluster.
- It has a notion of noise, and is robust to outliers.
- It needs two parameters and mostly insensitive to the order of the points in the database.
- It is designed to use with databases that can accelerate region queries ... etc

Its drawbacks over the previous similar algorithms are [8]

- Its quality depends on the distance measure used in the function regionQuery(P,ε). For high dimensional data the most common distance metric used is Euclidean distance, this metric can be useless due to "Curse of dimensionality", making it difficult to find an appropriate value for ε. The effect is also present in any other algorithm based on Euclidean distance. So because of the “curse of dimensionality” the Euclidean distance as the metric used in DBSCAN is not suitable for high-dimensional data sets.
- Data sets with large differences in densities can’t be clustered by DBSCAN since the minPts-ε combination can’t be chosen appropriately for all clusters.
- Choosing meaningful distance threshold ε can be difficult if the data and scale are not understood.

DOC/Fast DOC (Procopiuc et al., 2002) [9], is recently another density-based projective clustering algorithm has been proposed. This approach requires the maximum distance between attribute values as parameter in input, which follows an optimality criterion defined in terms of density of each cluster in its corresponding subspace. Then Monte Carlo procedure developed to approximate with high probability an optimal projective cluster. In practice it may be difficult to set the parameters of DOC, as each relevant attribute can have a different local variance.

2.2. SUBSPACE CLUSTERING METHODS

Subspace clustering is clustering algorithm that seeks to find clusters in different subspaces within a dataset (Often high dimensional data) which localizes the search for relevant dimensions allowing them to find clusters that exist in multiple, possibly overlapping subspaces.
Has two major branches, the **top-down** and **bottom-up approaches** methods [2]

The first one is **Top-down approaches** which starts from the full space and continue to search smaller and smaller subspaces recursively and its effective only if the locality assumption holds, that require that the subspace of a cluster can be determined by the local neighborhood. PROCLUS is one of its examples.

PROCLUS is type of a top-down subspace approach which first generates k potential cluster centers for a high-dimensional data set using a sample of the data set. It then refines the subspace clusters iteratively.
For each of the current k-medoids, it takes the local neighborhood of the medoid in the whole dataset and it identifies a subspace for the cluster by minimizing the standard deviation of the distances of the points in the neighborhood to the medoids on each dimension in each of its iterations. Each point in the data set is assigned to the closest medoids according to the corresponding subspace after all the subspaces for the medoids are determined. Clusters and possible outliers are identified. Then in the next iteration, new medoids replace existing ones if doing so improves the clustering quality

ORCLUS is an extension of PROCLUS which tries to find projected subspace clusters that are not parallel to the axes. By computing the covariance matrix, subspaces are relocated with the clusters. The closest pairs of clusters with similar directions are merged (by Aggarwal and Yu in 2000).

LAC is an algorithm which is proposed recently that discovers clusters in subspaces. With local weightings features, LAC is traversed by different combinations.
This approach mitigates the risk of loss of information encountered in feature selection or global dimensionality reduction techniques (by Domeniconi et al., 2004).

The second one is ***Bottom-up approaches*** which starts from low-dimensional subspaces and continue to search higher dimensional subspaces only when there may be clusters in those higher-dimensional subspaces. Different pruning techniques are explored to reduce the number of higher dimensional subspaces that need to be searched. From this techniques CLIQUE is an example of a bottom-up approach.

CLIQUE is type of a bottom-up subspace clustering approach; which is grid-based method for finding density based clusters in subspaces. CLIQUE divides every attribute into non overlapping intervals by partitioning the all embedding space of the data objects into cells. For identification of dense and sparse cells clique uses density threshold. If a cell is dense the number of objects mapped to it exceeds the density threshold.
A density based approach to clustering is used in clique; in this approach a cluster is a region that has a higher density of points than its surrounding region. Here the problem is to discover projections of input data into subset of attributes with regions that include high density of projections.
So Before giving a description of the problem of subspace clustering, we first give some explanation of our clustering model.

Our interest is to automatically identify subspaces of a high dimensional data space which allows better clustering of the data points than the original space. Only limiting our search to sub spaces of the original space allows much simpler and comprehensible presentation of the results instead of new attributes. Linear combinations of the original dimensions is an example for restricting our search

Every one of the original dimensions has a real meaning to the user while a simple linear combination of many dimensions may be hard to translate (Fayyad et al., 1996).

We use a density based approach to clustering, in this approach a cluster is a region that has a higher density of points than its surrounding region. Here the problem is to discover projections of input data into subset of attributes with regions that include high density of projections.

To discover the density of points we will slice the data space to find the number of points which lie in each cell which accomplished by partitioning each dimension into the same number of equal length of intervals. That is each unit will have the same volume and therefore the number of points inside will be used to find the density of the cell.

The remaining task is to find clusters in the projections that the data points are separated based on the density function after subspaces are found.
The clusters found are the unions of connected density units or cells within subspace; we will limit the clusters to axis-parallel hyper-rectangles for simplified description.

Every cluster is a collection of cells comprised in a *k*-dimensional subspace which is a conjunction of inequalities that is the intersection of 2*k* axis-parallel half spaces defined by *k* 1-dimensional intervals, which can be described with a DNF expression. Its description is obtained by cluster with a minimal number of maximal overlapping rectangles. [9].

2.3. PROBLEM STATEMENT

The problem of subspace clustering is described as follows.

Let : A = {A1, A2, A3……Ad}, is a set of bounded totally ordered domains and

S = A1x A2x A3x…….Ad a d-dimensional numerical space.

we will refer to A1, A2, A3…….. , Ad as the dimensions (attributes) of S.

The input consists of a set of d-dimensional points V = {v1, v2, v3 ……..... vm} where vj = vj1, vj2, vj3, ……...., vid. The jth component of vj is drawn from domain Aj [9].

We partition the data space S into non-overlapping rectangular units. By Partitioning every dimension into ξ which an input parameter, intervals of equal length we will get a unit or cell.

Each unit u is an intersection of one interval from each attribute. A unit u has the form {u1, u2, u3…….. ud} where ui = [li , hi) is a right-open interval in the partitioning of Ai .

We say that a point $v = v_1, v_2 \ldots v_d$ is contained in a unit $u = \{u_1, \ldots, ud\}$ if $l_i \leq v_i < h_i$ for all u_i.

The fraction of total data points contained in the unit is the selectivity of a unit. If a unit u is dense the selectivity u is greater than τ is density threshold and is another input parameter.
Similarly units are defined in all subspaces of the original d-dimensional space. Consider a projection of the data set V into At1 × At2 × At3 × ……× Atk, where k < d and $t_i < t_j$ if i < j.

A unit in the subspace is an interval points which intersects from each of the k attributes.
A cluster is a maximal set of units which are dense and connected in k-dimensions.
If two k-dimensional unit's u1 and u2 said to be connected they have common face. Or if they exists in another k-dimensional unit u3 such that u1 is connected to u3 and u2 is connected to u3.
Units u1 = { r_{t1}, r_{t2}, r_{t3},……., r_{tk}} and u2 = {r^1_{t1}, r_{t2}, r_{t3},……., r^1t_k} have a common face if there are k − 1 dimensions, assume dimensions A_{t1}, . . . , A_{tk-1}, such that $r_{tj} = r^1_{tj}$ and either $h_{tk} = l^1_{tk}$ or $h^1_{tk} = l_{tk}$, for j ∈ {1, …….. , k − 1}.

We are interested in those regions in k dimensions which are a union of connected units, in an axis-parallel rectangular k-dimensional set.

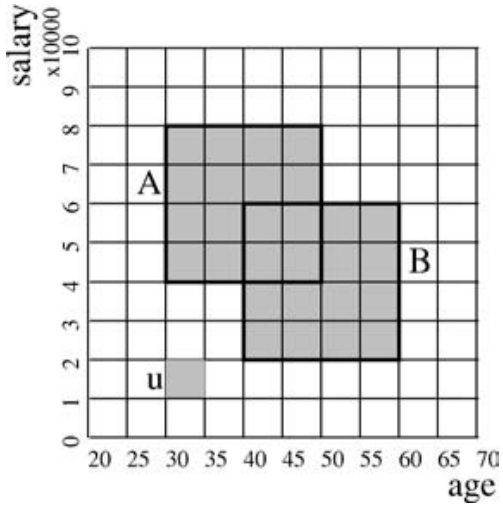


Fig 2.2 Illustration of definitions [9]

So in the above 2 dimensional (age and salary) figure, u stands for unit, A and B are regions

A region is described in DNF expression on intervals of the domains *Ai*, if we say a region *R* is *contained* in a cluster *C* if R intersection C is equal to R. If it is said to be *maximal* if no proper superset of *R* is contained in *C*.

When we say *minimal description* of a cluster covering with maximal regions is non redundant cluster, means that a set *R* of maximal regions such that their union equals *C* however the union of any proper subset of R is not equal to *C*.

For example: In figure 2.2, the two dimensional space of x-axis (age) and y-axis (salary) has been divided in to a 10×10 grid. A unit $u = (30 \leq \text{age} < 35) \wedge (1 \leq \text{salary} < 2)$ and A region A = $(30 \leq \text{age} < 50) \wedge (4 \leq \text{salary} < 8)$ and $B = (40 \leq \text{age} < 60) \wedge (2 \leq \text{salary} < 6)$.

Assuming that shaded are the dense units, $A \cup B$ is a cluster. DNF expression for minimal description for this cluster is: $((30 \leq \text{age} < 50) \wedge (4 \leq \text{salary} < 8)) \vee ((40 \leq \text{age} < 60) \wedge (2 \leq \text{salary} < 6))$.

In figure 2.3, there are no clusters in the original data space if we assume $\tau = 20\%$, no 2-dimensional unit is dense. But there are three 1-dimensional dense units if the points are projected on the salary dimension. Two of them are connected, and there are two clusters in the 1-dimensional salary subspace: $C1 = (5 \leq \text{salary} < 7)$ and $D1 = (2 \leq \text{salary} < 3)$.

Finally in the age subspace there is no cluster because there is no dense unit in that specific subspace.

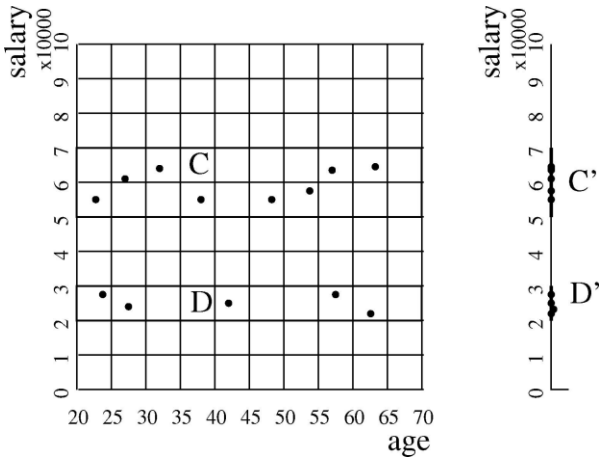


Fig 2.3 Cluster identification in areas of the original data space

In the following chapters we will analyze the problem of clique subspace clustering and give our detailed solution.

CHAPTER THREE

3. PROPOSED ALGORITHM: DENSITY-BASED ALGORITHM BY MEANS OF SPARK

To solve the defined research problem in (chapter 2) we introduce our clustering algorithm which is called density-based algorithm on the top of big data frameworks. We will discuss these techniques in detail as follows

The chapter has been divided into three parts. The first part deals with some basic subspace clustering definitions. The following part describe the detailed description of the techniques used and steps of our algorithm

3.1. DEFINITIONS

3.1.1. DATA AND DATASETS

Data is qualitative and quantitative collection of facts. For example words, numbers measurements, observations etc. [10]

- Qualitative data which focuses describing something.
- Quantitative data which focuses on numerical information.

Quantitative data can also be Discrete or Continuous:

- Discrete data takes only certain values (like whole numbers) and counted
  - It has finite or countable infinite set of values.
  - It is represented as integer variables often.
  - Binary attributes are special discrete attributes.
- Continuous data can take any value (within a range) and measured
  - It has real numbers as attribute values Examples: temperature, height, or weight.

- Continuous attributes are represented as floating-point variables and real values are measured using a finite number of digits.

A *Dataset* is a collection of data, which is usually structured in a tabular form that consists of rows and columns, a tree form with hierarchical structure or a graph form with interconnected nodes. Different structures are required by various applications.

3.1.2. SUBSPACES

A space that is fully contained in another space, or whose points or elements are all in another space

3.1.3. INPUT INTERVAL AND DENSITY THRESHOLD

Input interval (ξ) is the first input parameter which uses to partition the dataset into equal width interval to get the required bins.

Density threshold (τ), also called minimum support threshold, is the second parameter used to decide if the unit is dense or not.

3.1.4. UNIT

A unit is the intersection of one interval from each attribute.

3.1.5. DENSE UNITS

Dense units – if the units is above the given threshold it is said to be dense.

3.1.6. CONNECTED UNITS

Connected units – if two dense units are said to be connected they should have common face or adjacent edges

Before we begin our clustering technique will see how preprocessed our dataset:

3.2 PREPROCESSING

Integration of raw data collected from one or more sources may not be straight-forward. Real world data is often marred with errors, missing values and technician bias. Preprocessing phase is the opportunity to impute missing value, to treat noise, normalize, transformation, integration, mitigate inconsistencies and enhance reliability of data significantly, reduction and discretization.

3.2.1. MISSING VALUES IMPUTATION

Mostly missing values are common problem in the acquisition process of data mining techniques in the data set. This are datum which has not been stored or gathered due to a incorrect sampling process, limitations in the acquisition process which cannot be avoided in data analysis and will create difficulties if handled inappropriately which will easily lead to poor knowledge extraction and wrong conclusions. In many approaches it will be discarded in those instances that may contain missing values to tackle the problem caused by missing values so we discarded some missing values in our data preprocessing steps.

3.2.2. NOISE TREATMENT

Treating noise in data mining algorithm assumes that any dataset is a sample of underlying distribution with no disturbances. In previous sections data gathering are sometimes perfect and rarely corruptions appear. However the quality of results obtained is dependent on the quality of data and tacking the problem of noise data is mandatory in data mining techniques. If noise is present in input attributes which called attribute noise in supervised problems, it affects both input features and the output values or one of them which is called attribute noise. In these situations the worst case is when the noise affects the output attribute.

There are two main approaches to treat noise treat noise in data mining; these are commonly used in the data preprocessing literature. The first one is to correct the noise by using ***data polishing methods***, especially if it affects the labeling of an instance. Even partial noise correction is claimed to be beneficial, but it is a difficult task and usually limited to small amounts of noise. The second one and our approach selection is to use ***noise filters***, which identify and remove the noisy instances in the training data and do not require the data mining technique to be modified.

3.2.3. DATA DISCRETIZATION

It is a family of data transformation techniques in which continuous numerical values are transformed into a finite set of discrete intervals. For a numerical attribute A with an interval [a, b] as range, discretization of A is defined as a partition of the range into n intervals:

$\{[a_0, a_1),[a_1, a_2),\ldots\ldots,[a_{n-2}, a_{n-1}),[a_{n-1},a_n]\}$

Where $a_0 = a$, $a_n = b$, and $a_i < a_{i+1}$ for I = 0, 1, …………… , n−1. The numbers a_1, a_2, a_3, ………….., a_{n-1} are called cut-points. Discretization methods are called local if attributes are processed one at a time and global if all attributes are simultaneously considered towards selection of a best cut-point.

Discretization techniques are categorized based on ***class information*** or on which ***direction*** it proceeds. It is **supervised discretization** it uses class information else, **unsupervised**.
Top-down discretization or splitting in this technique starts by first finding one or a few points which is referred as split points or cut points to split the entire attribute range and then repeats recursively on the resulting intervals.
Bottom-up discretization which the second one starts from all continuous values as possible splitting points and removes some by merging neighborhood values which will form intervals, and then recursively enforces this process to the resulting intervals.

Types are discussed as follows

- ✓ Equal width intervals
- ✓ Equal frequency intervals
- ✓ Minimal class entropy method
- ✓ Entropy based discretization

Equal width intervals

This is unsupervised kind of discretization technique where entire range is partitioned into a number of equal width intervals. Range of the numeric attribute is sliced into a number of equal parts, or bins then when a numeric value falls into a bin; we will take the bin name as discretized version of the numeric value and the number of data points between bins may vary.

The algorithm divides the data into k intervals of equal size.

The width of intervals is:

$$w = (\text{maximum-minimum}) / k$$

And the interval boundaries are:

$$\text{Min} + w,\ \text{min} + 2w, \ldots\ldots\ldots, \text{min} + (k-1)\, w$$

Equal frequency intervals

Another unsupervised approach where interval widths may vary but sample frequency is same in every interval and therefore all discretized intervals have equal information content. Again, the desired number of intervals must be determined stochastically or supplied by the user. For example

- **Data :** 0, 4, 12, 16, 16, 18, 24, 26, 28
- **Equal width**
  - Bin 1: 0, 4 [-,10)
  - Bin 2: 12, 16, 16, 18 [10,20)
  - Bin 3: 24, 26, 28 [20,+)
- **Equal frequency**
  - Bin 1: 0, 4, 12 [-, 14)
  - Bin 2: 16, 16, 18 [14, 21)
  - Bin 3: 24, 26, 28 [21,+)

Fig 3.1 Equal width and frequency discretization example

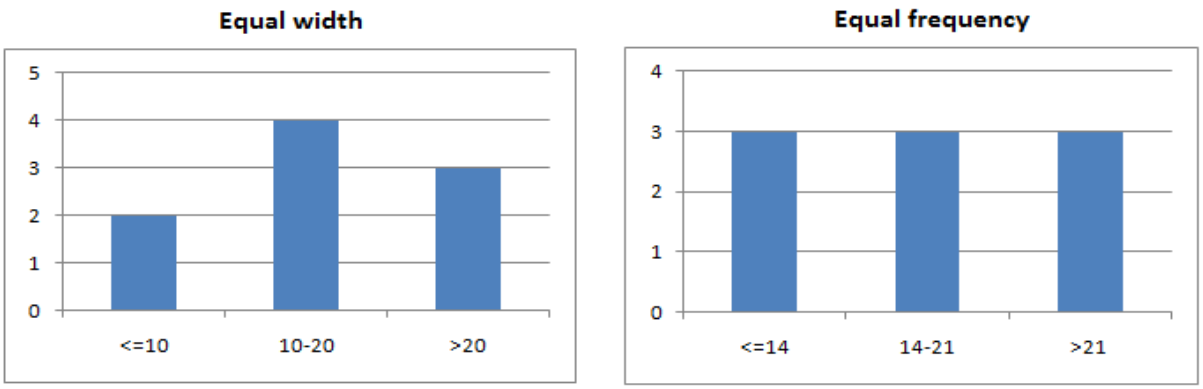


Fig 3.2 Equal width and frequency discretization seen by graph for example above

Minimal class entropy method

This method computes class entropy associated with subsets of values partitioned by the selected cut-point. Let class C has k concepts associated with a set S, then class entropy of S, E(S) is defined as:

$$E(S) = -\sum_{i=1}^{k} \frac{|c_i|}{|S|} \log_2 \frac{|c_i|}{|S|}$$

where |S| and $|C_i|$ are the cardinalities of S and ith concept respectively. Negative sign in the expression assures that the quantity is always positive; whose lower value implies closer association (or better fit) between set and class. To evaluate a cut-point q for an attribute A, weighted average of class entropies E(A, q; S) of the partitioned subsets S1 and S2 are determined as:

$$E(A, q; S) = \frac{|S_1|}{|S|} E(S_1) + \frac{|S_2|}{|S|} E(S_2)$$

This quantity is called the class information entropy. Binary discretization for an attribute is determined by computing $(A, q_i; S)$ for all possible cut-points and selecting the one for which the quantity is minimum. The process is recursively applied to the subsets until a stopping criterion is satisfied. Minimum description length principle (MDLP) criteria is one of such approaches that accepts cut-point if the result of partition leads to a positive information gain, otherwise recursion in the discretization process stops without further partitioning.

Entropy based discretization

Entropy based discretization takes into consideration the information content of both attribute and decision variables.

In our pre-processing step we extracted header and data separately from dataset, and pruning noises and missing values remove or mitigate inconsistencies and duplication, discretized our data using equal width intervals to bins our dataset into meaningful categorical attributes.

3.3. OUR CLUSTERING METHOD

Our clustering technique performs multidimensional clustering in two steps.

1. Identification of subspaces that contain clusters.
2. Identification of clusters.

3.3.1. IDENTIFICATION OF SUBSPACES THAT CONTAIN CLUSTERS

Identification of subspaces which contain clusters lays in finding dense units in different subspaces. In the first step, to identify subspaces the algorithm partitions the d-dimensional data space into non overlapping rectangular units, identifying the dense units among these. This is done (in 1-D) for each dimension. For example, Figure 3.3 shows dense rectangular units found with respect to *age* for the dimensions *salary* and (number of weeks of) *vacation*.

A *candidate* search space in which dense units of higher dimensionality are obtained after subspaces which represent the dense units are intersected.

The identification of the candidate search space is based on the ***Apriority property*** used in **association rule mining (see section 3.3.1.1)**. In general, the property employs prior knowledge of items in the search space so that portions of the space can be pruned. The property, adapted for CLIQUE, states the following: *If a k-dimensional unit is dense, then so are its projections in* (*k* -1)*-dimensional space*. That is, given a *k*-dimensional candidate dense unit, if we check its (*k*-1)-th projection units and find any that are not dense, then we know that the kth dimensional unit cannot be dense either. Therefore, we can generate potential or candidate dense units in *k*-dimensional space from the dense units found in (*k*-1)-dimensional space. In general, the resulting space searched is much smaller than the original space. The dense units are then examined in order to determine the clusters.

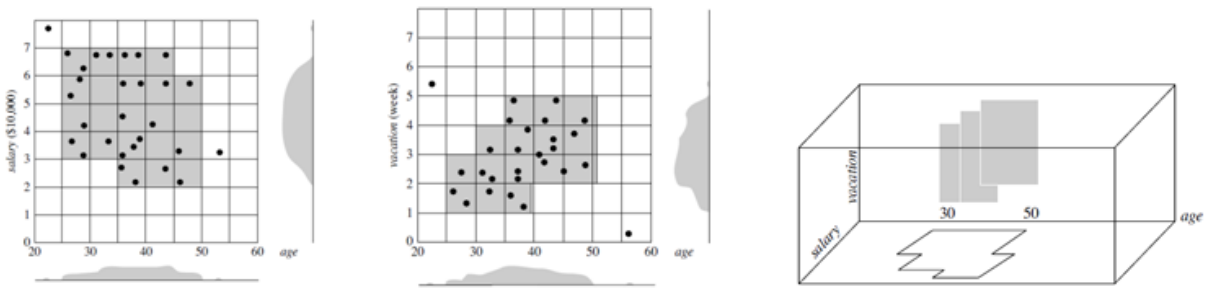


Figure 3.3 Dense units discovered with respect to *age* for the dimensions *salary* and *vacation* [2]

3.3.1.1 Association rule mining

In our algorithm this techniques used to discover frequent patterns /associations/correlations/ and other causal structures from data sets in rule mining in different types databases whether relational databases, transactional databases, or other forms of data repositories. [4]

Given a set of transactions T= $\{t_1, t_2, t_3, \ldots\ldots\ldots\ldots, t_n\}$, each transaction contain set of items (item set) An itemset is collection of items I= $\{ i_1, i_2, i_3, \ldots\ldots\ldots, t_m\}$, association rule mining aims to find the rules which enable us to predict the occurrence of a specific item

To find frequent itemset in our algorithm, first after we found the intervals in our preprocessing step we generate unit (intersection of each attribute intervals) then we count the number of points in the each unit satisfying the rule. Our rule or condition is the points should be between the two boundaries of the unit. After that we will find the dense units, if number of points in each unit is greater than the given density threshold it is said to be a dense unit.

So the bold steps in these categories are

A. Unit generation
B. Rule generation from intervals
C. Counting the number of points in each unit satisfying rule
D. Finding dense unit if the number of unit greater than the given density threshold which in turn prune the remaining points

3.3.2. FINDING CLUSTERS

In the **second** step, the input to the next step is a set of dense units *D*, all in the same *k*-dimensional subspace *S*. The output will be a partition of *D* into D1, D2, D3……., *Dq*, such that all units in *Di* are connected and no two units $ui \in Di$, $u\,j \in Dj$ with $i = j$ are connected. Each such partition is a cluster according to our definition.

The problem is equivalent to finding connected components in a graph defined as follows: Graph vertices correspond to dense units, and there is an edge between two vertices if the accompanying dense units have a common face.

Units corresponding to vertices in the same connected component of the graph are connected because there is a path of units that have a common face (adjacent) between them; therefore they are in the same cluster. In contrast, units accompanying to vertices in different components can't be connected so they can't be in the same cluster.

We use a **Depth-first search algorithm** (see section 3.3.2.1) to find the connected components of the graph. We start with some unit *u* in *D*, assign it the first cluster number, and find all the units it is connected to. Then, if there still are units in *D* that have not yet been visited, we find one and repeat the procedure.

Time complexity. In a given subspace the number of dense units can't be very large, because each dense unit must have at least τ selectivity. We assume therefore that the dense units in this and subsequent steps can be stored in memory. Asymptotic running times are given in terms of accesses performed on dense unit which are stored hash tree in a memory data structure (Aggarwal et al., 1996) which gives an efficient querying.

For each dense unit visited, the algorithm checks its $2k$ neighbors to find connected units.

If the total number of dense units in the subspace is *n*, the total number of data structure accesses is $2kn$. [9]

3.3.2.1 Depth-first search algorithm is an algorithm for searching graph data structures which starts by selecting some arbitrary node as the root in the case of a graph and explores along each branch before backtracking as far as possible.

Some of the applications of DFS are finding Connected Components of the graph (DFS is used to determine if a graph is connected); bi-connectivity and finding Strongly Connected Components (determine if a graph is strongly connected and finding strongly connected components using a stack)

Advantages

- Consumes less memory
- Less time and space complexity rather than Breadth First Search (BFS).
- Solution can be found out by without much more search.

Disadvantages

- Cut-off depth is smaller so time complexity is more
- Determination of depth until the search has proceeds

In this step given set of dense unit we will find connected dense unit, to do that we first CreateAjacentMatrix from dense units, by checking if two dense units are have the same & consecutive edge then set matrix to '1' else '0' (connected ‘1’, Not-connected ‘0’) then we find Neighbors for each source dense units after that will get connected dense units. If there is unvisited dense Unit the work will continue in the above fashion until all dense units are visited.

3.4. POST-PROCESSING

This step of data mining translates discovered patterns into forms acceptable for human beings. It may also make possible visualization of extracted patters

So in this last step of we map original points into connected dense units or clusters.

CHAPTER FOUR

4. IMPLEMENTATION

In this chapter we will describe tools and technology used to implement our algorithm and main steps and Pseudocode.

4.1 TOOLS AND TECHNOLOGIES

4.1.1 TECHNOLOGIES

Since we are dealing with both small and very large dataset, we have implemented our algorithm by using big data technologies.

Nowadays there are various big data technologies; we will discuss selected technologies for our algorithm as follows.

Big data has great potential to produce useful information for companies which can benefit the way they manage their problems. Its analysis is becoming indispensable for automatic discovering of intelligence that is involved in the frequently occurring patterns and hidden rules. For these huge data sets which are too large and complex for humans, it is difficult to extract information without the aid of computational tools. Spark and Hadoop framework are new and emerging technologies that offer new ways to process and transform big data which are (complex, unstructured, or large amounts of data) into meaningful knowledge [11]

Spark

It is a powerful parallel programming technique for distributed processing of vast amount of data on clusters and is a cluster computing platform designed to be fast and general purpose.
Spark supports computations, interactive queries, stream processing and others which are the extended functionalities of map–reduce model.

In processing large datasets speed is most important so spark offers the ability to run computations in memory and it is more efficient.

It offers API in Scala, java, python, SQL and built in libraries and also integrates with other tools mainly big data tolls. [11].

It parallelize tasks in clusters and covers complexities such as distributed system programming and also network communication, and fault tolerance. Enough control to monitor, inspect, and tune applications will be given by the system while allowing common tasks to be implemented quickly.

Its API Modular nature makes it to easily reuse its libraries and for local testing. It provides a wide variety of functionality, is easy to learn and use, and is mature and reliable.

These are some of the features of spark are

- It is in memory cluster computing platform
- It ecosystems are Spark SQL, Spark Graphx, Spark MLlib and Spark Streaming
- GraphX, a distributed graph system are integrated
- Iterative processing, interactive processing and event stream processing supported
- It run on Hadoop alongside other tools in the Hadoop ecosystem including Hive and Pig
- It is very flexible and powerful
- Supports Machine Learning Algorithms for Future Predictions
- Supports languages like Scala, Java, Python and R.
- HDFS and Hbase and Casendra and other storage systems can be integrated.

It has some limitations also

- A lot of memory consumed, and memory consumption issues which are not handled in a user friendly manner.
- Apache Spark would take large resources.

Hadoop

Hadoop is an open source virtual grid operating system for processing a storage which stores data and processes it; it's also a scalable and fault tolerant. It uses HDFS files system and runs on commodity hardware; HDFS is file system with fault-tolerant highest bandwidth clustered storage architecture. Hadoop runs spark for distributed data processing and works with structured data and unstructured ones. Tools such as Hive, Pig and Mahout which are parts of Hadoop and HDFS framework used in handling the velocity and heterogeneity of data, Hadoop and its File System are used by apache for storing and managing big data.

4.1.2 PROGRAMMING LANGUAGE

Since spark supports java, we have selected to write some of our code by using java programming language rather than Scala is because of familiarity with it and have some interesting features such as:

- ✓ ***Simplicity***: Java was designed to be easy to learn and use effectively. Complex operations such as handling memory leaks and garbage collection are taken care by automatic memory management and thus all the complexities are hidden from the programmer.

- ✓ ***Platform independent***: Both system software and machine architecture have been evolving continuously and therefore, one of the challenges for programmers is to maintain their own code for execution on different platforms and at different times. Java allows program to be written once and run anywhere/anytime

- ✓ ***Security***: Java restricts internet based applets to its own execution environment and therefore other system resources are protected from unauthorized access.

- ✓ ***Industry standard***: Platform independence gives a big advantage to any industry and the automatic memory management hides unnecessary technical jargon.

4.1.3. DATA STRUCTURES

A problem can be divided into a number of sub-problems and the solution can be reached in a different ways. Algorithms must be written to maximize the chances of achieving goal and minimize the amount of time and effort involved. The efficiency issue becomes most obvious when the size of input data is large. As an example, a poorly written algorithm for maximum subsequence sum takes 2.28 seconds for input size of small dataset but it fails to come up with solution for larger dataset of size. On the other hand, same problem can be solved in 0.0003 seconds with efficient algorithm. Keeping-up with the earlier discussion, data mining algorithms are highly complex, exploration intensive and goal oriented. Specific choices made in the course of action have profound impact on the quality of results.

Spark using Java provides a convenient facility for using desired data structures. The org.apache.spark.api.java package contains spark RDD data structure and java.util package contains powerful subsystems called collections which are Java's standard framework of handling group of objects. Those packages has highly efficient implementations of various fundamental data structures such as RDD, arrays, arraylist, list, stack, `concurrent hash maps`, etc. Some of the data structures used in this thesis work summarized below.

RDD (Resilient Distributed Datasets) is distributed collections of objects which immutable and spread across cluster. Each RDD is spliced into many parts which will be computed on different nodes of the cluster. RDD provides two types of operations the first one is transformations and the second one is actions. The contraction of new RDD from previous one is done in transformations and the computation of a result on RDD is done in actions, and both will be to the driver program or saved to an external storage system (e.g., HDFS). [11]

- ✓ RDD[T] has objects of type T
- ✓ Collections of objects across a cluster with user controlled partitioning & storage
- ✓ It can be automatically rebuilt on failure
- ✓ RDDs track lineage info to rebuild lost data its fault tolerance behavior
- ✓ RDDs know their partitioning functions

Arrays: which the collection of the same type fixed-size sequential elements.

List: List is a sequence of elements where duplicates are allowed and ordering is not important. Elements in the list are accessed by their position and elements at specific position can be inserted or removed.

Array List: In Java, it supports dynamic arrays that are created with an initial size and it can automatically grow or shrink during run time. Average running times for insert, delete and search operations are $O(n)$, $O(n)$ and $O(1)$ respectively.

Stack – allows data to be added (a 'push' operation), and removed (a 'pop' operation). Dozens of stacks support a read ahead (a 'peek' operation) to reads data without removing it. A stack is a Last in first out (LIFO) queue, which means that the last data to be added will be the first data to be removed. Its class represents a last-in-first-out (LIFO) stack of objects. There is no item when it is first created.

Concurrent Hash Map: This class uses a hash table to implement the Map interface which allows concurrent modification of the Map from several threads without the need to block them. Each thread will only add data to the map, reading is less frequent. Since speed is important in our algorithm we used this data structure for Good performance as compared to other structure and thread safe, fail safe in iteration, locks the portion in locking mechanism.

Loops: for and while loop are used for iteration purpose
In addition to well established data structures, we used spark has RDD Operations such as actions and transformations.

Some of RDD Operations in Spark listed in Table 4.1 [11] with examples, these RDD transformations and actions are used in our algorithm. Now we give the signature of each operation used in our algorithm,

Function Name	Its goal	Example of their usage
map()	Apply a function to each element in the RDD and return an RDD of the result.	rdd.map(x => x + 1)
filter()	Return an RDD consisting of only elements that pass the condition passed to filter()	rdd.filter(x => x != 1)
collect()	Return all elements from the RDD.	rdd.collect()
count()	Returns number of elements in the RDD.	rdd.count()
first()	Return the first element from RDD	rdd.first()
mapToPair()	Returns key value pair	Rdd.mapToPair();
reduceByKey(func)	Combine values with the same key.	rdd.reduceByKey((x, y) => x + y)

Table 4.1 RDD transformation and action operations used in our algorithm

Transformations are lazy operations which define a new RDD without immediately computing it, whereas actions launch a computation. Transformations such as map and filter functions. Actions are operations such as count and first functions.

In addition to these operators, users can ask for an RDD to persist or cache. Furthermore, users can get operations such as group by, reduceByKey and sort automatically result in a hash or range partitioned RDD.

Map that return a new dataset which is distributed by passing every element through a function.

Filter which will return a new dataset by selecting those elements through function and the function will return true

MapTopair which will return a new key value pair distributed dataset formed by passing each element of the source through a function.

ReduceByKey which will return which will returns a dataset of (K, V) pairs where the values for each key are combined using the given reduce function, which must be of type (V, V) => V.

Collect returns an array of dataset elements at the driver program. It is practicable after a filter or the other operation returns a sufficient subset of the data.

Count returns the number of elements in the dataset.

First return the beginning element of the dataset which is similar to take (1).

SaveAsTextFile (*path*) which writes the elements of the dataset as a text file or set of text files in a directory in the local file system, HDFS or any other Hadoop-supported file system. To convert each element to a line of text in file spark will uses function toString ().

Cache to persist a dataset in memory across operations spark uses this technique

Each node stores any kind of partitions it which will compute memory and recycles it in other actions for that dataset which will make future actions faster. This technique an essential tool for iterative algorithms and fast interactive use. An RDD can be persisted using function persist and cache. When this technique is computed in action for first time it will be kept in memory on nodes. It is fault tolerant which means if any partition of an RDD is lost; it will automatically be recomputed using the transformations which created it originally.

4.1.4. TOOLS

We used Eclipse integrated development environment (IDE) as main tool for developing our algorithm in personal computer which we installed spark and hadoop plug-in for it and to remotely work on server we used Putty terminal to log into dbdmgmtr.polito.it server and FileZilla for file transfer from local to server, we also used ScatterPlotClustering for plotting clustering result on graph .

4.2. MAIN STEPS IN OUR ALGORITHM

A. Pre-processing

1. In this step, cleaned the dataset from missing values, noises, duplication by using separate class
    I. Missing values removed
    II. Noises removed by writing regular expression for Number format
2. Equal-Width Interval approach - to divide each attribute of the input into equal number of bins which convert our numerical dataset to categorical interval.
    I. Extract each attribute column
    II. Find min & max for it
    III. Compute width
    IV. Get intervals (unit)

B. Identification of subspaces

3. Dense units generation by using Association rule mining / bottom up approach/ from low-high dimensionality
    I. Map all transactions into key value pair (Key-Unit, Value - Count)
    II. If each transaction is between Intervals which we got in preprocessing step Then put Count equals 1 for each unit
    III. Reduce each unit by its value (Count all 1's in that unit)
    IV. Filter each unit greater than density Threshold
    V. Return Dense Unit

C. Identification of clusters

4. Depth First Search Steps- to get connected dense units (node) to does the following
    a) CreateAjacentMatrix from nodes,
    b) Check if two nodes (dense unit) are have the same & consecutive edge then set matrix to '1' else 0 (connected-1,Not-connected-0)
    c) Find Neighbors for each source node passed (step-ii) from matrix
    d) Then you will get connected dense units
    e) If there is unvisited node the work will continue from step-ii

D. Post-processing

5. Mapping original points with each dense connected unit get the final cluster.

4.3. PSEUDOCODE

For the Pseudocode part I am using this assumptions: let say A= {A_1, A_2, A_3, ……., A_d} be a set of bounded, totally ordered domains and $S = A_1 \times A_2 \times A3 \times \ldots\ldots\ldots\ldots\ldots \times A_d$ a *d*-dimensional numerical space. We will refer to $A_1, \ldots\ldots\ldots, A_d$ as the *dimensions* (attributes) of *S*.

The input consists of a set of *d*-dimensional points V = {v_1, v_2, v_3, ……., v_m} where *vi* = (v_{i1}, v_{i2}, v_{i3} ……., v_{id}).

4.3.1 Main Pseudocode

Input: NumberInterval, DensityThreshold, and {$v_1, v_2, \ldots, v_m$}

Output: {v1 1, ..., v_m clusterNumber}

1. Read {$v_1, v_2, \ldots, v_m$}
2. Read NumberInterval
3. Get Dimension
4. Split Records by space
5. Rows = Records.filter() header, MissingInput() and NoiseInput(); //**pre-processing step**
6. For kk← 0 to Dimension do
    a. Column = Rows.Map().get(kk); //Extract each column from record
    b. Intervals = **Discritize.EqualWidth** (Column , NumberInterval); //go to section 4.3.2
    c. for j← 0 to Intervals.length do
        //store intervals to Mutli-dimensional array
        a. MultiIntervals.addToInnerArray(kk, j, intervals[j]);
    d. end for
7. End for;
8. Read DensityThreshold;
9. DenseUnits = Rows.mapToPair(eachRow){ // **Dense Unit generation step**
    return Tuple2<String, Integer> **GenerateUnit**(eachRow); //go to section 4.3.3
    }. reduceByKey(){
        return **i1 + i2**;
    }.filter(){
        if (arg0._2 > DensityThreshold) Then
            return **true**;
    }

10. ConnectedUnits = DenseUnits.map(){ **//Finding connected units step**
    return graph.**DepthfirstSearch**(DenseUnit, clusterid)) //go to section 4.3.4
    }.filter(){
        if (arg0.size() > 0) Then

```
                    return true;
                return false;
            }.cache();
```

11. Cluster = Rows.map(){ //**Post-Processing** - Mapping original points
    return MappingOriginalPoints(keyinc);
    } .filter() {
        if (arg0 != null) Then
            return true;
        return false;
    }
12. Output Cluster contain {v_1 , ..., v_m clusterNumber }

4.3.2 Pseudocode for Discretization:

//equal width intervals techniques

1. Min = FindMin (column);
2. Max = FindMax (column);
3. Intervals = ComputeWidth(Min, Max, NumberInterval);

4.3.3 Pseudocode to generate unit:

1. for j← 0 to dimension do {
2. for k← 0 to MultiIntervals.get(j).size()-1 ; k = k + 2 do {

- if (rowKey.get(j) > =MultiIntervals.get(j).get(k) && rowKey.get(j)< MultiIntervals.get(j).get(k + 1)) then
  jj ← 0;
  list = new ConcurrentHashMap<Integer, String>();

  a. for z← j + 1 to dimension do {
  b. for r← 0 to MultiIntervals.get(z).size() – 1; r = r + 2 do {
  c. if (rowKey.get(z) >= MultiIntervals.get(z).get(r) && rowKey.get(z) < MultiIntervals.get(z).get(r + 1)) then
  int rr ← r + 1;
  list.put(jj, (MultiIntervals.get(z).get(r) + "_" + MultiIntervals.get(z).get(rr)));
  d. }
  e. jj ← jj + 1;
  f. }

- return new Tuple2<String, Integer>(MultiIntervals.get(j).get(k)+ "_"+ MultiIntervals.get(j).get(k + 1)+ ", "+ list.values().toString(), 1);

3. }}

4.3.4 Pseudocode to Depth first Search:

DepthfirstSearch (Unit,clusterid) //Unit is the Node where the search starts

```
Stack S = {};                          // start with an empty stack
Unit u , node.visited = true
push u;
While (S is not empty) do
    element = pop S;                   // pop from stack
     neighbours = this.findNeighbours(matrix, element);
     for  i← 0 to neighbours.size() do       // for each unvisited neighbor
          n = neighbours.get(i);
          if (!n.visited) then
                stack.add(n);                  // add node(dense unit) to final cluster
                n.visited = true;
                clusters.add(n.data + "," + clusterNumber);
          end if
 End while
```

END **DepthfirstSearch** ()

As our algorithm stated above with Pseudocode now, I will discuss Main steps in Pseudocode for clarity, the first step as any algorithm it will read input data from command line (in **step 1&2**) which are multidimensional numerical dataset and Number of intervals, density threshold. Then from this record we will get dimension of the record (**step3**) and split it by space (**step 4**) by transforming record from RDD String to RDD list of strings for ease.

After that pre processing stage of the algorithm will continue by eliminating missing points and data noises from the record and will separate the header of the input from the main data for computation.(As described in **step 5**).

In **step 6,** which is main preprocessing step in big data mining, discretize the RDD records by Using Equal-width binning Approach in order to get intervals, so I extracted each column from

the records RDD and get the minimum and maximum values of each column and compute width by passing each of this values with number of intervals (i.e. number of bins) for width computing function. This function will subtract minimum value from the maximum value and divide it to number of bins to get the width, After getting the width the function will take the minimum value as initial point for the resulting intervals (i.e. intervals[0]) and will add width and the previous value to each of next ones (intervals[1]…intervals[n]).
For example the width is 5 and the initial intervals [0] value 3 and then intervals [1] is 3+5 =8, intervals[2] is 8+5 = 13 …. etc. Then we will get an array of intervals for each RDD column. Then I will add these intervals which I already get from columns into one multidimensional array for further computations.

In **step 9**, we will generate dense units, so to generate dense unit first we should generate unit from the record and intervals which is kept in multidimensional array, to generate unit we first mapped the records into key value pairs, then passed each row for unit generation function, this function will generate unit if the passed row satisfies the association rule which is if each row element is between the specified interval value the function will return this interval as a unit and mark as one for that unit as a key, because this unit has one point, like wise we will do the same for all and then reduce this by key and will continue to compute the number of records in that unit (i.e. collecting the same units as one and adding their point). After that will prune the resulting key value pair RDD filtering only the units with selectivity of unit is greater than the frequency threshold (support) so finally we will get the **dense unit**.

The next step (i.e. **step 10**) is identifying clusters,
To identify clusters, we must connected dense units (step 10) we used depth first search, finding connected units in the graph, Before passing each dense unit as anode for the depth first function I added each dense units as anode for a graph class, so I have complete set of nodes, then this function will create adjacent matrix from set of nodes which was added before and find neighbors for passed (specified) dense unit from created matrix, then it will assign same cluster number for each connected dense unit

The Las step (i.e. **step 11&12**) for this algorithm is post processing which is mapping each original values to cluster (connected dense units), this is achieve by the mapping original points function which will take each original input row and check if this row is in between each cluster intervals, then if it is it will return the row will its cluster id.

CHAPTER FIVE

5 PERFORMANCE EXPERIMENTS

This chapter discusses the experiments on our algorithm. The goals of the experiments are to assess the efficiency and accuracy of our clustering method, particularly their accuracy and run time for large numbers of objects in high-dimensional spaces and their abilities for searching subspaces.

- The efficiency is determined in on the basis of running time with the following measures
  - Size of dataset.
  - Dimensionality of the Data space.
  - Dimensionality of clusters.
- The accuracy checks if our algorithm recovers known clusters in some subspaces of a high dimensional data space.

The experiments are carried out with both synthetic data and real data. We used the small sized synthetic data, contains data which is not obtained by direct measurement, to easily determine accuracy. The accuracy is defined as the proportion of the number of correctly clustered objects to the real number of objects in that cluster. The real data contains the data gathered from real situations, and is more complicated and more challenging for clustering than the synthetic data.

5.1 COMPUTER PLATFORM

All experiments are carried out on my PC (personal computer) which has 4GB DDR3 RAM with 64 bit windows operating system, Processor (Intel® Core™ i3-2370M Processor, 2.4 GHz, 3MB L3 Cache) and a machine located in the labinf, laboratory of department of control and computer engineering, Polytechnic University of Turin. Machine configuration bigdata@polito cluster with a set of 30 servers 15 GiB of RAM with linux operating system

.

.

5.2. PARAMETER CONFIGURATION

Our algorithm uses two Parameters the first one is input interval and the second one is density threshold which is discussed in previous chapters, so by tuning this parameters we will get the actual clusters embedded in subspace of high dimensional dataset.

5.3. EXPERIMENTS

Used the following synthetic and real numerical dataset for testing purpose from low to high multi dimensions and instances

5.3.1 SYNTHETIC DATA

In this experiment with synthetic data the result with numeric type is tested to check if our algorithm will give us the desired outcome.

So we used some categorical clustering data sets from University of Eastern Finland machine learning website [12]

"**Aggregation**"

The first one is aggregation dataset which contains two attribute with a numerical data type and has 788 instances (records)

"**Jain**"

The second one is Jain dataset which contains two attribute with a numerical data type and has 373 instances (records)

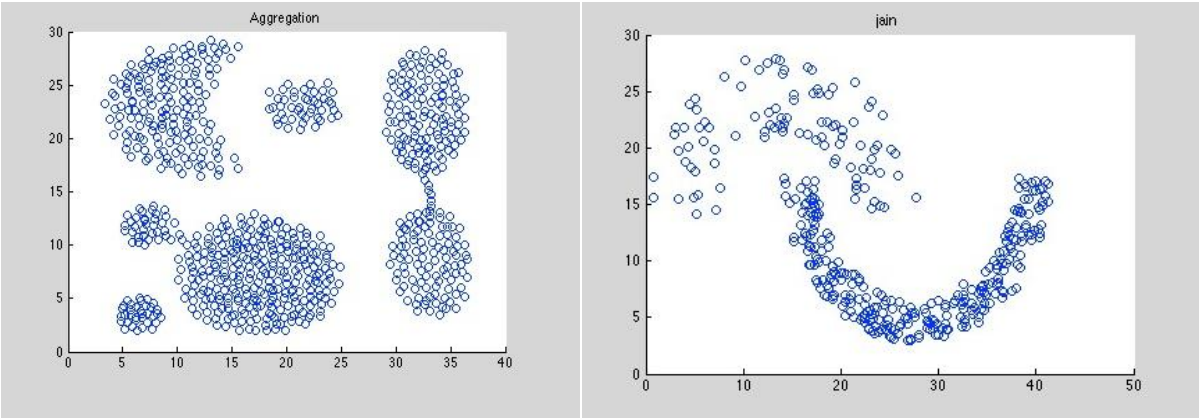


Figure 5.1 Aggregation and Jain datasets before testing in graph

Synthetic dataset results

After testing our algorithm I have got the following clustering results stated in table 5.2 for synthetic datasets described above with their input parameter Input interval and density threshold and number of clusters with each dataset number of dimension, instances.

Dataset Name	Number of dimension	Number of records	Input Interval	Density threshold	Number of cluster
Aggregation [13]	2	788	10	6	7
Jain [14]	2	373	10	4	2

Table 5.1 Synthetic data results

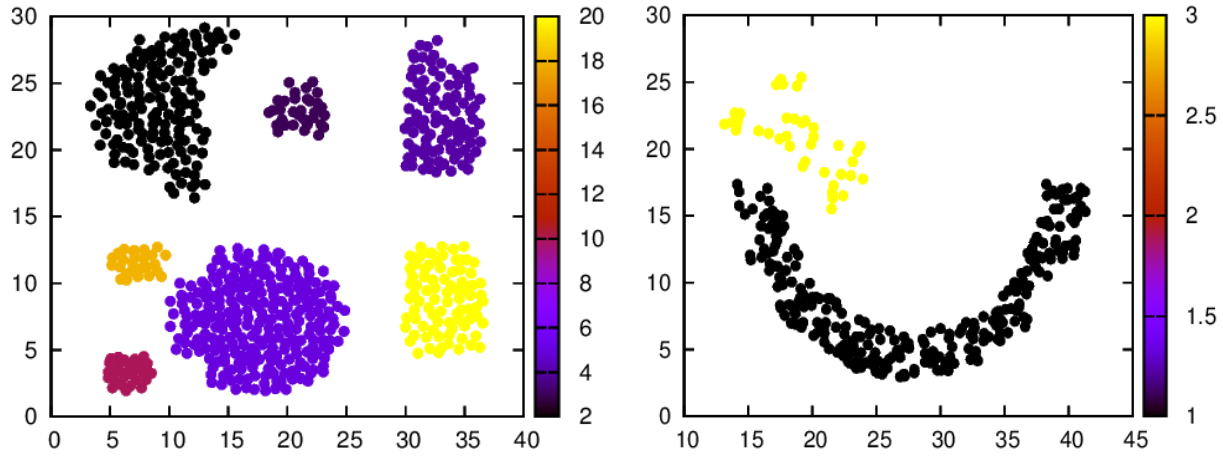


Figure 5.2 Aggregation and Jain datasets result after testing on graph

5.3.2 REAL DATA

In this experiment with real data the result of density-based clustering algorithm based on spark with numeric type is tested to check if our algorithm will give us the desired outcome.

Some of the real dataset which are found from UCI machine learning repository website [15] are used.

"**Iris**"

The data set "Iris"[1] was obtained from the UC Irvine Machine Learning Repository which includes 150 objects and 4 attributes. The attributes are: sepal length (a1), sepal width (a2), petal length (a3) and petal width (a4). The data set contains three clusters, where each cluster has 50 Objects and refers to a type of iris plant: Iris Setosa (S1), Iris Versicolour (S2) and Iris Virginica (S3).

"Seeds"

The data set "Seeds"[2] was obtained from the UC Irvine Machine Learning Repository which includes 210 objects and 7 attributes. The attributes are: area A, perimeter P, compactness C = 4*pi*A/P^2, length of kernel, width of kernel, asymmetry coefficient length of kernel groove which all of these parameters were real-valued continuous. The dataset contains kernels which belong to 3 different varieties of wheat namely Rosa, Kama and Canadian, 70 elements.

"Skin"

The data set "Skin"[3] was obtained from the UC Irvine Machine Learning Repository which includes 245057 objects and 4 attributes. B, G, R are attributes which contains x1, x2, and x3 features, and facial images values from various age groups (young, middle, and old), race groups (white, black, and asian), and genders obtained from FERET database and PAL. The dataset contains kernels which belong to 3 different varieties of wheat namely Rosa, Kama and Canadian, 70 elements). The data set comprises of 50859 skin samples and 194198 is non-skin samples

[1] https://archive.ics.uci.edu/ml/datasets/Iris
[2] https://archive.ics.uci.edu/ml/datasets/seeds
[3] https://archive.ics.uci.edu/ml/datasets/Skin+Segmentation

Real dataset results

After testing our algorithm we have got the following clustering result stated in table 5.4 for real datasets described above with their parameter Input interval and density threshold, number of clusters and each dataset number of dimension, instances.

Dataset Name	Number of dimension	Number of records	Input Interval	Density threshold	Number of cluster
Iris	4	150	3	4	3
Seed	7	211	3	7	3
Skin	3	245058	2	18550	2

Table 5.2 Real data results

5.4. EVALUATION

Finally in this section of this chapter we evaluated our algorithm from data mining requirement perspective which is stated in chapter one, if this algorithm has effectively addressed the problem statement described above by measuring its accuracy and efficiency of clustering result

5.4.2.1. Accuracy

In all the above experiments by using synthetic and real dataset shows, our clustering technique able to recover original clusters in subspace of original space.

5.4.2.2. Run Time

As shown in below table 5.3 and 5.4, the execution time of our algorithm is listed and when doing real data experiment for dataset “Skin”, we used a data set with a 3-dimensional space and 2 clusters in different subspaces. The number of objects was increased from 780 to >200000 in different tests, in our personal computer and in lab server the specification is described in the section 5.2; we have seen an increase in execution time for this dataset which runs in our PC because of different capacity with respect to server. So we say that when the number of object in dataset increases the execution time also increases. We will show relationship in figure 5.3 a and b.

Dataset Name	Number of dimension	Number of records	Input Interval	Density threshold	Number of cluster	Execution time
Aggregation [13]	2	788	10	6	7	11 sec
Jain [14]	2	373	10	4	2	10 sec
Iris	4	150	3	4	3	10 sec
Seed	7	211	3	7	3	10 sec
Skin	3	245058	2	18550	2	180 sec

Table 5.3 Execution time of our algorithm experimented on PC

Dataset Name	Number of dimension	Number of records	Input Interval	Density threshold	Number of cluster	Execution time
Aggregation [13]	2	788	10	6	7	11 sec
Jain [14]	2	373	10	4	2	10 sec
Iris	4	150	3	4	3	10 sec
Seed	7	211	3	7	3	10 sec
Skin	3	245058	2	18550	2	11 sec

Table 5.4 Execution time of our algorithm experimented on Labinf

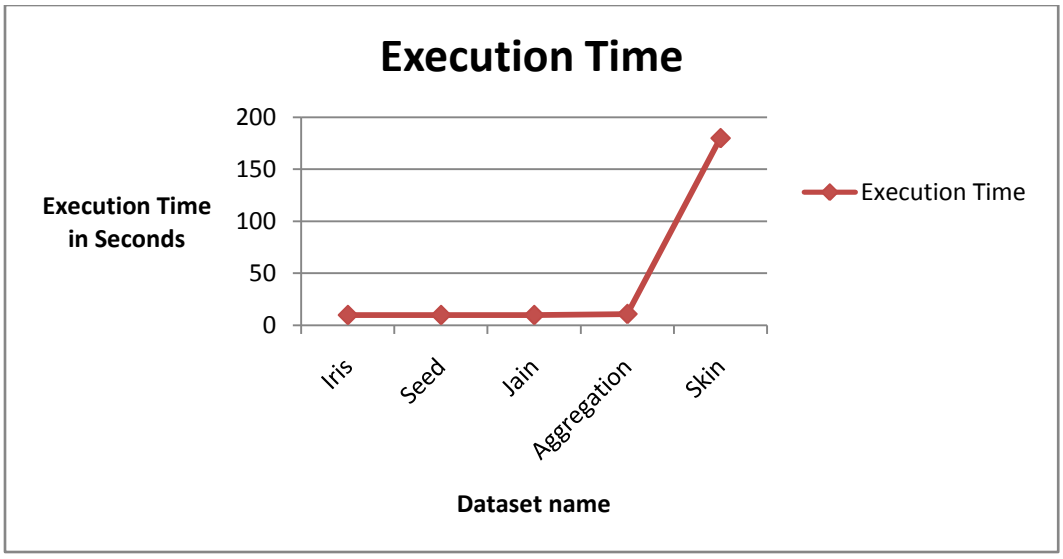


Figure 5.3a Run time for each dataset

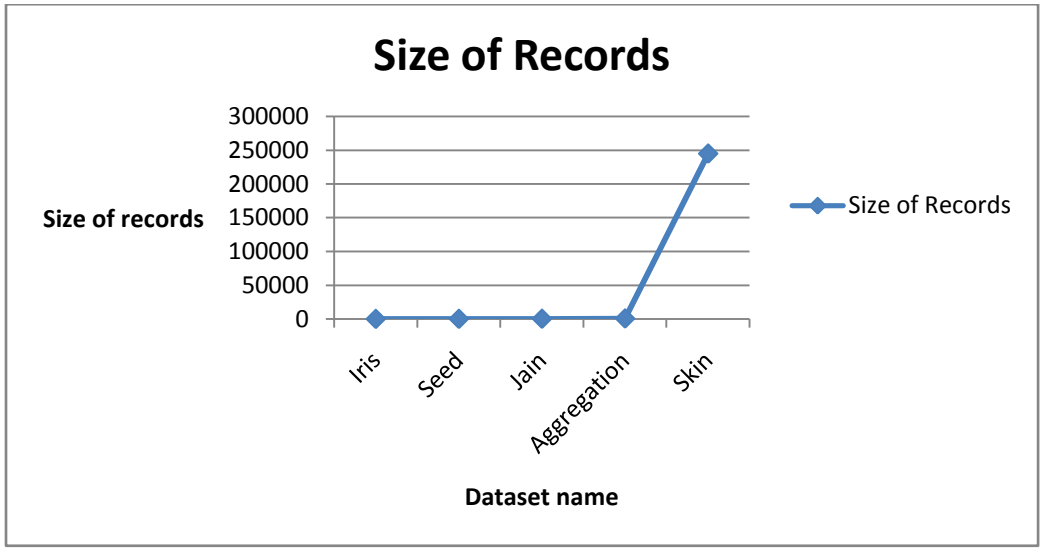


Figure 5.3b Size of records for each dataset

CHAPTER SIX

6 CONCLUSIONS AND FUTURE WORK

In this chapter, the conclusion and future work part the entire study is summarized and areas that require further research are also recommended.

6.1 CONCLUSIONS

In this thesis paper we have introduced subspace clustering algorithm by using association rule mining on top of big data frameworks and experimented the result. We have implemented this algorithm by first identifying subspace that contain clusters by association rule mining to identify the potential frequent cells and infer the content of this cell to find dense cells and connected these dense cells by depth first search technique to finally find the cluster embedded in subspace of high dimensional dataset on the top of spark and hadoop frameworks.

So we found this approach discovers subspaces of the highest dimensionality automatically such that high density clusters exist in those subspaces. And has efficient running time with respect to size and dimensionality of dataset, and it is insensitive to the order of input objects. It scales with the size of input linearly and has better scalability as the number of dimensions in the data se is increased. However, obtaining meaningful clustering results is dependent on proper tuning of the grid size and the density threshold.

This is particularly difficult because the grid size and density threshold are used across all combinations of dimensions in the data set. Thus, clustering results accuracy may be degraded at as a result of this method.

6.2 FUTURE WORK

For the future work different approach are suggested. We can use adaptive, data-driven strategy Instead of using fixed bins for each of the dimensions to determine the bins for each dimension dynamically based on data distribution statistics. Alternatively, instead of using a density threshold, we would use entropy as a measure of the quality of subspace clusters.

Another future study could be the adaption of the subspace clustering methods to various data types, such as time series or multimedia data.

ACRONYMS

CABSH – Clustering algorithms by using spark and hadoop

ASCHDD – Automatic subspace clustering of high dimensional data

HDD - High dimensional data

SC - Subspace clustering

DBSCAN - Density based clustering,

CLARANS- Clustering Algorithm based on Randomized Search

BIRCH - Balanced iterative reducing and clustering using hierarchy

CURE - Clustering Using REpresentatives

CLARA - Clustering Large Applications

CLIQUE - CLustering In QUEst

DFS - Depth-first search algorithm

BFS - Breadth-first search algorithm

RDD – Resilient Distributed Dataset

LIST OF TABLES

LIST OF FIGURES

BIBLIOGRAPHY

[1] Architectures And Data Analytics course materials available on Politechnico database and data mining group website. [Online]. http://dbdmg.polito.it/wordpress/teaching/big-data-architectures-and-data-analytics/#Materials-1

[2] Jiawei Han and Michelle Kamber, *Data Mining: Concepts and Techniques. Morgan Kaufmann*., 2001.

[3] Ehtesham Haque, Huan Liu Lance Parsons, "Subspace Clustering for High Dimensional Data: A Review," vol. Vol. 6, pp. pp. 90–105, 2004.

[4] Techopedia. [Online]. https://www.techopedia.com/definition/30306/association-rule-mining

[5] sites.google.com. [Online]. https://sites.google.com/site/dataclusteringalgorithms/home

[6] slidewiki.org. [Online]. http://slidewiki.org/slide/23976

[7] sites.google.com. [Online]. https://sites.google.com/site/dataclusteringalgorithms/hierarchical-clustering-algorithm

[8] Wikipedia. [Online]. https://en.wikipedia.org/wiki/DBSCAN

[9] Johannes Gehrke, Dimitrios Gunopulos, Prabhakar Raghavan Rakesh Agrawal, "Automatic Subspace Clustering of High Dimensional Data for Data," *AGRAWAL ET AL*, 2005.

[10] mathsisfun. [Online]. https://www.mathsisfun.com/data/data.html

[11] Andy Konwinski,Patrick Wendell & Matei Zaharia Holden Karau, *Learning Spark, LIGHTNING-FAST DATA ANALYSIS*. 1005 Gravenstein Highway North, Sebastopol, CA 95472., United States of America, 2015.

[12] Generate data. [Online]. http://www.generatedata.com/

[13] H. Mannila, and P. Tsapara A. Gionis, "Clustering aggregation," in *ACM Transactions on Knowledge Discovery from Data (TKDD)*., 2007, pp. 1(1): p. 1-30.

[14] A. Jain and M. Law, "Data clustering: A user's dilemma. Lecture Notes in Computer Science," in *Data clustering: A user's dilemma. Lecture Notes in Computer Science*., 2005., pp. 3776: p. 1-10..

[15] A. K. Jain and R. C. Dubes Data., *Algorithms for Clustering.* Prentice Hall :, 1988.

[16] Tom White, *Hadoop: The Definitive Guide,*. 1005 Gravenstein Highway North, Sebastopol, CA 95472, United States of America., 2015.

[17] Uri Laserson,Sean Owen & Josh Wills Sandy Ryza, *Advanced Analytics with Spark, PATTERNS FOR LEARNING FROM DATA AT SCALE*. 1005 Gravenstein Highway North, Sebastopol, CA 95472, United States of America, 2015.

[18] Guojun, Chaoqun Ma, and Jianhong Wu Gan, *Data Clustering: Theory, Algorithms, and Applications,ASA-SIAM Series on Statistics and Applied*. SIAM, Philadelphia, ASA, Alexandria, VA., United States of America, 2007.

[19] Guojun, Chaoqun Ma, and Jianhong Gan, *Data Clustering: Theory, Algorithms, and Applications*. Alexandria, VA: SIAM, Philadelphia, 2007.

[20] Architectures And Data Analytics course materials available on Politechnico database and data mining group website. [Online]. http://dbdmg.polito.it/wordpress/teaching/big-data-architectures-and-data-analytics/#Materials-1

[21] wikipedia. [Online]. https://en.wikipedia.org/wiki/DBSCAN

[22] unsupervised_binning at saedsayad. [Online]. http://www.saedsayad.com/unsupervised_binning.htm

[23] Pasi Fr\"anti et al. (2015) Clustering datasets. [Online]. http://cs.uef.fi/sipu/datasets/