INCSUBCLU
Incremental Density-Connected Subspace Clustering for High-Dimensional Data

Aditya Rajesh Patil, Bhasker Goel, Kartikeya Saxena, V Anirudh
{aditya18, bhasker18, kartikey18a, anirudh18}@iitg.ac.in
180101004, 180101015, 180101034, 180101084
dataHacks - Indian Institute Of Technology Guwhati

ABSTRACT

So many domains today like molecular biology or geography produce a tremendous amount of data, which in turn becomes very difficult to manage without the help of efficient clustering algorithms. Clustering is one the primary data mining tasks. Where traditional clustering algorithms fail on high-dimensional, inherently sparse data spaces, Subspace clustering algorithms come in to detect meaningful clusters in the subspaces of the original data space. In a lot of environments, when new data is added, the entire batch algorithm is run all over again. Due to very large size of Datasets, it is highly desirable that these updates are made incrementally and inplace. In this paper, we present incremental SUBCLU, an algorithm that can make insertions and deletions in the present clustering incrementally. Static DBSCAN in the original SUBCLU algorithm is replaced with a modified version of Incremental DBSCAN. Performance evaluation is done on 4 datasets of different size and dimensionality using the Silhouette coefficient. Incremental SUBCLU is found to use approximately the same amount of memory as the Static version while providing a significant speed-up.

1 INTRODUCTION

Modern methods in several application domains such as molecular biology, astronomy, geography, etc. produce a tremendous amount of data. Since all this data can no longer be managed without the help of automated analysis tools, there is an ever increasing need for efficient and effective data mining methods to make use of the information contained implicitly in that data. One of the primary data mining tasks is clustering which is intended to help a user discovering and understanding the natural structure or grouping in a data set. In particular, clustering is the task of partitioning objects of a data set into distinct groups (clusters) such that two objects from one cluster are similar to each other, whereas two objects from distinct clusters are not.

Many clustering algorithms deal with the curse of dimensionality. To cope with this, dimensionality reduction techniques or projected clustering algorithms are used. But in doing so, the clustering loses its interpretability, and is able to present clusters in only one subspace of the original dataspace. The task of subspace clustering was introduced to overcome these problems. Subspace clustering is the task of automatically detecting clusters in subspaces of the original feature space. SUBCLU is an effective answer to the problem of subspace clustering.

The next part of the algorithm is particularly relevant in this age of massive amounts of data in Data Warehouse environments. Typically, a data warehouse is not updated immediately when insertions and deletions on the operational databases occur. Updates are collected and applied to the data ware- house periodically in a batch mode, e.g., each night. Then, all patterns derived from the warehouse by data mining algorithms have to be updated as well. This update must be efficient enough to be finished when the warehouse has to be available for users again. Due to the very large size of the databases, it is highly desirable to perform these updates incrementally so as to consider only the old clusters and the objects inserted or deleted during the day, instead of applying the clustering algorithm to the (very large) updated database.

In this paper, we present an incremental subspace clustering algorithm. Our algorithm is based on an optimised version of Incremental DBSCAN and SUBCLU. Due to the density-based nature of DBSCAN, the insertion or deletion of an object affects the current clustering only in the neighborhood of this object in all its subspaces. We also provide a proof, as to why replacing the static version of DBSCAN by the incremental version does not affect the working of the apriori algorithm underlying SUBCLU. We demonstrate the high speed-up with only a reasonably small drop in accuracy of the incremental clustering on the Iris Flower Dataset along with 3 other datasets mentioned in latter sections.

The rest of this paper is organized as follows. We discuss the underlying static clustering algorithm in section 2. In section 3, we briefly talk about some related Incremental algorithms that inspired our work. The approach we've taken in designing our algorithm is presented in section 4 and the datasets used are listed down in section 5. The results obtained are presented in section 6. Section 7 concludes with a summary and the learning values from the project and finally Section 8 talks about some directions for future research.

2 BASE STATIC ALGORITHM

SUBCLU (density-connected Subspace Clustering) is an efficient and effective algorithm to cluster high dimensional data in all subspaces. SUBCLU takes advantage of the monotonicity of the DBSCAN algorithm and efficiently prunes subspaces while building clustering in a bottom up way. In recent times due to availability of large amounts of data, finding meaningful clusters on the data is

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

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

becoming more challenging. SUBCLU has been developed to tackle clustering high-dimensional data or the curse of dimensionality. SUBCLU takes min-points and epsilon as input and returns clusters in all subspaces. Clusters obtained for each subspace is same as running DBSCAN algorithm on each subspace separately.

2.1 Definitions

Let DB be the database.

(1) ϵ-Neighbourhood of point o:-
Let $\epsilon \in \mathbb{R}$ and o ∈ DB. ϵ neighbourhood of o in subspace S is denoted by $N_\epsilon^S(o)$ where
$N_\epsilon^S(o) = \{x \in DB | dist(\pi_S(o), dist(\pi_S(x)) < \epsilon\}$
$\pi_S(o)$ is the projection of point o onto subspace S
(2) Core point :-
Let$\epsilon \in \mathbb{R}$ and $m \in \mathbb{N}$.Point $o \in DB$ is considered core point in subspace S if its epsilon neighbourhood has a size of at least m. $CORE_{\epsilon,m}^S(o) \iff |N_\epsilon^S(o)| > m$
(3) DIRECTLY DENSITY REACHABLE:-
Let$\epsilon \in \mathbb{R}$, $m \in \mathbb{N}$ and $p, q \in DB$. p is directly reachable from q in subspace S if q is a core point and $p \in N_\epsilon^S(q)$
$DIRREACH_{\epsilon,m}^S(q,p) \iff CORE_{\epsilon,m}^S(q) \cap \wedge p \in N_\epsilon^S(q)$
(4) DENSITY-RECHABILITY:-
Let$\epsilon \in \mathbb{R}$, $m \in \mathbb{N}$ and $p, q \in DB$. p is directly reachable from q in subspace S,if there is a chain of objects $p_1, ..., p_n, p_1 = q, p_n = p$ such that p_{i+1} is directly density-reachable from p_i.
$REACH_{\epsilon,m}^S(q,p) \iff \exists p_1, ..., p_n \in DB : p_1 = q \wedge p_n = p \wedge \forall i \in \{1....n-1\} : DIRREACH_{\epsilon,m}^S(p_i, p_{i+1})$
(5) DENSITY-CONNECTIVITY:-
Let$\epsilon \in \mathbb{R}$, $m \in \mathbb{N}$ and $p, q \in DB$. p and q are said have denstiy connectivity if there exists a point $o \in DB$ such that both p and q are density reachable from o.
$CONNECT_{\epsilon,m}^S(q,p) \iff \exists o \in DB : REACH_{\epsilon,m}^S(o,p) \wedge REACH_{\epsilon,m}^S(o,q)$
(6) DENSITY-CONNECTED SET:-
Let$\epsilon \in \mathbb{R}$, $m \in \mathbb{N}$ and $C \subseteq DB$ be a non-empty set. Then C is density connected set in subspace S if all points in C are density connected in S.
$ConSet_{\epsilon,m}^S(C) \iff \forall o, q \in C : CONNECT_{\epsilon,m}^S(q,0)$
(7) Cluster:-
Let $C \subseteq DB$ be a non-empty set. Then C is a cluster in subspace S if C is a maximal density connected set.

2.2 Pruning and SpeedUp

Since SUBCLU aims to produce clustering for all possible subspaces, it might have to visit all 2^d subspaces. This will lead to large computational time. To speed up the process, pruning is used to eliminate subspaces which will produce no clusters. Further a speed up is also used to compute clusters in higher dimensions. The following two properties of clusters are observed form the definition of cluster:-

(1) A cluster in a lower dimension subspace might not be a cluster in it's higher dimension subspace. It might lose points from the cluster of lower dimension, but will never gain other points. As moving to higher dimension the distance between two points will increase, points will be lost from each others ϵ-neighbourhoods.
(2) If two points are not in a cluster in a particular dimension, then they will not be in the same cluster in any of it's higher dimensions. Consider two subspaces S and T such that $T \subset S$. If two points are not in same cluster in T, they will not be in the same cluster in S.

The above two properties of clusters help in pruning and speeding up the algorithm.

(1) Consider a k dimension space S. Let T be one of it's k-1 dimension subspace. If T has no clusters present in it, S will also have no clusters. If S has clusters, then two points are in a cluster in S but were not a part of a cluster in T , violating the property of clusters.
(2) Consider a k dimension space S. Let T be one of it's k-1 dimension subspace. Running DBSCAN separately on each of T's clusters will produce clusters of S. This holds true as new points are not added to a cluster on moving up the dimension.

2.3 Algorithm

SUBCLU algorithm starts from 1st Dimension and builds up in apriori fashion.

(1) C^S denotes the set of all clusters of subspce S which are obtained on running DBSCAN algorithm in S.
(2) S_k denotes the set of all k-dimensional subspaces containing at least one cluster.
(3) C_k denotes the set of sets of all clusters in k-dimensional subspaces.

SUBCLU algorithm begins with performing DBSCAN on each of 1-D subspace. 1-D subspace with at least one cluster is added to S_1 and their clusters are added to C_1.
Now the Algorithm performs iterations till end condition is met. In $k-1^{th}$ step, candidate subspaces of dimension k are generated by calling GenerateSubspaces on S^{k-1}. Lets denote this by $CandS_k$.Every $k-1$ size subspace of objects in $CandS^k$ will have at least one cluster. All the k dimension spaces which have at least one $k-1$ dimensional subspace with no clusters will be pruned by GenerateSubspace [2].
For each k dimension space in $CandS_k$ a $k-1$ dimension bestSubspace is obtained. $k-1$ dimension subspace with least number of points present in all clusters is the bestSubspace of a k dimension space. To obtain the clustering in the k dimension space, DBSCAN is run on each cluster of the bestSubspace. If a candidate in $CandS_k$ has at least one cluster, it is added to S_k and clusters are added to C_k. This process is reapeated until C_k is empty i.e none of the k dimension spaces have at least one cluster.

3 RELATED INCREMENTAL ALGORITHMS

There is no existing incremental version of SUBCLU but there is an incremental version of DBSCAN which is a major subroutine in SUBCLU. The algorithm is as follows:

(1) **Insertion**
First a set UpdSeed is computed after insertion and deletion of points. UpdSeed is a set of core points which are directly

Data: Real ϵ, Integer m, Database DB
Result: clusters in all sub-spaces
$S_1 = \phi$;
$C_1 = \phi$;
for *each 1-D subspace* **do**
    Find clustering in 1-D sub-spaces using DBSCAN;
    if *clusters are present* **then**
        add subspace to S_1;
        add clusters to C_1 ;
    end
end
for $k = 2$; $k <$ *dimension*; $k++$ **do**
    `/* generate k dimension subspaces from S */`
    $CandS_k$ = GenerateSubspaces(S_{k-1});
    for *each cadidate in* $CandS_k$ **do**
        Find the bestSubspace of the candidate;
        for *each cluster in bestSubspace* **do**
            run DBSCAN on cluster;
            Add clusters generated to $C^{candidate}$
        end
        if $C^{candidate}$ *is not empty* **then**
            Add candidate to S_k Add $C^{candidate}$ to C_k
        end
    end
    if C_k *is empty* **then**
        break;
    end
end

Algorithm 1: SUBCLU

Data: S_{k-1}
Result: pruned k dimension subspaces
$CandS_k = \phi$;
for $s_1 \in S_{k-1}$ **do**
    for $s_2 \in S_{k-1} \wedge s_2 != s_1$ **do**
        if *first k-2 attributes of* s_1, s_2 *are same* **then**
            add $s_1 \cup s_2$ to $CandS_k$;
        end
    end
end
for *each* $Cand \in CandS_k$ **do**
    for *each* $s \subset Cand \wedge |s| = k-1$ **do**
        if $s \notin S_{k-1}$ **then**
            delete $Cand$ from $CandS_k$;
            break;
        end
    end
end
return $CandS_k$;

Algorithm 2: Generate Subspaces subroutine

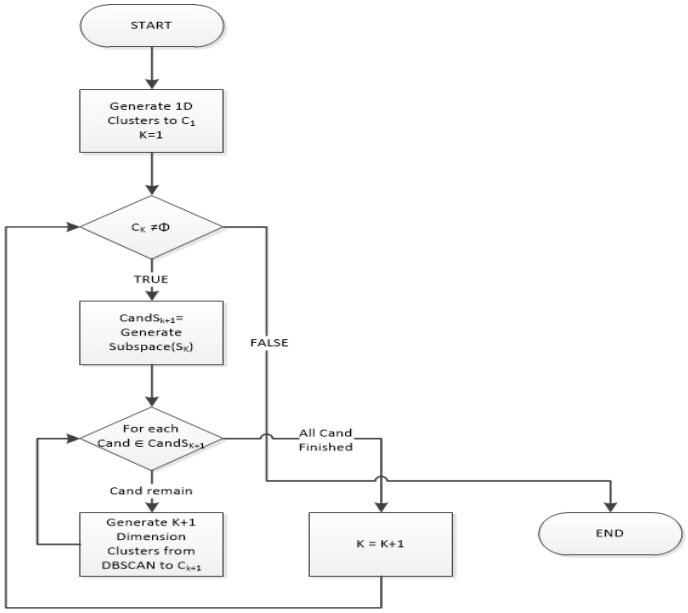

Figure 1: SUBCLU Algorithm

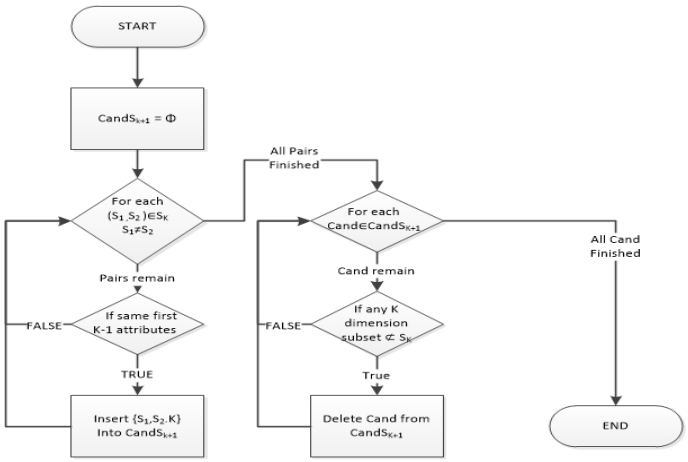

Figure 2: GenerateSubspace Subroutine

reachable from new core points formed after insertion of points.

- If UpdSeed is empty, then the inserted point is noise and no change is needed.
- If points in UpdSeed do not belong to a cluster then a new cluster is created along with p
- If the points in UpdSeed are from same cluster then inserted point is absorbed into that cluster along with some noise possibly.
- If the points in UpdSeed are from different clusters then these clusters are merged to form a single cluster.

(2) **Deletion**
First a set UpdSeed is computed after insertion and deletion of points. UpdSeed is a set of core points which are directly reachable from points which were core points before deletion but do not remain core points after deletion.

- If UpdSeed is empty, then the cluster corresponding to p is removed.
- If all points in UpdSeed are directly reachable from each other then few points in UpdSeed reduce to noise after removal of the deleted point.
- If all the points are not directly density reachable then there is a chance for potential split.

Remarks: From the above two ideas in the incremental dbscan we can observe that only a small section of points are being affected on inserting a new point i.e points which are in the epsilon neighbourhood of inserted/deleted point and their directly reachable points. Computation on these small numbers of points results in a speedup of the algorithm compared to running complete dbscan again without much loss of accuracy.

4 OUR APPROACH

4.1 Intuition behind our approach and related theorems

Since the algorithm is double-stepped i.e DBSCAN over apriori, we saw two ways to move forward. One being incremental apriori with vanilla DBSCAN. The second being Incremental DBSCANs connected by an apriori algorithm. Since, DBSCAN is the primary algorithm on which the clustering is based on, and also because this was the bottleneck of the running time of the algorithm, we chose to move along the second path.
To avoid running incremental DBSCAN on all the subspaces after every insertion and deletion, we used the following subspace pruning technique.
A space may change its cluster structure only if there is change in cluster structures in all its subspaces. Because if there is a subspace which has not changed, we can run DBSCAN with this subspace as the bestSubspace resulting in exactly the same clusters as before.

Remark. To minimize the cost of the runs of DBSCAN in cand, we choose that subspace bestSubspace $\subset$ cand from S_k in which a minimum number of objects are in the cluster. This heuristics only minimize the number of range queries necessary during the runs of DBSCAN in S and has no other effect on the results of the algorithm.

THEOREM 4.1. *If there is a change in a (k + 1)-D space then there is a change in all its k-D subspaces*

Proof: Contrapositive of the above statement is as follows:

Contrapositive. *If there is no change in at least one of the k-D subspaces then the (k + 1)-D space does not change.*

Proof by contradiction
Assume that a (k + 1)-D space S changed its cluster structure due to insertion/deletion but there exists a k-D subspace T with no change in its cluster structure. Let C be the cluster structure of S before change and C' after the change. C can be obtained by running DBSCAN on the clusters of T (See Remark). This property holds true even after deletion/insertion of points. But we already have C' as the cluster structure of S which is different from C. This is not possible. Hence contradiction.
=><=

COROLLARY 4.2. *If there is a change in a space then there is a change in all its subspaces.*

Incremental SUBCLU uses the above theorem to prune subspaces.

4.2 Our approach to incremental DBSCAN

We plan to restrict the changes to the new point's neighbourhood only. We believe that in due time with enough changes the algorithm will stabilize to give quite accurate results with a significant speed up.
For each point we maintain the count of number of points in its epsilon neighbourhood in each subspace.

(1) **Point Insertion**
For each subspace, get the points in the neighbourhood of the newly inserted point and increment their count by 1. Then, set the count for the inserted point. Now the following cases will arise after the insertion:

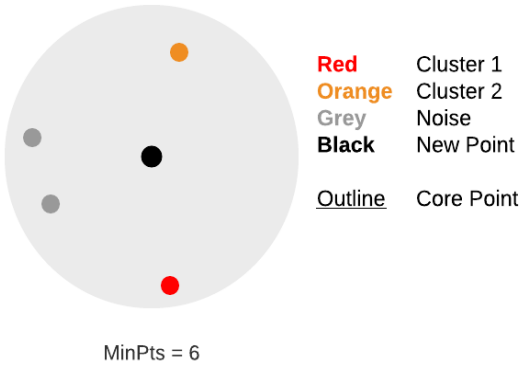

Figure 3: insertion 1

- If neighbourhood has no core point then label the inserted point as noise.
- Else, some points in neighbourhood are core points:

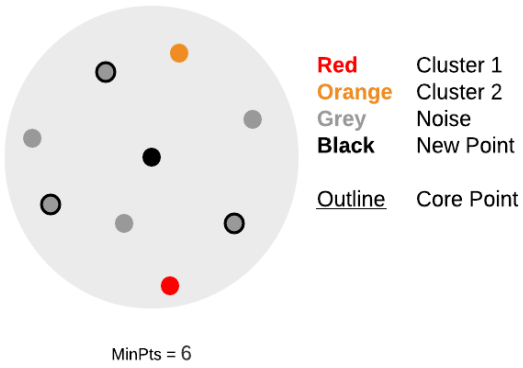

Figure 4: insertion 2

– **If those core points were all noise before:** Form a new cluster with all the noise points in the neighbourhood of the new point and extend the neighbourhood of the core points.

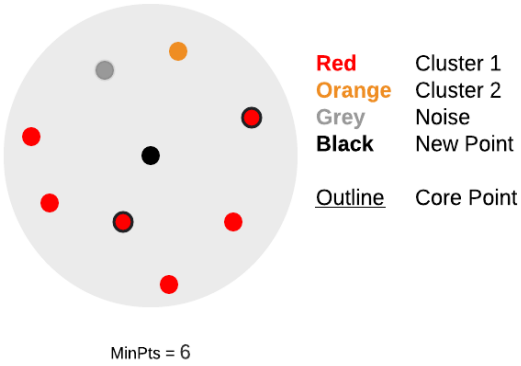

Figure 5: insertion 3

– **If some of those core points belonged to a single cluster and others were noise before:** Add the newly inserted point to that cluster and if it is a core point, add all the noise points in its neighbourhood to that cluster too.

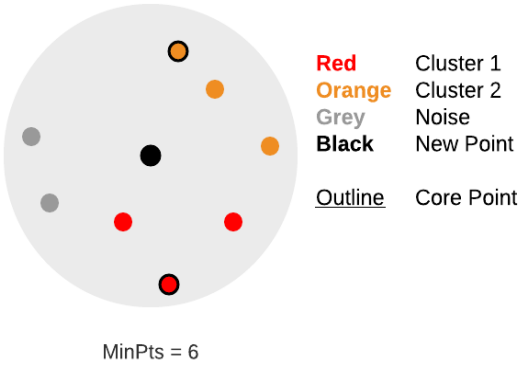

Figure 6: insertion 4

– **If those core points belonged to different clusters:** If the newly inserted point is a core point, merge all those clusters into a single cluster and add the newly inserted point to that cluster, otherwise just add the newly inserted point arbitrarily to any of those clusters with no merging.

(2) **Point Deletion**
For each subspace, get the points in the neighbourhood of the deleted point and decrement their count by 1. Then remove the deleted point from the subspace. Now the following cases will arise after the deletion:

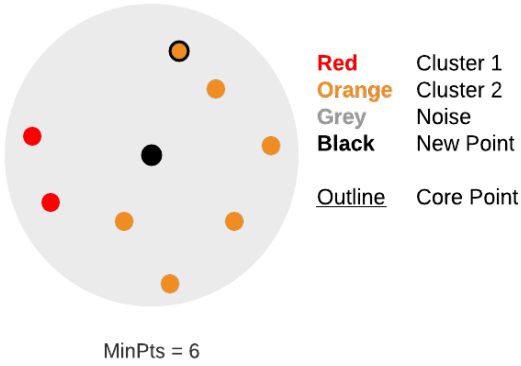

Figure 7: deletion 1

- **If the deleted point was a core point:** Temporarily, mark all points in the deleted point's neighbourhood as noise. Then, assign these points to the cluster of the nearest core point (with distance < epsilon) if such core point exists. If none does, then the point stays as noise.

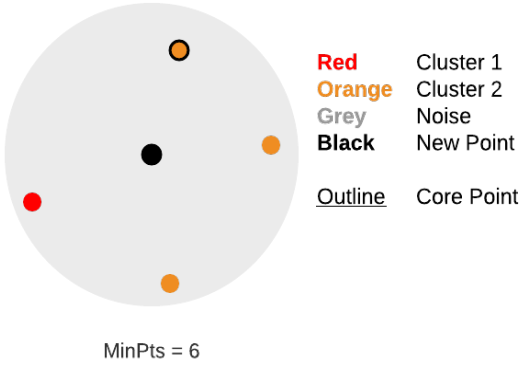

Figure 8: deletion 2

- **Else, the deleted point was not a core point:**
  – If none of the points in the neighbourhood loose their core property due to deletion, then simply remove the point to be deleted and do no perform any changes.
  – If points in the neighbourhood loose their core property, then run part 1(If the deleted point was a core point) considering each of these points to be deleted.
- **Split:** A split count is maintained for each cluster which is incremented every time a deleted point creates the possibility of a potential split (as described by incremental DBSCAN). When this count reaches a particular threshold (specified as a parameter), we run a static-DBSCAN on this cluster so that all the missed splits are accounted for.

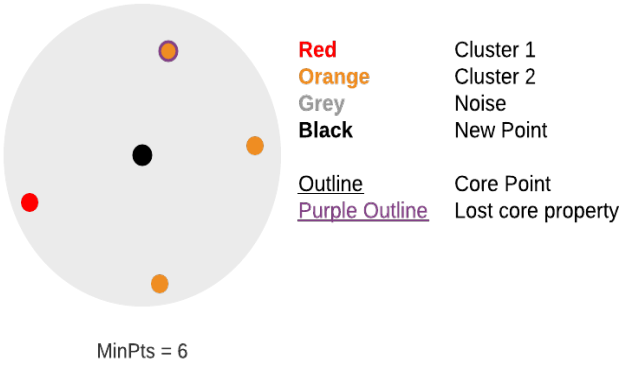

Figure 9: deletion 3

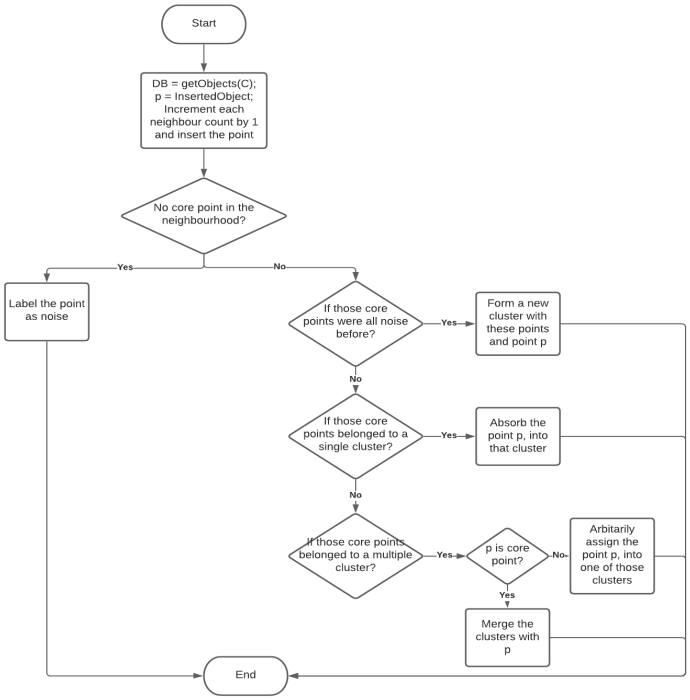

Figure 10: Incremental DBSCAN insertion

5 DATASETS

(1) **Seeds Dataset**

No. of Attributes - 7
Data points - Initial: 168
Update: Insert: 42 , Delete: 24
Data Collection Year - 2012
Real Valued.
Measurements of geometrical properties of kernels belonging to three different varieties of wheat. A soft X-ray technique and GRAINS package were used to construct all seven, real-valued attributes.

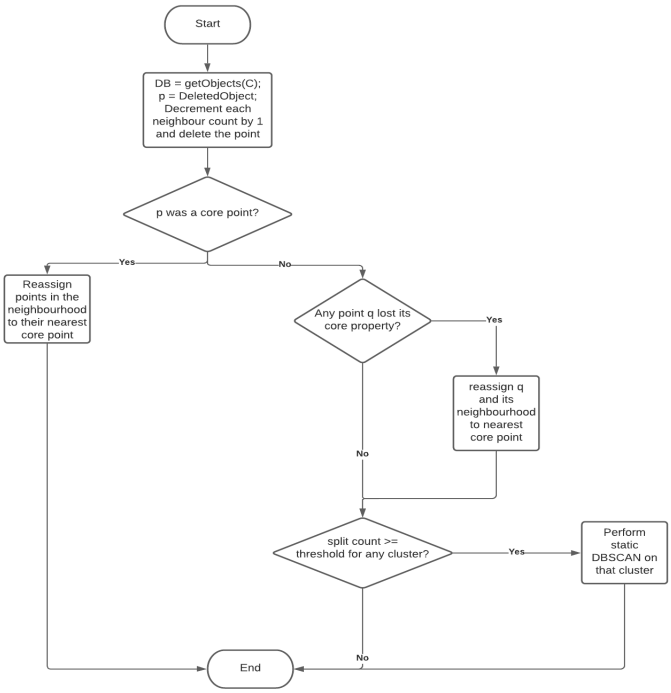

Figure 11: Incremental DBSCAN deletion

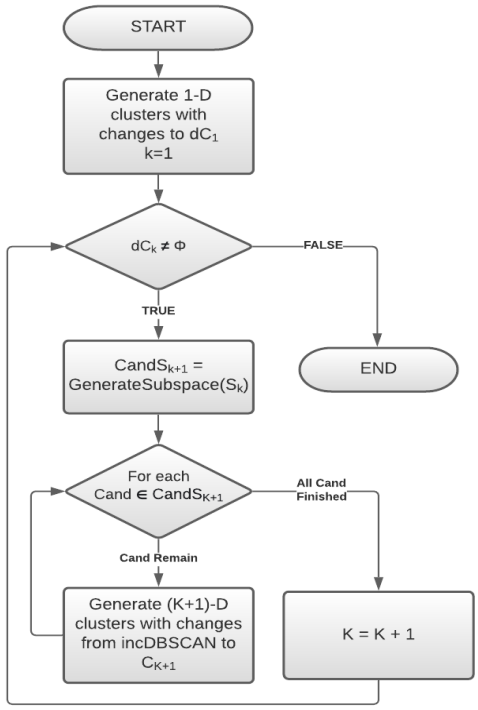

Figure 12: :Incremental SUBCLU

Data: Real ϵ, Integer m, NewObject p
Result: Updated clusters in all sub-spaces
/* STEP 1: Update all 1-D clusters */
$dS_1 = \Phi$; // 1-D subspaces with changes in them
for *each* $a_i \in A$ **do**
  Clusters Ca_i = getClusters(a_i) ; // original clusters found by DBSCAN
  Bool change = IncDBSCAN(Ca_i,{a_i},ϵ,m,p);
  if *change* **then**
    $dS_1 = dS_1 \cup \{a_i\}$;
  end
end
/* STEP 2: Update (k + 1)-D clusters from k-D clusters */
k = 1;
while dS_k!= Φ **do**
  /* STEP 2.1: Generate (k+1)-D candidate subspaces */
  dS_{k+1} = GenerateCandidateSubspaces(dS_k);
  /* STEP 2.2: Test and Update (k+1)-D subspaces */
  for *each cand* $\in dS_{k+1}$ **do**
    Clusters C_{cand} = getClusters(cand);
    Bool change = IncDBSCAN(C_{cand},cand,ϵ,m,p);
    if *change == false* **then**
      $dS_{k+1} = dS_{k+1}$-{cand};
    end
  end
  k = k+1;
end

Algorithm 3: Incremental SUBCLU

(2) **User Knowledge Modelling Dataset**
No. of Attributes - 5
Data points - Initial: 328
Update: Insert: 75 , Delete: 70
Data Collection Year - 2013
Integer Valued
It is the real dataset about the students' knowledge status about the subject of Electrical DC Machines. The dataset had been obtained from Ph.D. Thesis of students

(3) **Iris Dataset**
No. of Attributes - 5
Data points - Initial: 130
Update: Insert: 20 , Delete: 30
Data Collection Year - 1936
Real Valued
The dataset contains a set of 150 records under five attributes - sepal length, sepal width, petal length, petal width and species to distinguish between three species of Iris - Iris setosa, Iris virginica and Iris versicolor.

(4) **Mouse (ELKI)**
No. of Attributes - 2
Data points - Initial: 400
Update: Insert: 100 , Delete: 36
Real Valued
The Dataset has been used as a toy dataset for comparing K-means and EM clustering in the ELKI Library. It has 4 clusters, namely head, ear-left, ear-right and noise.

6 EXPERIMENTAL RESULTS

System Specs
Processor: Intel® Core™ i5-8250U CPU @ 1.60GHz × 8
RAM: 8GB
Cores: 8
Swap Space: 2GB

6.1 Testing Measures

(1) Purity:-
Having generated clusters for both static run as well as incremental run of the algorithm, accuracy should tell how well incremental run performs compared to the static run. For this, for each cluster in the static run, we look for the most similar cluster in the clusters formed by the incremental run. The similarity measure is calculated by counting the number of points common to both the clusters. The number of common points are classified as *Correct predictions* while the rest are classified as *Incorrect predictions*. In case two or more different clusters formed by static run has the same most similar cluster, then the static cluster with most number of common points is assigned that cluster while the rest are assigned to the next most similar cluster. If static clusters have same number of common points too, ties are broken by the assigning incremental cluster to the static cluster with the least number of total points. If no more incremental clusters can be assigned to static clusters all its points are considered *Incorrect predictions*.

(2) Silhouette Coefficient:-
The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.[18]
For data point $i \epsilon C_i$ let

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \epsilon C_i, i \neq j} d(i, j) \tag{1}$$

where d(i, j) is the distance between data points i and j. a(i) is a measure of how ell i is assigned to its cluster.

For data point $i \epsilon C_i$ let

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \epsilon C_k} d(i, j) \tag{2}$$

b(i) is a measure of how poorly the data point is matched to its neighbouring clusters.

Silhoutte value s(i) is defined as

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}, \text{if } |C_i| > 1 \quad (3)$$

$$s(i) = 0, \text{if } |C_i| = 1 \quad (4)$$

The Silhouette Coefficient is the maximum value of the mean Silhoutte value per k cluster over all data of the entire dataset.

6.2 Test Results

First and the third column indicate the memory in KB consumed by static and incremental algorithm respectively. Second and fourth column indicate the time in seconds consumed by static and incremental algorithm respectively.

Table 1: Memory and Time

Data	St(KB)	St(S)	Inc(KB)	Inc(S)
test.csv	5832	236.78	6016	10
test_insert.csv	6288	141.19	5784	5.37
Iris_update.csv	4324	4.58	4308	0.57
seeds_update.csv	8384	118.66	7648	8.97
Mouse_update.csv	4272	28.6	4528	1.03

First and the second columns indicate the silhouette constant of the height dimension subspace obtained from static and incremental algorithm respectively. Third column indicated the purity of clusters obtained from incremental algorithm. MinPoints and Epsilon are the user inputs.

Table 2: Accuracy

Data	Static	Incr	Purity	MinPoints	Epslion
test.csv	1	1	0.78	15	0.25
test_insert.csv	0.34	0.32	0.85	15	0.25
Iris_update.csv	0.46	0.46	0.97	4	0.4
seeds_update.csv	0.47	0.36	0.96	4	0.7
Mouse_update.csv	0.57	0.52	0.90	12	0.05

6.3 Conclusion

(1) Time : Incremental SUBCLU is consistently performing better in time compared to static SUBLU. Speed up is observed between 10 to 30 times of static algorithm.The reason being, for both insert and delete, incremental algorithm is mostly updating the points in epsilon neighbourhood.
(2) Memory : Incremental SUBLU and static SUBLU are performing very similar in terms of memory consumption. Storin subspace clustering consumes maximum amount of memory. Since both the algorithms on reaching the end obtain similar subspace clusterings, they have very similar memory.
(3) Accuracy : Incremental SUBLU is not completely accurate. Since the update is mostly performed for the points in epsilon neighbourhood, points beyond epsilon neighbourhood are not always updated. This leads to noise points being considered as cluster points and vice versa, leading to a loss of accuracy. In the test results, least observed accuracy is 78% and the rest of the cases it is more than 85%. Even in silhouette coefficient, incremental run is performing poorly compared to static run. In the case of seeds database the decrease has been 0.11 and in the rest of the databases the decrease has been less than 0.06.The speed up observed is worth the loss of accuracy.
(4) In the accuracy vs number of insertions graph, accuracy was measured for addition of every 15 points. With increase in the number of points added, decrease in accuracy has been observed. But a decrease in the rate of accuracy fall (slope of line) is also observed. Hence on a longer run the accuracy is expected to saturate.

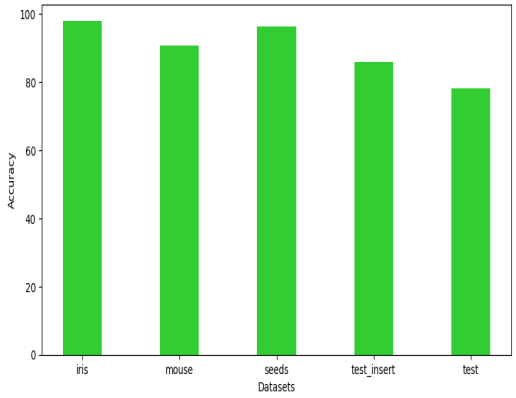

Figure 13: Accuracy bar graph

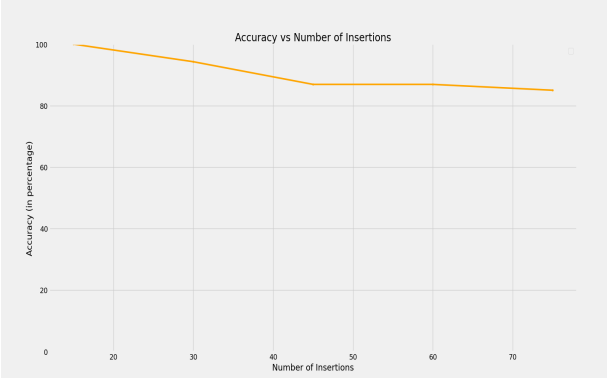

Figure 14: test_insert

7 LESSONS LEARNT FROM THIS PROJECT

The first and foremost thing we would like to mention is finding, reading and writing research papers efficiently and effectively. Having searched and read several research papers, we have developed better research paper reading habits. This skill will be beneficial to us in the future. We also learnt about clustering in detail beyond the scope of what textbooks typically cover. We came across many clustering methodologies that various researchers have tried and implemented and their pros and cons. During the course of the semester, we studied some density-based clustering algorithms, clustering methods using dimensionality reduction, and also the concept of Subspace-based clustering. From the sheer amount of clustering related research papers, we could infer its importance in academia as well as the industry. We learnt of various new domains which use different types of clustering like Geography, Astronomy, Biology and what not.

In the process of understanding SUBCLU, we collaterally studied the apriori algorithm as well as DBSCAN in depth. While implementing the static SUBCLU from the ELKI implementation, we learned better ways of designing code architecture than we normally are used to. We understood the true power of a neatly modularised object oriented code.

Reading Incremental Clustering research papers, we realised the importance of this class of clustering algorithms in Warehousing environments, and the fallacies of Batch Algorithms. We also embraced the idea of allowing a small fall in accuracy in returns of Speed up or efficient memory usage. In developing the incremental algorithm, we came across Incremental DBSCAN, which being time-consuming, we were then inclined to build a modified algorithm that does not give accurate results, but with a small drop in accuracy gives a significant speed-up. We think that for us, as computer scientists, this was the first time we developed an algorithm by ourselves and naturally, we learnt a lot about designing, coding, debugging and testing from this process. Finally, the entire process followed of weekly reports and presentations vivas has taught us the importance and the advantages of maintaining a timely report of the work happening in the project.

8 FUTURE WORKS

(1) Although we get a speedup compared to static SUBCLU, we also lose some accuracy. This speedup vs accuracy tradeoff is something that needs more research.
(2) We have kept minimum points and epsilon thresholds constant through all the subspaces, which results in more inferior results in some subspaces. It needs more research to understand how these thresholds should be varied to obtain better results.
(3) The implemented code saves the clusterings found in various subspaces on the disk. But deleting an entry from an array is an expensive operation as it is a linear-time operation in the worst-case scenario. Moreover, since deleting a point changes the indices of other points in the database, the clustering data we have saved on the disk would need to be recomputed and saved again. Considering these factors, we have replaced the deleted values with a sentinel, which even though wastes a lot of storage space, saves a lot of time and re-computations. Therefore, a more robust implementation is needed, possibly one equipped with a database management system.

9 TABLE OF CONCLUSIONS

Table 3: Table of Conclusions

Questions	Answers
Link to the base paper	[1]
Link to the code of the base paper	[6] [7] [8]
Link to the datasets used	[9] [10] [11] [12] [13]
Link to your code	[17] [8]
Is our code working?	Yes
Maximum speed up	27.76 times
Average Extra amount of memory	≈ 0
Average Accuracy	89.2%
Accuracy measures	Silhoutte coeff.[18] and purity
Interested in further improvement	No (3rd Year Internships of Everyone)

REFERENCES

[1] Density-Connected Subspace Clustering for High-Dimensional Data(SUBCLU), https://www.dbs.ifi.lmu.de/Publikationen/Papers/sdm04-subclu.pdf
[2] Density-based spatial clustering of applications with noise(DBSCAN), https://en.wikipedia.org/wiki/DBSCAN
[3] Density-based spatial clustering of applications with noise(DBSCAN), http://www2.cs.uh.edu/ ceick/7363/Papers/dbscan.pdf
[4] SUBCLU variant 1, http://j.mecs-press.net/ijitcs/ijitcs-v9-n6/IJITCS-V9-N6-4.pdf
[5] SUBCLU variant 2, https://www.scielo.sa.cr/scielo.php?pid=S0379-39822018000300074&script=sci_arttext&tlng=en
[6] ELKI SUBCLU documentation, https://elki-project.github.io/releases/current/doc/de/lmu/ifi/dbs/elki/algorithm/clustering/subspace/SUBCLU.html
[7] SUBCLU R data analysis package, https://rdrr.io/cran/subspace/man/SubClu.html
[8] Static Code Implementation https://github.com/bg2404/CS568-Data-Mining/tree/main/static
[9] Seeds dataset, https://archive.ics.uci.edu/ml/datasets/seeds
[10] IRIS dataset, https://archive.ics.uci.edu/ml/datasets/Iris
[11] User Knowledge Modelling dataset, https://archive.ics.uci.edu/ml/datasets/User+Knowledge+Modeling
[12] Yeast dataset, http://archive.ics.uci.edu/ml/datasets/yeast
[13] ELKI Mouse dataset, https://github.com/elki-project/elki/blob/master/data/synthetic/Vorlesung/mouse.csv
[14] Enhanced incremental DBSCAN, https://www.sciencedirect.com/science/article/pii/S1110016815001489
[15] Incremental DBSCAN, https://www.researchgate.net/publication/281556454_Incremental_DBSCAN_for_Green_Computing
[16] Incremental DBSCAN algorithm https://www.dbs.ifi.lmu.de/Publikationen/Papers/VLDB-98-IncDBSCAN.pdf
[17] Incremental Code Implementation https://github.com/bg2404/CS568-Data-Mining/tree/main/incremental
[18] Silhoutte Coefficient, https://en.wikipedia.org/wiki/Silhouette_(clustering)