ROCK: A Robust Clustering Algorithm for Categorical Attributes

Group: MinersInPandemic

Aayush Patni
170101001

Aviral Gupta
170101014

Naveen Kumar Gupta
170101041

Rishi Pathak
170101054

Abstract

In data mining, clustering is helpful in discovering distribution patterns in the data. Clustering algorithms usually employ a distance metric-based (e.g., euclidean) similarity measure to partition the database. Data points in the same cluster are more similar than points in different clusters. For this review, we consider the following paper titled 'ROCK: A Robust Clustering Algorithm for Categorical Attributes' [2]. We give a brief explanation of the algorithm and datasets for evaluating its performance, Besides we discuss the need for this algorithm and its real-world applications.

1 Introduction

The data mining task is the semi-automatic or automatic analysis of large quantities of data to extract previously unknown, interesting patterns such as groups of data records (cluster analysis), unusual records (anomaly detection), and dependencies (association rule mining, sequential pattern mining). A record of non-numeric data point can be transformed to a record of boolean attributes, which are a particular case of categorical attributes.

Clustering analysis is broadly used in many applications such as market research, pattern recognition, data analysis, and image processing. Clustering can also help marketers discover distinct groups in their customer base. Moreover, they can characterize their customer groups based on purchasing patterns. 'ROCK' is a clustering algorithm that uses the idea of linking data points instead of clustering based on the distance between data points.

The two key steps in the static algorithm are the computation of links and merging of clusters based on goodness measure. [7]. In this paper we try to introduce an incremental version of ROCK clustering algorithm, which adds new points to original database in batches.

2 Review of ROCK Clustering Algorithm

2.1 ROCK Clustering Algorithm

2.1.1 Brief Overview. A random sample is drawn from the database. After that, the sampled points are added to a hierarchical clustering algorithm that employs links. Finally, the remaining data points on the disk are allocated to the relevant clusters, only the clusters involving the sampled points are used.

2.1.2 Goodness Measure. We define the goodness measure $g(C_i, C_j)$ for combining the pair of clusters C_i and C_j as follows, $link[C_i, C_j]$ stores the number of cross link between clusters C_i and C_j, and n_k is the size of cluster C_k .

$$g(C_i, C_j) = \frac{link[C_i, C_j]}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}} \quad (1)$$

The strongest pair of clusters to be combined in any given state is the pair of clusters for which the above calculation of goodness measure is optimum.

2.1.3 Clustering Algorithm. Initially, each point represents a single separated cluster. We maintain a local heap $q[i]$ for each of the cluster which contains every other cluster j such that $link[i, j]$ is non-zero, arranged in a descending order on the basis on their goodness measure values.

We also maintain a global heap Q which contains all the clusters, arranged in a descending order on the basis of their goodness measure values. The goodness measure for a cluster j is calculated with the cluster having the maximum value in $q[j]$.

During each iteration, the pair of clusters with the highest goodness is merged, and the goodness measure of the corresponding clusters is updated. The algorithm terminates either when the desired number of clusters has been reached, or when all the clusters are well separated.

2.1.4 Computation of Links. Before clustering, the number of links between each point in the set is computed. This is done by squaring the adjacency matrix that defines the graph of neighbors whereby, the number of links between two points i and j can be computed by multiplying i^{th} row and j^{th} column of the adjacency matrix A.

2.1.5 Time and Space Complexity. The worst case complexity of ROCK algorithm, along with computation of neighbor lists and links, is $O(n^2 + nm_m m_a + n^2 \log n)$. The ROCK clustering algorithm's space complexity is the same as that of link computation, that is, $O(min\{n^2, nm_m m_a\})$.

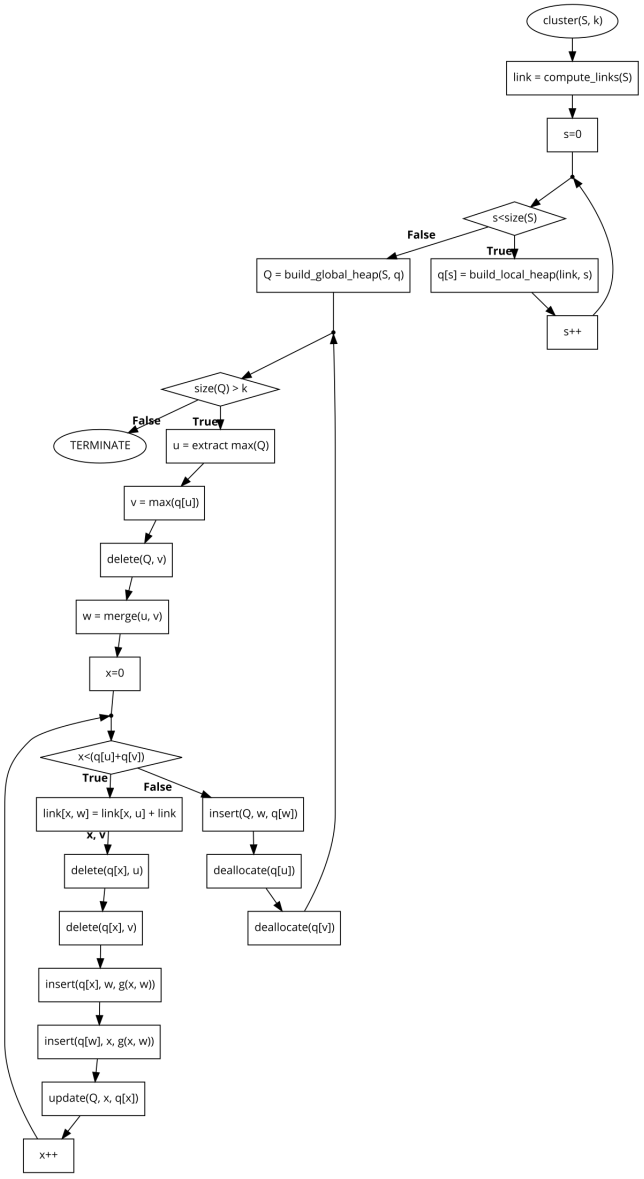

Figure 1. Flow Chart for Static ROCK Clustering Algorithm

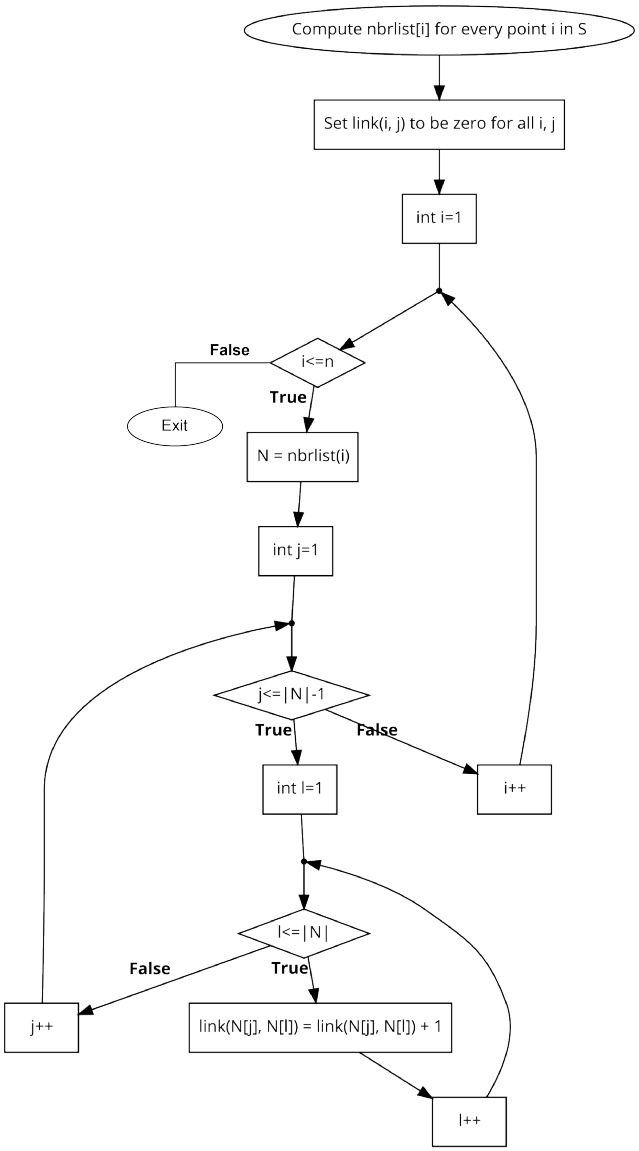

Figure 2. Flow Chart for Computing Links

2.1.6 Miscellaneous Issues. Random sampling can be done to reduce the number of points to be considered for clustering and thus, help in speeding up the algorithm. Outliers can be handled easily by ROCK through choosing an appropriate value of θ or by filtering the relatively isolated clusters that are formed during the intermediate stages of ROCK.

3 Related work

Clustering is one of the most researched areas in recent times. Various techniques for accurate clustering have been proposed, e.g. K-MEAN, CURE, BIRCH, ROCK and many other. We researched and found that there is no existing incremental version of the ROCK clustering algorithm. We found some incremental clustering algorithms for mixed and categorical data like [6] and [4], which were not related to ROCK algorithm.

Also, over the period many variants of the ROCK clustering algorithms are proposed. One variant of ROCK is QROCK[1] which says that the final clusters formed by ROCK are the connected components of the graph with data points as vertices. Another variant is GE-ROCK[9] which improves ROCK by introducing genetic optimisation with clustering. IROCK[10] introduces concept of link weight overlaps instead of link count method for textual data. EROCK[3] proposes enhanced ROCK with improved similarity measure

and storage efficiency for improving the organization of document data. DROCK[8] is a distributed version of the ROCK clustering algorithm that employs preliminary calculations at different processors.

Algorithm 1 Static ROCK Algorithm

```
1:  procedure ROCK(S, k, θ)
2:      for all i ∈ S do
3:          for all j ∈ S do
4:              if j_coefficient(i, j) > θ then
5:                  link_matrix[i][j] := true
6:              end if
7:          end for
8:      end for
9:      for all s ∈ S do
10:         clusters[s] := initial(s)
11:     end for
12:     while size(clusters) > k do
13:         max_goodness := 0
14:         u := −1; v := −1;
15:         for all i ∈ clusters do
16:             for all (j > i) ∈ clusters do
17:                 if goodness(i, j) > max_goodness then
18:                     max_goodness := goodness(i, j)
19:                     u := i; v := j
20:                 end if
21:             end for
22:         end for
23:         delete(u); delete(v)
24:         w := merge(u, v)
25:         insert(w)
26:     end while
27:     return clusters
28: end procedure
```

Algorithm 2 Incremental ROCK Algorithm

```
1:  procedure INCREMENTAL_ROCK(B)
2:      for all i ∈ S do                          ▷ Add new columns
3:          for all j ∈ B do
4:              if j_coefficient(i, j) > θ then
5:                  link_matrix[i][j] := true
6:              end if
7:          end for
8:      end for
9:      for all i ∈ B do                          ▷ Add new rows
10:         for all j ∈ (B + S) do
11:             if j_coefficient(i, j) > θ then
12:                 link_matrix[i][j] := true
13:             end if
14:         end for
15:     end for
16:     for all s ∈ B do
17:         clusters.insert(initial(s))
18:     end for
19:     while size(clusters) > k do
20:         max_goodness := 0
21:         u := −1; v := −1;
22:         for all i ∈ B do
23:             for all j ∈ clusters do
24:                 if goodness(i, j) > max_goodness then
25:                     max_goodness := goodness(i, j)
26:                     u := i; v := j
27:                 end if
28:             end for
29:         end for
30:         delete(u); delete(v)
31:         w := merge(u, v)
32:         insert(w)
33:     end while
34:     return clusters
35: end procedure
```

4 Proposed Incremental ROCK Algorithm

Let the original dataset be S. After an initial clustering *clusters*, let B be the changes introduced to the dataset converting it from S to S'. Our algorithm deals with the insertion changes in batches. So let the size of one batch be b.

4.1 Intuition

We have k clusters, and we get a batch of b new points. Now three cases can happen with any of these points.

1. They can merge among them to form a new cluster. (This will only happen if they don't have links with any existing clusters.)
2. They can merge with any of the existing clusters.
3. Some point from the existing cluster leaves that cluster and merge with the new points.

Now we argue that the 3rd case cannot happen because in that case, there will be k+1 clusters, and the new cluster that is formed will have links with the cluster from which the point has arrived, and hence they will merge. Our proposed algorithm takes first and second cases into account.

4.2 Insertion Phase

First, we add a batch of b points and calculate links between the new points and the existing points and update the adjacency matrix. Considering each new point of the batch as its own cluster, we obtain total $b + k$ clusters. For each of the b new cluster, we find a cluster with the best goodness measure and merge them. We repeat the step till there are k clusters or the existing clusters have no links between them. After this, we remove the outliers from the formed clusters. This represents one iteration. We continue this process for B/b times.

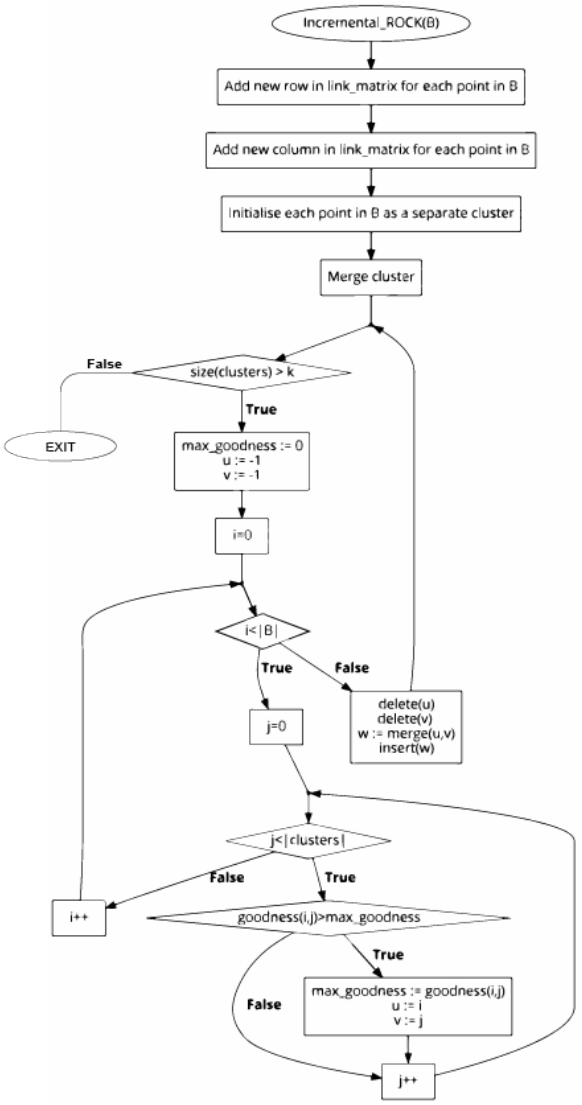

Figure 3. Flow Chart of Incremental ROCK Algorithm

4.3 Outlier Removal

We consider the clusters with size less than or equal to five percent of the largest cluster size as outliers. The total number of clusters after outlier removal remains greater than k. In the incremental algorithm, we remove the outliers every time we finish processing a batch addition. While in the static algorithm, we remove the outliers when the number of clusters falls below ten percent of total points.

4.4 Time Complexity

The time complexity of the proposed incremental algorithm for one iteration of batch is $O(nb^2 + b^3)$, where N points are added by static algorithm. Line 2-8 has a time complexity of $O(nb)$. Line 9-15 has a time complexity of $O(nb + b^2)$. Line 16-18 has a time complexity of $O(b)$. While loop from lines 19-33 will run for $O(b)$ times. In each iteration, for all of the new b points, we find the link with all the other points, the maximum such points are $n + b$. So the worst case for any iteration is $O(nb + b^2)$. Hence out total complexity is $O(nb^2 + b^3)$. So if we are adding n new points in batches of size b, there will be a total n/b iteration on incremental algorithm leading to a time complexity of $O(nNb + nb^2)$ which is approximate $O(nN)$ (as $b << n$). This is much less than the $O((n+N)^3)$ time complexity of the static algorithm.

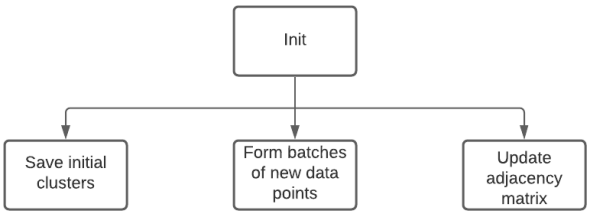

Figure 4. Code Initiation of Incremental algorithm

5 Implementation

5.1 Code Description

This code uses an object-oriented approach to implement the algorithm efficiently. We made a class called Rock with public attributes such as adjacency_matrix, data, and theta, clusters, no_clusters. This class makes an object of Rock, which generates clusters from the given data/CSV file. Then we made a class, Incremental, with attributes initial_clustering, which is a Rock object and contains all the data and function of static algorithm. Other attributes include new_data, batch_size, and no_batches. The code is executed in the following way:

Initiation: Function `Incremental` is used to create an instance (Figure 3).

- **Incremental():** Constructor function creates an instance of the Incremental ROCK and then runs a for loop which calls update_adjacency_matrix() to compute newly formed links and incremental_process() to make k clusters for B/b times.
- **update_adjacency_matrix():** Calculate links of the newly added batch with the existing data points and among themselves.An entry corresponding to the i^{th} row and j^{th} column in the matrix is true if result returned by j_coefficient(data[i], data[j]) is greater than or equal to theta and vice versa.
- **j_coefficient():** This function calculates the jaccardś coefficient of two data points.

Data Processing: In this step cluster analysis is done on input batch data and later processed data is saved in instance. Figure 3 shows code flow of data processing.

- **incremental_process():** Performs cluster analysis in line with rules of Incremental ROCK algorithm.

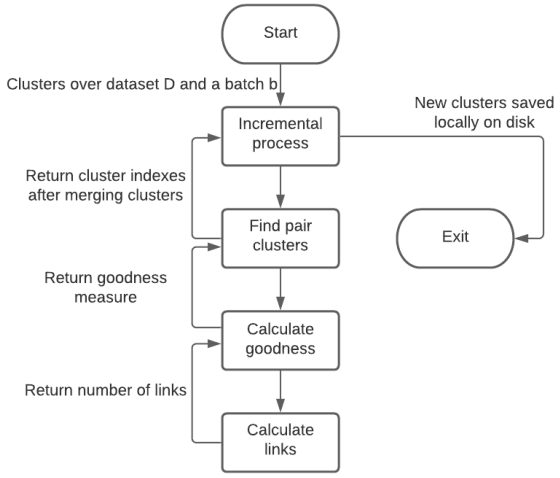

Figure 5. Code Flow of Incremental Data Processing

- **find_pair_clusters():** Returns pair of clusters that are best candidates for merging in line with goodness measure.The pair of clusters for which the above goodness measure is maximum is the best pair of clusters to be merged.
- **calculate_goodness():** Calculates coefficient 'goodness measurement' between two clusters. The coefficient defines level of suitability of clusters for merging.
- **calculate_links():** Returns number of link between two clusters. Link between objects (points) exists only if distance between them less than connectivity radius.
- **calculate_f():** This function calculates the value of normalisation degree as $1 + 2 * \frac{1-\theta}{1+\theta}$ where θ is the threshold.
- **remove_outliers():** Removes the clusters with size less than 5% of the biggest cluster's size, until the number of clusters remains larger than k.

6 Dataset

We experimented over two real-life datasets *Mushroom* and *Congressional Votes Datasets*. These datasets contain only categorical attributes and both can be obtained from UCI Machine Learning Repository.

6.1 Mushroom Dataset

A data record contain a label (poisonous or edible) for a mushroom and 22 attributes to describe its characteristics. The number of edible and poisonous mushroom in dataset are 4208 and 3916 respectively.

6.2 Congressional Votes Dataset

The *Congressional Votes dataset* had a total of 435 data points and of which 267 were democrat and remaining 168 were republican. Each record of dataset has one label and 16 categorical attributes.

7 Experimental Results of Incremental Algorithm

Original ROCK Algorithm was tested over *Mushroom* and *Congressional Votes Dataset* and we tested our Incremental ROCK Algorithm over the same dataset. All experiments were performed on a linux machine with 8 GB of RAM and running on Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, 2592 Mhz, 6 Core(s) CPU. The metric used for result comparison are running time, memory footprint, clustering quality metrics as defined in [5] and data distribution in clusters.

7.1 Real-Life Datasets

7.1.1 Mushroom Dataset. The ROCK algorithm was run over a partition of mushroom dataset which contained 1500 data points of which 1200 were processed statically and then 300 were added incrementally with $\theta = 0.8$, k=20 and batch size=120.

Algorithm Strategy	Static ROCK	Incremental ROCK
run time (sec)	85.69	47.61
memory usage (KB)	9208	8856

Table 1. Execution time and memory comparison for 1500 data points

From Table 1. we can clearly see the speedup we get by using incremental. Of 47.61 seconds running time of ROCK incremental, 42.96 seconds was used to process 1200 data points statically while only 4.65 seconds was used to process next 300 data points incrementally. On the other hand ROCK static took 85.69 seconds to process 1500 data points.

Clustering in Incremental ROCK					
S.No.	Edible	Poisonous	S.No.	Edible	Poisonous
1	0	7	2	0	29
3	28	0	4	17	0
5	206	0	6	0	427
7	0	32	8	275	0
9	72	0	10	1	0
11	1	0	12	0	17
13	4	0	14	0	10
15	19	0	16	1	0
17	3	0	18	1	0
19	0	247	20	4	0

Table 2. Clustering result over a part(1500 data points) of Mushroom Dataset

The Table 2. describes the distribution of data points among 20 clusters. From the distribution we can see our cluster contain very less false positive, in this case none and thus proving high quality clustering.

7.1.2 Congressional Votes Dataset.

For recording performance of incremental ROCK algorithm over *Congressional Votes* dataset with $\theta = 0.65$, k=2 and batch size=35. Of total 435 data points, 350 were processed statically and then 85 were added incrementally.

Algorithm Strategy	Static ROCK	Incremental ROCK
run time (sec)	2.34	1.38
memory usage (KB)	5168	4972

Table 3. Execution time and memory comparison

Clustering in Incremental Rock					
S.No.	Democrat	Republic	S.No.	Democrat	Republic
1	210	0	2	32	149

Table 4. Clustering result over Congressional Votes Dataset

Table 3 shows time complexity and memory usage in the execution of the algorithm. Of 1.38 seconds, only 0.13 seconds was used to add 85 data points incrementally.

7.2 Comparison of Incremental ROCK and Static ROCK with random sampling

As the time complexity of static ROCK is exponential it was not feasible for a large dataset, and to tackle this issue authors of the original ROCK algorithm suggested clustering based on random sampling. In static ROCK with random sampling, first, a sufficient amount of data points were randomly sampled over the given dataset. Second, the randomly sampled data points were clustered using static ROCK algorithm and then the remaining points were later added to clusters.

For comparing the accuracy of our algorithm we compare the data points distribution among clusters formed using incremental ROCK and results in original ROCK paper.

7.2.1 Mushroom Dataset.

In testing of algorithm over *Mushroom Dataset*, out of a total of 8124 data points of *Mushroom dataset* 1500 were processed statically and then remaining 6624 were added incrementally. The parameters in the experiment were $\theta = 0.8$, k=20 and batch size=1. The original paper also used the same parameters.

7.2.2 Congressional Votes Dataset.

For testing over *Congressional Votes* dataset with $\theta = 0.65$, k=2 and batch size=1. Of total 435 data points, 350 were processed statically and then 85 were added incrementally.

From Table 5 and Table 6, we can verify clustering quality of our Incremental ROCK algorithm to be almost same as clustering quality of Static ROCK with random sampling(result of clustering on Static ROCK with random sampling was directly taken from the original ROCK paper[2]).

Clustering in Static ROCK with random sampling					
S.No.	Edible	Poisonous	S.No.	Edible	Poisonous
1	96	0	2	0	256
3	704	0	4	96	0
5	768	0	6	0	192
7	1728	32	8	0	32
9	0	1296	10	0	8
11	48	0	12	48	0
13	0	288	14	192	0
15	32	72	16	0	1728
17	288	0	18	0	8
19	192	0	20	16	0
21	0	36			
Clustering in Incremental ROCK					
S.No.	Edible	Poisonous	S.No.	Edible	Poisonous
1	83	0	2	704	0
3	764	0	4	0	192
5	1728	0	6	48	0
7	192	0	8	48	0
9	0	288	10	0	1296
11	0	32	12	0	8
13	84	156	14	0	8
15	16	0	16	32	72
17	288	0	18	0	36
19	0	1728	20	192	0

Table 5. Clustering result over Mushroom Dataset

Clustering in Static Rock with random sampling					
S.No.	Democrat	Republic	S.No.	Democrat	Republic
1	201	5	2	22	144
Clustering in Incremental Rock					
S.No.	Democrat	Republic	S.No.	Democrat	Republic
1	210	0	2	32	149

Table 6. Clustering result over Congressional Votes Dataset

7.3 Scalability with respect to θ

As, we increase θ, the number of links in our dataset decreases (less number of links between data points of same class) and quality of links increases (high chance of forming links with of same class). So, we need to experimentally find a θ which give us the best clustering result.

7.4 Clustering quality with respect to batch size

From the result, we can see that the lower the batch size, the better the clustering. We also know that the time complexity of the incremental algorithm is directly proportional to batch size. Hence we can conclude that the value of batch size should be ideally 1, i.e., batch size should not be a parameter.

7.5 Clustering quality with respect to k

If we increase the k, generally False Negatives and True Positives increases while False Positives and True Negative

decreases. In an ideal case, where k is equal to the number of class labels, both False Positives and False Negatives are zero. Hence, for best clustering, we need to have k close to the number of the class labels but k is a variable in our algorithm, so, it is not advisable to use rand index, recall, and f measure to calculate accuracy.

k	clusters formed	rand index	recall	f1 measure
2	3	0.810852	0.527784	0.690914
3	3	0.810852	0.527784	0.690914
4	4	0.763593	0.472086	0.641384
5	5	0.682845	0.40096	0.572408
6	6	0.65862	0.389114	0.560233
7	7	0.592902	0.349055	0.517481

Table 7. Variation of accuracy measures wrt k ($\theta = 0.8$)

For example, when we run algorithm with different values of k over 600 data points of *Mushroom dataset* of which 500 were processed statically and 100 were added incrementally, we observe decrease in accuracy measures. The Table 6. summarizes the variation in accuracy measures with respect to change in k.

7.6 Clustering quality with respect to points added incrementally

From the result, we can also see that our clustering quality decreases as the number of points added incrementally increases. The reason for the decrease in clustering quality can be seen with an example. For Example, consider two points p and q having a high goodness measure between them. In the static algorithm, they will form a cluster initially. However, in the incremental algorithm, suppose point q is added incrementally, then the point p may already be part of a particular cluster, say C. For q to be merged in C goodness measure will be calculated between C and q. Since C has many points, the goodness measure is not that high, and q is merged with some other cluster, resulting in a decrease in clustering quality. The graphs in Fig 8. shows variation in accuracy measures with outlier removal.

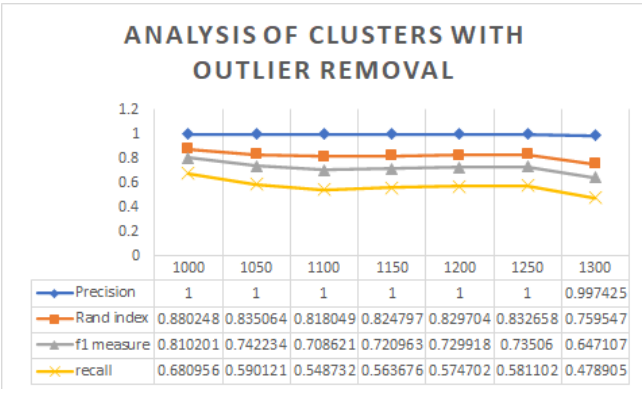

Figure 6. Analysis of Clusters with Outlier Removal

The incremental ROCK algorithm was tested on 1500 data points of *Mushroom dataset* of which 1000 were processed statically and then 50 were added incrementally at each iteration for analysing different accuracy measures. The parameters chosen for experiment are θ= 0.8, k=2 (same as number of class labels for detailed analysis using f score, rand index and recall) and batch size=10.

8 Lessons learnt

Throughout the course of this project, we learnt a lot of valuable lessons. Consistent weekly reports and evaluation helped us to get familiar with research tools like Latex, websites like Google Scholar and accessing journals like ACM. We also learnt about the process of reviewing a research paper and following up with the authors of the original paper to get datasets. We also learnt about the importance of robustly testing our code and making our results reproducible for others. Finally we also realized that the reason behind an algorithm like ROCK not having an incremental version was probably due to the fact that ROCK with random sampling was pretty similar to an incremental clustering algorithm.

9 Future Works and Conclusion

In this report we briefly reviewed ROCK algorithm and proposed an Incremental version of ROCK which deals with insertion of points. We then showed that the quality of clustering produced by our incremental version was in most cases similar or better than that obtained by the static algorithm, despite our algorithm running in significantly less time. Further work can be done to extend the incremental algorithm to deal with deletions. We also observed that in our incremental algorithm, the best clusters were formed when batch size was 1 that is when individual points were added to the existing clusters like in the static algorithm with random sampling.

References

[1] M. Dutta, A. Kakoti Mahanta, and Arun K. Pujari. 2005. QROCK: A quick version of the ROCK algorithm for clustering of categorical data. *Pattern Recognition Letters* 26, 15 (2005), 2364 – 2373. https://doi.org/10.1016/j.patrec.2005.04.008

[2] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. 2000. Rock: A robust clustering algorithm for categorical attributes. *Information Systems* 25, 5 (2000), 345 – 366. https://doi.org/10.1016/S0306-4379(00)00022-3

[3] Aasia Khanum. 2010. Document Topic Generation in Text Mining by Using Cluster Analysis with EROCK. *International Journal of Computer Science and Security* 4 (06 2010).

[4] Dr. Thangavel Kuttiyannan. 2010. Incremental Algorithm to Cluster the Categorical Data with Frequency Based Similarity Measure. 37 (01 2010).

[5] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval.* Cambridge University Press, USA.

[6] Fakhroddin Noorbehbahani, Sayyed Rasoul Mousavi, and Abdolreza Mirzaei. 2015. An incremental mixed data clustering method using a new distance measure. *Soft Computing* 19, 3 (2015), 731–743.

Link to the base paper	https://ieeexplore.ieee.org/document/754967
Link to the code of the base paper	- (We tried to contact the authors of the base paper but we didn't get a response. We coded the static algorithm ourselves)
Link to the datasets used	• Congressional Voting - https://bit.ly/2RiU8q1 • Mushroom Data Set - https://bit.ly/2Q7P474
Link to your code	https://github.com/rishipathak6/CS568-Group-Project
Is your code working	Yes
What is the maximum speed up you are getting	1.8 (On mushroom dataset without losing the quality of clustering)
What is the extra amount of memory required by your incremental algorithm	Extra memory required is only for loading old dataset and its result
How did you measure the accuracy?	precision (cannot use rand index, recall and f score as explained in section 7.5)
Are you interested in further improve your work during the summer?	No

[7] Andrei V. Novikov. 2019. PyClustering: Data Mining Library. *Journal of Open Source Software* 4, 36 (2019), 1230. https://doi.org/10.21105/joss.01230

[8] A. Patidar, R. Joshi, and S. Mishra. 2011. Implementation of distributed ROCK algorithm for clustering of large categorical datasets and its performance analysis. In *2011 3rd International Conference on Electronics Computer Technology*, Vol. 2. 79–83. https://doi.org/10.1109/ICECTECH.2011.5941659

[9] Qiongbing Zhang, Lixin Ding, and Shanshan Zhang. 2010. A genetic evolutionary ROCK algorithm. In *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, Vol. 12. V12–347–V12–351. https://doi.org/10.1109/ICCASM.2010.5622305

[10] Shaoxu Song and Chunping Li. 2006. Improved ROCK for Text Clustering Using Asymmetric Proximity. In *SOFSEM 2006: Theory and Practice of Computer Science*, Jiří Wiedermann, Gerard Tel, Jaroslav Pokorný, Mária Bieliková, and Július Štuller (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 501–510.