Algorithms for Incremental Clustering

A thesis submitted in the partial fulfilment of

the requirements for the degree of

Master of Technology

by

Arindam Chowdhury

Roll No: 154102013

Under the guidance of

Dr. Prithwijit Guha

June, 2017

Department of Electronics and Electrical Engineering

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI - 781 039

Declaration of Authorship

I hereby declare that the work presented in this thesis entitled **"Algorithms for Incremental Clustering"**, and the work presented in it, submitted for the award of degree for Master of Technology in Signal Processing at Indian Institute of Technology, Guwahati, is a bonafide work carried out under the supervision of **Dr. Prithwijit Guha**. The books, journals, articles and websites, which I have made use of are acknowledged at the respective place in the text. The contents of this thesis, in full or in parts, have not been submitted to any other institute or university for the award of any degree or diploma.

Arindam Chowdhury

June, 2017 Dept. of Electronics and Electrical Engineering

Guwahati. Indian Institute of Technology, Guwahati, Assam-781039, India.

Certificate

This is to certify that the work presented in this thesis entitled, **"Algorithms for Incremental Clustering"**, by **Arindam Chowdhury, Roll-154102013,**", has been carried out in Multimedia Analytics Lab, Department of Electronics and Electrical Engineering, Indian Institute of Technology, Guwahati, under my supervision and that the work has not been submitted elsewhere for a degree.

Dr. Prithwijit Guha
Assistant Professor
Dept. of Electronics and Electrical Engineering
Indian Institute of Technology
Guwahati, Assam-781039, India.

Dedicated to my parents

Acknowledgements

I would like to express my most sincere gratitude to my supervisor, **Dr. Prithwijit Guha** for his valuable guidance and suggestions during the course of the project, without which this work could not have been possible. I also feel it as an immense privilege to express my gratitude to my parents who indirectly helped me and remain an inspiration to write the thesis. I would also like to thank the Head of the Department and other faculty members for their help in carrying out this work. I would also like to extend my sincere gratitude non-teaching staff of the department for providing me all necessary facilities for doing my research.

I also wish to express my special thanks to **Mr. Raghvendra Kannao, Mr. Mathew Francis, Mr. Suman Paul Choudhury**, Research Scholars, Dept. of EEE, IIT Guwahati, for their valuable guidance. I also take this opportunity to express my deepest gratitude to my friends **Karthik**, **Sudhir** and **Anushree** for having their tremendous support and love all through.

Arindam Chowdhury
IIT Guwahati
May, 2017

Abstract

Two incremental clustering algorithms are presented. The first one (Incremental K-Means) assumes hyper-ellipsoid cluster structures which are easy to localize with respect to a new point. The second algorithm (Incremental DBSCAN) is free from any assumptions on cluster structures but require extensive search and distance computations for inserting a new point. Our proposal marries both the approaches to achieve a fast and memory inexpensive incremental clustering methodology. The available dataset (or its part) is first subjected to incremental K-means to estimate a crude distribution over the input space. These points are also used to construct the DBSCAN clusters. A new point is first localized with respect to distribution estimated using incremental K-means. Its neighbors are searched from members of localized host clusters. This leads to a much reduced search for insertion of the new point into a DBSCAN cluster set. The Incremental K-means is also extended to Hierarchical Incremental K-means for further reduction of this search time.

Contents

List of Figures

Chapter 1

Introduction

Clustering techniques are among the most popular unsupervised machine learning algorithms. These algorithms are extremely useful in learning representations for unlabeled data [1–4]. A variety of clustering techniques have been formulated that utilize inter-point distances *(Centroid Based)*[5], spatial linkage *(Agglomerative Clustering)*[6], connectivity graph *(Spectral Clustering)*[7–9], density connections *(Density Based)* and also data distribution[10] to learn both deterministic and probabilistic models for data. Clustering is used widely for data mining applications in Biology and Bioinformatics [11], Computer Vision and Image Processing [12], Social Media Analysis [13, 14], Text Mining [15] etc. This work focuses on "Incremental Clustering Algorithms" that deal with streaming data problems.

1.1 Previous Work

Incremental Clustering algorithms have been very popular over the years as they have an edge over their offline counterparts by being less memory expensive as, unlike offline methods, they do not load the entire database into memory for processing, rather pick the data points one by one, which also brings down the processing time.

Can et. al. [16] proposed an incremental clustering method for very large document databases which periodically updated the clusters. The algorithm used a data structure that had very low computational complexity and negligible storage overhead. It makes C^2ICM (Cover-Coefficient-based Incremental Clustering

Methodology) suitable for very large dynamic document databases. Gupta et. al. [17] propose a single pass incremental clustering algorithm that divides the data space into segments in a manner similar to windowed k-means. Starting with a set of initial points as mean, the algorithms keeps adding subsequent points to these clusters depending on distance. Clusters are maintained, killed and created periodically depending on the probability of points in them.

Lughofer et. al. [18] extend the concept of Vector Quantization to incremental clustering by incorporating a vigilance parameter, which maintains steers the trade-off between plasticity and stability during incremental online learning. The paper proposes two novel extensions: one concerns the incorporation of the sphere of influence of clusters in the vector quantization learning process by selecting the winning cluster based on the distances of a data point to the surface of all clusters. Another one introduces a deletion of cluster satellites and an online split-and-merge strategy: clusters are dynamically split and merged after each incremental learning step. Both strategies prevent the algorithm to generate a wrong cluster partition due to a bad a priori setting of the most essential parameter(s).

The single pass incremental algorithms are computationally inexpensive, require less memory and quite fast but often suffer from instability and also dependency on the order of streaming data. But they can be very effective pre-clustering tools for more stable clustering techniques such as the density based methods. For example, Density based clustering techniques come in handy while clustering data that has highly non-convex spatial distribution.

DBSCAN (Density Based Spatial Clustering with Applications for Noise) was proposed by Ester et. al. [19]. It makes use of certain spatial distance constraints to detect density connections between data points to form suitable clusters. Points which have a certain minimum number of neighbouring points are called core points and clusters are formed by a group of core points that are directly or indirectly connected to each other. It also efficiently removes noise from the data by suitable choice of clustering parameters. One disadvantage of the process is that finding the density connections(region query) is computationally expensive and therefore slows down the process for big data.

There have been several extensions of DBSCAN, some of which are as follows. Zhou et. al. [20] propose a fast DBSCAN algorithm (FDBSCAN) which considerably speeds up the original DBSCAN algorithm. Unlike DBSCAN, FDBSCAN

uses only a small number of representative points in a core point's neighborhood as seeds to expand the cluster such that the execution frequency of region query and consequently the I/O cost are reduced. Borah et. al. [21] proposes a similar extension which efficiently finds representatives in the neighbourhood of a given core point for region query, by choosing only those points which lie at the boundary of a circle centered around the core point having a user defined radius. Birant et. al. [22] suggests a very interesting extension of DBSCAN in which both spatial and non-spatial attributes are used in forming clusters. For example, if we have image coordinates as data, both the spatial coordinates and the intensity values would be used to find the neighbours of a given data point. The added constraint augments the original method in detecting non-convex clusters which inspite of being spatially distant are detected because of similarity in the non-spatial attributes.

Inspite of achieving superior performance over other offline clustering techniques, the algorithms are not free from the typical disadvantages associated with non-incremental techniques. An incremental version of DBSCAN was prposed in [23] which is used for mining large databases. In this algorithm, the insertion or deletion of an object affects the existing cluster set in the neighbourhood of this object. It efficiently speeds-up DBSCAN even for large numbers of daily updates in a data warehouse.

1.2 Our Proposal

The work presented in this thesis proposes a clustering algorithm that combines two existing algorithms. The first is an incremental version of the k-means algorithm. As the data cannot be stored, the algorithm approximates a set of clusters which are parameterized by mean, variance and weight. The output of this algorithm is a point set covered by hyper-ellipsoidal cluster boundaries. The second algorithm clusters data based on density connections(DBSCAN) among them and is highly efficient in detecting clusters of data even with a complicated, non-convex spatial distribution. However, it suffers from high computation and memory requirements. This is because, for each point to be clustered, the algorithm looks for its neighbours among all points that were clustered before it and therefore as the size of data grows, the number of distance computations grows with it. In our proposed method, a section of the database is sampled initially and is used to generate a set of clusters by incremental k-means. This section of the data is also

processed by DBSCAN to generate a set of clusters. Now, as a new point comes in, it is first passed through the incremental k-means algorithm to find a set of host clusters. These clusters are first updated for the new point and then passed to DBSCAN as a reduced search space for finding its neighbours. This results in reduced computations and memory requirements.

Thesis Organization – Incremental k-Means Clustering is discussed in Chapter **??**. The hierarchical extension of incremental k-means is described in Chapter 3. DBSCAN, its incremental version and the proposed algorithm are presented in Chapter 4. Finally, we conclude in Chapter **??** and sketch the future extensions of the present work.

Chapter 2

Incremental Clustering

Consider the problem of estimating a set of clusters on streaming data $\boldsymbol{X}_t$. In most practical cases, the "streaming" data can not be stored. Hence, the traditional offline clustering algorithms (e.g. K-means) are not applicable here. In such cases, we can not estimate the "accurate" cluster parameters but can maintain an "approximate" set of clusters on the data seen so far.

This approximate cluster set $\zeta(t) = \{C_i(t); i = 1, ..., m_t\}$ is a finite set of clusters of size m_t, where a cluster at the t^{th} instant is given by,

$$C_i(t) = \{\boldsymbol{\mu_i}(t), \boldsymbol{\Sigma_i}(t), \pi_i(t)\} \tag{2.1}$$

where, $\boldsymbol{\mu_i}(t), \boldsymbol{\Sigma}_i(t)$ and $\pi_i(t)$ are the respective mean vector, covariance matrix and the weight of $C_i(t)$ at the t^{th} time instant. Note that, $\boldsymbol{\mu_i}(t)$ and $\boldsymbol{\Sigma}_i(t)$ are estimated only from those $n_i(t)$ number of data points which have gone to $C_i(t)$ till the t^{th} instant and $\pi_i(t) = \dfrac{n_i(t)}{t}$

2.1 The Algorithm

This section provides a detailed description of the proposed algorithm in a modular fashion. The complete mathematical framework is provided along with necessary deductions.

Initialization – The cluster set is initialized with a single cluster $C_1(1) = \{\boldsymbol{\mu_1}, \boldsymbol{\Sigma}_{init}, 1.0\}$, where $\boldsymbol{X}_1$ is the data point at $t = 1$ and $\boldsymbol{\Sigma}_{init}$ is the initial variance whose value is assigned from the domain knowledge. The first point is assigned the label 1.

Determine Host Clusters – To identify the clusters which are eligible for update at t^{th} instant, we follow a two-check rule. First, a dimension wise Chebyshev's inequality check is performed on $\boldsymbol{X}_t$ with respect to each cluster,

$$\{\boldsymbol{X}^t[j] - \boldsymbol{\mu}_i^{(t-1)}[j]\}^2 \leq \lambda^2 \boldsymbol{\Sigma}_i^{(t-1)}[j][j] \ for \ j = 1, \ldots, \ d \tag{2.2}$$

where i is the cluster index, d is data dimension and λ is a user defined threshold.

The total number of violations of the above inequality are noted. A normalized violation count is then generated $V_{count} = \dfrac{Number\ of\ Violations}{Number\ of\ Dimensions}$, which is compared with a user defined threshold, V_{thres}. The clusters which fail to satisfy the inequality $V_{count} \leq V_{thres}$ are filtered out as non-hosts. For the rest, a membership based inequality check is performed which is of the form,

$$(\boldsymbol{X} - \boldsymbol{\mu_i})^\intercal \boldsymbol{\Sigma_i^{-1}} (\boldsymbol{X} - \boldsymbol{\mu_i}) \leq d\lambda^2 \tag{2.3}$$

where d is the data dimension and i being the cluster index.

The set of clusters which satisfy this inequality is called the host clusters set, denoted by ζ_{host} and set of non-host clusters is denoted by $\zeta_{nonhost}$.

Penalize Non Host Clusters – Parameter $\pi(t)$ for the set $\zeta_{nonhost}$ is updated as per the following rule,

$$\pi_i(t) = (1 - \alpha_t)\pi_i(t-1) \ \forall i \in \zeta_{nonhost} \tag{2.4}$$

where the update rate is defined as $\alpha_t = \dfrac{1}{t}$.

Update Host Cluster Parameters – In this section we deduce the equations for updating the parameters of clusters set $\zeta(t-1)$ learned till the $(t-1)^{th}$ instant to $\zeta(t)$ with the current data point x_t.

The weight update rule, at t^{th} instant, is as follows,

$$\pi_i(t) = (1 - \alpha_t)\pi_i(t-1) + \alpha_t \ \forall i \in \zeta_{host} \tag{2.5}$$

To update the mean, of the i^{th} cluster in set ζ_{host} at the t^{th} instant, we proceed as follows,

$$\begin{aligned} \boldsymbol{\mu_i}(t) &= \frac{1}{n_i(t)} \sum_{\boldsymbol{X} \in C_i(t)} \boldsymbol{X} \\ &= \frac{n_i(t-1)\boldsymbol{\mu_i}(t-1) + \boldsymbol{X}_t}{n_i(t)} \\ &= \frac{(t\pi_i(t) - 1)\boldsymbol{\mu_i}(t-1) + \boldsymbol{X}_t}{t\pi_i(t)} \\ &= (1 - \beta_i(t))\boldsymbol{\mu_i}(t-1) + \beta_i(t)\boldsymbol{X}_t \end{aligned} \tag{2.6}$$

where $\beta_i(t) = \dfrac{\alpha_t}{\pi_i(t)}$. Note that, the update factor $\beta_i(t)$ is not only a function of time but also the cluster index.

Similarly, the variance of i^{th} cluster in set ζ_{host} at t^{th} instant is updated as,

$$\begin{aligned} \boldsymbol{\Sigma}_i(t) &= \frac{1}{n_i(t)} \sum_{\boldsymbol{X} \in C_i(t)} [\boldsymbol{X} - \boldsymbol{\mu_i}(t)]^\intercal [\boldsymbol{X} - \boldsymbol{\mu_i}(t)] \\ &= \frac{1}{n_i(t)} \sum_{\boldsymbol{X} \in C_i(t)} \boldsymbol{X}\boldsymbol{X}^\intercal - \boldsymbol{\mu_i}(t)\boldsymbol{\mu_i}(t)^\intercal \end{aligned} \tag{2.7}$$

By futher simplication, we have the following

$$\begin{aligned} n_i(t)(\boldsymbol{\Sigma}_i(t) + \boldsymbol{\mu_i}(t)\boldsymbol{\mu_i}(t)^\intercal) &= \sum_{\boldsymbol{X} \in C_i(t-1)} \boldsymbol{X}\boldsymbol{X}^\intercal + \boldsymbol{X}_t\boldsymbol{X}_t^\intercal \\ &= n_i(t-1)(\boldsymbol{\Sigma}_i(t-1) + \boldsymbol{\mu_i}(t-1)\boldsymbol{\mu_i}(t-1)^\intercal) + \boldsymbol{X}_t\boldsymbol{X}_t^\intercal \\ \boldsymbol{\Sigma}_i(t) + \boldsymbol{\mu_i}(t)\boldsymbol{\mu_i}(t)^\intercal &= \frac{n_i(t-1)}{n_i(t)}(\boldsymbol{\Sigma}_i(t-1) + \boldsymbol{\mu_i}(t-1)\boldsymbol{\mu_i}(t-1)^\intercal) + \frac{1}{n_i(t)}\boldsymbol{X}_t\boldsymbol{X}_t^\intercal \end{aligned} \tag{2.8}$$

Now, by substituting the update rule for $\boldsymbol{\mu}_i(t)$ on the left hand side of 2.8, it can be shown that the updated variance is given by,

$$\boldsymbol{\Sigma}_i(t) = (1 - \beta_i(t))[\boldsymbol{\Sigma}_i(t-1) + \beta_i(t)(\boldsymbol{X}_t - \boldsymbol{\mu_i}(t))(\boldsymbol{X}_t - \boldsymbol{\mu_i}(t))^\intercal] \tag{2.9}$$

With no loss in generality, for higher dimensional data, we modify the covariance matrix to be a diagonal matrix making all the off-diagonal elements zero. As our purpose is only to find an approximate fit on the data and not the exact shape, the orientation of the ellipsoids donot matter much. Moreover, this reduces the number of computations making the algorithm faster.

Create New Cluster – A special case arises when $\zeta_{host} = \emptyset$. This essentially means that the data point does not belong to any of the existing clusters with a certain minimum membership value. Therefore, all the existing clusters belong to $\zeta_{nonhost}$. In such a case, a new cluster $C_i(t) = \{\boldsymbol{X}_t, \boldsymbol{\Sigma}_{init}, \alpha_t\}$ is created. Practically, if there is a limit on the maximum number of permissible clusters, then the new cluster replaces a cluster from the set $\zeta(t-1)$ which has the minimum weight.

Merge Clusters – As data points keep streaming in, the clusters drift around in space. Due to the drift, two clusters may come closer to each other by a certain minimum average distance, causing them to combine and form a single cluster. To determine the clusters eligible to merge, we create a graph with each cluster mean as a node and the edges being the average Mahalonbish distances between them given by,

$$D(C_i, C_j) = \frac{1}{2}(\boldsymbol{\mu_i} - \boldsymbol{\mu_j})^\intercal(\boldsymbol{\Sigma_i^{-1}} + \boldsymbol{\Sigma_j^{-1}})(\boldsymbol{\mu_i} - \boldsymbol{\mu_j}) \tag{2.10}$$

where i &j are cluster indices to be merged.

Now, the smallest edge is determined and is compared with a merging threshold, which is a user defined constant. The pair of nodes which satisfy the mentioned criterion, become eligible for merging.

To deduce the equations for the parameters of the merged cluster, let us consider two clusters $C1 = \{\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, \pi_1\}, C2 = \{\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2, \pi_2\}$. Let us denote the number of points in C_1 by N_1 and C_2 by N_2.

$$\begin{aligned}
\mu_{combined} &= \frac{\sum_{\boldsymbol{X} \in C_1 \bigcup C_2} \boldsymbol{X}}{N_{combined}} \\
&= \frac{\sum_{\boldsymbol{X} \in C_1} \boldsymbol{X} + \sum_{\boldsymbol{X} \in C_2} \boldsymbol{X}}{N_1 + N_2} \\
&= \frac{N_1 \boldsymbol{\mu}_1 + N_2 \boldsymbol{\mu}_2}{N_1 + N_2} \\
&= \frac{\pi_1 N \boldsymbol{\mu}_1 + \pi_2 N \boldsymbol{\mu}_2}{\pi_1 N + \pi_2 N} \\
&= \frac{\pi_1 \boldsymbol{\mu}_1 + \pi_2 \boldsymbol{\mu}_2}{\pi_1 + \pi_2}
\end{aligned} \tag{2.11}$$

where N is the total number of points till t^{th} iteration.

Similarly, the combined variance is given by,

$$\begin{aligned}
\boldsymbol{\Sigma_{combined}} &= \frac{\sum_{\boldsymbol{X} \in C_1 \bigcup C_2} \boldsymbol{X}\boldsymbol{X}^\intercal}{N_{combined}} - \boldsymbol{\mu_{combined}}\boldsymbol{\mu}_{\boldsymbol{combined}}^\intercal \\
&= \frac{\sum_{\boldsymbol{X} \in C_1} \boldsymbol{X}\boldsymbol{X}^\intercal + \sum_{\boldsymbol{X} \in C_2} \boldsymbol{X}\boldsymbol{X}^\intercal}{N_1 + N_2} - \boldsymbol{\mu_{combined}}\boldsymbol{\mu}_{\boldsymbol{combined}}^\intercal \\
&= \frac{N_1(\boldsymbol{\Sigma}_1 + \boldsymbol{\mu}_1\boldsymbol{\mu}_1^\intercal) + N_2(\boldsymbol{\Sigma}_2 + \boldsymbol{\mu}_2\boldsymbol{\mu}_2^\intercal)}{N_1 + N_2} - \boldsymbol{\mu_{combined}}\boldsymbol{\mu}_{\boldsymbol{combined}}^\intercal \\
&= \frac{\pi_1 N \boldsymbol{\Sigma}_1 + \pi_2 N \boldsymbol{\Sigma}_2}{\pi_1 N + \pi_2 N} + \frac{\pi_1 N \boldsymbol{\mu}_1\boldsymbol{\mu}_1^\intercal + \pi_2 N \boldsymbol{\mu}_2\boldsymbol{\mu}_2^\intercal}{\pi_1 N + \pi_2 N} - \boldsymbol{\mu_{combined}}\boldsymbol{\mu}_{\boldsymbol{combined}}^\intercal \\
&= \frac{\pi_1 \boldsymbol{\Sigma}_1 + \pi_2 \boldsymbol{\Sigma}_2}{\pi_1 + \pi_2} + \frac{\pi_1 \boldsymbol{\mu}_1\boldsymbol{\mu}_1^\intercal + \pi_2 \boldsymbol{\mu}_2\boldsymbol{\mu}_2^\intercal}{\pi_1 + \pi_2} - \boldsymbol{\mu_{combined}}\boldsymbol{\mu}_{\boldsymbol{combined}}^\intercal
\end{aligned} \tag{2.12}$$

Finally, the combined weight is the sum of the weights of the two clusters, $\pi_{combined} = \pi_1 + \pi_2$

2.2 Practical Considerations

This section talks about some changes that were made in the algorithm, within the mathematical framework discussed above, from a practical implementation perspective.

Learning Rate – In a practical applications setting, $\alpha_t = \frac{1}{t}$ will not work as $\alpha_t \to 0$ as $t \to \infty$. Now, $\boldsymbol{\mu}_t = (1-\alpha_t)(\boldsymbol{\mu}_{t-1}) + \alpha_t x_t$ i.e., $\boldsymbol{\mu}_t$ is a linear combination of $\boldsymbol{\mu}_{t-1}$ and $\boldsymbol{X}_t$ where α_t decides the update factor of $\boldsymbol{X}_t$.If we set $\alpha_t = \frac{1}{t}$ then for large t, α_t is very small and $\boldsymbol{\mu}_{t-1}$ will not be updated after some time. Thus, it is better to clip the value of α_t to some finite value α.

$$\alpha_t = \begin{cases} \frac{1}{t} & ; t \leq t_c \\ \alpha & ; t > tc \end{cases} \tag{2.13}$$

where, $t_c = round(1/\alpha)$

Now, if we look at the update rule for $\boldsymbol{\mu}$

$$\boldsymbol{\mu}_j(t) = \left(1 - \frac{\alpha_t}{\pi_j(t)}\right)\boldsymbol{\mu}_j(t-1) + \frac{\alpha(t)}{\pi_j(t)}\boldsymbol{X}_t \tag{2.14}$$

Simply replacing α_t by α will cause a lot of trouble as $\frac{\alpha}{\pi_j(t)}$ may become greater than one. Whereas, our condition is $\beta_j(t) = \frac{\alpha_t}{\pi_j(t)} \leq 1$. Therefore, to satisfy the condition, we redefine $\beta_j(t)$ as a membership function. A standard membership function could be,

$$\beta_j(t) = exp\left(-\frac{1}{2d}(\boldsymbol{X}_t - \boldsymbol{\mu_j(t)})^\intercal \boldsymbol{\Sigma_j^{-1}(t)}(\boldsymbol{X}_t - \boldsymbol{\mu_j}(t))\right) \tag{2.15}$$

Multiple Iterations, Partitioning and Order of Streaming Data – While performing experiments, it was found that the algorithm sometimes suffers from the cluster drift problem. As the data points keep streaming in, the clusters tend to drift away from their intitial positions, resulting in generation of orphan points. These are the points which initially belonged to a certain cluster but as a result of the drift, they no longer belong to that cluster.

To avoid this, multiple iterations of the algorithm can be performed on the entire dataset which stabilizes the clusters. However, this is applicable only if the database is available offline.

The algorithm is also sensitive to the order in which the data points stream in. A random shuffling of the data points in each iteration can also stabilize the clusters.

A scheme was developed in which the entire dataset is spilt into multiple segments and the order in which the segments are loaded is chosen in random in each iteration. This provides the required randomness in data streaming and also brings down the memory required in loading the entire dataset which is especially useful in case of big data.

2.3 Visualization

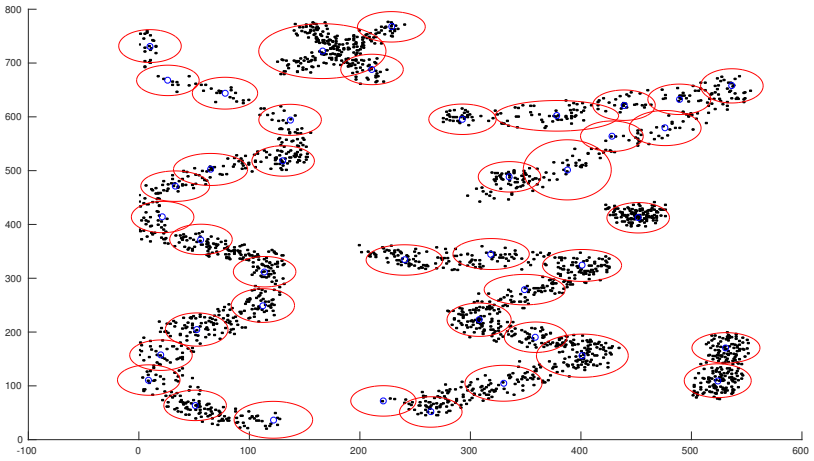

FIGURE 2.1: Incremental Clustering Output on Toy Dataset 1
Default Variance = 50

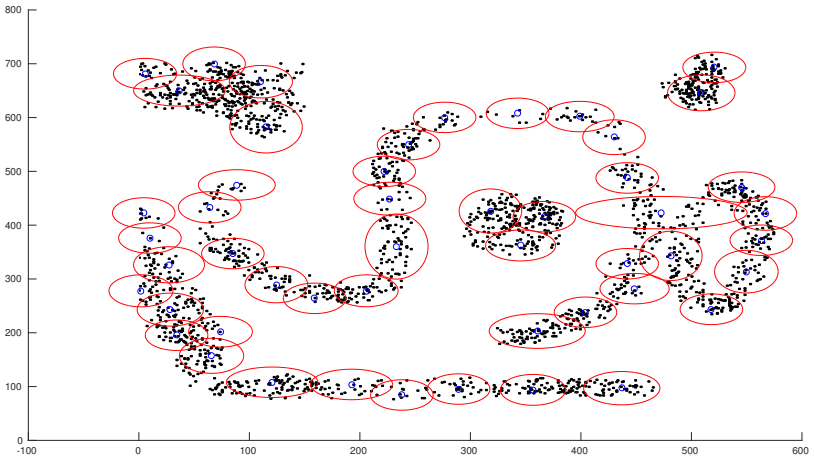

FIGURE 2.2: Incremental Clustering Output on Toy Dataset 2
Default Variance = 50

Effects of Parameter Variations

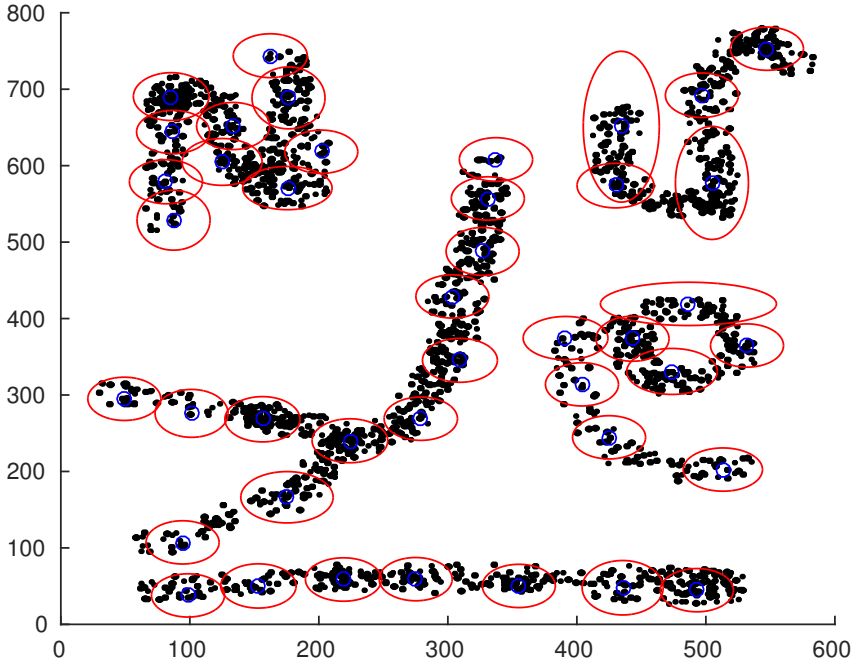

FIGURE 2.3: Incremental Clustering with Merging Output on Toy Dataset 3
Default Variance = 50 *Number of Clusters* = 42

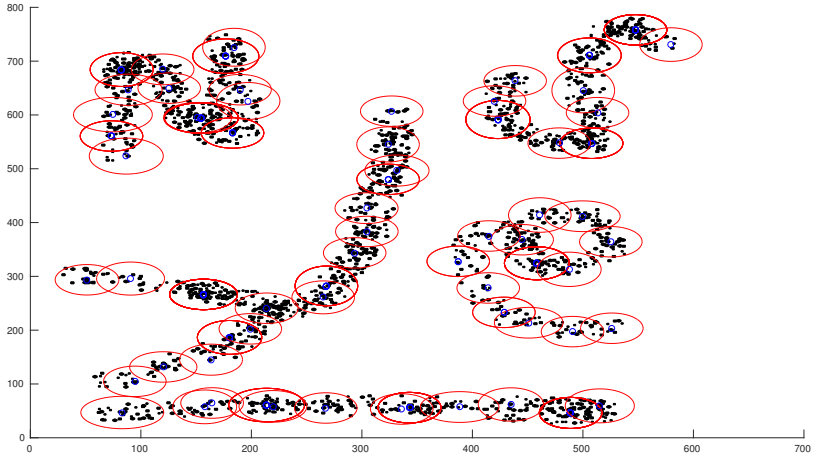

FIGURE 2.4: Incremental Clustering without Merging Output on Toy Dataset 3
Default Variance = 50 *Number of Clusters* = 152

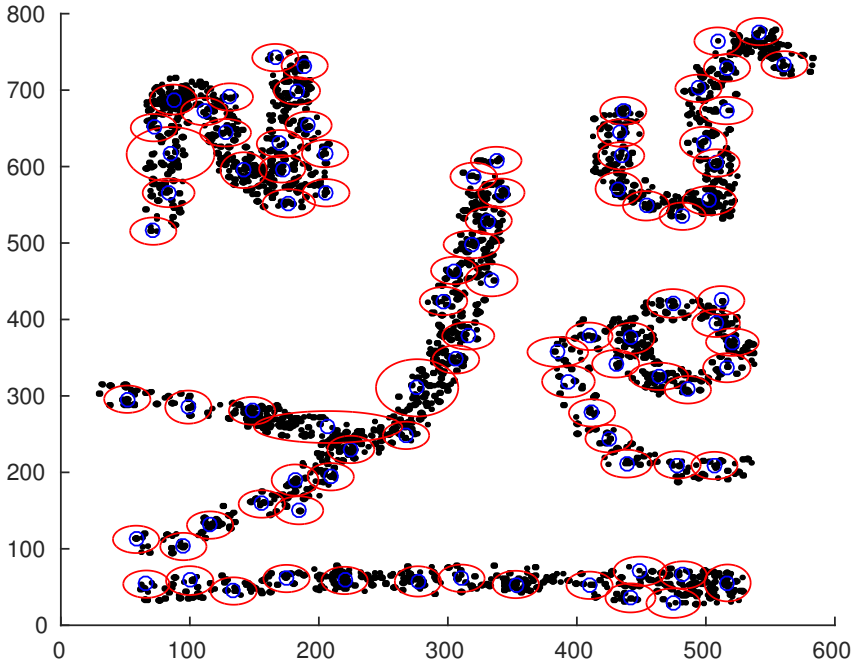

FIGURE 2.5: Incremental Clustering with Merging Output on Toy Dataset 3
Default Variance = 20 *Number of Clusters* = 83

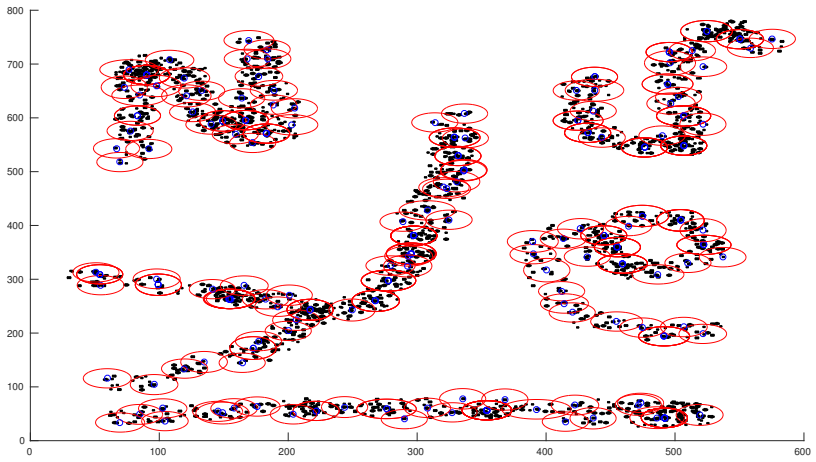

FIGURE 2.6: Incremental Clustering without Merging Output on Toy Dataset 3
Default Variance = 20 *Number of Clusters* = 235

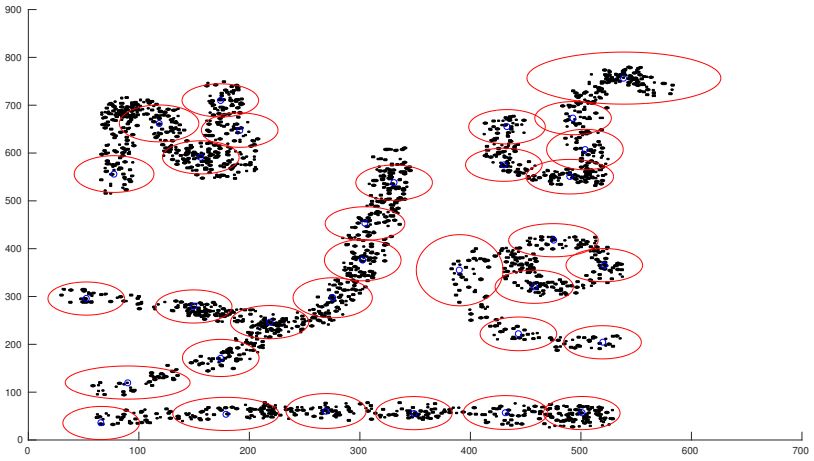

FIGURE 2.7: Incremental Clustering with Merging Output on Toy Dataset 3
Default Variance = 75 *Number of Clusters* = 35

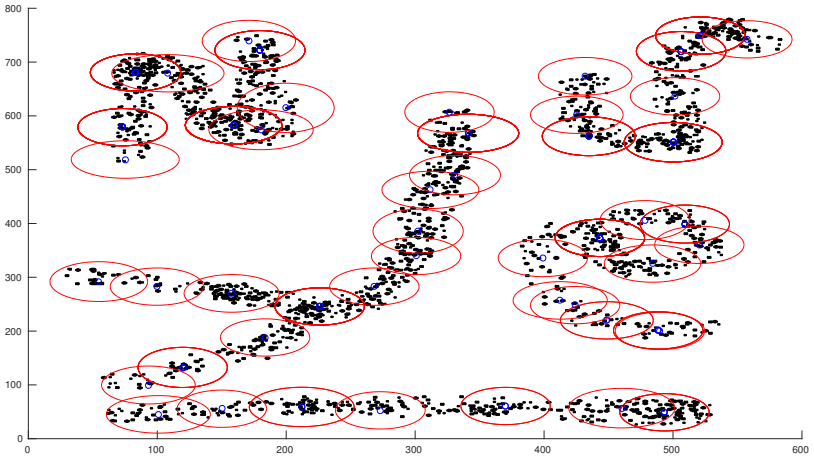

FIGURE 2.8: Incremental Clustering without Merging Output on Toy Dataset 3
Default Variance = 75 *Number of Clusters* = 147

2.4 Discussion

The algorithm discussed above turns out to be a very useful method that can be applied as a pre-processing technique to other clustering algorithms like k-means which is initialized with random means. The cluster means that are generated by the proposed algorithm can be used as the initial means. These means, being stabilzed after multiple iterations, would possibly result in faster convergence of k-means.

Also, this algorithm can be used as a pre-processing technique for DBSCAN clustering method, which has been discussed in detail in chapter 4.

Certain aspects of the algorithm that came up while implementation and experimentation are discussed below.

- **Default Variance** provides a lower limit to the size of any given cluster. Setting it to a very low value would result in uncontrolled shrinkage of clusters and therefore generate too many clusters whereas a very high value may result in too few clusters. An optimum value has to be chosen from domain knowledge or based on experimentation. For offline data, non-parametric bandwidth estimation techniques can be chosen which estimates the variance of the data in a single pass, which can then be set as the default value.

  A rule of thumb estimator, known as Silverman's(1986) estimator is given by

  $$h = \left(\frac{4\hat{\sigma}^5}{3n}\right)^{-1/5} \approx 1.06\hat{\sigma}n^{-1/5} \tag{2.16}$$

  where $\hat{\sigma}$ is the standard deviation of the data and n is the total number of data points.

- **Merging** after each iteration may become infeasible as the scheme computes pairwaise distances between all clusters and will slow the system down for large values of K. Therefore, merging needs to be done at certain intervals.

Now, the interval can be a user-defined constant, for example, after every 500 iterations or so. In this case, the optimum value can be chosen experimentally. Or else, the iterval can be chosen adaptively by keeping track of how fast the cluster means are changing their positions with respect to each other. The reciprocal of the average change can give us the average number of iterations after which merging should be applied.

Incremental Clustering [Membership Weighted]

Algorithm 1 Incremental Clustering [membership weighted]

1: **Initialization**
$\mathbb{C}_1(1) = \{\ \boldsymbol{\mu}_1(1) = \boldsymbol{X}_1, \boldsymbol{\Sigma}_1(1) = \boldsymbol{\Sigma}_d, \pi_1(1) = 1.0\ \}$
2: **Iteration** $\mathbb{C}_1(t-1), \ldots \mathbb{C}_K(t-1);\ \boldsymbol{X}_t$
Compute $d_j(t) = (\boldsymbol{X} - \boldsymbol{\mu_j})^\intercal \boldsymbol{\Sigma_j^{-1}} (\boldsymbol{X} - \boldsymbol{\mu_j})\ ; \forall j = 1, \ldots K$
Let, $S_H(t) = \{r : d_j(t) \leq \lambda^2\}$
Penalize the weights of all non- host clusters
i.e. $\forall l \notin S_H(t)\ \pi_l(t) = (1 - \alpha_t)\pi_l(t-1)$
Case I : $S_H(t) \neq \emptyset$
Update mean, variance and weight $\forall r \in S_H(t)$
$\beta_r(t) = exp\left(-\frac{1}{2d} (\boldsymbol{X_t} - \boldsymbol{\mu_r(t)})^\intercal \boldsymbol{\Sigma_r^{-1}(t)} (\boldsymbol{X_t} - \boldsymbol{\mu_r(t)}) \right)$ [Or any membership function of your choice]
Mean Update, $\boldsymbol{\mu_r}(t) = (1 - \beta_i(t))\boldsymbol{\mu_i}(t-1) + \beta_i(t)\boldsymbol{X}_t$
Variance Update, $\boldsymbol{\Sigma}_r(t) = (1 - \beta_r(t))[\boldsymbol{\Sigma}_r(t-1) + \beta_r(t)(\boldsymbol{X}_t - \boldsymbol{\mu_r}(t))(\boldsymbol{X}_t - \boldsymbol{\mu_r}(t))^\intercal]$
if $\boldsymbol{\Sigma}_r(t) < \ \boldsymbol{\Sigma}_d$ then $\boldsymbol{\Sigma}_r(t) = \boldsymbol{\Sigma}_d$ [Clip the variance to $\boldsymbol{\Sigma}_d$]
Weight Update, $\pi_r(t) = (1 - \alpha_t)\pi_r(t-1) + \alpha_r(t)$
where, $\alpha_r(t) = \alpha_t \frac{\beta_r(t)}{\sum_{l \in S_H} \beta_l(t)}$
Case II : $S_H(t) = \emptyset$
Create new cluster, $\mathbb{C}_{K+1}(t) = \{\boldsymbol{\mu}_{K+1}(t) = \boldsymbol{X}_t, \boldsymbol{\Sigma}_{K+1}(t) = \boldsymbol{\Sigma}_d, \pi_{(K+1)}(t) = \alpha_t\ \}$
3: **Merging**
Compute $d_{ij} = \frac{1}{2}(\boldsymbol{\mu_i} - \boldsymbol{\mu_j})^\intercal (\boldsymbol{\Sigma_i^{-1}} + \boldsymbol{\Sigma_j^{-1}})(\boldsymbol{\mu_i} - \boldsymbol{\mu_j}); \forall i = 1, \ldots K; \forall j = i+1, \ldots K;$
Find the minimum, $argmin\ d_{ij}$
Merging Candidate $\mathbb{C}_i, \mathbb{C}_j\ if\ d_{ij} \leq MergingThreshold$
Update the parameters of the merged cluster
Mean Update, $\boldsymbol{\mu_{combined}} = \frac{\pi_1 \boldsymbol{\mu}_1 + \pi_2 \boldsymbol{\mu}_2}{\pi_1 + \pi_2}$
Variance Update, $\boldsymbol{\Sigma_{combined}} = \frac{\pi_1 \boldsymbol{\Sigma}_1 + \pi_2 \boldsymbol{\Sigma}_2}{\pi_1 + \pi_2} + \frac{\pi_1 \boldsymbol{\mu}_1 \boldsymbol{\mu}_1^\intercal + \pi_2 \boldsymbol{\mu}_2 \boldsymbol{\mu}_2^\intercal}{\pi_1 + \pi_2} - \boldsymbol{\mu_{combined}} \boldsymbol{\mu_{combined}^\intercal}$
Weight Update,$\pi_{combined} = \pi_i + \pi_j$

Chapter 3

Hierarchical Incremental k-Means

In this chapter we extend the idea of Incremental k-means, discussed in the previous chapter, to Hierarchical Clustering. The main objective here is to have a tree structure, in which each level will have a set of clusters parameterized by a user defined set of values and in each subsequent level, the clusters formed will be enclosed by the clusters of the previous level. Following the tree nomenclature, the enclosing cluster will be called the *parent node* and the enclosed, *children nodes*.

As this is an extension of *Algorithm 1*, the basic clustering technique remains same and is therefore not discussed in this chapter. The practical implementation details are described below.

The two major operations that are performed are *Classification* and *Update*. Apart from these, cluster *Initialization* is also discussed below.

3.1 Initialization

The process of initialization of a new cluster in HIC involves creation of cluster in each depth level of the cluster tree. The process can be visualized as formation of concentric circles with the radii getting smaller with depth. This means, the cluster formed in each level will have the same mean vector as its parent but will have a different default variance, as defined by the user for the given level.

Three possible cases here are

- *Initialization at root* This case arises as the process of clustering begins with the first data point. A single branch is created at root going down till last level, each cluster being assigned the weight 1.0.
- *Initialization at leaf* This is a special case where only one cluster will be created at the leaf level and will not have the provision to store children.
- *Initialization at an arbitrary non-leaf location* A new set of clusters can be created from a given location in the tree, going down till the last level.

3.2 Classification

- A top-down approach is chosen for classification. As the data points keep streaming in, we start from the root node and sequentially move to all its *child* clusters.
- Here, we determine the host ζ_{host} and non-host $\zeta_{non-host}$ clusters, thereby maintaining a list of both sets.
- The list of ζ_{host} is updated in the following way
  - For a given host cluster, its children are pushed into the list, provided they satisfy the host criterion, and a counter keeps track of the *end of the list*.
  - Another counter is set at the *start of the list* which is incremented after each host cluster in the list is processed.
  - This process continues till *start of the list* and *end of the list* coincide.
- The list of $\zeta_{non-host}$ is updated in the following way
  - Clusters that donot satisfy the host criterion are pushed into the list.
  - For each non-host cluster, all its children are pushed into the list.
  - In a manner similar to the host list, two counters are maintained that store the current position and end of the list.
  - Proceeding this way, the entire sub-tree starting from a given non-host is pushed into the list.

- If the data point does not belong to any of the clusters in a given level, then a new branch is initialized from the host cluster in the previous level to which the data point had the highest membership.
- Remaining clusters in subsequent levels are then classified as non-host.

3.3 Update

- Classification generates a list of host and non-host clusters.
- Host cluster parameters are updated as per equations 2.5, 2.6 and 2.7.
- Non-Host cluster weights are penalized as per equation 2.4.

3.4 Discussion

Hierarchical Incremental Clustering (HIC) can be viewed as a process of quantizing of space. The main aim of our Incremental Clustering (IC) algorithm was to find a set of points that would reperesent a segment of the area in space over which the data is distributed. Mostly, IC is used as a preprocessing step for a more stable clustering algorithm, where it contributes in initializing the algorithm or reducing the search space, et cetra.

But often, we donot know what should be the minimum size of the clusters required for a given application. Therefore, it becomes useful to apply IC with a varying set of cluster sizes. This also lets us look at the spatial distribution of data at multiple resolutions.

In the proposed algorithm, the user defines the default variance for the lowest level and also provides a scaling factor that sets up the default variance for the higher levels.

HIC becomes particularly useful as a pre-processing technique for Incremental DBSCAN, which is discussed in next chapter.

Chapter 4

Incremental DBSCAN

This chapter describes the algorithm of DBSCAN*(Density Based Spatial Clustering of Applications with Noise)* and an incremental version of the same. It further discusses certain drawbacks in the algorithm followed by the proposed improvements.

4.1 DBSCAN

The algorithm DBSCAN was first proposed in [19]. The main idea here is that each point in a cluster should satisfy a certain density constraint. Mathematically, the hyperspherical neighbourhood of a given radius *eps* around the data point, denoted by $\mathcal{N}_{eps}$, must have a minimum number of points *minPts*.

Certain important definitions in this context are

Definition 4.1. (Eps - Neighbourhood) The *Eps-Neighbourhood* of a point $\mathbf{p}$, denoted by $\mathcal{N}_{eps}(\mathbf{p})$ is defined by $\mathcal{N}_{eps}(\mathbf{p}) = \{ \mathbf{q} \in D \mid dist(\mathbf{p}, \mathbf{q}) \leq eps\}$.

Definition 4.2. (Core Point) A point $\mathbf{p}$ that satisfies the condition, $\mathcal{N}_{eps}(\mathbf{p}) \geq minPts$ is defined as a *core point*.

Definition 4.3. (Border Point) A point $\mathbf{q}$ that fails to satisfy the condition, $\mathcal{N}_{eps}(q) \geq minPts$ but lies in the $\mathcal{N}_{eps}(\mathbf{p})$ of a *core point* $\mathbf{p}$, is defined as a *border point*.

Definition 4.4. (Direct Density-Reachability) A point $\mathbf{q}$ is *directly density-reachable* from a point $\mathbf{p}$ w.r.t. $minPts$ if,

1. $\mathbf{q} \in \mathcal{N}_{eps}(\mathbf{p})$
2. $\mathbf{p}$ is a *core point*

Definition 4.5. (Density-Reachability) A point $\mathbf{q}$ is *density-reachable* from a point $\mathbf{p}$ w.r.t. *eps*, *minPts* if there are a chain of points $\mathbf{q}_1, \ldots, \mathbf{q}_n$, $\mathbf{q}_1 = \mathbf{p}$, $\mathbf{q}_n = \mathbf{q}$ such that $\mathbf{q}_{i+1}$ is *directly density-reachable* from $\mathbf{q}_i$

Definition 4.6. (Density Connectivity) A point $\mathbf{q}$ is *density-connected* to a point $\mathbf{p}$ w.r.t. *eps*, *minPts* if there exists a point $\mathbf{o}$ such that both $\mathbf{q}$ and $\mathbf{p}$ are *density-reachable* from $\mathbf{o}$

Definition 4.7. (Cluster) Let D be a database of points. A *cluster* C w.r.t. *eps*, *minPts* is defined as a non-empty subset of D satisfying the following

1. $\forall\ \mathbf{p}, \mathbf{q} :\ if\ \mathbf{p} \in C$ and $\mathbf{q}$ is *density-reachable* from $\mathbf{p}$ then $\mathbf{q} \in C$.
2. $\forall\ \mathbf{p}, \mathbf{q} \in C :\ \mathbf{p}$ is *density-connected* to $\mathbf{q}$ w.r.t. *eps*, *minPts*.

The first point in the above definition states that any point that is *density-reachable* from a *core point* in a cluster, will also be a member of the cluster. Therefore, in a cluster, all core points will be *density-reachable* from one another.

The second point states that any given pair of points in a cluster must be atleast *density-connected*. This implies that, each cluster will have *border points* which earn their position in the cluster by being in the $\mathcal{N}_{eps}$ of a *core point* of the cluster.

Definition 4.8. (Noise) Let $C_1, \ldots, C_k$ be the clusters in a database D. Then, *noise* is defined as the set of points in D, not belonging to any of the clusters C_i, i.e. $noise = \{\ \mathbf{p} \in D \mid \forall\ i :\ \mathbf{p} \notin C_i\ \}$.

The basic version of the algorithm is explained below.

- The clustering starts by labeling all points as *noise* initially.
- Then, the first point is picked and its $\mathcal{N}_{eps}$ is retrieved.
- If the point is found to be a *core point* then its $\mathcal{N}_{eps}$ is pushed in a queue data structure. A new cluster label is assigned to the point.
  - Then, after each pop operation from the queue, the point retrieved is checked for its status.

  - If its a core point, then its $\mathcal{N}_{eps}$ is retrieved and pushed into the queue and the current point is assigned the label of the current cluster.
  - If the point is not a core point then it becomes a border point, by definition, as it belongs to the $\mathcal{N}_{eps}$ of a core point, and it is assigned the current cluster label.
  - This process is repeated until the queue is empty.

- If the queue is empty, the next point is processed, following the same steps, provided the point is labelled as *noise*.

4.2 Incremental DBSCAN

The extension of DBSCAN to streaming data, called Incremental DBSCAN was proposed in [23]. The algorithm assumes that offline DBSCAN has been implemented on a small subset of the database D to be clustered, producing a set of clusters $C_1, \ldots, C_k$. Under this assumption, the algorithm performs two major operations, *insertion* of a new point from database D to the cluster set and *deletion* of an existing point from the cluster set. Here, we are only concerned with *insertion*, which is described below.

To add a point to the existing cluster set, the main objective is to find all points in its $\mathcal{N}_{eps}$ in the existing set. Among its $\mathcal{N}_{eps}$, all such points are located, whose status will change from *non-core* to *core* after the *insertion* as only the $\mathcal{N}_{eps}$ of such points will be directly affected as a result of the *insertion* operation, as they will now be in the $\mathcal{N}_{eps}$ of a *core point* and therefore eligible to be in a cluster. All the affected points are stored in a list *UpdSeedIns*.

- For a given point $\mathbf{p}$ to be clustered, a region query is performed to retrieve its $\mathcal{N}_{eps}$ from the existing cluster set and stored in *UpdSeedIns.*.
- Points in $\mathcal{N}_{eps}(\mathbf{p})$, whose $\mathcal{N}_{eps}$ have $minPts-1$ number of points are located. These points would become *core points* after insertion.
- Further, the $\mathcal{N}_{eps}$ of these points are also retrieved and stored in *UpdSeedIns*.
- Depending on the type of points in *UpdSeedIns*, four possible cases may arise.

- **Noise** If *UpdSeedIns* is empty, there are no "new" core points formed as a result of *insertion*. Therefore, the new point is labelled as noise and nothing else changes.
- **Creation** If *UpdSeedIns* contains only *core points* which did not belong to any cluster previously, then a new cluster is created with the set *UpdSeedIns*.
- **Absorption** If *UpdSeedIns* contains *core points* which belong to a single cluster, then the new point is also absorbed into that cluster and assigned the label of that cluster.
- **Merging** If *UpdSeedIns* contains *core points* which belong to multiple clusters, then all the clusters are merged into a single cluster.

Drawbacks – The drawbacks of the above algorithm are listed below

- To retrieve the $\mathcal{N}_{eps}$ of a given point, the region query is performed over the entire set of clustered points. For a large dataset, the search space would be quite big.
- It also requires application of offline DBSCAN to a subset of the database to be clustered.

4.3 Proposed Solution

The proposed method aims at reducing the search space, to retrieve $\mathcal{N}_{eps}$ of streaming data points. Also, it does not require offline DBSCAN to be implemented initially.

Pre-Clustering – As a pre-clustering step, the incremental k-means algorithm, discussed in chapter 3, is applied on a set of data points from D(say, 10% of the total size, randomly sampled). It segments the spatial distribution of the data using multiple hyper-ellipsoidal blobs and generates a *mean vector* for each blob. Further, for each point, top K host cluster ids, are also stored.

Main Clustering – The proposed algorithm clusters the data in two passes. In the first pass, the *eps* neighbourhood of each point is retrieved and stored. In

the second pass, the points are assigned cluster labels based on connectivity, as explained below.

Region Query – Once the host clusters are determined, the data points present in the host clusters become the new search space for the current data point. To retrieve all points in the $\mathcal{N}_{eps}$ of the current point, the following method is applied.

- Each cluster is assumed to be a hyper-cuboid with the mean vector being the centre.
- Taking the centre to be the zero vector, the cluster is made to resemble a rectangular coordinate system and all cluster members are assigned new coordinates by subtracting the mean.
- Further, the current data point is also assigned a new coordinate by mean subtraction.
- Now, all points that lie within a hyper-cuboidal bounding box of side *eps* around the data point is found, simply, by dimension wise comparison.
- Only among the points that satisfy the previous step, an euclidean distance check is performed to get the final set of points in the hyper-spherical $\mathcal{N}_{eps}$ of the current point.

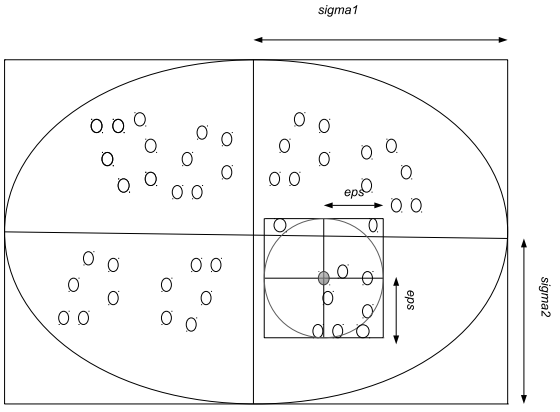

FIGURE 4.1: Illustration of Region Query: Assume host cluster to be a hyper-cuboid. Substract mean to assign coordinates. Find *eps* neighbourhood of given point in a rectangular window of size 2*eps*

This method reduces the number of distance computations for a given point.

Label Assignment – Proceeding in the manner described above, a table is populated, which stores the number of *eps* neighbours of a point, its status as *core, border* or *noise*, the neighbouring point indices and the cluster label.

The label assignment process is explained below.

- Initially, status of all points except the *core points* are set as *noise* and all labels are set to 0, which is equivalent to noise.
- Now each point is processed in a sequence.
- If the point is found to be a *core point*, but with a label 0, it is stored in a list. A new cluster label is assigned to it.
  - Next, all its neighbours are pushed into the list. A *start of the list* and an *end of the list* counter is maintained which point to the current neighbour to be processed and the last entry in the list respectively.
  - After a point is processed, its neighbours are pushed into the list.
  - The processsed point is assigned the current cluster label and if it is found to be *non-core*, its status is updated to *border*.
  - The process continues till the two counters coincide.
- All points that are not *core points* are skipped.

Clustering Unseen Points – The algorithm till now clusters an initial set of points from D. Now, as a new data point which is unseen to both the algorithms, comes in, it is first passed through the incremental k-means algorithm, where the host clusters are determined and then passed to the DBSCAN algorithm as a reduced search space for the point. Also, the cluster set maintained by the incremental k-means is updated.

The point is now inserted in the existing incremental DBSCAN cluster set by following the same steps as described in the previous section.

In a special case, if the point does not find any host cluster in the first step, then it is marked as noise in DBSCAN as well.

4.4 Visualization

For the purpose of visualization, we generated two dimensonal toy datasets which are black and white images having random shapes made using spray paint tool in an image editing software. These images are then loaded in memory and the cartesian coordinates of the black pixels are provided as data points.

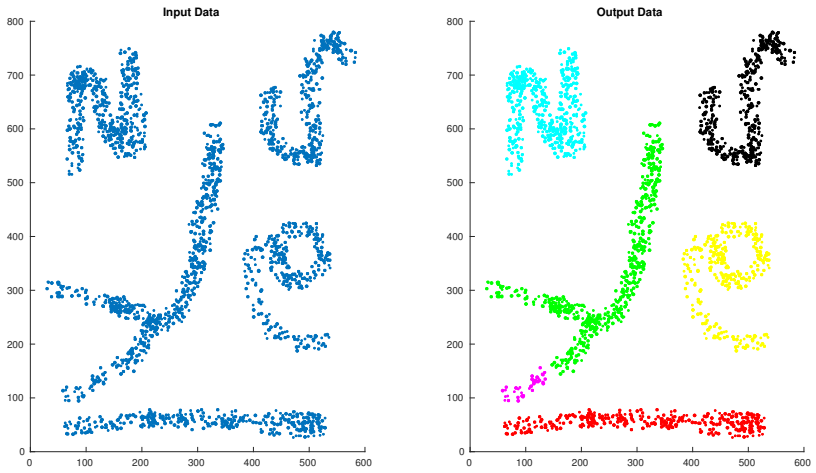

FIGURE 4.2: Incremental DBSCAN Output on Toy Dataset 1
$eps = 16$ $minPts = 5$

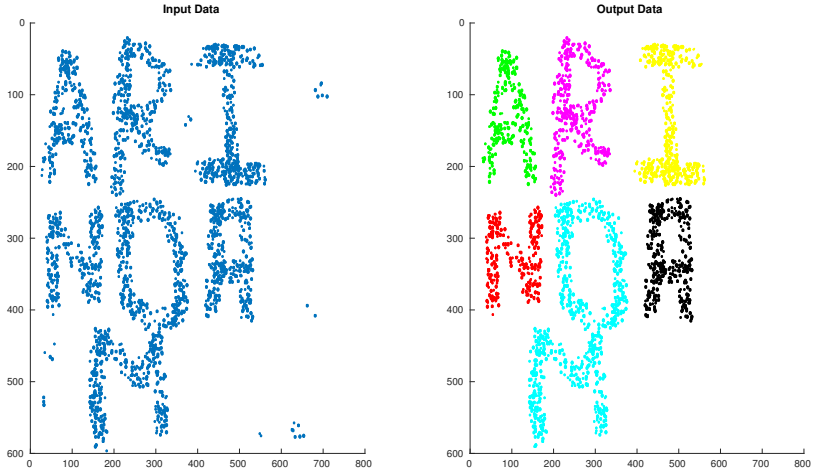

FIGURE 4.3: Incremental DBSCAN Output on Toy Dataset 1
$eps = 14$ $minPts = 5$

As shown in Fig 4.2, the algorithm effectively detects clusters and removes noise.

4.5 Discussion

In our proposed solution, the number of host clusters K for each data point needs to be determined experimentally. But this would be a difficult task as the value of K may be different for all points, depending on how dense the spatial distribution of data is around the given point.

To overcome this, we find a simple solution of setting the default variance v_d in pre-clustering step to be c times eps, where c is a positive integer greater than 1. Setting an optimum value of c will make sure that the minimum cluster size of all clusters will be greater than the size of $\mathcal{N}_{eps}$ of the data point and therefore it will be sufficient to choose the first few host clusters, to which the point has higher membership, for region query. From our experiments, we found that top 3 host clusters covered the entire $\mathcal{N}_{eps}$ of a given point.

To find an optimum value of c we can replace Incremental k-means by its hierarchical extension, in which each level can have a default variance that is $c \times v_d$, with the value of c increasing at a chosen rate as we go up from the leaf nodes to the root.

The authors suggest a heuristic approach to determine the values of eps and $minPts$.

For each point in the database D, the distance d of its k^{th} nearest neighbour is computed. Therefore, the point shall have atleast $k+1$ points within a sphere of radius d. This distance is denoted as $k-dist$. All the points in D are sorted based on their $k-dist$ and a graph is plotted for the same in which x-axis denotes point indices and y-axis, the correponding $k-dist$. Points with the highest $k-dist$ values are considered noise and the value of $k-dist$ at the first valley in the curve is chosen as eps. And, k is chosen as $minPts$.

The authors found experimentally that, $k-dist$ graphs do not vary much for values of k greater than 4, for two dimensional data. Therefore, $minPts$ can be set to 4 and the corresponding $4-dist$ is chosen as eps.

Chapter 5

Conclusion

We have integrated two incremental clustering algorithms to use the advantages of each while overcoming their individual drawbacks. The incremental K-means algorithm assumes hyper-ellipsoidal cluster structures and estimates their mean and bandwidth parameters. A new point can be easily localized and inserted through the computation of a relatively smaller set of Mahalanobis distance computations. Incremental DBSCAN, on the other hand, makes no assumption on cluster structures and hence, provides better clustering outputs. However, localizing a new point needs distance computations with a large number of existing points. Our proposal aims at reducing this search by first finding a host cluster from incremental K-means cluster set. Distances from members of this host cluster are only computed to localize the new point in DBSCAN cluster set. This heavily reduces the search required for indexing new points in DBSCAN. We have also developed a hierarchical version of incremental K-means for efficient localization of a new point.

The immediate future extension of this work will involve the integration of the present framework with incremental hierarchical DBSCAN. Also, this framework can be applied to unsupervised or semi-supervised classification applications like event discovery, image/video indexing etc.

Bibliography

[1] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[2] Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[3] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[4] Charu C Aggarwal and Chandan K Reddy. *Data clustering: algorithms and applications*. CRC press, 2013.

[5] Dragomir R Radev, Hongyan Jing, Małgorzata Styś, and Daniel Tam. Centroid-based summarization of multiple documents. *Information Processing & Management*, 40(6):919–938, 2004.

[6] Doug Beeferman and Adam Berger. Agglomerative clustering of a search engine query log. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 407–416. ACM, 2000.

[7] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[8] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[9] Thomas Schultz and Gordon L Kindlmann. Open-box spectral clustering: applications to medical image analysis. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2100–2108, 2013.

[10] Jeff A Bilmes et al. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute*, 4(510):126, 1998.

[11] Mehmet Gönen and Adam A Margolin. Localized data fusion for kernel k-means clustering with application to cancer biology. In *Advances in Neural Information Processing Systems*, pages 1305–1313, 2014.

[12] Maoguo Gong, Yan Liang, Jiao Shi, Wenping Ma, and Jingjing Ma. Fuzzy c-means clustering with local information and kernel metric for image segmentation. *IEEE Transactions on Image Processing*, 22(2):573–584, 2013.

[13] Linhong Zhu, Aram Galstyan, James Cheng, and Kristina Lerman. Tripartite graph clustering for dynamic sentiment analysis on social media. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1531–1542. ACM, 2014.

[14] Mohsen JafariAsbagh, Emilio Ferrara, Onur Varol, Filippo Menczer, and Alessandro Flammini. Clustering memes in social media streams. *Social Network Analysis and Mining*, 4(1):237, 2014.

[15] Wu He, Shenghua Zha, and Ling Li. Social media competitive analysis and text mining: A case study in the pizza industry. *International Journal of Information Management*, 33(3):464–472, 2013.

[16] Fazli Can. Incremental clustering for dynamic information processing. *ACM Transactions on Information Systems (TOIS)*, 11(2):143–164, 1993.

[17] Chetan Gupta and Robert Grossman. Genic: A single pass generalized incremental algorithm for clustering. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pages 147–153. SIAM, 2004.

[18] Edwin Lughofer. Extensions of vector quantization for incremental clustering. *Pattern Recognition*, 41(3):995–1011, 2008.

[19] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases ‘ with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[20] Shuigeng Zhou, Aoying Zhou, Wen Jin, Ye Fan, and Weining Qian. Fdbscan: a fast dbscan algorithm. *Ruan Jian Xue Bao*, 11(6):735–744, 2000.

[21] B Borah and DK Bhattacharyya. An improved sampling-based dbscan for large spatial databases. In *Intelligent Sensing and Information Processing, 2004. Proceedings of International Conference on*, pages 92–96. IEEE, 2004.

[22] Derya Birant and Alp Kut. St-dbscan: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, 60(1):208–221, 2007.

[23] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Michael Wimmer, and Xiaowei Xu. Incremental clustering for mining in a data warehousing environment. In *VLDB*, volume 98, pages 323–333, 1998.