# INVESTIGATING CHURN DETECTION IN DYNAMIC NETWORKS

by

Serdar Baran Tatar

Submitted in partial fulfillment of the
requirements for the degree of
Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
June 2015

*To my Mom and Dad for their unwavering support throughout my education.*

## Table of Contents

**List of Figures** . . . . . **v**

**List of Tables** . . . . . **viii**

**Abstract** . . . . . **x**

**List of Abbreviations and Symbols Used** . . . . . **xi**

**Acknowledgements** . . . . . **xiii**

**Chapter 1 Introduction** . . . . . **1**

**Chapter 2 Literature Survey** . . . . . **3**

**Chapter 3 Methodology** . . . . . **6**

3.1 CluStream . . . . . 9

3.1.1 Online Phase . . . . . 13

3.1.2 Offline Phase . . . . . 14

3.2 DenStream . . . . . 15

3.2.1 Core-micro-cluster . . . . . 16

3.2.2 Potential core-micro-cluster . . . . . 16

3.2.3 Outlier-micro-cluster . . . . . 17

3.3 ClusTree . . . . . 18

3.4 Flockstream . . . . . 23

3.4.1 MSF Rules . . . . . 25

3.4.2 Initialization . . . . . 29

3.4.3 Maintenance and Clustering . . . . . 29

**Chapter 4 Evaluation** . . . . . **33**

4.1 Parameterization . . . . . 33
4.2 Evaluation Metrics . . . . . 34
4.3 Parameter Sensitivity Analysis for Flockstream . . . . . 36
4.3.1 Electricity Data Set . . . . . 37
4.3.2 Forest Covertype Data Set . . . . . 39
4.3.3 Commercial-Small Data Set . . . . . 40
4.3.4 KDD 2009 Churn Data Set . . . . . 44
4.4 Discussion on the Performances of Algorithms Employed . . . . . 45

**Chapter 5 Conclusions and the Future Work** . . . . . **49**

**Bibliography** . . . . . **51**

**Appendix A MOA Result Figures** . . . . . **55**

A.1 Electricity . . . . . 55
A.2 Covertype . . . . . 60
A.3 KDD 2009 Churn . . . . . 64
A.4 Commercial-Small . . . . . 69
A.5 Commercial-Big . . . . . 73

## List of Figures


Figure 3.1 Class Distribution of Data Sets . . . . . 9

Figure 3.2 ClusTree Flowchart . . . . . 22

Figure 3.3 Visibility and Defense Range of an Agent . . . . . 24

Figure 3.4 3D Representation of Toroidal Virtual Space . . . . . 24

Figure 3.5 Alignment . . . . . 26

Figure 3.6 Separation . . . . . 27

Figure 3.7 Cohesion . . . . . 28

Figure A.1 MOA Results of DenStream on Electricity Data Set: F1-P . . 55

Figure A.2 MOA Results of DenStream on Electricity Data Set: F1-R . . 56

Figure A.3 MOA Results of DenStream on Electricity Data Set: Purity . 56

Figure A.4 MOA Results of CluStream on Electricity Data Set: F1-P . . 57

Figure A.5 MOA Results of CluStream on Electricity Data Set: F1-R . . 57

Figure A.6 MOA Results of CluStream on Electricity Data Set: Purity . . 58

Figure A.7 MOA Results of ClusTree on Electricity Data Set: F1-P . . . 58

Figure A.8 MOA Results of ClusTree on Electricity Data Set: F1-R . . . 59

Figure A.9 MOA Results of ClusTree on Electricity Data Set: Purity . . . 59

Figure A.10 MOA Results of DenStream on Covertype Data Set: F1-P . . 60

Figure A.11 MOA Results of DenStream on Covertype Data Set: F1-R . . 60

Figure A.12 MOA Results of DenStream on Covertype Data Set: Purity . 61

Figure A.13 MOA Results of CluStream on Covertype Data Set: F1-P . . . 61

Figure A.14 MOA Results of CluStream on Covertype Data Set: F1-R . . 61

Figure A.15 MOA Results of CluStream on Covertype Data Set: Purity . . 62

Figure A.16 MOA Results of ClusTree on Covertype Data Set: F1-P . . . . 62

Figure A.17 MOA Results of ClusTree on Covertype Data Set: F1-R . . . 62

Figure A.18 MOA Results of ClusTree on Covertype Data Set: Purity . . . 63

Figure A.19 MOA Results of DenStream on KDD2009 Data Set: F1-P . . 64

Figure A.20 MOA Results of DenStream on KDD2009 Data Set: F1-R . . 65

Figure A.21 MOA Results of DenStream on KDD2009 Data Set: Purity . . 65

Figure A.22 MOA Results of CluStream on KDD2009 Data Set: F1-P . . . 66

Figure A.23 MOA Results of CluStream on KDD2009 Data Set: F1-R . . . 66

Figure A.24 MOA Results of CluStream on KDD2009 Data Set: Purity . . 67

Figure A.25 MOA Results of ClusTree on KDD2009 Data Set: F1-P . . . . 67

Figure A.26 MOA Results of ClusTree on KDD2009 Data Set: F1-R . . . . 68

Figure A.27 MOA Results of ClusTree on KDD2009 Data Set: Purity . . . 68

Figure A.28 MOA Results of DenStream on Commercial-Small Data Set: F1-P . . . . . 69

Figure A.29 MOA Results of DenStream on Commercial-Small Data Set: F1-R . . . . . 69

Figure A.30 MOA Results of DenStream on Commercial-Small Data Set: Purity . . . . . 70

Figure A.31 MOA Results of CluStream on Commercial-Small Data Set: F1-P 70

Figure A.32 MOA Results of CluStream on Commercial-Small Data Set: F1-R 70

Figure A.33 MOA Results of CluStream on Commercial-Small Data Set: Purity . . . . . 71

Figure A.34 MOA Results of ClusTree on Commercial-Small Data Set: F1-P 71

Figure A.35 MOA Results of ClusTree on Commercial-Small Data Set: F1-R 71

Figure A.36 MOA Results of ClusTree on Commercial-Small Data Set: Purity 72

Figure A.37 MOA Results of DenStream on Commercial-Big Data Set: F1-P 73

Figure A.38 MOA Results of DenStream on Commercial-Big Data Set: F1-R 73

Figure A.39 MOA Results of DenStream on Commercial-Big Data Set: Purity 74

Figure A.40 MOA Results of CluStream on Commercial-Big Data Set: F1-P 74

Figure A.41 MOA Results of CluStream on Commercial-Big Data Set: F1-R 74

Figure A.42 MOA Results of CluStream on Commercial-Big Data Set: Purity 75

Figure A.43 MOA Results of ClusTree on Commercial-Big Data Set: F1-P 75

Figure A.44 MOA Results of ClusTree on Commercial-Big Data Set: F1-R 75

Figure A.45 MOA Results of ClusTree on Commercial-Big Data Set: Purity 76

## List of Tables


Table 2.1 Information on the data sets used in the previous works. . . . . 5

Table 3.1 Information on the Electricity data set. . . . . 7

Table 3.2 Information on the Forest Covertype data set. . . . . 8

Table 3.3 Summary of all the data sets used in this thesis. . . . . 8

Table 3.4 An example of stored snapshots for $\alpha = 2$, $l = 1$ and $T = 22$ . . 13

Table 4.1 Descriptions of parameters of each algorithm. . . . . 34

Table 4.2 An Example of a Confusion Matrix . . . . . 36

Table 4.3 Results of Flockstream on Electricity data set: Change in Epsilon. 37

Table 4.4 Results of Flockstream on Electricity data set: Change in Stream Speed. . . . . 38

Table 4.5 Results of Flockstream on Electricity data set: Change in Initial Agents. . . . . 38

Table 4.6 Results of Flockstream on Covertype data set: Change in Epsilon. 39

Table 4.7 Results of Flockstream on Covertype data set: Change in Stream Speed. . . . . 40

Table 4.8 Results of Flockstream on Covertype data set: Change in Initial Agents. . . . . 40

Table 4.9 Results of Flockstream on Commercial-Small data set: Change in Epsilon. . . . . 41

Table 4.10 Results of Flockstream on Commercial-Small data set: Change in Stream Speed. . . . . 42

Table 4.11 Results of Flockstream on Commercial-Small data set: Change in Initial Agents. . . . . 42

Table 4.12 Results of Flockstream on First Inner Cluster of Commercial-Small data set: Change in Epsilon. . . . . 43

Table 4.13 Results of Flockstream on First Inner Cluster of Commercial-Small data set: Change in Stream Speed. . . . . 43

Table 4.14 Results of Flockstream on Second Inner Cluster of Commercial-Small data set: Change in Epsilon. . . . . 44

Table 4.15 Results of Flockstream on Second Inner Cluster of Commercial-Small data set: Change in Stream Speed. . . . . 45

Table 4.16 Results of Flockstream on KDD 2009 Churn data set: Change in Epsilon. . . . . 46

Table 4.17 Results of Flockstream on KDD 2009 Churn data set: Change in Stream Speed . . . . . 46

Table 4.18 Performances of Algorithms on Electricity Data Set . . . . . 47

Table 4.19 Performances of Algorithms on Covertype Data Set . . . . . 47

Table 4.20 Performances of Algorithms on Commercial-Small Data Set . . 48

Table 4.21 Performances of Algorithms on Commercial-Big Churn Data Set 48

Table 4.22 Performances of Algorithms on KDD 2009 Churn Data Set . . 48

## Abstract

Retaining users and customers is one of the most important challenges for the service industry from mobile communications to online gaming. As the users of these services form dynamic networks that grow in size, predicting churners becomes harder and harder. The changing behavior of users and type of services changing day by day make it difficult to monitor the mobility of customers. However, from the service providers point of view, convincing a customer to keep using their services is more efficient way than the gaining a new customer.

In this thesis, I explore the use of anomaly detection for churn prediction. Due to the reason that users generate a huge amount of data during the use of services, I approach to the problem in terms of stream clustering methods. To this end, I evaluate bio-inspired and massive online data analysis techniques on public data sets, which are well known for clustering and classification tasks, as well as real world cell phone and online gaming data sets. I discuss the results of each technique from the perspective of usage of efficient features, sensitivity analysis on the parameters of the respective techniques as well as their performance.

## List of Abbreviations and Symbols Used

**ANFIS** Adaptive Neuro-Fuzzy Inference System.

**ANN** Artificial Neural Network.

**CF** Clustering Feature.

**CRM** Customer Relationship Management.

**ET** Execution Time.

**GA** Genetic Algorithms.

**GSM** Global System for Mobile.

**k-NN** k-Nearest Neighbor.

**KDD** Knowledge Discovery and Data Mining.

**LS** Linear Sum.

**MOA** Massive Online Analysis.

**MSF** Multiple Species Flocking.

**NN** Neural Networks.

**NoC** Number of Clusters.

**NSW** New South Wales.

**ROC** Receiver Operating Characteristic.

**SS** Squared Sum.

**SSQ** Sum of Squared Distance.

**SVM** Support Vector Machines.

**TV** Television.

**UCI** University of California, Irvine.

**VIP** Very Important Person.

## Acknowledgements

I consider myself the luckiest man in the world for having the love of my life, Ecenaz Ruscuklu. From the beginning to the end, she was always there for me with her love, kindness and encouragement. I owe my deepest gratitude to her. Also, I would like to thank my parents, Dursun and Çelik Tatar, and my sister Dilan Tatar for their never-ending love and support.

I am grateful to my second family, my friends, Egemen Tamcı, İpek Pakkaner, Kamer Aydın, Duygu Akbaş, Gökşen and Cansu Akbaş Demirel, Güneş Altay, and Burcu Kaya. Despite more than seven thousand kilometers, I felt them always beside me. Additionally, I am thankful to Gürcan and Bilge Gerçek, and Eray Balkanlı for their kindly friendship during my stay in Canada. On the other hand, I would like to thank two professors, Dr. İrem Özgören Kınlı and Dr. Mustafa Sakallı for encouraging me to apply master's degree, and being great mentors to me during my undergraduate years.

Finally, this thesis would not have been possible without my supervisor, Dr. Nur Zincir-Heywood. With her enthusiasm and knowledge, she guided me perfectly during all steps of my study. It was pleasure to me to work with such an amazing person, and I am grateful for everything. Also, I am indebted to all professors and colleagues in Network Information Management and Security (NIMS) Lab to support me.

This research is supported by the Mitacs Accelerate Internship award, and is conducted as part of the Dalhousie NIMS Lab at: https://projects.cs.dal.ca/projectx/

> "Try not to resist the changes, which come your way. Instead let life live through you. And do not worry that your life is turning upside down. How do you know that the side you are used to is better than the one to come?"
>
> — Shams Tabrizi

# Chapter 1

# Introduction

Day by day, customers are offered to better products and services from various providers apart from their own. It becomes difficult for companies to retain customers for a long period of time. From the company side, the threat of low brand loyalty may cause a decrease in usage of services or may even end up with a loss of customers. Those customers, who terminate their subscription of a service, are called churners and the rate of churn in a certain period of time is referred to a churn rate. Companies endeavour to keep churn rate as low as possible.

In today's competitive world, the importance of churn detection is indisputable, especially for the service sector. The literature on churn detection reveals a variety of approaches applied to different kinds of industries such as telecommunication, insurance, finance, Internet service providers, online services, TV providers and so on. Notably, studies on telecommunication [21], [34], [35], [18], [28], [17], [32], [22] and finance [36], [20],[40], [30], [7] sectors attract great attention from researchers.

In general, the common characteristic of the data sets being studied is the behaviors of customers who are taking the particular service. Additionally, the perceptions and demographics of customers, as well as their interactions with the company can be used to predict churn [4]. One of the problems of churn data sets is their unbalanced nature; generally, only small portion of the data consists of churning customers [38]. Nevertheless, predicting that small amount of churn can be a valuable source for companies to retain their customers who are about to quit the service [34]. Moreover, small precautionary measures to retain customers help companies to increase their profit considerably [30] as this is much cheaper than finding a new customer [24].

Although the data sets being studied by researchers are relatively small, in the real world, they are much larger and being generated continuously. Through the developing technologies, it becomes easy to store large amounts of data. However, processing it and revealing meaningful inferences from it, are the major challenges

because of the infinite nature and evolving characteristics of the data streams [2]. In other words, we can assume that data flow will never stop and its features may change over time. For these reasons, it is expected that a stream clustering algorithm will form clusters of arbitrary shape with no prior information about the number of clusters, as well as handle outliers [8]. In recent years, there has been considerable interest in literature on this subject [8], [2], [19], [13], [39], [25].

In this research, I concentrate on the Flockstream [13] algorithm, as well as three state-of-the-art algorithms; ClusTree, CluStream, and DenStream. The aim of this research is to give a comprehensive analysis of these algorithms in terms of clustering performance for churn detection on continuously streaming and evolving data. Algorithms are evaluated by experiments on three data sets; KDD 2009 churn data set which is publicly available, and two commercial data sets. In the evaluation phase, I present parameter sensitivity analysis of Flockstream and then discuss the results of all algorithms based on well-known performance metrics.

The thesis is organized as follows. In Chapter 2, a literature review takes place on churn detection and stream clustering techniques. Chapter 3 presents the algorithms employed. Chapter 4 details the experimental evaluations on the real life data sets employed. Finally, conclusions are drawn and the future work is discussed in Chapter 5.

# Chapter 2

# Literature Survey

A variety of classification techniques such as decision trees [9], neural networks (NN), k-Nearest Neighbor (k-NN), support vector machines (SVM), logistic regression, genetic algorithms (GA) have been applied to churn detection (prediction). According to the results of the churn-modeling tournament presented by Neslin et al. in [26], logistic regression and decision trees are the two most popular methods with a usage ratio of 68% in total. Moreover, a high percentage of the participants used more than one technique which is very common in most of the studies. In the literature, there have been numerous publications presented using either one or a combination of these methods. Lee et al. argue that k-NN based classification using time series performs better than the other classification techniques which use transformed version of time-series attributes into one or more non-time-series attributes using statistical methods [21]. In their research, they used the data gathered from one of the largest telecommunication companies in Taiwan over a four-month period. As a neural network approach, Mozer et al. [24] studied a wireless telecommunication data set containing approximately $47,000$ subscribers. The experimental tests are performed on two randomly selected groups of subscribers. The results show that the churn rate in treatment group in which potential subscribers were contacted by the company is 40% less than the control group where no action was taken by the company for the subscribers.

Huang et al. [17] compared decision tree, neural network and SVM methods for churn modelling. They used the telecommunication data set where roughly 23% of the data consisted of churners and results indicate that neural networks and multi-class SVMs performed better. Another comparison is made by Zhao and Dang [38] using artificial neural network (ANN), decision tree (C4.5), logistic regression and naive Bayesian classifiers on financial data sets containing information about one of the commercial banks' VIP customers. In the evaluation of the techniques, they consider

the accuracy, hit, and covering rate, as well as the lift coefficient which is calculated as a ratio between accuracy rate and customer churn rate. According to the results, SVM shows the best performance compared to other techniques. GA is another technique used by Pendharkar [29] with a combination of neural networks. The data set involves nearly $200,000$ records from a wireless company and evaluations are made in terms of ROC performance metric. A comparative study between GA-based ANN algorithm developed by researcher and statistical z-score classification model indicates that the GA-based model performs better than the statistical one. Instead of classification techniques, the effect of fuzzy c-means clustering for separating churning customers is studied by Karahoca and Karahoca in [18]. As in many studies, they ran the algorithm on telecommunication data which includes randomly selected loyal and churner subscribers. In the comparison with the data mining approaches (Ridor, ANFIS and decision trees) it is observed that the proposed clustering technique is better than the other algorithms with respect to sensitivity, specificity, precision, and correctness performance metrics. The information about the data sets used in literature are summarized in Table 2.1.

Working on churn prediction itself is impractical for long term analysis when the changing behavior of the subscribers is considered. For that reason, I have decided to consider churn detection problem with the challenges of streaming data. The literature on stream clustering shows a variety of approaches. CLARANS [27] algorithm is used to measure performance of DenStream. Both algorithms were run on some synthetic and real world (SEQUOIA 2000 benchmarking) data sets. The results show that DenStream performs much better and also finds clusters in arbitrary shapes. Analogously to DenStream, the D-Stream algorithm, proposed by Tu and Chen in [33], is another density-based method by mapping data points to grid space. Its two versions, DS0 and DS1, are compared with the CluStream and SSQ on the KDD Cup 1999 Network Intrusion Detection data set. Authors state that both versions of the D-Stream outperforms CluStream in terms of the evaluation measures used. StreamKM++ is another approach to stream clustering proposed by Ackermann et al. [1]. It takes advantage of k-means++ algorithm to determine first values of the clusters. In the further step, it computes samples using the corset tree data structure, introduced by the same authors. They compare their algorithm with the

two algorithms, StreamLS and BIRCH. Five different data sets were used from the UCI Machine Learning Repository, which are Spambase, Intrusion, Covertype, The Tower, and Census 1990. The algorithms are evaluated with respect to their execution times and average SSQ. The results reveal that while BIRCH performs faster than the other techniques used, StreamKM++ and StreamLS are better than BIRCH in terms of SSQ. However, performances of the StreamKM++ and StreamLS are very similar in terms of the average cost (SSQ). However, StreamKM++ works faster than StreamLS.

In this work, I have focused on three well known stream clustering algorithms (CluStream [2], DenStream [8], ClusTree [19], and Flockstream [13]), which are discussed in detail in the next chapter.

Table 2.1: Information on the data sets used in the previous works.

| | Gathered from | Access | Type | Year |
|---|---|---|---|---|
| Neslin et al. | The Teradata Center for CRM | Public | Batch | 2001 |
| Lee et al. | Taiwan Telecommunication Company | Commercial | Batch | - |
| Mozer et al. | Anonymous Telecommunication Company | Commercial | Batch | 1998 |
| Huang et al. | Eircom Telecom Company | Commercial | Batch | 2008 |
| Zhao et al. | China Construction Bank VIP Customers | Commercial | Batch | 2007 |
| Pendharkar et al. | The Teradata Center for CRM | Public | Batch | 2001 |
| Karahoca et al. | GSM Operator from Turkey | Commercial | Batch | - |

# Chapter 3

# Methodology

This thesis focuses on investigating churn detection using the state-of-the-art clustering techniques. In this context, I conducted my research on five data sets gathered from both commercial and public sources. First three data sets are Forest Covertype [6], Electricity [16] and KDD Cup 2009 [15], which are publicly available and used in many studies related to classification and clustering tasks. On the other hand, I employed two more data sets from one of the leading gaming companies. Because of the privacy concerns, they are mentioned as commercial throughout the following sections.

Electricity data set was gathered from the Australian New South Wales Electricity Market. It contains $45,312$ instances showing electricity price changes between May 7, 1996 and December 5, 1998. This data set contains 8 features in which first three features are specified as the date (i.e. 980331 refers to March 31, 1998), the day of the week (i.e. 1 refers to Monday and 7 refers to Sunday), and the time period (i.e. the time period 37 refers to the 37th portion of 48 half hours in a day). Data instances are sorted with respect to these three features. Next four features represent the electricity price and demand in two states, NSW and Victoria. The final feature denotes the scheduled power transfer between these states. Table 3.1 presents the summary of features and their types. The last attribute of Electricity data is the class value which is a binary label indicating whether the current price is upper or lower than the daily average price.

Another public data set that I used for measuring clustering performance of algorithms is Forest Covertype data set. Data is especially published for classification task to predict type of cover in $30 * 30$ meters patches of forest. It contains $581,012$ number of instances with 54 attributes. In fact, 44 attributes are represented by only two features, *Wilderness_Area* and *Soil_Type*. That's why, we may assume 12 features in total. The last attribute takes a value from 1 to 7 which specifies the categories of

Table 3.1: Information on the Electricity data set.

| Feature Name | Type | Description |
|---|---|---|
| date | numeric | Date |
| day | [1-7] | Day of week |
| period | [1-48] | Time |
| nswprice | numeric | NSW electricity price |
| nswdemand | numeric | NSW electricity demand |
| vicprice | numeric | Victorian electricity price |
| vicdemand | numeric | Victorian electricity demand |
| transfer | numeric | Scheduled electricity transfer between states |
| class | [Up, Down] | Class Value |

the cover types. Data is sorted ascending by the *Elevation* feature. Table 3.2 outlines the summary information for this data set.

In addition to Electricity and Covertype data sets, three more data sets were chosen specifically for the churn detection task. The first of these is KDD Cup 2009, explained in detail by Guyon et al. [15], which was presented by the French Telecommunication company, Orange, for the KDD Cup 2009 challenge. The aim of the challenge was to predict propensity of the customers to change their provider. Two data sets were presented as churn data, including one large and one small. In this thesis, I picked the small data set which consists of $50,000$ instances and 230 attributes because the large one consists of $15,000$ attributes, which may cause a significant load on the system resources. The data includes a large amount of missing values ($\sim$60%). Additionally, the first 190 variables are numerical, whereas the last 40 variables are categorical. Due to the privacy concerns, this data set is not fully documented. Furthermore, categorical variables were replaced with meaningless codes. For that reason, the data is preprocessed before being used in the experiments such that the categorical attributes and features containing no value for all instances are identified and deleted. Additionally, the mean of each feature is calculated column-wise and missing values are filled with the mean of the corresponding feature. Before running experiments on the data sets, they are normalized such that the range of the values is limited between 0 and 1. At the end of this process, I had a normalized churn data set with 175 numerical attributes.

The other two churn data sets is gathered from the commercial sources. Particularly, the two commercial data sets used are named according to their sizes as big and

small. They are composed of a set of activities by customers during the game. The main purpose is to detect churners within the upcoming $24 - 48$ hour time window. The data sets are quite unbalanced with only about 9% and 8% churn rates for small and big data sets, respectively. The small data set consists of $1,669,593$ number of instances, whereas the big one has $9,618,868$ instances (records). There are 16 numerical features for both sets. At the request of the company, I can not share the meaning of these features but basically, each instance represents one play with a collection of player behavior during the game. It should be noted here that more than one play might be played by only one player. The identification number of each player is used to group the players and preprocess them to extract their behaviours throughout the game. Similar to the KDD 2009 data set, values are normalized before being processed. Information about data sets is summarized in Table 3.3.

Table 3.2: Information on the Forest Covertype data set.

| Feature Name | Type | Description |
|---|---|---|
| Elevation | numeric | Elevation in meters |
| Aspect | numeric | Aspect in degrees azimuth |
| Slope | numeric | Slope in degrees |
| Horizontal_Distance_To_Hydrology | numeric | Horz. dist. to nearest surface water feature |
| Vertical_Distance_To_Hydrology | numeric | Vert Dist to nearest surface water feature |
| Horizontal_Distance_To_Roadways | numeric | Horz Dist to nearest roadway |
| Hillshade_9am | [0-255] | Hillshade index at 9am, summer solstice |
| Hillshade_Noon | [0-255] | Hillshade index at noon, summer soltice |
| Hillshade_3pm | [0-255] | Hillshade index at 3pm, summer solstice |
| Horizontal_Distance_To_Fire_Points | numeric | Horz. dist. to nearest wildfire ignition points |
| Wilderness_Area (4 columns) | binary | Wilderness area designation |
| Soil_Type (40 columns) | binary | Soil Type designation |
| Cover_Type (7 types) | [1-7] | Forest Cover Type designation |

Table 3.3: Summary of all the data sets used in this thesis.

| Data Sets | # Instances | # Features | # Classes | Size (MB) |
|---|---|---|---|---|
| Electricity | 45,312 | 8 | 2 | 3.1 |
| KDD Cup 2009 | 50,000 | 175 | 2 | 122.1 |
| Forest Covertype | 581,012 | 54 | 7 | 103.4 |
| Commercial | 1,669,593 | 16 | 2 | 118.8 |
| Commercial | 9,618,868 | 16 | 2 | 809.1 |

In this thesis, the analysis was performed with four stream clustering techniques; CluStream [2], DenStream [8], ClusTree [19], and Flockstream [13]. The first three

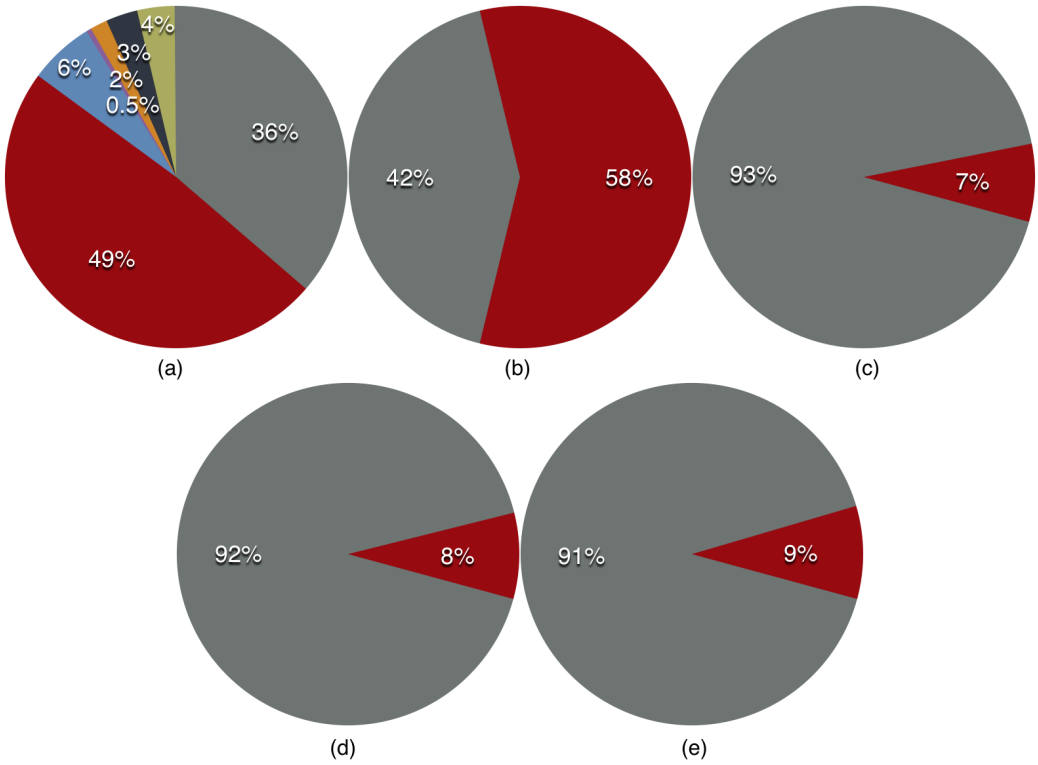


Figure 3.1: Class distribution of data sets. (a) Covertype, (b) Electricity, (c) KDD Cup 2009, (d) Commercial-Big, (e) Commercial-Small

algorithms are implemented via the Massive Online Analysis (MOA) [5], which is a free open source software to perform clustering and classification techniques for data stream mining. On the other hand, I implemented the Flockstream algorithm using the Java programming language.

### 3.1 CluStream

CluStream clustering process is divided into two separate phases to reduce the density caused by a fast-flowing data, as well as comparing the present results with those of the past [2]. First phase is called online micro-clustering where summary statistics of data points are gathered and stored in an efficient manner. Subsequently, the obtained statistics can be used to analyze clusters in accordance with the user demand, which is known as the offline phase. One of the most important features of this approach is to give more relevant results on evolving data for a long time when analyzed retrospectively. A deeper understanding of the process can be achieved by

clarifying the core concepts. Eventually, these concepts will be mentioned or applied in the analysis of other algorithms. After that, online and offline phases will be examined in detail.

**Clustering Feature Vector**

The clustering process can be problematic for big data sets, not only due to the computational cost, but also the space. In order to minimize memory requirements, it has been suggested by Zhang et al. [37] that features of data points can be represented as a vector. This vector, namely *Clustering Feature* (CF), has three components and is defined as $CF = (N, LS, SS)$. Here, $N$, $LS$, and $SS$ represent the total number, linear sum ($\sum_{i=1}^{N} \vec{X}_i$) , and squared sum ($\sum_{i=1}^{N} \vec{X}_i^{\,2}$) of the data points ($X_i$) in the cluster, respectively.

- **N:** Total number of data points in a cluster. When we assume $\vec{X}_i$ as a d-dimensional data points in a cluster, than its range can be shown as $i = 1, 2, 3, ..., N$.
- **LS:** Linear sum of the all data points in a cluster such that $\sum_{i=1}^{N} \vec{X}_i$ .
- **SS:** Squared sum of the all data points in a cluster such that $\sum_{i=1}^{N} \vec{X}_i^{\,2}$ .

When a new data point $\vec{x_i}$ arrives to the cluster, total number of points $N$ is incremented by 1, $\vec{x_i}$ and $\vec{x_i}^2$ are added to $\vec{LS}$ and $SS$, respectively. If two CFs are need to be merged, then a new CF vector is comprised of sum of their components respectively such that, $CF_1 + CF_2 = (N_1 + N_2, LS_1 + LS_2, SS_1 + SS_1)$.

$$\begin{bmatrix} N_1 + N_2 \\ LS_1 + LS_2 \\ \mathrm{SS}_1 + \mathrm{SS}_2 \end{bmatrix} \leftarrow \begin{bmatrix} N_1 \\ LS_1 \\ \mathrm{SS}_1 \end{bmatrix} + \begin{bmatrix} N_2 \\ LS_2 \\ \mathrm{SS}_2 \end{bmatrix}$$

These vector components allow us to measure three fundamental features of clusters such as *centroid*, *radius*, and *diameter*. Centroid is defined as a mean of the linear sum of a cluster. Radius is the distance between centroid and the

farthest data point, whereas diameter is the distance between the most distant data points. Mathematical representation of these definitions are given in the following equations 3.1, 3.2, and 3.3.

$$\mathrm{C}_c \;=\; \frac{\sum_{i=1}^{N} x_i}{N} \tag{3.1}$$

$$R_c \;=\; \sqrt{\frac{\sum_{i=1}^{N} (x_i \;-\; \mathrm{C}c)^2}{N}} \tag{3.2}$$

$$D_c \;=\; \sqrt{\frac{\sum_{i=1}^{N} \sum_{j=1}^{N} (x_i \;-\; x_j)^2}{N \;\cdot\; (N-1)}} \tag{3.3}$$

In the equations, $C_c$, $R_c$, and $D_c$ denote the center, radius, and diameter of a cluster. Additionally, $N$ represents the total number of data points, while $x$ represents the data point itself.

**Micro-Clustering**

In CluStream, data points are processed in a time interval and each of them has a timestamp. Based on the CF structure explained above, authors of CluStream have introduced a broader concept called *micro-cluster*. Two additional time dependent components, $\mathrm{CF1}^t$ and $\mathrm{CF2}^t$ distinguish a micro-cluster from a clustering feature. $\mathrm{CF1}^t$ contains sum of all time stamps belonging to the data points in a cluster. On the other hand, $\mathrm{CF2}^t$ stands for the squared sum of the all data points in a cluster. In terms of computation, $\mathrm{CF1}^t$ and $\mathrm{CF2}^t$ are the same with $LS$ and $SS$ except for the time variable.

A micro-cluster [2] can be represented with the $(2 * d + 3)$ tuple for a set of d-dimensional data points $X_{i_1}, \ldots, X_{i_N}$ and time stamps $T_{i_1}, \ldots, T_{i_N}$ as ($\overline{CF2^x}$, $\overline{CF1^x}$, $CF1^t$, $CF2^t$, $N$). The benefit of this extension is to facilitate the access of saved micro-clusters at different time intervals. In this respect, users are able to extract old clusters from the history and examine them on demand. In addition to the *additive* property, discussed for the clustering feature, micro clustering satisfies the *subtractive* property. This property gives the user an

opportunity to get clusters in a specified time period or *horizon*. Assuming that $t_c$ represents the current time and $h$ represents the time period backwards from the current time that the user wants to obtain clusters. In such a situation, the higher level clusters stored in the time horizon $\{(t_c - h), t_c\}$ is derived by using the macro-clustering algorithm discussed below. As a result, the micro-clustering technique allows the user to reach older clusters and compare them with the current results.

Although the micro-clustering technique allows users to reach older clusters and compare them with the current results, it is not feasible to store every micro-cluster in the memory. That is why researchers have suggested an efficient way to store micro-clusters which is called *pyramidal time frame*.

**Pyramidal Time Frame**

This technique [2] offers a solution to space requirements arising from the saving every micro-cluster in the system. In order to achieve that, it stores snapshots hierarchically with respect to specific rules. Assume that $T$ indicates the clock time which is elapsed since the beginning of the process. Then, the snapshots can be sorted in an *order* $i$, where $i$ can be a value between 0 to $\log T$. Following definitions show how maintenance of snapshots is satisfied.

- Orders of snapshots are determined according to $\alpha^i$, where $\alpha$ is an integer and $\alpha \geq 1$. The snapshot $S$ is saved to every order $i$ simultaneously unless it is divisible by $\alpha^i$.

- For each order $i$, maximum $\alpha^l + 1$ snapshots can be stored, where $l$ is a user defined parameter. If an order reaches its maximum capacity, then oldest snapshot inside is deleted.

Consider a case when $\alpha = 2$, $l = 1$ and the clock time $t_c = 16$. Then, $t_c$ can be divisible by $2^0$, $2^1$, $2^2$, $2^3$, and $2^4$. So, there are five different orders from zero to four to place the snapshot. Moreover, each order may contain at most $2^1 + 1 = 3$ snapshots. Even though capacity of one of the orders is full, the

oldest snapshot can be deleted before storing the new one. In accordance with these cases, authors [2] have made the following observations.

1. The maximum number of order for $T$ in a streaming process is $\log_\alpha T$.
2. The maximum number of snapshots stored at time $T$ throughout the process is $(\alpha^l + 1) \cdot \log_\alpha T$
3. It is ensured that at least one stored snapshot can be reachable by user within $2 \cdot h$ units, where $h$ is user specified time horizon. It is formally proved in [2].

Table 3.4 illustrates the stored snapshots for $\alpha = 2$, $l = 1$ and $T = 22$. As its name indicates, a pyramidal shape emerges when snapshots ordered from top to bottom. It is obvious that while the order $i$ increases, distance between the snapshots in the same order decreases, or vice versa. That's why, better granularity can be seen between recently added snapshots. As an instance, at clock time 22, micro-clusters at times of 8, 12, 16, 18, 20, 21, and 22 are stored in the system.

Table 3.4: An example of stored snapshots for $\alpha = 2$, $l = 1$ and $T = 22$

| Order of Snapshots | Clock Times |
|---|---|
| 0 | 22 - 21 - 20 |
| 1 | 22 - 20 - 18 |
| 2 | 20 - 16 - 12 |
| 3 | 16 - 8 |
| 4 | 16 |

### 3.1.1 Online Phase

As mentioned earlier, generation and maintenance of micro-clusters are handled in two phases. The online phase begins with a sufficient amount of micro-clusters generated by using standard k-means algorithm. After that the system starts to accept new data points. When a new point arrives to the system, there are two cases in order to achieve maintenance: (i) Either the new point joins a micro-cluster, in which it fits most; or (ii) a new micro-cluster is initiated by including the new point. In the first scenario, a convenient micro-cluster can be found by calculating the distance

between the new point and the centroid of the micro-clusters. However, the closest micro-cluster is not always suitable because the new point may be an outlier or a first instance of evolving data where a new micro-cluster should be initiated. Hence it is hard to predict how the data will evolve, authors [2] have suggested to look at whether the new point is within the *maximum boundary* of the micro-cluster or not. If it is not, then a new micro-cluster should be created. However, because of the memory constraints, old micro-clusters are maintained by either deleting an old cluster (if it is safe) or merging two old clusters.

The decision of which cluster will be deleted is made according to the recency of micro-clusters. So that, the micro-clusters, which have recently accepted data points, are not considered to be deleted. In order to make a decision, authors have proposed a way to approximate for finding recency level of a micro-clusters by using the mean and the standard deviation of time stamps, which is called the *relevance stamp*. In order to determine whether a micro-clusters will be deleted or not, its relevance stamp is compared to a user-defined threshold value $\delta$. If it is below $\delta$, then the micro-cluster is remained in the system. Otherwise, it is picked to be removed from the system. It is also possible that relevance stamp of every micro-cluster currently being in the system is above the threshold. In that case, two closest micro-clusters are merged together and a new unique id is assigned to the new micro-cluster. In order to identify the merged micro-clusters, new id is assigned as a combination of their ids.

Meanwhile, at every clock time divisible by $\alpha^i$, micro-clusters that are in the system at that time are saved with their id list and time of storage for macro-clustering. As mentioned earlier, if order $i$ is full of snapshots and if the current snapshot can not be divisible by $\alpha^{i+1}$, then the least recent snapshot is deleted from the order $i$.

### 3.1.2 Offline Phase

The purpose of this phase [2] is to provide in depth analysis of micro-clusters for the user in a given time horizon $h$. Hence, the summary statistics are prepared in online phase, it is easy to extract relevant micro-clusters by using its additive and subtractive properties described before. It is also mentioned in the explanation of pyramidal time frame that there is always a snapshot between the current time $t_c$ and the time horizon $h'$, which is an admissible horizon within the interval $(t_c, h)$.

In the first step, relevant clusters are extracted from the user-specified time interval. Assume that $S(t_c)$ represents the set of micro-clusters at current time, and $S(t_c - h')$ represents the set of micro-clusters in the time interval $(t_c, h')$. Hence the list of micro-cluster *ids* in $S(t_c)$, can be reached and the micro-clusters in $S(t_c - h')$ can be eliminated by subtracting their cluster feature vectors. So that, summary statistics gathered before the time horizon $h$ will not dominate over the results. At the end of this step, a set of micro-clusters $N(t_c, h')$ that are the output of the subtraction process are obtained. For the next step, the modified version of k-means explained in [2], is run over $N(t_c, h')$ to find high level of clusters. A more detailed description of CluStream can be found in [2].

### 3.2 DenStream

Cao et. al. [8] proposed a density-based clustering algorithm for streaming data. It extends the *core point* concept introduced with *DBSCAN* [12], by using the micro-clustering technique from CluStream explained in the previous section. While DenStream adapts several concepts of CluStream, it also originates new ones. A core point is defined as an object in the $\varepsilon$ neighbourhood where the weight of the points is at least an integer $\mu$. Instead of a binary decision of data point inclusion to the cluster, a *damped window model* is preferred to determine the importance of historical data. In this model, each data point has a weight depending on time $t$. During the existence of data points in the system, importance of them gradually decreases according to the fading function $f(t) = 2^{-\lambda \cdot t}$, where $\lambda > 0$. In addition to the weight of data points, the data stream has its own weight, which is specified as a constant $W = v/1 - 2^{-\lambda}$, where $v$ is the speed of the stream, i.e. the number of points arrive to the system in a one time unit.

Generally, it is difficult to separate streaming data points into dense regions. In order to achieve that, the authors [8] have introduced three concepts, namely *core-micro-cluster*, *potential core-micro-cluster*, and *outlier-micro-cluster*.

### 3.2.1 Core-micro-cluster

A core-micro-cluster (abbr. c-micro-cluster) [8] is defined as $CMC(w, c, r)$ at time $t$ for a group of close points $p_{i_1}, \ldots, p_{i_n}$ with time stamps $T_{i_1}, \ldots, T_{i_n}$. The components in $CMC(w, c, r)$ denote the weight, center, and radius of the c-micro-cluster, respectively. Mathematical representations of are provided in Eq. 3.4, 3.5, and 3.6.

$$w \;=\; \sum_{j\,=\,1}^{n} f\left(t \;-\; T_{i_j}\right) \;,\; where\, w \;\geq\; \mu \tag{3.4}$$

$$c \;=\; \frac{\sum_{j\,=\,1}^{n} f\left(t \;-\; T_{i_j}\right)\; p_{i_j}}{w} \tag{3.5}$$

$$r \;=\; \frac{\sum_{j\,=\,1}^{n} f\left(t \;-\; T_{i_j}\right)\; dist\left(p_{i_j} \;,\; c\right)}{w} \;,\; where\, r \;\leq\; \varepsilon \tag{3.6}$$

In equation 3.6, $dist(p_{i_j}, c)$ denotes the Euclidean distance between the point $p_{i_j}$ and the center of the c-micro-cluster, $c$. It is explicitly stated that the weight of the c-micro-clusters must be greater than or equal to $\mu$. The authors [8] consider that the set of micro clusters can be used to represent the clusters with an arbitrary shape. However, the clusters need to be distinguished from outliers. Therefore, they introduced potential core-micro-cluster, and outlier-micro-cluster, similar to those in [3].

### 3.2.2 Potential core-micro-cluster

A potential c-micro-cluster (abbr. p-micro-cluster) [8] at time $t$ is defined as $\overline{CF^1}$, $\overline{CF^2}$, $w$ for a group of close points $p_{i_1}, \ldots, p_{i_n}$ with time stamps $T_{i_1}, \ldots, T_{i_n}$. It is possible that the set of points representing p-micro-cluster become a micro-cluster; however, it may be demoted if no new data point joins into it because its weight tends to decrease by fading function. The weight function is the same as the one used in the c-micro-cluster except that it must be greater than or equal to $\beta\mu$. $\beta$ is defined as an outlier threshold and takes a value between 0 and 1. In the following equations, $c$ and $r$ represent the center and radius of a p-micro-cluster (Eq. 3.7 and Eq. 3.8), respectively. Whereas $\overline{CF^1}$ is the weighted linear sum of the points (Eq. 3.9), $\overline{CF^2}$ is the wighted square sum of the points (Eq. 3.10).

$$c \;=\; \frac{\overline{CF^1}}{w} \tag{3.7}$$

$$r \;=\; \sqrt{\frac{\overline{CF^2}}{w} \;-\; \left(\frac{\overline{CF^1}}{w}\right)^2}\;,\; where\, r \;\leq\; \varepsilon \tag{3.8}$$

$$\overline{CF^1} \;=\; \sum_{j\,=\,1}^{n} f\left(t \;-\; T_{i_j}\right)\; p_{i_j} \tag{3.9}$$

$$\overline{CF^2} \;=\; \sum_{j\,=\,1}^{n} f\left(t \;-\; T_{i_j}\right)\; p_{i_j}^2 \tag{3.10}$$

### 3.2.3 Outlier-micro-cluster

An outlier micro-cluster (abbr. o-micro-cluster) [8] at time $t$ is defined as $\overline{CF^1}$, $\overline{CF^2}$, $w$, $t_o$ for a group of close points $p_{i_1}, \ldots, p_{i_n}$ with time stamps $T_{i_1}, \ldots, T_{i_n}$. Except for $t_o$, all definitions are the same as the ones used in the p-micro-clustering and $t_o = T_{i_1}$ denotes the creation time of the o-micro-cluster. Contrary to p-micro-clustering, the weight is below the threshold $\beta\mu$, i.e $w < \beta\mu$. However, o-micro-cluster can be promoted to p-micro-cluster, if its weight exceeds the threshold while accepting new data points.

Analogous to CluStream, there are two phases; the online phase where the micro-clusters are updated and maintained, and the offline phase where the clusters are extracted on user demand. In the first phase, assume that the data point $p$ is one of the streaming data coming into the system. There are three scenarios:

i. Merging the data point $p$ into the nearest p-micro cluster $c_p$ such that, if it satisfies the condition that the new radius $r_p$ of $c_p$ is less than or equal to $\varepsilon$, i.e. $r_p \leq \varepsilon$, then the merging occurs. Otherwise, it is passed to the second scenario.

ii. Merging the data point $p$ into the nearest o-micro cluster $c_o$ such that, if it satisfies the condition that the new radius $r_o$ of $c_o$ is less than or equal to $\varepsilon$, i.e. $r_o \leq \varepsilon$, then the merging occurs. If the weight of $c_o$ is greater than $\beta\mu$, i.e. $w \geq \beta\mu$, then it is promoted to p-micro-clustering.

iii. If the conditions above are not satisfied, then a new o-micro-cluster is generated by $p$.

In the offline phase, the DenStream algorithm makes use of the variant of DBSCAN to find the final clusters. The p-micro-clusters are examined as virtual points in this section. The variant of DBSCAN includes two parameters, $\varepsilon$ and $\mu$, which are used to determine the density area by using concepts *density-reachable* and *density-connected*. As a result, final clusters are specified by density-connected p-micro-clusters. The definitions of these concepts are given below. In addition, Algorithm 1 shows the control flow of the DenStream.

- *Directly density-reachable* : A p-micro-cluster $c_p$ is directly density-reachable from a p-micro-cluster $c_q$, if $w_{c_q} > \mu$ and $dist(c_p, c_q) \leq 2 \cdot \varepsilon$, wrt. $\varepsilon$ and $\mu$. Two p-micro-clusters are considered as density-reachable, if the distance between their centers is less than or equal to $2 \cdot \varepsilon$, and $r_p + r_q$, where $r_p$ and $r_q$ represents the radiuses of $c_p$ and $c_q$, respectively.
- *Density-reachable* : A p-micro-cluster $c_p$ is density-reachable from a p-micro-cluster $c_q$ wrt. $\varepsilon$ and $\mu$, if there is a chain of p-micro-clusters $c_{p_1}, \ldots, c_{p_n}$ , $c_{p_1} = c_q$, $c_{p_n} = c_p$ such that $c_{p_{i+1}}$ is directly density-reachable from $c_{p_i}$.
- *Density-connected* : A p-micro-cluster $c_p$ is density-connected to a p-micro-cluster $c_q$ wrt. $\varepsilon$ and $\mu$, if there is a p-micro-cluster $c_m$ such that both $c_p$ and $c_q$ are density-reachable from $c_m$ wrt. $\varepsilon$ and $\mu$.

### 3.3 ClusTree

Kranen et al. [19] have introduced a self-adaptive clustering algorithm, ClusTree, for mining data streams. They proposed a parameter-free solution, which is able to adapt for different stream speeds. Micro-clustering and R-Tree structure [14] form the basis of the ClusTree algorithm. As noted at the beginning of this section, clustering feature (CF) tuple stores the summary of information related to the data stream. Tree structure makes it possible to maintain micro-clusters into different levels of the

**Algorithm 1** Denstream Algorithm

$T_p = \left\lceil \frac{1}{\lambda} \log \left( \frac{\beta\mu}{\beta\mu - 1} \right) \right\rceil$;
Get the next point p at current time $t$ from data stream $DS$
Merging(p)
**if** $(t \mod T_p) = 0$ **then**
  **for** each p-micro-cluster $c_p$ **do**
    **if** $\omega_p$(the weight of $c_p$)$< \beta\mu$ **then**
      Delete $c_p$;
    **end if**
  **end for**
  **for** each o-micro-cluster $c_o$ **do**
    $\xi = \frac{2^{-\lambda(t-t_o+T_p)}-1}{2^{-\lambda T_p}-1}$;
    **if** $\omega_o$(the weight of $c_o$)$< \xi$ **then**
      Delete $c_o$;
    **end if**
  **end for**
**end if**
**if** a clustering request arrives **then**
  Generating clusters;
**end if**

hierarchy. The main idea is to place an arriving object to the optimal micro-cluster, searching the tree from the root to leaf node. However, there might not be enough time for the point insertion process, therefore, authors have suggested to keep those points in a *local aggregate*, a buffer to keep the points, which have not yet completed and can be processed at a later time. ClusTree is explained with the parameters $m$, $M$, $l$, and $L$ as follows.

- While inner nodes have the entries from $m$ to $M$, leaf nodes have the entries from $l$ to $L$.
- An inner node contains the summary information of the objects both it stores and buffers. In addition to that, it keeps a pointer, which points to the child

node.

- A leaf node stores the CF of the data points it represents.
- The tree is balanced, which means that any path from the root to the leaf nodes has always the same length.

Insertion of an object is a continuous process, every data point travels from the root through the leaf nodes by choosing the subtree with the closest mean. In the case that an object can not reach the leaf nodes, the process is interrupted and the current CF is saved to the buffer of the subtree. Whenever that subtree is accessed by another object, then the saved entry is taken as a "hitchhiker". Unless their paths differ from each other, they descend together. If their paths need to be separated, then the hitchhiker is saved again to the buffer and the current insertion continues on its path. When an object reaches the leaf node, then it causes a split if there is still time. Otherwise, the closest entries are merged and their ids are saved to a list as a pair.

In the maintenance of clusters, an exponential decay function is chosen similar to DenStream, i.e. $\omega(\Delta t) = \beta^{-\lambda \Delta t}$, where $\beta = 2$. This method put emphasis on new data, rather than the old. On the other hand, summary information of the subtrees should be accurately stored by using weighted CF components (see Eq 3.11, Eq 3.12, and Eq 3.13).

$$n^{(t)} = \sum_{i=1}^{n} \omega\,(t - t_{\mathrm{S}_i}) \tag{3.11}$$

$$LS^{(t)} = \sum_{i=1}^{n} \omega\,(t - t_{s_i}) \cdot x_i \tag{3.12}$$

$$SS^{(t)} = \sum_{i=1}^{n} \omega\,(t - t_{s_i}) \cdot x_i^2 \tag{3.13}$$

where $t_{s_i}$ is the timestamp of the object $x_i$ which denotes the insertion time of it to the CF. Also, $n$ represents the number of objects in the CF.

In the updating process, all entries in the node are updated considering their CF, buffer and last update time. If a leaf node needs to be split, then the least significant entry in the system can be discarded. In that situation, the related entry is subtracted

from the path through the root. This ensures that no entry or CF is discarded if an object is added to it before the last snapshot. Moreover, it guarantees that each entry is stored in at least one snapshot.

In general, the definitions and concepts explained so far form the base of the ClusTree algorithm. Additionally, authors [19] analyzed the performance of the algorithm according to different stream speeds. Firstly, they examine the case when the data stream flows faster than algorithm can handle. This causes short-term interruptions at the very beginning of the process. Consequently, objects that can not descend to leaf nodes start to aggregate on the top level of the tree. This leads to a *global aggregate* which is problematic because completely dissimilar objects may come together in the same aggregate. Instead of global aggregates, the authors [19] proposed to create various aggregates for dissimilar objects and insert interrupted objects into their closest aggregate. The closeness decision is made by considering the distance of objects to the mean of the aggregate. If it is lower than the threshold value $max_{radius}$, then it is inserted into that aggregate. Otherwise, a new aggregate is initiated with that object. Whenever the algorithm finds time to process aggregated objects, it first picks the outnumbering aggregate. If there is more than one aggregate, it chooses the oldest one. Pleasantly, $max_{radius}$ need not to be defined by the user. Instead, it is set with respect to the average variance of the leaves. As a summary, Figure 3.2 illustrates the general working mechanism of the ClusTree with a flow chart.

Whereas fast data streams come with several problems, slow data streams lets authors to try different descent strategies on the algorithm. Normally, ClusTree tries to reach leaf level by choosing the closest child node, which is mentioned as *single-try depth first search*. However, computation time of this strategy is relatively low, which causes a lot of idle time for the algorithm. Alternative descent strategies examined in the scope of this research are as follows.

- *Priority breadth first traversal*
- *Best first traversal*
- *Iterative depth first descent*

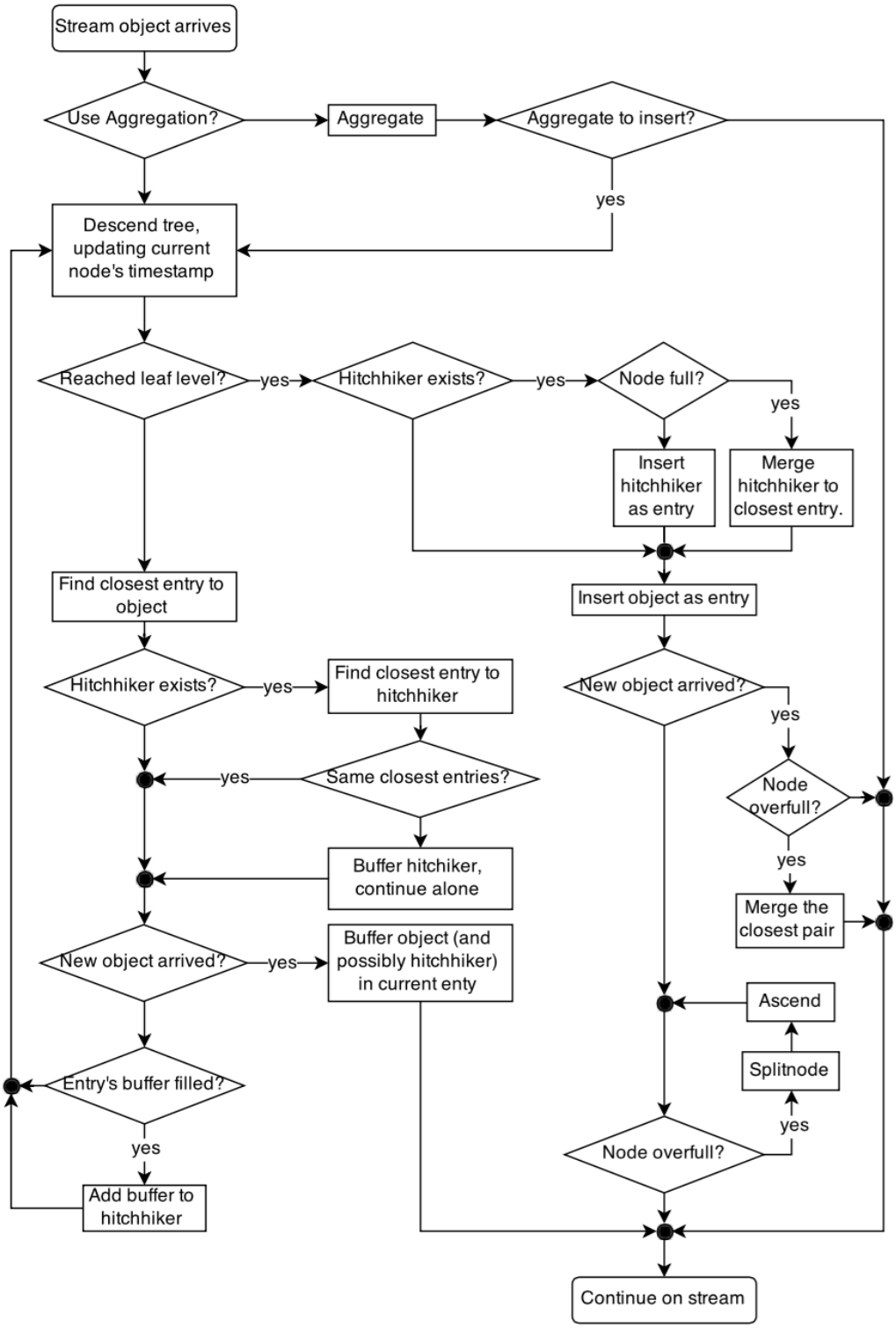


Figure 3.2: ClusTree Flowchart [19]

### 3.4 Flockstream

Forestiero et al. [13] have introduced a new, bio-inspired, agent-based approach for single pass stream clustering. They adapt the micro-clustering approach from DenStream to the Flockstream algorithm and unlike the algorithms discussed previously, single-pass paradigm is adopted instead of a two phase approach. Analogous to DenStream, a damped window model is chosen to fade the importance of the data points throughout the process. According to this model, the importance, or weight, of data points is determined according to the time variable, $t$. In this model, as the duration of agents in the system increases, the importance of the represented data points decrease with respect to the fading function, $f(t) = 2^{-\lambda \cdot t}$, where $\lambda > 0$. By using this fading function, the weight of a group of data points in a cluster can be calculated as $w = \sum_{j=1}^{n} f(t - T_{i_j})$, at time $t$ for time stamps $T_{i_1}, \ldots, T_{i_n}$.

The *flocking model*, first proposed by [31] and developed as a computational model by [11], forms the basis of this algorithm. According to this model, *boids* or *agents* interact with each other in an environment without sharing any information. Agents can interact with only their neighbour agents in their visibility range. Additionally, they keep some distance between them to avoid collisions. Their movements in the environment are coordinated based on three steering rules: *alignment*, *separation*, and *cohesion*.

- *Alignment* : Steering toward so that the direction and velocity would be the same with neighbors
- *Separation* : Steering away from neighbors to avoid collision.
- *Cohesion* : Steering toward the midpoint of neighbors

In addition to these rules, the *Multiple Species Flocking* (MSF) model [10] presents an additional rule, feature similarity. However, the authors [13] have modified the MSF by considering the flocking rules with similarity or dissimilarity of an agent with its neighbors. In order to define these rules, the concept of *velocity vector* $\vec{v}$ is presented in the MSF model.

The algorithm works on two spaces; $d$-dimensional feature space represented as $\mathcal{R}^d$, and two dimensional Cartesian space named virtual space represented as $\mathcal{R}_v^2$.

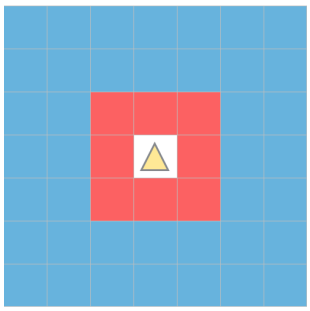

Figure 3.3: Visibility and Defense Range of an Agent

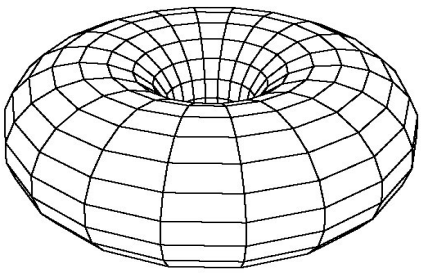

Figure 3.4: 3D Representation of Toroidal Virtual Space

The feature space is the data space in which each data point is stored. On the other hand, the virtual space is a toroidal grid system, as illustrated in Fig. 3.4, for agents to move on and interact with each other. It is designed to be discrete instead of continuous. Each cell contains only one agent at a time. Every agent deployed to the virtual space represents a data point from the feature space. In Flockstream, an Agent A is defined as $A = (P, \vec{v})$, such that $P$ is the position of the agent A in $\mathcal{R}_v^2$, i.e. $P = (x, y)$ and $\vec{v}$ is the velocity vector of the agent A, i.e. $\vec{v} = (m, \theta)$. In addition to this, I add two more features to the agent; $id_A$ to identify the agent A, and $id_C$ to keep the *id* of a *conductor* agent which conduce the agent A to join or form a flock.

An agent can observe the environment over a limited range, which is mentioned as a visibility range. Since I work on a grid environment, I assumed that an agent can observe only $v$ number of cells ahead. Also, an agent needs to specify its defense range, which is $d$ the number of cells ahead of the agent. As an instance, visibility and defense ranges for $v = 3$ and $d = 1$ are illustrated in Fig. 3.3. Blue and red cells represent the visibility range, and red cells represent the defense range for the yellow agent located in the white cell. It is useful to note that, every agent in the $\mathcal{R}_v^2$ can move only one cell at a time. The velocity component $m$ denotes the magnitude of the vector, which is fixed to 1, and $\theta$ denotes the angle between $\vec{v}$ and the positive x axis.

Before describing the formal definitions of the flocking rules, let us consider the following variables and functions. Let $p_c$ be a data point in feature space, which is represented by the agent $A_c$ in the virtual space. Each agent can interact with only the neighbor agents in a range with radius $R_1$ and defend itself from collision

with other agents within the range of $R_2$. Assume that the neighbor agents in $A_c$'s visibility range are denoted as $F_1, \ldots, F_n$. The distance between two agents, i.e. $d_v(A_i, F_i)$, is the Euclidean distance between positions of the agents in the virtual space, i.e. $P_{F_i} = (x_{F_i}, y_{F_i})$, $P_{A_i} = (x_{A_i}, y_{A_i})$. On the other hand, $dist(p_c, p_i)$ specifies the Euclidean distance between the data points $p_c$ and $p_i$, where $p_i$ is the data point of the neighbor agent $F_i$. It is used to determine the similarity between two agents such that, if $dist(p_c, p_i) \leq \varepsilon$, then two points are assumed to be similar. The maximum threshold value $\varepsilon$ mentioned in Eq. 3.6 specifies the radius of a micro-cluster.

Basically, the flocking behavior is the combination of velocity vectors of $\vec{v}_{ar}$, $\vec{v}_{sr}$, $\vec{v}_{cr}$ that are related to the rules of alignment, separation and cohesion, respectively. Before each movement of the agent $A_c$, these velocities are combined together to find the *target* velocity, i.e. $\vec{v}_{ar} + \vec{v}_{sr} + \vec{v}_{cr}$, and normalized to obtain a unit vector. It is worth noting that to steer the agent A towards a target point, it is needed to subtract target velocity from the current velocity. The final velocity vector determines the movement of the agent $A_c$. Please note that flocking rules can be applied with neighbor agents within agents' visibility range for each individual agent. The formal definitions and conditions of the flocking rules are described below.

### 3.4.1 MSF Rules

In the following sections, MSF rules are explained in detail. In which conditions are these rules applied and how they effect to the movement of an agent are described in this section.

#### 3.4.1.1 Alignment

In order to satisfy the alignment rule, an agent changes its direction towards the same direction of its neighbor agents that are similar. Typically, it tries to adjust its velocity vector to the average velocity vector of the others. This rule is applied as Eq.3.14, if the following condition is satisfied:

$$\text{“}dist(p_i, p_c) \leq \varepsilon \ \wedge \ d_v(A_c, F_i) \leq R_1 \ \wedge \ d_v(A_c, F_i) \geq R_2 \ , \ \ for\ i \ \in \ \{i, \ldots, n\}\text{”}$$

$$\vec{v}_{ar} \ = \ \frac{\sum_{i=1}^{n} \vec{v}_i}{n} \tag{3.14}$$

Briefly, the condition that $p_i$ is within the neighbourhood of $p_c$ for neighbor agents $F_i$ that are located between the visibility and defense range of the agent $A_c$, then $\vec{v}_{ar}$ is considered for determining the new velocity of $A_c$. Fig. 3.5 shows a sample case where the agent $A_c$ (yellow triangle) adjusts its velocity according to the alignment rule. The figure is composed of two snapshots of the agent $A_c$; the left one represents the state where $A_c$ calculates its velocity considering the neighbor agents (blue triangles), and the right one represents the state right after it applies the force to satisfy new velocity. While green dotted circle represents the limit of the visibility range of $A_c$, the red dotted one stands for defense range. The black dot at the top corner of the triangles (agents) indicate the direction to where agents are heading. Assume the horizontal axis as 0°, then $A_c$ is heading to 135°, and neighbour agents are heading to 90°. In order $A_c$ to move in the same direction with other agents, it must rotate by an angle of 45°. In order to achieve that, a force of 45°is needed to be applied. This is shown as a purple vector. Thus, $A_c$ now points in direction the same direction with the neighbor agents as shown in the right snapshot.

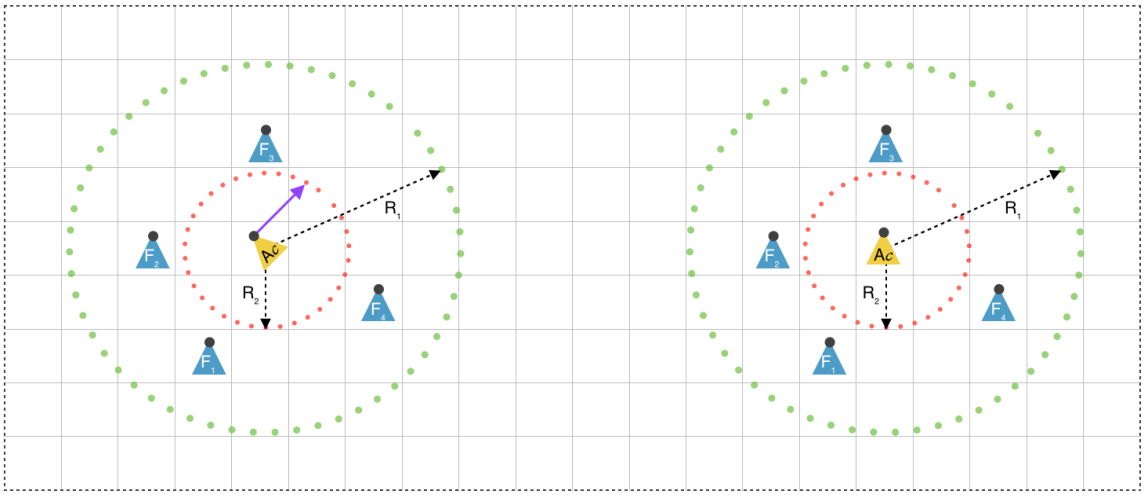


Figure 3.5: Graphical representation of the alignment rule.

#### 3.4.1.2 Separation

The separation rule is applied to agent $A_c$ not only to protect itself from any collision but also move away from those agents, which are not similar in the feature space. The new velocity vector is determined as a vector from the centroid of the neighbor

agents $C_{db}$, which are dissimilar and violates the defense range of the agent $A_c$, to the current position $P_c$ of $A_c$ in the virtual space. The midpoint or centroid of a set of agents can be calculated such that $C = \left(\frac{1}{n}\sum_{i=1}^{n} P_i\right)$. This rule is applied as $\overrightarrow{C_{db}P_c}$, if the following condition is satisfied:

"$dist(p_i, p_c) > \varepsilon \quad \vee \quad d_v(A_c, F_i) \leq R_2\ , \quad for\ i\ \in\ \{i, \ldots, n\}$"

Consider Fig. 3.6, which plots an example case for the separation rule. In the first snapshot, you will see four agents, one yellow and three blue agents. Similar to the case in Fig. 3.5, the yellow agent represents the current agent $A_c$, and blue agents represent the neighbor agents. It is obvious that all blue agents violate the defense range of the yellow agent. In this case, the yellow agent tries to move in the opposite direction of the middle point of neighbor agents. This is indicated as a black star in the figure. In that point, the yellow agent moves to the right by one cell. The final position of $A_c$ is illustrated in the second snapshot. Unfortunately, two of the neighbor agents are still in the defense range and as described before, agents can move only one cell at a time. However, in every timestamp, agents calculate their new velocity and take actions accordingly.

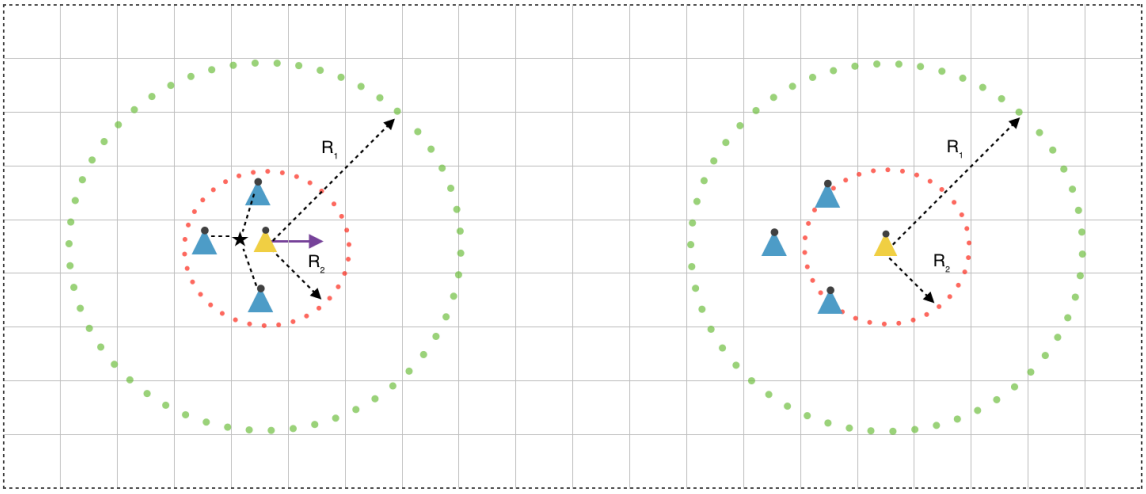


Figure 3.6: Graphical representation of the separation rule.

#### 3.4.1.3 Cohesion

In this rule, an agent tends to move toward the centroid of neighboring agents which are similar, in the neighborhood of $\varepsilon$, and in the visibility range of agent $A_c$. Consider

that moving toward the centroid must not cause a violation for the agents in the neighborhood. The direction of the vector is specified as a vector from the current position $P_c$ of an agent $A_c$ to the centroid of the neighbor agents $C_{nb}$. It can be denoted as $\overrightarrow{P_c C_{nb}}$, if the following condition is satisfied:

"$dist(p_i, p_c) \leq \varepsilon \ \wedge \ d_v(A_c, F_i) \leq R_1 \ \wedge \ d_v(A_c, F_i) \geq R_2 \ , \ \ for\ i \ \in \ \{i, \ldots, n\}$"

Fig. 3.7 shows an example of an agent behavior to satisfy the cohesion rule. Similar to previous representations, teh yellow agent is the current agent $A_c$, and the blue agents are the neighbor agents within the visibility range of $A_c$. Again, Fig. 3.7 shows the previous and the next state of the current agent from left to right, respectively. The black star denotes the middle point of three blue agents. In this situation, $A_c$, the yellow agent, needs to change its location in the direction of the purple vector, so that it is located in the middle of the flock. This provides better solidarity between similar agents. As you can see in the second snapshot, the final position of the yellow agent ensures the cohesion rule.

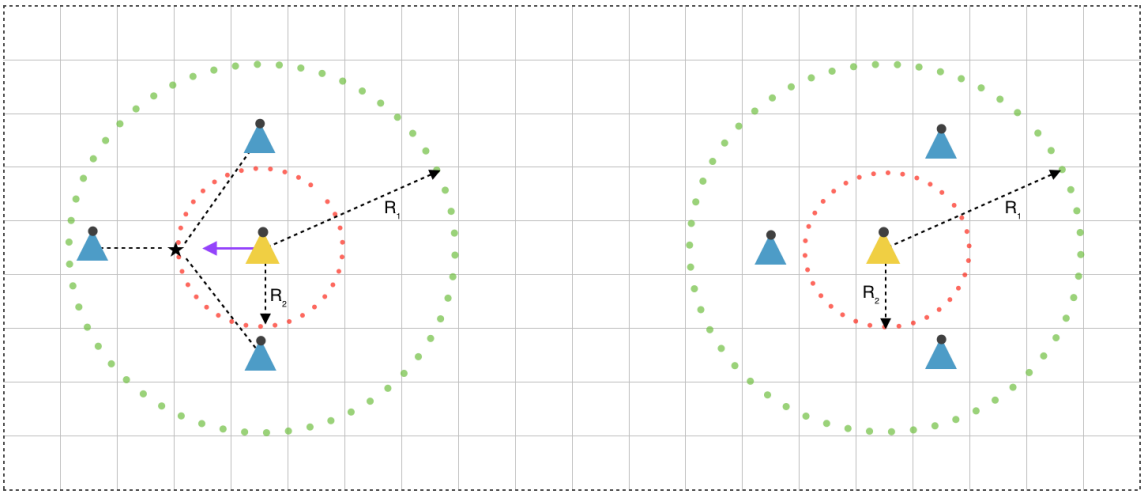


Figure 3.7: Graphical representation of the cohesion rule.

As mentioned, Flockstream is an agent-based algorithm and there are three types of agents used in the clustering process. These are: (i) *basic* agents, which represent the new data points arriving to the system; (ii) *p-representative* agents, which represent the p-micro-clusters; and (iii) *o-representative* agents, which represent the o-micro-clusters. The algorithm is composed of two phases, which are the initialization phase, and the maintenance and clustering phase. In the initialization phase,

the first bulk of basic agents are deployed to the system. After initialization, micro-cluster maintenance and clustering phase takes place for the all three types of agents present in the virtual space.

### 3.4.2 Initialization

At the initialization phase, the predetermined number of basic agents are created and deployed to random positions in the virtual space. Initially, the velocity vectors of basic agents $\vec{v} = (m, \theta)$ are assigned such that the magnitude $m$ equals to 1 and the angle $\theta$ is a random value within the range $[0, 360]$. Agent *ids* are incrementally assigned from 1 to $n$, where n is the total number of data points.

For a predefined number of iterations, the agents move and interact with each other on the virtual space simultaneously according to the MSF rules described above. Each basic agent is influenced by the neighbor agents in its visibility range. In every iteration, the velocity vector of each basic agent is calculated and assigned for the next move. During this process, similar agents (based on a Euclidean distance between the data points from the feature space) are apt to move together as a flock, whereas dissimilar agents move away from the flocks. Although an agent can be involved in a flock, it may disjoin due to the change in flocking behavior or effect of the MSF rules. At the end of this phase, two kind of basic agents remain; those that belong to one of the flocks in the system and those that do not.

At the end of the initialization phase, formed flocks turn into representative agents and the corresponding basic agents in the flocks will be represented by those agents. Representative agents can be divided into two types, p-representative and o-representative agents. The definitions of these concepts are explained in detail in the DenStream section such that, p-micro-cluster $c_p = \{CF^1, CF^2, w\}$ and o-micro-cluster $c_o = \{CF^1, CF^2, w, t_o\}$, respectively. In the mean time, the summary statistics of the basic agents in the flocks are computed and stored by the corresponding representative agent. Finally, these basic agents are discarded from the system.

### 3.4.3 Maintenance and Clustering

When initialization has finished, there are three types of agents remaining in the system; p-representative, o-representative, and basic agents that were not involved in

a flock at the initialization process. The purpose of this phase is both to maintain the $p$ and $o$ representative agents associated with the $p$ and $o$ micro-clusters in the feature space, and to perform online clustering. In the next iteration, a new bulk of basic agents is accepted to the system. Note that, the stream speed and the maximum number of iterations are specified by the user at the beginning of the process. Similar to the initialization phase, agents move around the environment with respect to the MSF rules. However, because the types of the agents are not the same, similarities between those agents are calculated as follows.

- Basic $\rightarrow$ Basic : In the case that a basic agent A, associated with a data point $p_A \in \mathcal{R}^d$ meets basic agent B, associated with a data point $p_B \in \mathcal{R}^d$, then they are considered similar if $dist\,(p_A, p_B) \leq \varepsilon$.

- Basic $\rightarrow$ Representative : In the case that a basic agent A meets either a p-representative B, i.e. p-micro-cluster $c_p^B$ , or an o-representative B, i.e. o-micro-cluster $c_o^B$, then $p_A$ is added to the copy of micro-cluster $c_p^B$ or $c_o^B$ to obtain a new radius. If the new radius $r_p$ or $r_o \leq \varepsilon$, then they are considered similar.

- Representative $\rightarrow$ Basic : In the case that a p- or o-representative agent A, meets a basic agent B associated with a data point $p_B$, then they are considered similar if Euclidean distance between the center of the micro-cluster and $p_B \leq \varepsilon$.

- Representative $\rightarrow$ Representative : In the case that a p-representative A, i.e., p-micro-cluster $c_p^A$ , or an o-representative A, i.e., o-micro-cluster $c_o^A$ , meets another representative agent, then they are considered to be similar if Euclidean distance between the centers of the micro-clusters less than or equal to $\varepsilon$.

Unlike the original Flockstream, agents can move on the virtual space by using the similarity functions described above. When the number of maximum iterations achieved, flocks are formed containing similar agents from all types. In that case, the maintenance is performed using the following algorithm. So that, basic agents can interact with not only each other, but also with the representative agents. Additionally, the representative agents can be compared with each other with respect to the centers of their micro-clusters. Moreover, I added one more feature to the

agents. This is called *flock-by*, which differs from the original paper. In this case, let us assume $A_c$ as the current agent. If $A_c$ encounters with another similar agent from a flock during its movement in the virtual space, then the $A_c$ may join to the flock if the conditions below are satisfied. When it is joined to the flock, that similar agent's ID will be stored by the $A_c$. In the further iterations, if $A_c$ encounters with another agent, and its flock-by agent is not present in its visibility range, then it may change its flock with the new similar agent's flock, if the conditions below are satisfied.

i. If flock consist of only basic agents, then a new o-representative agent is created representing the basic agents in the flock. After that, those basic agents are removed from the virtual space.

ii. If flock contains only one representative agent and others are basic agents, then all basic agents are added to the representative agent. Once this is done, its weight is calculated to compare with the value $\beta\mu$, which is a threshold for promoting representatives mentioned in DenStream. If representative agent is a p-representative, corresponding to p-microcluster and its weight $w$ below $\beta\mu$, then its type demoted to o-representative. On the other hand, if the weight of an o-representative $w$ is above $\beta\mu$, it is promoted to to the p-representative.

iii. If there are more than one representatives in a flock, then they become a swarm agent by merging the micro-clusters of representative agents. Later in the process, it acts as a representative agent for the calculations.

**Algorithm 2** Interaction of agents after the initialization phase.

**for** each flock $F$ in the Virtual Space **do**
  check the type of each agent in $F$
  **if** the type of all agents is basic **then**
    **if** number of agents $\geq \mu$ **then**
      create a new p-representative agent
    **else if** number of agents $\leq \mu$ **then**
      create a new o-representative agent
    **end if**
  **end if**
  **if** there is only one representative agent $A_r$ in $F$ **then**
    insert all other basic agents to $A_r$
    **if** $A_r$ is p-representative $\wedge$ its weight $\omega \leq \beta\mu$ **then**
      diminish $A_r$ to o-representative
    **else if** $A_r$ is o-representative $\wedge$ its weight $\omega \geq \beta\mu$ **then**
      promote $A_r$ to p-representative
    **end if**
  **end if**
  **if** there is more than one representative agent in $F$ **then**
    merge representative agents and insert basic agents into it
    label new representative agent as a *swarm*
  **end if**
**end for**

# Chapter 4

# Evaluation

In this section, I present the evaluation metrics, which are used to compare the algorithms, and the experiments that are performed on all of the data sets employed in this thesis. In the following experiments, the criteria of "best" can be stated as generating the purest possible clusters with a reasonable precision in lowest possible execution time.

## 4.1 Parameterization

The DenStream parameters are a subset of those used for Flockstream; hence, a common process for parameterization is assumed. In the case of CluStream and ClusTree, the recommendations from the MOA distribution are assumed. Table 4.1 summarizes the resulting parameterization for the three MOA sourced algorithms. Flockstream assumes the parameters of DenStream, plus: 1) $MaxIterations$ defining the maximum number of iterations; and, 2) $d$ defining the size of the virtual space. The authors of Flockstream suggest that if the stream speed is $v$ (aka size of the non-overlapping window interface to the data stream), then the size of the virtual space parameterized such that $d \times d \geq 4 \times v$. The maximum stream speed used in my experiments is 1000. Therefore, the *minimum* size of the virtual space would be $64 \times 64$. A virtual space value of $100 \times 100$ was adopted in order to reduce the congestion resulting from agent the immobility.

Flockstream includes two additional parameters, which are $MaxIterations$ for the maximum number of iterations and $d$ for the dimensions of the virtual space. They are fixed to 800 and 100, respectively. The authors [13] of Flockstream suggest that if the stream speed is $v$, then dimensions of the virtual space can be specified such that $d \times d \geq 4 \times v$. The maximum stream speed used in my experiments is 1000. Therefore, dimensions of thevirtual space should be roughly more than $64 \times 64$. However, I picked a $100 \times 100$ virtual space to reduce teh congestion resulting from

immobility. Another difference between parameters is the offline multiplier used in DenStream to calculate the final value of epsilon. In Flockstream, I only use epsilon. The following sections cover the topics of evaluation metrics.

Table 4.1: Descriptions of parameters of each algorithm.

| Algorithms | Parameters | Descriptions |
|---|---|---|
| CluStream | -h (d: 1000) | Range of the window. |
| | -k (d: 100) | Maximum number of micro kernels to use. |
| | -t (d: 2) | Multiplier for the kernel radius. |
| | -M | Evaluate the underlying micro-clustering instead of the macro-clustering. |
| DenStream | -h (d: 1000) | Range of the window. |
| | -e (d: 0.02) | Defines the epsilon neighbourhood. |
| | -b (d: 0.2) | Beta ($\beta$) constant. |
| | -m (d: 1) | Mu ($\mu$) constant. |
| | -i (d: 1000) | Number of points to use for initialization. |
| | -o (d: 2) | Offline multiplier for epsilon. |
| | -l (d: 0.25) | Lambda ($\lambda$) constant. |
| | -s (d: 100) | Number of incoming points per time unit. |
| | -M | Evaluate the underlying micro-clustering instead of the macro-clustering. |
| ClusTree | -h (d: 1000) | Range of the window. |
| | -H (d: 8) | The maximal height of the tree |
| | -M | Evaluate the underlying micro-clustering instead of the macro-clustering. |
| FlockStream | -d (d: 100) | Dimensions of Virtual Space. |
| | -e (d: 0.1) | Defines the epsilon neighbourhood. |
| | -b (d: 0.2) | Beta ($\beta$) constant. |
| | -m (d: 10) | Mu ($\mu$) constant. |
| | -i (d: 300) | Number of points to use for initialization. |
| | -x (d: 800) | Maximum number of iterations. |
| | -l (d: 0.25) | Lambda ($\lambda$) constant. |
| | -s (d: 300) | Number of incoming points per time unit. |

## 4.2 Evaluation Metrics

All algorithms used in this thesis are evaluated according to three performance metrics, which are *macro-purity*, *micro-precision*, and *recall*. In MOA, these metrics are defined as *Purity*, *F1-P*, and *F1-R* respectively [23]. Additionally, I used *micro-purity* and *macro-precision* in the parameter sensitivity analysis. In fact, Purity and F1-P from MOA correspond to *macro-purity* and *micro-precision*, respectively.

Micro and macro measures are calculated in the same manner except with respect to the confusion matrices they use. Whereas micro confusion matrix is used for micro metrics, macro confusion matrix is used for macro metrics. The only difference

between matrices is that the macro confusion matrix groups all the clusters that share the same majority label in the micro confusion matrix. Before defining each of these metrics, let us consider the confusion matrix in Table 4.2. In this table, $CR_i$ represents the found cluster, whereas $CS_j$ represents the class value. In every found cluster, the number of data points in each class is represented with $v_{i_j}$. The precision, recall and $F_1 - Score$ of the found cluster $CR_i$ can be calculated with the equations Eq.4.1, Eq.4.2, and Eq.4.3, respectively.

$$precision_{CR_i} = \frac{\max(v_{i1}, \ldots, v_{im})}{\sum_{j=1}^{m} v_{ij}} \tag{4.1}$$

$$recall_{CR_i} = \frac{\max(v_{i1}, \ldots, v_{im})}{\sum_{i=1}^{n} v_{ij}} \tag{4.2}$$

$$F_1 - Score_{CR_i} = 2 \cdot \frac{precision_{CR_i} \cdot recall_{CR_i}}{precision_{CR_i} + recall_{CR_i}} \tag{4.3}$$

In the recall equation, $j$ is the index of the maximum class value for the cluster $CR_i$. The arithmetic means of the $precision_{CR_i}$ and $F_1 - Score_{CR_i}$ give us the Purity and F1-P metrics (Eq. 4.4 and Eq. 4.5), respectively. In the equations, $n$ denotes the total number of found clusters.

$$Purity = \frac{\sum_{i}^{n} precision_{CR_i}}{n} \tag{4.4}$$

$$F1 - P = \frac{\sum_{i=1}^{n} F_1 - Score_{CR_i}}{n} \tag{4.5}$$

The final performance measure of the MOA used in the experiments is the F1-R. In order to calculate it, first the maximum $F_1 - Score$ (see Eq. 4.6 and Eq. 4.7) need to be calculated for each value $v_{i_j}$ in the class $CS_j$ such that the arithmetic mean of the sum of maximum $F_1 - Scores$ with respect to the total number of classes $m$ gives the final result for F1-R as shown in Eq. 4.8.

$$F1 - Score_{v_{ij}} \;=\; 2 \;\cdot\; \frac{precision_{v_{ij}} \;\cdot\; recall_{v_{ij}}}{precision_{v_{ij}} \;+\; recall_{v_{ij}}} \tag{4.6}$$

$$Max\ F1 - Score_{CS_j} \;=\; \max\left(F1 - Score_{v_{1j}},\ F1 - Score_{v_{2j}},\ \ldots,\ F1 - Score_{v_{nj}}\right) \tag{4.7}$$

Table 4.2: An Example of a Confusion Matrix

| | $CS_1$ | ... | $CS_m$ | $Sum\ of\ Cluster\ Values$ |
|---|---|---|---|---|
| $CR_1$ | $v_{1_1}$ | ... | $v_{1_m}$ | $v_{1_1} + \ldots + v_{1_m}$ |
| $\vdots$ | $\vdots$ | ... | $\vdots$ | $\vdots$ |
| $CR_n$ | $v_{n_1}$ | ... | $v_{n_m}$ | $v_{n_1} + \ldots + v_{n_m}$ |
| $Sum\ of\ Class\ Values$ | $v_{1_1} + \ldots + v_{n_1}$ | ... | $v_{1_m} + \ldots + v_{n_m}$ | $Total$ |

$$F1 - R \;=\; \frac{\sum_{j\,=\,1}^{m} Max\ F1 - Score_{CS_j}}{m} \tag{4.8}$$

### 4.3 Parameter Sensitivity Analysis for Flockstream

In this section, I present the preliminary results of modified Flockstream algorithm on each data set. I have excluded Commercial-Big because it shares similar features with the Commercial-Small data set. The results are evaluated according to the metrics discussed in the previous section. Additionally, two metrics, specifically, the number of clusters obtained and the total execution time in minutes, are added for comparison.

It is important to note that the parameter epsilon plays a vital role in the experiments and it differs from data to data. For that reason, I have chosen different epsilon values for data sets as a default value. Default values of other parameters are summarized in Table 4.1. Based on those parameters, I performed several experiments by tuning epsilon, stream speed and the initial number of agents.

In the analysis of commercial data sets, I realized that most of the time, a high percentage of the data is represented by only one cluster. In order to obtain purer clusters, I made an inner cluster analysis so that the data points of the most populated

cluster is processed one more time with Flockstream. I used this technique only for commercial data sets because clusters generated by public data sets contain relatively less number of data points and there were no need to make any further analysis for the most populated cluster. As a result of this recursive technique, the clusters formed seem to be purer. The results are presented below for both the public and the commercial data sets, as well as the inner cluster analysis of the small commercial data set.

### 4.3.1 Electricity Data Set

Initially, I begin with Electricity data set. As can be seen from Fig. 3.1, it is the most balanced data set compared to others with a ratio of 1 : 1.38. Besides, it contains the minimum number of instances with only 8 features that provide significantly low execution times in the experiments.

While the parameters stream speed and initial agents are set to 300, epsilon is increased from 0.1 to 1.0 by 0.1. Table 4.3 presents the results with each evaluation metrics. The first of these metrics is macro-purity. It starts with 78% and gradually decreases till to 59%. As you notice even from this result, epsilon has a great impact on the performance. On the other hand, micro-purity shows almost the same trend with macro-purity. Simply, micro-purity is always less than macro-purity with a $2\% - 10\%$ difference.

Table 4.3: Results of Flockstream on Electricity data set: Change in Epsilon.

| -e | Macro Purity | Micro Purity | Macro F-Score | Micro F-Score | NoC | ET (min) |
|---|---|---|---|---|---|---|
| 0.1 | 0.7845 | 0.7390 | 0.7274 | 0.0022 | 1301 | 106 |
| 0.2 | 0.7281 | 0.6672 | 0.6566 | 0.0066 | 389 | 26 |
| 0.3 | 0.7051 | 0.6266 | 0.5913 | 0.0102 | 225 | 13 |
| 0.4 | 0.6640 | 0.5966 | 0.5288 | 0.0141 | 147 | 9 |
| 0.5 | 0.6829 | 0.5815 | 0.4292 | 0.0196 | 95 | 7 |
| 0.6 | 0.6492 | 0.5798 | 0.4002 | 0.0257 | 67 | 7 |
| 0.7 | 0.6345 | 0.5773 | 0.3856 | 0.0261 | 66 | 7 |
| 0.8 | 0.6149 | 0.5805 | 0.4505 | 0.0357 | 49 | 6 |
| 0.9 | 0.6268 | 0.5778 | 0.3918 | 0.0352 | 43 | 6 |
| 1.0 | 0.5958 | 0.5762 | 0.3760 | 0.0368 | 44 | 6 |

Moreover, Table 4.4 shows the results of stream speed experiments. The rate of micro purity is likely to remain steady, although it oscillates between $58 - 62\%$. This experiment reveals that there has been a gradual increase in macro-precision, and also

a slight decrease in micro-precision with respect to the raise in stream speed. Also, it does not make a major impact on the execution time. The number of clusters found shows an increasing trend from top to bottom. This is because of the larger amount of data points deployed all over the virtual space causing more interaction between the agents. Thus, the higher numbers of clusters are created throughout the process.

Table 4.4: Results of Flockstream on Electricity data set: Change in Stream Speed.

| -s | Macro Purity | Micro Purity | Macro F-Score | Micro F-Score | NoC | ET (min) |
|---|---|---|---|---|---|---|
| 100 | 0.6189 | 0.5756 | 0.3682 | 0.0503 | 18 | 5 |
| 200 | 0.5927 | 0.5757 | 0.3715 | 0.0548 | 24 | 6 |
| 300 | 0.5958 | 0.5762 | 0.3760 | 0.0368 | 44 | 6 |
| 400 | 0.5938 | 0.5770 | 0.3817 | 0.0313 | 53 | 6 |
| 500 | 0.6134 | 0.5770 | 0.3874 | 0.0338 | 50 | 7 |
| 600 | 0.5992 | 0.5765 | 0.3831 | 0.0301 | 59 | 7 |
| 700 | 0.5994 | 0.5781 | 0.3967 | 0.0231 | 79 | 8 |
| 800 | 0.5919 | 0.5794 | 0.3920 | 0.0244 | 76 | 8 |
| 900 | 0.5833 | 0.5772 | 0.4290 | 0.0274 | 68 | 9 |
| 1000 | 0.5873 | 0.5766 | 0.4087 | 0.0267 | 70 | 9 |

In Table 4.5, the results of experiments on change in initial agents are presented. It is not easy to say that there is a pattern on the obtained macro-purity values. Similar to stream speed experiments, micro-purity is fixed to 57%. Trend in purity values is almost valid for precision values as well. In terms of macro perspective, I obtain precision values within $37 - 41\%$ without an order. Also, micro-precision has almost never changed. Naturally, there is no change in the execution time and the numbers of clusters are almost the same for all experiments.

Table 4.5: Results of Flockstream on Electricity data set: Change in Initial Agents.

| -i | Macro Purity | Micro Purity | Macro F-Score | Micro F-Score | NoC | ET (min) |
|---|---|---|---|---|---|---|
| 100 | 0.6204 | 0.5771 | 0.3848 | 0.0322 | 44 | 6 |
| 200 | 0.6148 | 0.5771 | 0.3814 | 0.0315 | 45 | 6 |
| 300 | 0.5958 | 0.5762 | 0.3760 | 0.0368 | 44 | 6 |
| 400 | 0.6219 | 0.5787 | 0.4071 | 0.0325 | 49 | 6 |
| 500 | 0.6151 | 0.5773 | 0.4018 | 0.0338 | 45 | 6 |
| 600 | 0.6204 | 0.5777 | 0.3850 | 0.0422 | 39 | 6 |
| 700 | 0.6048 | 0.5769 | 0.3771 | 0.0389 | 41 | 6 |
| 800 | 0.6178 | 0.5780 | 0.3956 | 0.0318 | 49 | 6 |
| 900 | 0.6186 | 0.5769 | 0.3760 | 0.0385 | 39 | 6 |
| 1000 | 0.6331 | 0.5780 | 0.4032 | 0.0295 | 54 | 6 |

### 4.3.2 Forest Covertype Data Set

In the experiments on Covertype data set, the default epsilon value has been determined as 0.8. As you can see in Table 4.6, the results for $epsilon = 0.1$ can not be generated. This value is so low that most of the similarity comparisons returns false. Thus, the number of agents remain in the system gradually increases and at one point virtual space becomes fully loaded to take another agent. That is why, the process can not be completed for that experiment.

In the epsilon experiments, approximately 10% change in macro-purity can be observed. Although the maximum macro-purity is achieved when $epsilon = 0.2$, execution time of that experiment is infeasible. Instead, the experiment with $epsilon = 1.0$ provides a reasonable purity with much lower execution time and less number of clusters. The same situation for purity is valid for micro perspective as well. On the other hand, macro-precision decreases from 51% to 23%, while epsilon value increases. However, micro-precision shows an increasing trend for a similar change in epsilon.

Table 4.6: Results of Flockstream on Covertype data set: Change in Epsilon.

| -e | Macro Purity | Micro Purity | Macro F-Score | Micro F-Score | NoC | ET (min) |
|---|---|---|---|---|---|---|
| 0.1 | - | - | - | - | - | - |
| 0.2 | 0.8564 | 0.7115 | 0.5116 | 0.0066 | 988 | 1217 |
| 0.3 | 0.8310 | 0.6786 | 0.4527 | 0.0127 | 443 | 594 |
| 0.4 | 0.8099 | 0.6488 | 0.4364 | 0.0191 | 276 | 375 |
| 0.5 | 0.8028 | 0.6283 | 0.4298 | 0.0200 | 246 | 267 |
| 0.6 | 0.8201 | 0.6499 | 0.4244 | 0.0233 | 207 | 215 |
| 0.7 | 0.8108 | 0.6386 | 0.4015 | 0.0250 | 178 | 179 |
| 0.8 | 0.7913 | 0.6459 | 0.3558 | 0.0267 | 133 | 153 |
| 0.9 | 0.7766 | 0.6079 | 0.3036 | 0.0315 | 84 | 123 |
| 1.0 | 0.7646 | 0.5739 | 0.2319 | 0.0327 | 57 | 115 |

It is hard to estimate what may be the next purity value in the stream speed experiments on Covertype. In Table 4.7, it can be seen that macro-purity increases until $streamspeed = 600$, and then becomes steady around 77%. Likewise, micro-purity has variable values between $53 - 61\%$. Increase in stream speed effects the precision in the opposite direction of the way it effects the epsilon. Whereas as the macro-precision increases, the micro-precision decreases. Additionally, the execution time and the number of clusters increase as the stream speed increases.

As observed in experiments of Electricity data set, change in the number of initial agents does not have that much impact on the results, as shown in Table 4.8. There

Table 4.7: Results of Flockstream on Covertype data set: Change in Stream Speed.

| -s | Macro Purity | Micro Purity | Macro F-Score | Micro F-Score | NoC | ET (min) |
|---|---|---|---|---|---|---|
| 100 | 0.7467 | 0.5043 | 0.1284 | 0.0363 | 25 | 89 |
| 200 | 0.7797 | 0.5432 | 0.2294 | 0.0432 | 42 | 100 |
| 300 | 0.7646 | 0.5739 | 0.2319 | 0.0327 | 56 | 115 |
| 400 | 0.7788 | 0.6069 | 0.3201 | 0.0295 | 97 | 143 |
| 500 | 0.7815 | 0.6217 | 0.2812 | 0.0211 | 144 | 158 |
| 600 | 0.7932 | 0.6077 | 0.3173 | 0.0178 | 163 | 171 |
| 700 | 0.7772 | 0.6372 | 0.3384 | 0.0179 | 179 | 184 |
| 800 | 0.7777 | 0.6343 | 0.3527 | 0.0174 | 212 | 197 |
| 900 | 0.7703 | 0.6241 | 0.3300 | 0.0168 | 216 | 205 |
| 1000 | 0.7689 | 0.6270 | 0.3413 | 0.0156 | 236 | 218 |

are some changes, but it is not easy to make a definite decision on trends.

Table 4.8: Results of Flockstream on Covertype data set: Change in Initial Agents.

| -i | Macro Purity | Micro Purity | Macro F-Score | Micro F-Score | NoC | ET (min) |
|---|---|---|---|---|---|---|
| 100 | 0.7275 | 0.5794 | 0.2266 | 0.0299 | 60 | 127 |
| 200 | 0.7649 | 0.5914 | 0.2400 | 0.0328 | 63 | 122 |
| 300 | 0.7646 | 0.5739 | 0.2319 | 0.0327 | 56 | 115 |
| 400 | 0.7852 | 0.6065 | 0.2471 | 0.0304 | 64 | 123 |
| 500 | 0.7541 | 0.5308 | 0.1838 | 0.0238 | 60 | 122 |
| 600 | 0.7520 | 0.5840 | 0.2550 | 0.0297 | 72 | 124 |
| 700 | 0.7424 | 0.5817 | 0.2636 | 0.0345 | 65 | 122 |
| 800 | 0.7440 | 0.5976 | 0.2556 | 0.0359 | 60 | 122 |
| 900 | 0.7453 | 0.5713 | 0.2373 | 0.0326 | 58 | 124 |
| 1000 | 0.7625 | 0.5694 | 0.2471 | 0.0322 | 62 | 122 |

### 4.3.3 Commercial-Small Data Set

In this section, I discussed the results of the Flockstream on the small commercial data set both as a whole and two level of inner clusters. In this case, the epsilon parameter is fixed to 0.2. I observed the distribution of the results according to change in the parameters of epsilon, stream speed and initial number of agents. First, I will cover the results of the data set as a whole, and then continue with the inner cluster analysis.

#### 4.3.3.1 Overall Data Set Analysis

In the experiments on Commercial-Small data set, I have specified epsilon value as 0.2. The results are examined according to the change in epsilon and stream speed.

Table 4.9, 4.10, and 4.11 summarize the results. The parameters are mostly chosen to examine the wide range of values. However, given the memory and time limitations, I was compelled to restrict the range of values in some of the experiments.

Table 4.9 shows the results of the Flockstream based on the epsilon values 0.05, 0.1, 0.2, and 0.3. The decrease in epsilon value shows a tendency to increase in macro-purity and micro-purity measures. Lower epsilon values decrease the similarity threshold, more similar data points are thus aggregated and more pure clusters are formed. However, one of the disadvantages of this approach is that the execution time gradually increases. The number of agents retained in the virtual space during the maintenance process causes more comparison and significantly increase the execution time. As an example, the experiment with 0.1 epsilon is three times faster than the one with 0.05 epsilon. Nevertheless, it is obvious that the lower epsilon values provide lots of pure clusters, compared to the higher values of epsilon.

In the stream speed experiments, provided in Table 4.10, I tried a large scale of values from 200 to 1000. The change in stream speed does not impact the execution time as much as epsilon experiments. Similarly, there is no pattern in macro F-Score, but a slight change can be observed the micro one. Micro purity, meanwhile, remains almost the same. However, there is approximately 5% change in macro purity from the stream speed 200 to 1000. Although I obtain fewer number of clusters in lower stream speed, they are more pure than the ones in higher speeds.

The change in the number of initial agents does not make a significant effect to the results, as you can see in Table 4.11. There are some slight changes in macro measures but it is hard to say that there is a pattern. For that reason, I did not pursue the sensitivity analysis of the number of initial agents for the inner cluster analysis.

Table 4.9: Results of Flockstream on Commercial-Small data set: Change in Epsilon.

| -e | Macro Purity | Micro Purity | Macro F-Score | Micro F-Score | NoC | ET (min) |
|---|---|---|---|---|---|---|
| 0.05 | 0.9705 | 0.9231 | 0.6043 | 0.0025 | 840 | 1097 |
| 0.1 | 0.9552 | 0.9199 | 0.6052 | 0.0078 | 228 | 371 |
| 0.2 | 0.9365 | 0.9132 | 0.5051 | 0.0158 | 90 | 297 |
| 0.3 | 0.9456 | 0.9175 | 0.5699 | 0.0061 | 269 | 524 |

Table 4.10: Results of Flockstream on Commercial-Small data set: Change in Stream Speed.

| -s | Macro Purity | Micro Purity | Macro F-Score | Micro F-Score | NoC | ET (min) |
|---|---|---|---|---|---|---|
| 200 | 0.9454 | 0.9135 | 0.5130 | 0.0192 | 75 | 252 |
| 300 | 0.9365 | 0.9132 | 0.5051 | 0.0158 | 90 | 297 |
| 400 | 0.9244 | 0.9153 | 0.5444 | 0.0149 | 109 | 302 |
| 500 | 0.9299 | 0.9165 | 0.5593 | 0.0109 | 144 | 381 |
| 600 | 0.9221 | 0.9149 | 0.5219 | 0.0103 | 137 | 345 |
| 700 | 0.9033 | 0.9166 | 0.5651 | 0.0091 | 178 | 364 |
| 800 | 0.8886 | 0.9157 | 0.5488 | 0.0080 | 192 | 391 |
| 900 | 0.8938 | 0.9162 | 0.5815 | 0.0075 | 220 | 413 |
| 1000 | 0.8999 | 0.9198 | 0.6240 | 0.0083 | 220 | 439 |

Table 4.11: Results of Flockstream on Commercial-Small data set: Change in Initial Agents.

| -i | Macro Purity | Micro Purity | Macro F-Score | Micro F-Score | NoC | ET (min) |
|---|---|---|---|---|---|---|
| 100 | 0.9462 | 0.9135 | 0.4980 | 0.0171 | 92 | 279 |
| 200 | 0.9305 | 0.9142 | 0.5119 | 0.0163 | 86 | 277 |
| 300 | 0.9365 | 0.9132 | 0.5051 | 0.0158 | 90 | 297 |
| 400 | 0.9441 | 0.9142 | 0.5402 | 0.0143 | 100 | 274 |
| 500 | 0.9353 | 0.9135 | 0.4936 | 0.0152 | 89 | 274 |
| 600 | 0.9333 | 0.9133 | 0.5024 | 0.0161 | 95 | 274 |
| 700 | 0.9354 | 0.9130 | 0.4907 | 0.0138 | 98 | 276 |
| 800 | 0.9312 | 0.9129 | 0.4961 | 0.0141 | 97 | 275 |
| 900 | 0.9361 | 0.9144 | 0.5787 | 0.0201 | 90 | 274 |
| 1000 | 0.9331 | 0.9171 | 0.5735 | 0.0165 | 95 | 274 |

#### 4.3.3.2 Inner Cluster Set Analysis

As mentioned earlier, the purpose of the inner cluster analysis is to split highly populated clusters obtained from the Flockstream process into well-grained and pure clusters. In order to achieve this, I examined two level of inner clusters because the first layer analysis did not provide us the expected granularity.

##### 4.3.3.2.1 First Layer

In the first layer, I have chosen the epsilon value as 0.2 because it is observed that it outputs relatively small number of clusters and less execution time. Although it results in less purity in the epsilon experiments, this still provides more than 90% percent purity. In that experiment, almost 58% of the data points (almost 1M points) were represented by one cluster. That's why, I ran the Flockstream for that cluster by using its data points as the input. The benefit of analyzing inner clusters is that

it allows us to work with lower epsilon values in less amount of time. As a result, the better granularity is achieved on those data points.

Table 4.12 and Table 4.13 refer to the results of epsilon and stream speed experiments. The range of the epsilon values is determined between the interval of 0.025 and 0.2. It is clear that the execution time of the first layer analysis is almost three times faster than the whole data set analysis, when I compared the experiments with the same epsilon value. Additionally, the purity of the experiments tend to increase, similar to the previous experiments. The effect of the stream speed is even less, only 2% change on macro purity between stream speeds. Apart from that, other metrics show similar characteristics with the analysis of stream speed on the whole data set. By means of first layer analysis, I achieved approximately 60% decrease on the most populated data set with less than half a million data points. However, it was still insufficient to interpret the data correctly. That is why I analyzed one more level. Similarly, the cluster with the highest number of data points is selected from the related experiment and its data points were processed in the second layer.

Table 4.12: Results of Flockstream on First Inner Cluster of Commercial-Small data set: Change in Epsilon.

| -e | Macro Purity | Micro Purity | Macro F-Score | Micro F-Score | NoC | ET (min) |
|---|---|---|---|---|---|---|
| 0.025 | 0.9880 | 0.9588 | 0.6411 | 0.0019 | 1144 | 1146 |
| 0.05 | 0.9762 | 0.9532 | 0.5307 | 0.0048 | 358 | 381 |
| 0.075 | 0.9795 | 0.9523 | 0.5134 | 0.0086 | 170 | 205 |
| 0.1 | 0.9627 | 0.9517 | 0.5053 | 0.0127 | 97 | 182 |
| 0.2 | 0.92512 | 0.9518 | 0.5098 | 0.0282 | 37 | 185 |

Table 4.13: Results of Flockstream on First Inner Cluster of Commercial-Small data set: Change in Stream Speed.

| -s | Macro Purity | Micro Purity | Macro F-Score | Micro F-Score | NoC | ET (min) |
|---|---|---|---|---|---|---|
| 100 | 0.9606 | 0.9515 | 0.5081 | 0.0199 | 60 | 170 |
| 200 | 0.9528 | 0.9515 | 0.4980 | 0.0123 | 91 | 175 |
| 300 | 0.9627 | 0.9517 | 0.5053 | 0.0127 | 97 | 182 |
| 400 | 0.9399 | 0.9521 | 0.5162 | 0.0101 | 120 | 201 |
| 500 | 0.9426 | 0.9518 | 0.5089 | 0.0096 | 144 | 218 |
| 600 | 0.9691 | 0.9518 | 0.5041 | 0.0087 | 150 | 237 |
| 700 | 0.9564 | 0.9520 | 0.5074 | 0.0083 | 181 | 257 |
| 800 | 0.9534 | 0.9527 | 0.5216 | 0.0061 | 219 | 276 |
| 900 | 0.9574 | 0.9524 | 0.5183 | 0.0059 | 247 | 286 |
| 1000 | 0.9446 | 0.9528 | 0.5281 | 0.0062 | 254 | 296 |

**4.3.3.2.2 Second Layer** For the second layer, I selected the experiment where epsilon is 0.05. It is the lowest epsilon value before the execution time jumps to $\sim 19h$. In the output of that experiment, 358 clusters are formed and the most populated one has almost half a million data points. In order to provide better a separation, I analyzed it with lower epsilon values, Table 4.14. At this time, I was able to apply epsilon 0.025 and it took one third of the total execution time in first layer analysis respectively. I have observed that the second layer is more‘effective with 76.3% decrease in number of points aggregated in one cluster.

There is an inverse proportion between macro and micro precision according to change in epsilon value. Since the decrease in epsilon causes more but pure clusters, the precision also increases for the macro, i.e 63%. The micro purity is almost never affected by the different epsilon or stream speed values. However, it achieves pretty good macro purity when I consider the combination of the clusters with the same label. As discussed in the other stream speed experiments, its effect on the number of clusters and the execution time is limited as shown in Table 4.15. Hence the layers bring more similar data points together, it is easy to notice that the response to the change is little. In summary, even though this recursive technique causes additional processing, it improves the distribution of the data points for the clusters of the overall data set.

Table 4.14: Results of Flockstream on Second Inner Cluster of Commercial-Small data set: Change in Epsilon.

| -e | Macro Purity | Micro Purity | Macro F-Score | Micro F-Score | NoC | ET (min) |
|---|---|---|---|---|---|---|
| 0.025 | 0.9846 | 0.9575 | 0.6304 | 0.0034 | 621 | 397 |
| 0.05 | 0.9779 | 0.9530 | 0.5361 | 0.0127 | 129 | 104 |
| 0.075 | 0.9854 | 0.9520 | 0.4877 | 0.0209 | 53 | 96 |
| 0.1 | 0.9838 | 0.9520 | 0.4877 | 0.0331 | 33 | 95 |

### 4.3.4 KDD 2009 Churn Data Set

In the evaluation of the public data set, the parameters are tuned to the values shown in the Table 4.1 except the epsilon parameter. Since every data set shows different characteristics from each other, the parameters should be tuned accordingly.

In the evaluation of the public data, the default value for epsilon is chosen as 0.8. Table 4.16 contains ten experiments for various epsilon values from 0.4 to 4.0. The

Table 4.15: Results of Flockstream on Second Inner Cluster of Commercial-Small data set: Change in Stream Speed.

| -s | Macro Purity | Micro Purity | Macro F-Score | Micro F-Score | NoC | ET (min) |
|---|---|---|---|---|---|---|
| 100 | 0.9756 | 0.9527 | 0.5118 | 0.0219 | 68 | 98 |
| 200 | 0.9786 | 0.9526 | 0.5028 | 0.0127 | 102 | 95 |
| 300 | 0.9779 | 0.9530 | 0.5361 | 0.0127 | 129 | 104 |
| 400 | 0.9720 | 0.9528 | 0.5088 | 0.0115 | 134 | 115 |
| 500 | 0.9687 | 0.9529 | 0.5365 | 0.0097 | 164 | 128 |
| 600 | 0.9710 | 0.9542 | 0.5718 | 0.0092 | 195 | 151 |
| 700 | 0.9692 | 0.9543 | 0.5810 | 0.0080 | 241 | 166 |
| 800 | 0.9721 | 0.9541 | 0.5518 | 0.0066 | 274 | 181 |
| 900 | 0.9675 | 0.9542 | 0.5612 | 0.0056 | 332 | 197 |
| 1000 | 0.9674 | 0.9532 | 0.5436 | 0.0050 | 364 | 201 |

time needed to execute these experiments is not as much as Commercial-Small data set. Although KDD 2009 Churn data set includes much larger number of attributes (more than 10 times), it only adds more load to the distance calculations. However, the main problem is to handle continuously streaming data, which means continuously adding agents to the system in conjunction with both virtual and feature space.

Since this data set consist of only $50,000$ instances, the longest experiment does not exceed an hour, see Table 4.16. The evaluation metrics micro purity and macro F-Score remained relatively stable for every epsilon value. In contrast to the commercial data analysis, it is not clear that the macro purity values follow a particular pattern. On the micro F-score, I observe a gradual increase until $epsilon = 2.0$. Nevertheless, with a rate of 93% purity, Flockstream performed well for $epsilon = 0.8$. Therefore, I used this epsilon in the stream speed experiments.

Similar to the other experiments, increasing the stream speed has an impact on number of clusters and execution time with a rising trend. The best purity achieved in these experiments is 93% for stream speed 300, which is in line with the best result from the epsilon experiments. From the macro perspective, precision values are around 48%. Micro precision values gradually decrease while the stream speed increases.

## 4.4 Discussion on the Performances of Algorithms Employed

In this section, I present experimental results of the selected stream clustering algorithms on both public and commercial data sets. All related experiments are done

Table 4.16: Results of Flockstream on KDD 2009 Churn data set: Change in Epsilon.

| -e | Macro Purity | Micro Purity | Macro F-Score | Micro F-Score | NoC | ET (min) |
|---|---|---|---|---|---|---|
| 0.4 | 0.9112 | 0.9306 | 0.4868 | 0.0134 | 133 | 59 |
| 0.5 | 0.9036 | 0.9284 | 0.4825 | 0.0178 | 103 | 26 |
| 0.6 | 0.9125 | 0.9273 | 0.4811 | 0.0231 | 71 | 16 |
| 0.7 | 0.8956 | 0.9267 | 0.4823 | 0.0257 | 61 | 12 |
| 0.8 | 0.9302 | 0.9265 | 0.4809 | 0.0328 | 40 | 11 |
| 0.9 | 0.9256 | 0.9264 | 0.4809 | 0.0378 | 31 | 11 |
| 1.0 | 0.9226 | 0.9264 | 0.4809 | 0.0484 | 27 | 11 |
| 2.0 | 0.9225 | 0.9265 | 0.4809 | 0.1071 | 10 | 8 |
| 3.0 | 0.9332 | 0.9265 | 0.4809 | 0.0917 | 13 | 8 |
| 4.0 | 0.9184 | 0.9266 | 0.4809 | 0.0953 | 12 | 8 |

Table 4.17: Results of Flockstream on KDD 2009 Churn data set: Change in Stream Speed

| -s | Macro Purity | Micro Purity | Macro F-Score | Micro F-Score | NoC | ET (min) |
|---|---|---|---|---|---|---|
| 100 | 0.8781 | 0.9265 | 0.4809 | 0.0661 | 16 | 8 |
| 200 | 0.9024 | 0.9265 | 0.4814 | 0.0467 | 31 | 10 |
| 300 | 0.9302 | 0.9265 | 0.4809 | 0.0328 | 40 | 11 |
| 400 | 0.9144 | 0.9267 | 0.4815 | 0.0273 | 51 | 14 |
| 500 | 0.9315 | 0.9266 | 0.4809 | 0.0280 | 52 | 15 |
| 600 | 0.9145 | 0.9267 | 0.4815 | 0.0248 | 67 | 17 |
| 700 | 0.9205 | 0.9265 | 0.4815 | 0.0249 | 71 | 19 |
| 800 | 0.9287 | 0.9267 | 0.4809 | 0.0236 | 75 | 21 |
| 900 | 0.9202 | 0.9266 | 0.4809 | 0.0219 | 80 | 20 |
| 1000 | 0.9133 | 0.9268 | 0.4821 | 0.0187 | 94 | 22 |

with the MOA Release 2014.04. For the sake of simplicity, parameters of the MOA algorithms are kept as default except DenStream. Instead of default values of the parameters used in DenStream, I specified them according to the original Flockstream algorithm and my experiments discussed in the previous section. The parameters are set to values as discussed at the beginning of this chapter except the epsilon value.

Table 4.18 and Table 4.19 show the performances of algorithms employed on Electricity and Covertype data sets, respectively. I have chosen the experiments where epsilon parameter is 0.3 and 0.6 on parameter sensitivity analysis of Electricity and Covertype data sets, respevtively. On the other hand, Table 4.20, 4.21 and 4.22 summarize the performances of algorithms on Commercial-Big, Commercial-Small, and KDD 2009 churn data sets and epsilon values are set as 0.1, 0.1, and 0.8, respectively. In order to prevent confusion, the names of evaluation metrics are given as the same names mentioned in MOA, such that F1-P, F1-R, and Purity.

As seen in Table 4.18, the algorithms perform almost the same on Electricity

Table 4.18: Performances of Algorithms on Electricity Data Set

| Algorithms | F1-P | F1-R | Purity |
|---|---|---|---|
| Flockstream | 0.0102 | 0.0880 | 0.7051 |
| DenStream | 0.1460 | 0.1564 | 0.7601 |
| CluStream | 0.0831 | 0.0972 | 0.7431 |
| ClusTree | 0.0832 | 0.1009 | 0.7456 |

data set. The best precision is achieved by DenStream with 14%. CluStream and ClusTree follow with 8% and Flockstream clusters with 1% precision. On the other hand, the found recall measures increase starting from 8% one by one for Flockstream, CluStream, and ClusTree respectively. The purity values of CluStream and ClusTree are pretty much same with 74%. Whereas the purity of Flockstream clusters is 70% and the purity of DenStream clusters is 76%.

Table 4.19: Performances of Algorithms on Covertype Data Set

| Algorithms | F1-P | F1-R | Purity |
|---|---|---|---|
| Flockstream | 0.0233 | 0.2939 | 0.8201 |
| DenStream | 0.0264 | 0.0381 | 0.8102 |
| CluStream | 0.0310 | 0.0393 | 0.7938 |
| ClusTree | 0.0306 | 0.0387 | 0.7934 |

On the Covertype data set, Table 4.19, Flockstream achieves a purity measure of 82%. While both CluStream and ClusTree performs 79%, DenStream exceeds them by 2%. Recall values are around 3% for the all algorithms with only slight differences. In the precision side, Flockstream and DenStream performs 2%, following behind the CluStream and ClusTree with 3% precision.

The first churn data set chosen for performance comparison is Commercial-Small, where Table 4.20 presents the performances. Flockstream generally provides lower precision and higher recall than the MOA algorithms. From the purity point of view, it exhibited an outstanding performance with 25% better purity than DenStream. The CluStream and ClusTree results are almost the same. The biggest difference between them is revealed in purity level where CluStream results are 1% better than the ClusTree.

Table 4.21 presents the results on the bigger commercial data set. Similar to the results on the small one, Flockstream achieves 98% purity. DenStream also slightly

Table 4.20: Performances of Algorithms on Commercial-Small Data Set

| Algorithms | F1-P | F1-R | Purity |
|---|---|---|---|
| Flockstream | 0.0078 | 0.4014 | 0.96 |
| DenStream | 0.3197 | 0.2467 | 0.83 |
| CluStream | 0.0384 | 0.0281 | 0.54 |
| ClusTree | 0.0369 | 0.0270 | 0.53 |

increases its purity but the big increase comes from the Clustree and CluStream with 20% and 18%, respectively. Best precision value is achieved by DenStream, around 14%. Again, Flockstream returns the highest recall value of all the tested algorithms.

Table 4.21: Performances of Algorithms on Commercial-Big Churn Data Set

| Algorithms | F1-P | F1-R | Purity |
|---|---|---|---|
| Flockstream | 0.0093 | 0.4431 | 0.98 |
| DenStream | 0.1359 | 0.1060 | 0.85 |
| CluStream | 0.0824 | 0.0629 | 0.72 |
| ClusTree | 0.0859 | 0.0654 | 0.73 |

The results of public data set shows that the performance of Flockstream is even more pronounced over MOA algorithms. As in the case with the commercial data sets experiments, the purity of the Flockstream on public data set is the highest. The other three algorithms produced much lower purity rates with an average of 20%. Although the values in the F1-P and F1-R columns are low in general, the results of the Flockstream are still better on the basis of precision and recall. From the recall point of view, MAO algorithms results are almost the same. On the other hand, recall for the Flockstream is much higher, around 48%.

Table 4.22: Performances of Algorithms on KDD 2009 Churn Data Set

| Algorithms | F1-P | F1-R | Purity |
|---|---|---|---|
| Flockstream | 3.28E-2 | 4.75E-1 | 0.93 |
| DenStream | 4.67E-4 | 2.33E-4 | 0.21 |
| CluStream | 2.15E-4 | 1.07E-4 | 0.20 |
| ClusTree | 6.82E-4 | 3.41E-4 | 0.19 |

# Chapter 5

# Conclusions and the Future Work

The objective of this thesis was to investigate the performance of stream clustering techniques on dynamic networks. To this end, the performance of the Flockstream, bio-inspired stream clustering algorithm, and three other state-of-the-art stream clustering algorithms; namely CluStream, DenStream, and ClusTree are designed, built and evaluated on the churn detection task. I have performed my experiments on five data sets and discussed the results using the evaluation metrics; precision, recall and purity values of the found clusters. In my experiments, I observed that the epsilon value plays a vital role in the performance of Flockstream. Larger epsilon values cause irrelevant data points to aggregate in the same cluster. In contrast, only substantially similar data points became a swarm and large number of data points remain in the system for the lower epsilon values. Thus, the data points that can not become or join a flock continue to occupy a place in the virtual space. Due to the limited number of cells in the virtual space, it becomes full after a certain period of time and the system can not accept incoming data. Therefore, the choice of epsilon value is very important for Flockstream functionality.

Based on the performances of the algorithms employed in this research on both the commercial and the public data sets, it is clear that Flockstream presents remarkable results, especially on purity. While DenStream is the closest follower on the commercial data sets, there is still a considerable difference between them. In the experiments on the public data sets, superiority of Flockstream on the purity is incontestable. When I examine the results from the precision point of view, both DenStream and Flockstream produce the best results for the commercial and the public data sets. Due to the reason that the precision values are evaluated from the micro perspective, resulting values are rather low for all the analyzed algorithms. Finally, when I consider recall values for the algorithms, Flockstream has the highest recall

values on all data sets. The average of the maximum F-Scores of each class determines the final value of the recall metric. Because Flockstream produces relatively a large number of clusters, the probability of achieving better F-Scores is higher than the other algorithms. For that reason, I would expect high recall values compared to the other algorithms.

In conclusion, the effect of stream clustering techniques on churn detection research is notable. This approach can be applied to various service industries in addition to the gaming and the telecommunication sectors. Future work will investigate how to speed up the Flockstream algorithm as well as how to improve the precision measurements.

## Bibliography


[1] Ackermann, M.R., Märtens, M., Raupach, C., Swierkot, K., Lammersen, C., Sohler, C.: Streamkm++: A clustering algorithm for data streams. Journal of Experimental Algorithmics (JEA) 17, 2–4 (2012)

[2] Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: Proceedings of the 29th international conference on Very large data bases-Volume 29. pp. 81–92. VLDB Endowment (2003)

[3] Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for projected clustering of high dimensional data streams. In: Proceedings of the Thirtieth international conference on Very large data bases-Volume 30. pp. 852–863. VLDB Endowment (2004)

[4] Ali, Ö.G., Arıtürk, U.: Dynamic churn prediction framework with more effective use of rare event data: The case of private banking. Expert Systems with Applications 41(17), 7889–7903 (2014)

[5] Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: Moa: Massive online analysis, a framework for stream classification and clustering. Journal of Machine Learning Research 10, 1601–1604 (2010)

[6] Blackard, J.A.: Comparison of neural networks and discriminant analysis in predicting forest cover types. Colorado State University (1998)

[7] Burez, J., Van den Poel, D.: Separating financial from commercial customer churn: A modeling step towards resolving the conflict between the sales and credit department. Expert Systems with Applications 35(1), 497–514 (2008)

[8] Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: SDM. vol. 6, pp. 328–339. SIAM (2006)

[9] Chu, B.H., Tsai, M.S., Ho, C.S.: Toward a hybrid data mining model for customer retention. Knowledge-Based Systems 20(8), 703–718 (2007)

[10] Cui, X., Potok, T.E.: A distributed agent implementation of multiple species flocking model for document partitioning clustering. In: Cooperative Information Agents X, pp. 124–137. Springer (2006)

[11] Eberhart, R.C., Shi, Y., Kennedy, J.: Swarm intelligence. Elsevier (2001)

[12] Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the KDD Conference (AAAI). pp. 226–231 (1996)

[13] Forestiero, A., Pizzuti, C., Spezzano, G.: Flockstream: a bio-inspired algorithm for clustering evolving data streams. In: Tools with Artificial Intelligence, 2009. ICTAI'09. 21st International Conference on. pp. 1–8. IEEE (2009)

[14] Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: ACM SIGMOD. pp. 47–57 (1984)

[15] Guyon, I., Lemaire, V., Boullé, M., Dror, G., Vogel, D.: Analysis of the kdd cup 2009: Fast scoring on a large orange customer database. JMRL Workshop and Conference Proceedings 7, 1–22 (2009)

[16] Harries, M., Wales, N.S.: Splice-2 comparative evaluation: Electricity pricing (1999)

[17] Huang, B.Q., Kechadi, T.M., Buckley, B., Kiernan, G., Keogh, E., Rashid, T.: A new feature set with new window techniques for customer churn prediction in land-line telecommunications. Expert Systems with Applications 37(5), 3657–3665 (2010)

[18] Karahoca, A., Karahoca, D.: Gsm churn management by using fuzzy c-means clustering and adaptive neuro fuzzy inference system. Expert Systems with Applications 38(3), 1814–1822 (2011)

[19] Kranen, P., Assent, I., Baldauf, C., Seidl, T.: The clustree: indexing micro-clusters for anytime stream mining. Knowledge and information systems 29(2), 249–272 (2011)

[20] Larivière, B., Van den Poel, D.: Investigating the role of product features in preventing customer churn, by using survival analysis and choice modeling: The case of financial services. Expert Systems with Applications 27(2), 277–285 (2004)

[21] Lee, Y.H., Wei, C.P., Cheng, T.H., Yang, C.T.: Nearest-neighbor-based approach to time-series classification. Decision Support Systems 53(1), 207–217 (2012)

[22] Lemmens, A., Croux, C.: Bagging and boosting classification trees to predict churn. Journal of Marketing Research 43(2), 276–286 (2006)

[23] Moise, G., Sander, J., Ester, M.: P3c: A robust projected clustering algorithm. In: Data Mining, 2006. ICDM'06. Sixth International Conference on. pp. 414–425. IEEE (2006)

[24] Mozer, M.C., Wolniewicz, R., Grimes, D.B., Johnson, E., Kaushansky, H.: Predicting subscriber dissatisfaction and improving retention in the wireless telecommunications industry. Neural Networks, IEEE Transactions on 11(3), 690–696 (2000)

[25] Nasraoui, O., Uribe, C.C., Coronel, C.R., Gonzalez, F.: Tecno-streams: Tracking evolving clusters in noisy data streams with a scalable immune system learning model. In: Data Mining, 2003. ICDM 2003. Third IEEE International Conference on. pp. 235–242. IEEE (2003)

[26] Neslin, S.A., Gupta, S., Kamakura, W., Lu, J., Mason, C.H.: Defection detection: Measuring and understanding the predictive accuracy of customer churn models. Journal of marketing research 43(2), 204–211 (2006)

[27] Ng, R.T., Han, J.: Clarans: A method for clustering objects for spatial data mining. Knowledge and Data Engineering, IEEE Transactions on 14(5), 1003–1016 (2002)

[28] Owczarczuk, M.: Churn models for prepaid customers in the cellular telecommunication industry using large data marts. Expert Systems with Applications 37(6), 4710–4712 (2010)

[29] Pendharkar, P.C.: Genetic algorithm based neural network approaches for predicting churn in cellular wireless network services. Expert Systems with Applications 36(3), 6714–6720 (2009)

[30] Van den Poel, D., Lariviere, B.: Customer attrition analysis for financial services using proportional hazard models. European journal of operational research 157(1), 196–217 (2004)

[31] Reynolds, C.W.: Flocks, herds and schools: A distributed behavioral model. ACM Siggraph Computer Graphics 21(4), 25–34 (1987)

[32] Tsai, C.F., Chen, M.Y.: Variable selection by association rules for customer churn prediction of multimedia on demand. Expert Systems with Applications 37(3), 2006–2015 (2010)

[33] Tu, L., Chen, Y.: Stream data clustering based on grid density and attraction. ACM Transactions on Knowledge Discovery from Data (TKDD) 3(3), 12 (2009)

[34] Verbeke, W., Dejaeger, K., Martens, D., Hur, J., Baesens, B.: New insights into churn prediction in the telecommunication sector: A profit driven data mining approach. European Journal of Operational Research 218(1), 211–229 (2012)

[35] Xiao, J., Xie, L., He, C., Jiang, X.: Dynamic classifier ensemble model for customer classification with imbalanced class distribution. Expert Systems with Applications 39(3), 3668–3675 (2012)

[36] Xie, Y., Li, X., Ngai, E., Ying, W.: Customer churn prediction using improved balanced random forests. Expert Systems with Applications 36(3), 5445–5449 (2009)

[37] Zhang, T., Ramakrishnan, R., Livny, M.: Birch: an efficient data clustering method for very large databases. In: ACM SIGMOD. pp. 103–114 (1996)

[38] Zhao, J., Dang, X.H.: Bank customer churn prediction based on support vector machine: taking a commercial bank's vip customer churn as the example. In: Wireless Communications, Networking and Mobile Computing, 2008. WiCOM'08. 4th International Conference on. pp. 1–4. IEEE (2008)

[39] Zhou, A., Cao, F., Qian, W., Jin, C.: Tracking clusters in evolving data streams over sliding windows. Knowledge and Information Systems 15(2), 181–214 (2008)

[40] Zopounidis, C., Mavri, M., Ioannou, G.: Customer switching behaviour in greek banking services using survival analysis. Managerial Finance 34(3), 186–197 (2008)

# Appendix A

# MOA Result Figures

## A.1 Electricity

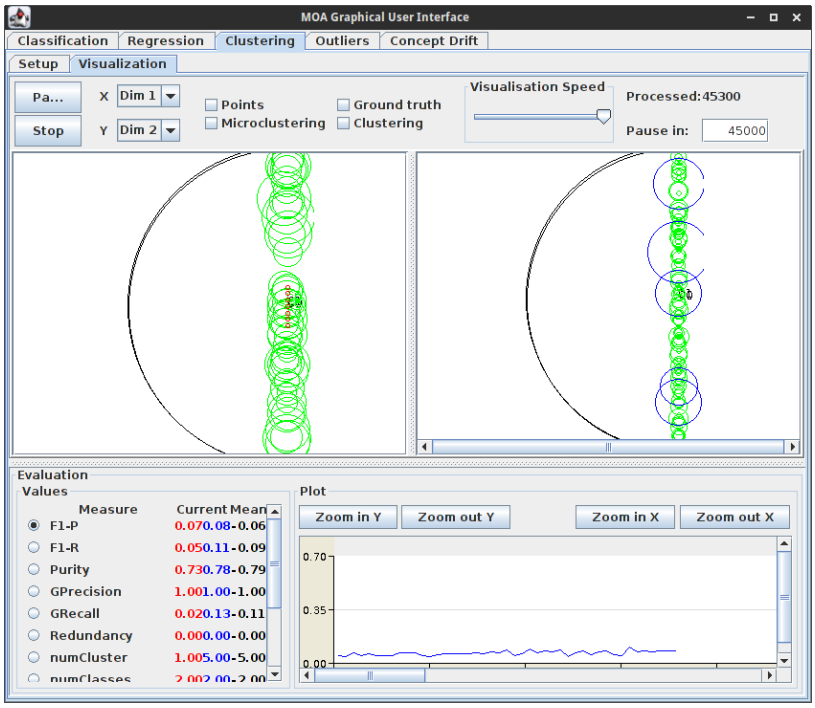


Figure A.1: MOA Results of DenStream on Electricity Data Set: F1-P

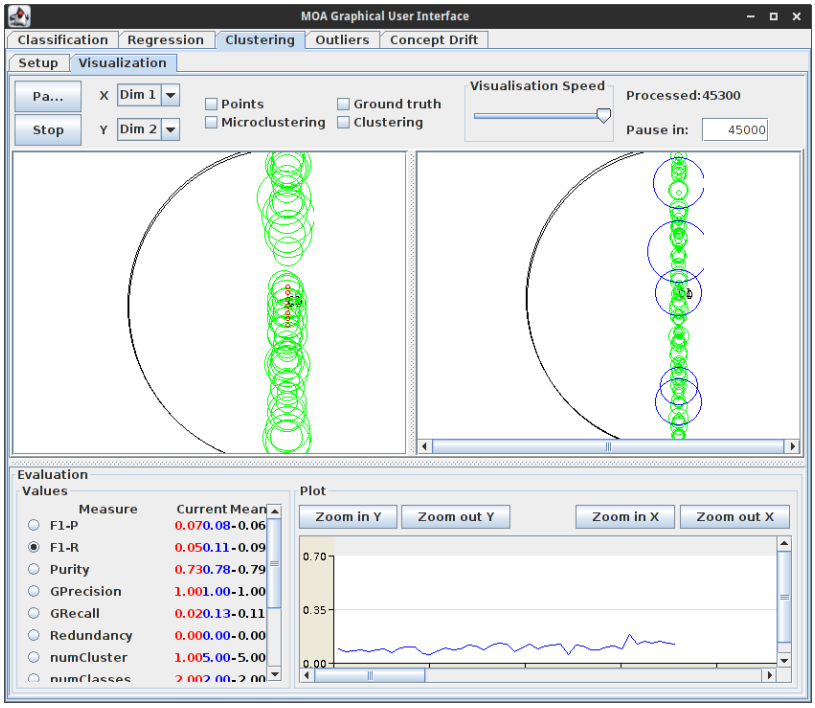


Figure A.2: MOA Results of DenStream on Electricity Data Set: F1-R

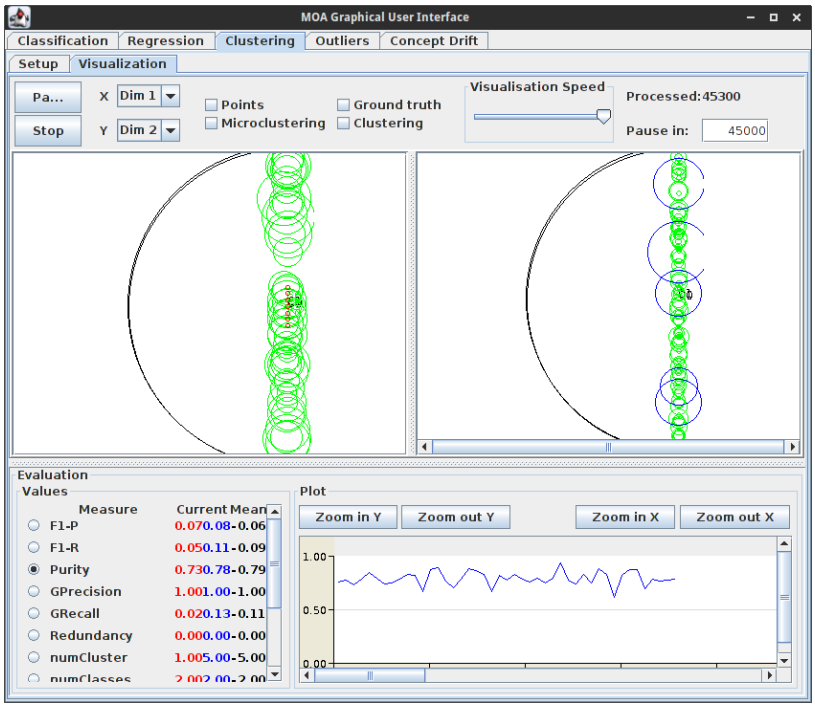


Figure A.3: MOA Results of DenStream on Electricity Data Set: Purity

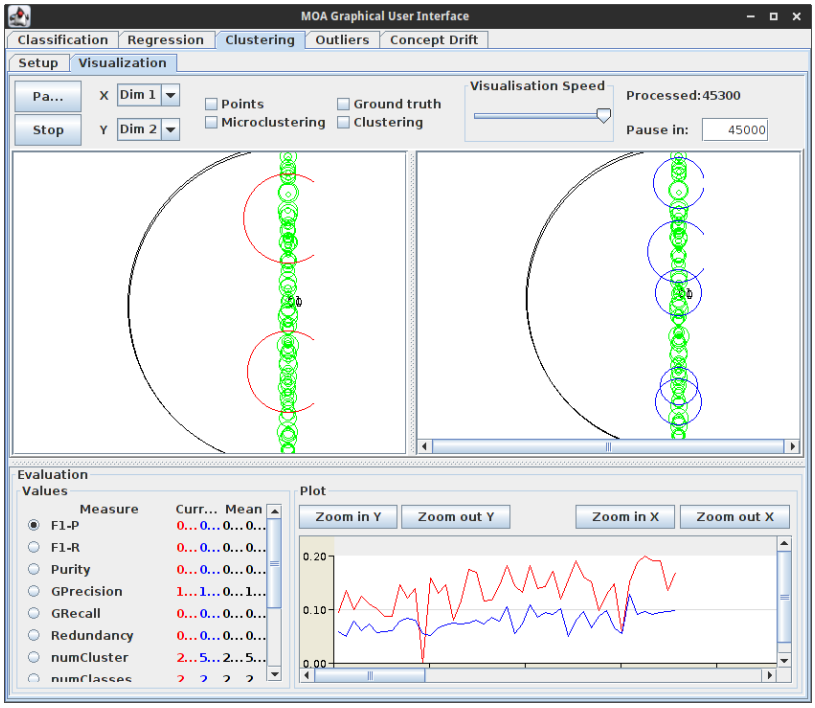


Figure A.4: MOA Results of CluStream on Electricity Data Set: F1-P

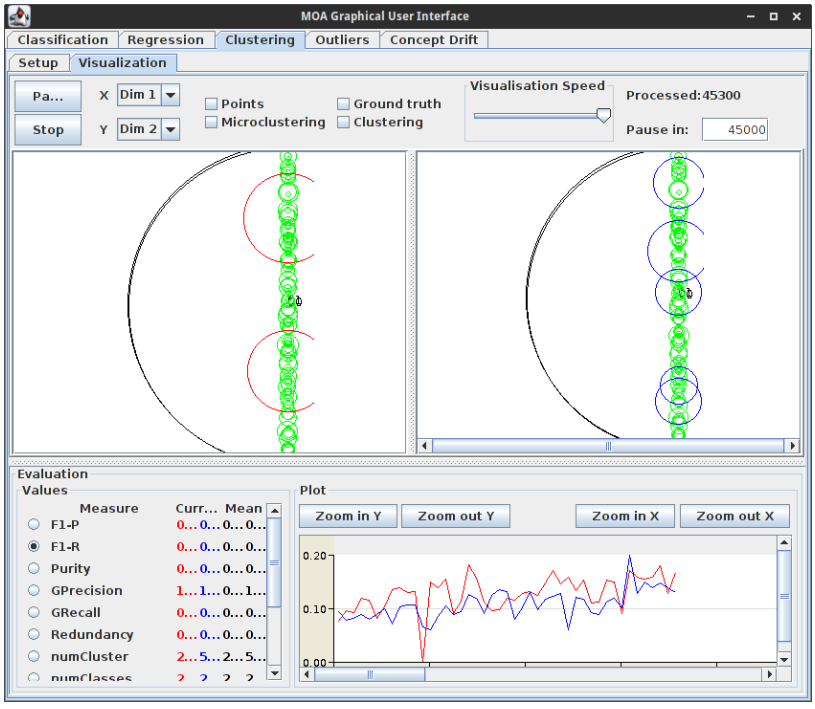


Figure A.5: MOA Results of CluStream on Electricity Data Set: F1-R

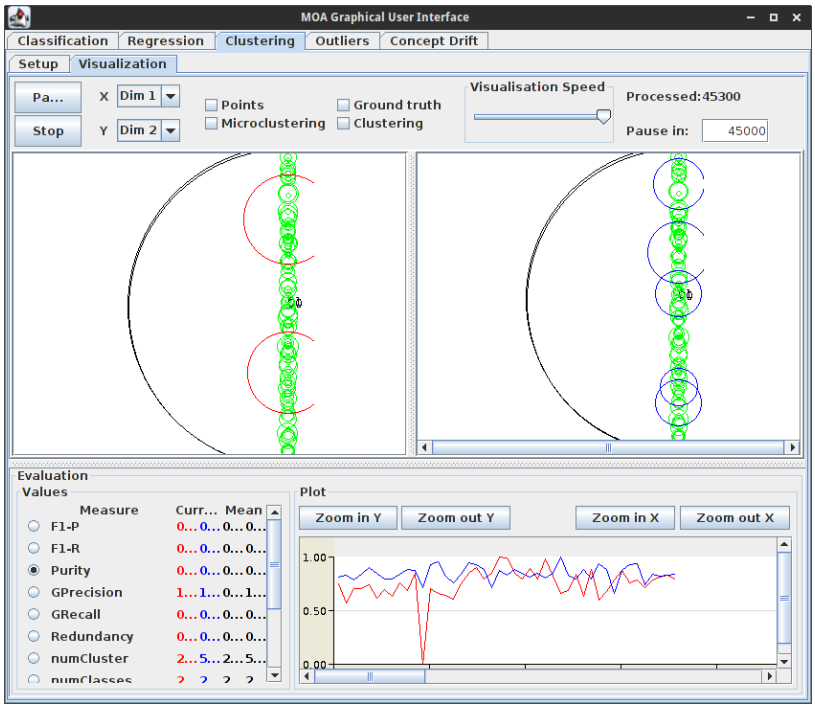


Figure A.6: MOA Results of CluStream on Electricity Data Set: Purity

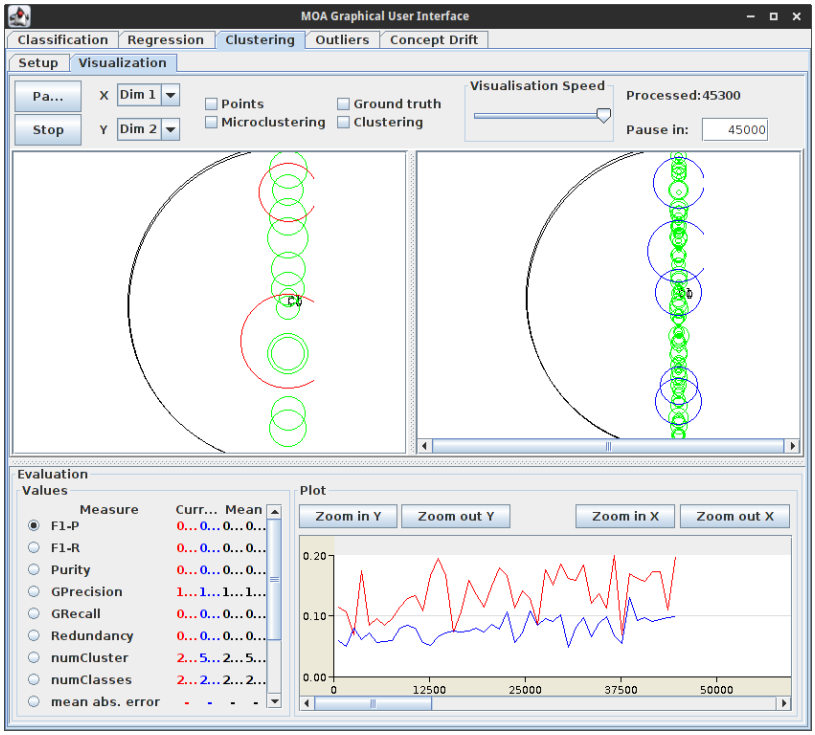


Figure A.7: MOA Results of ClusTree on Electricity Data Set: F1-P

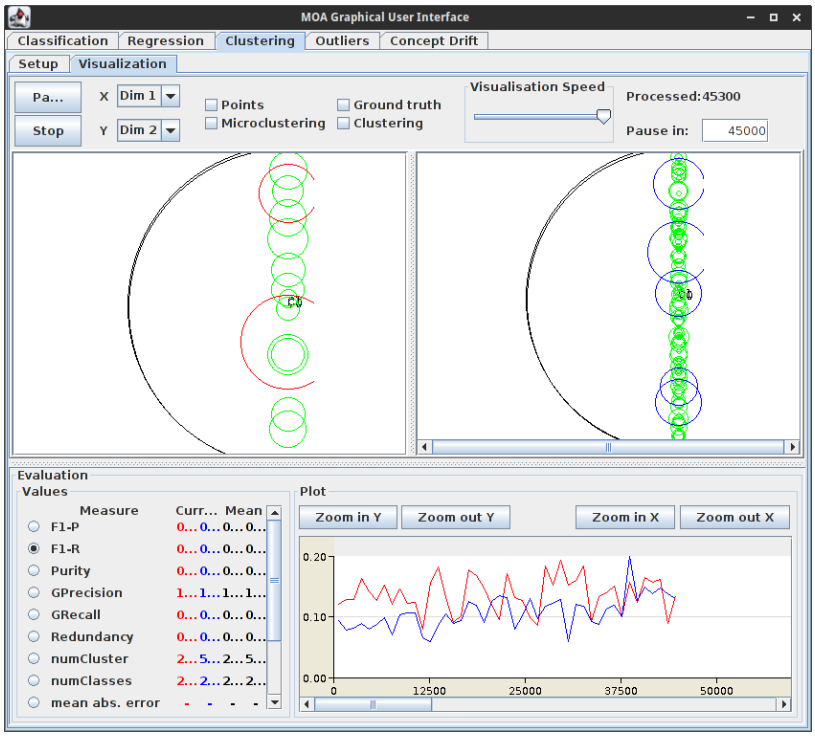


Figure A.8: MOA Results of ClusTree on Electricity Data Set: F1-R

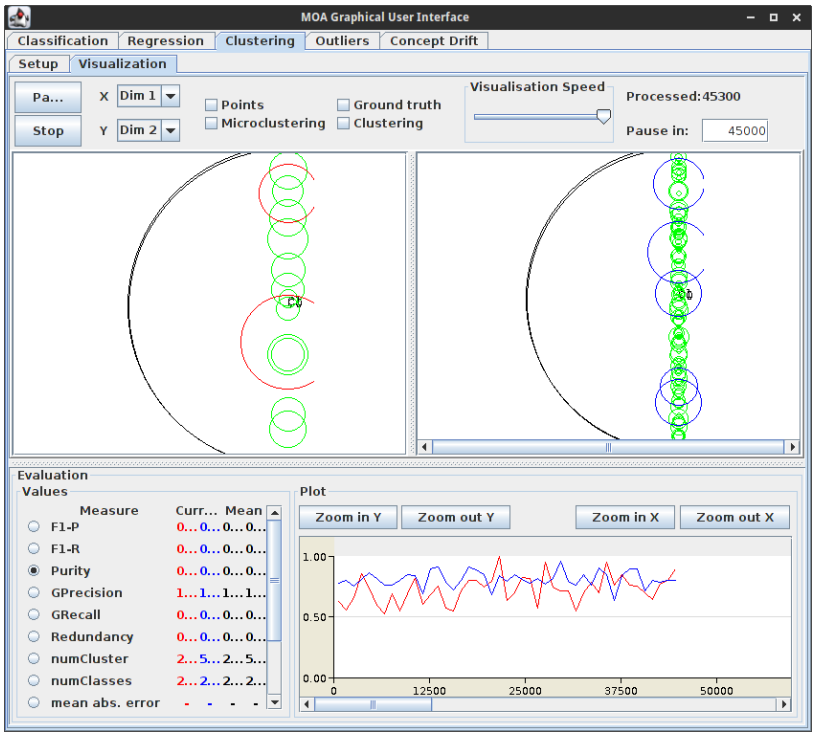


Figure A.9: MOA Results of ClusTree on Electricity Data Set: Purity

## A.2 Covertype

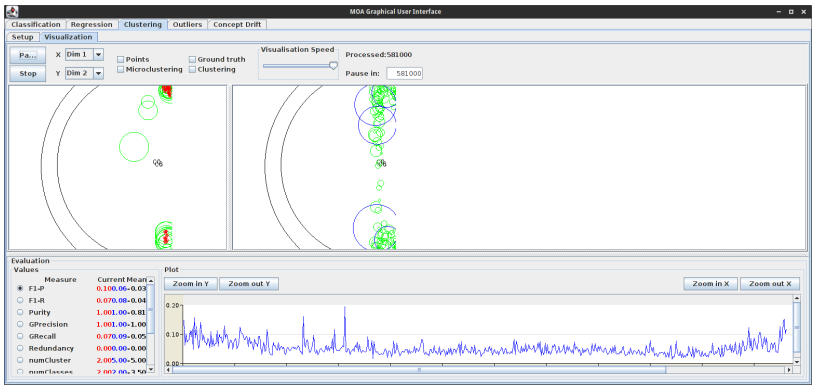


Figure A.10: MOA Results of DenStream on Covertype Data Set: F1-P

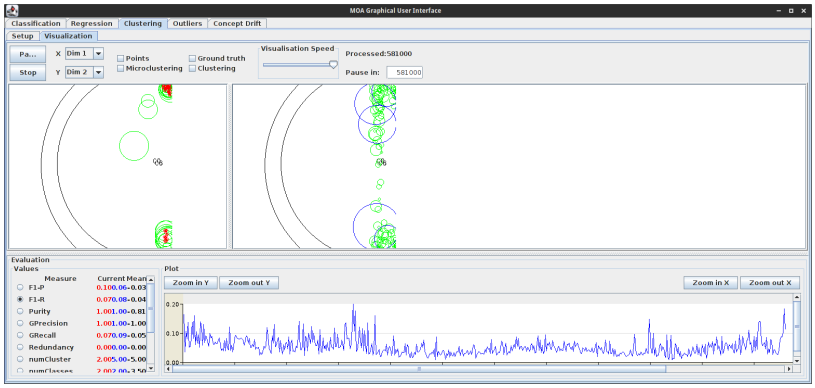


Figure A.11: MOA Results of DenStream on Covertype Data Set: F1-R

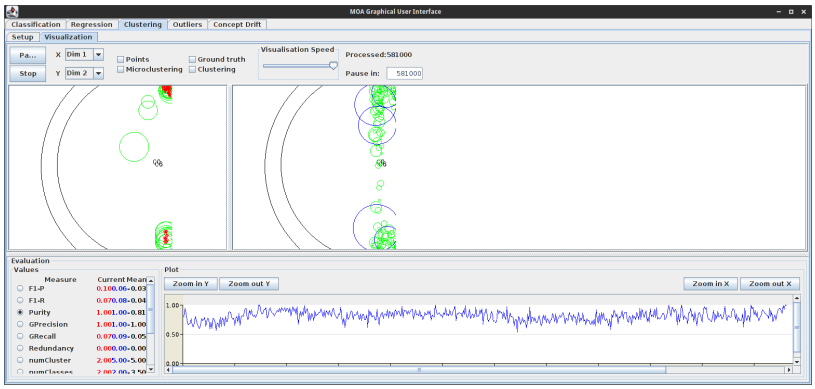


Figure A.12: MOA Results of DenStream on Covertype Data Set: Purity

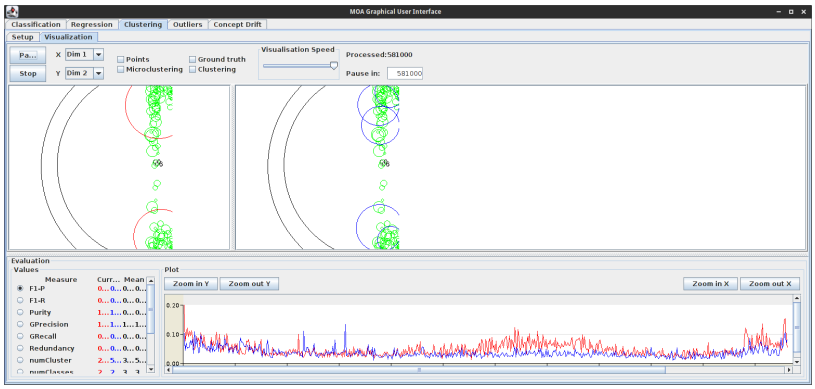


Figure A.13: MOA Results of CluStream on Covertype Data Set: F1-P

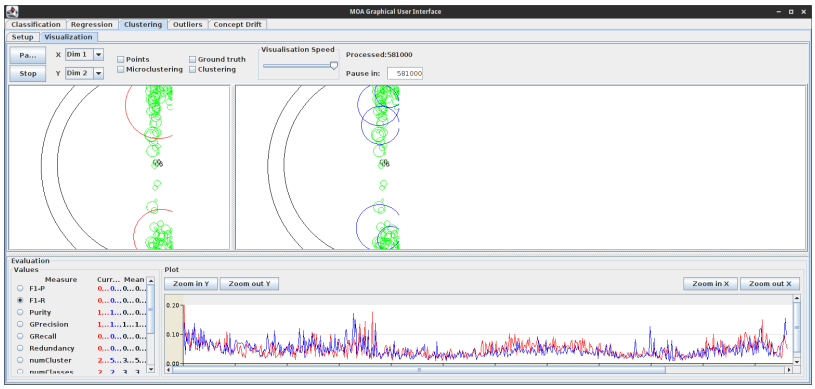


Figure A.14: MOA Results of CluStream on Covertype Data Set: F1-R

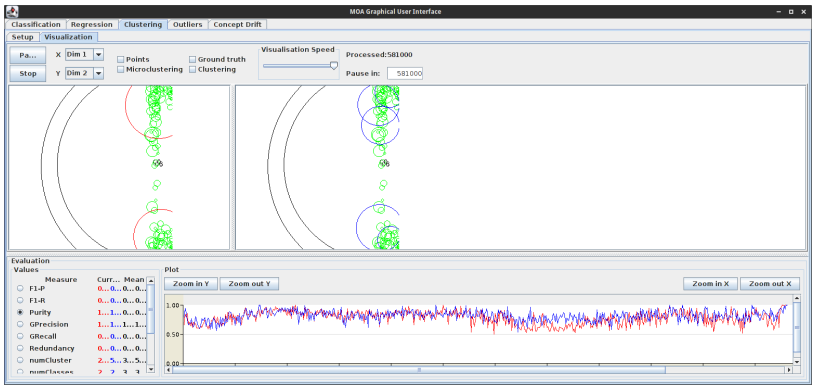


Figure A.15: MOA Results of CluStream on Covertype Data Set: Purity

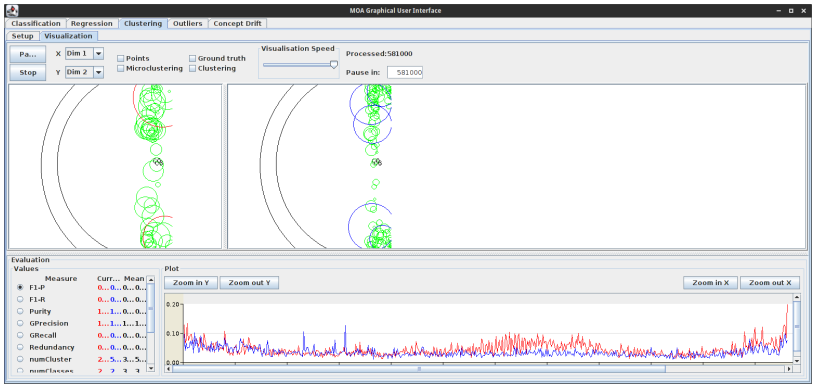


Figure A.16: MOA Results of ClusTree on Covertype Data Set: F1-P

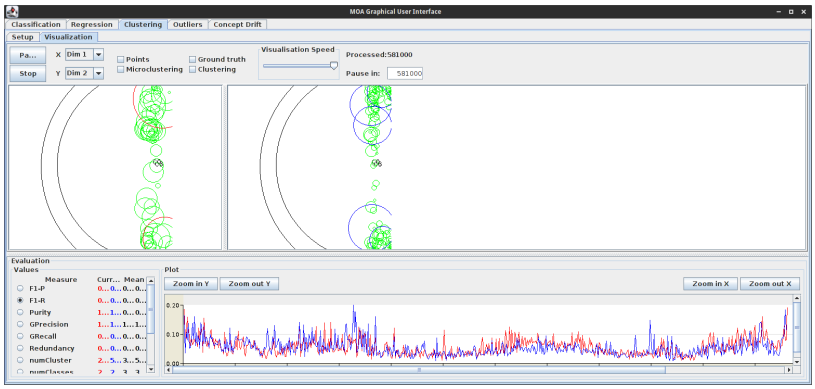


Figure A.17: MOA Results of ClusTree on Covertype Data Set: F1-R

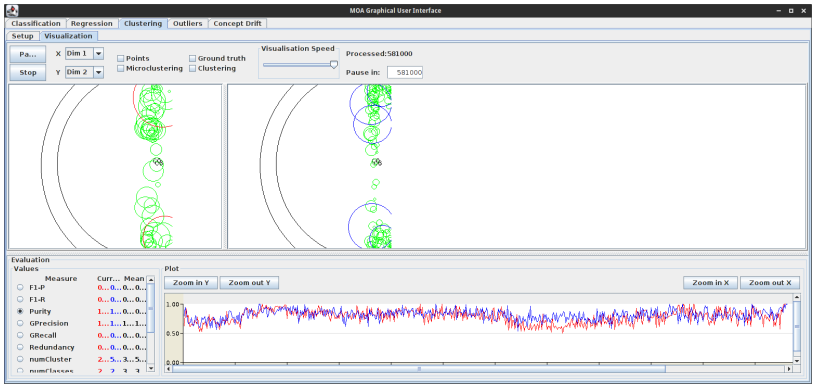


Figure A.18: MOA Results of ClusTree on Covertype Data Set: Purity

## A.3 KDD 2009 Churn

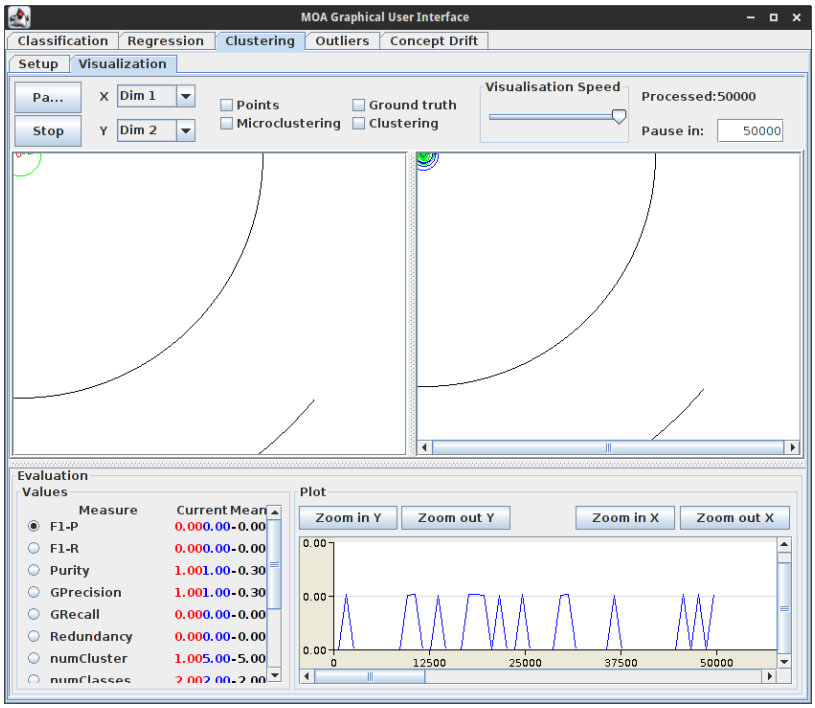


Figure A.19: MOA Results of DenStream on KDD2009 Data Set: F1-P

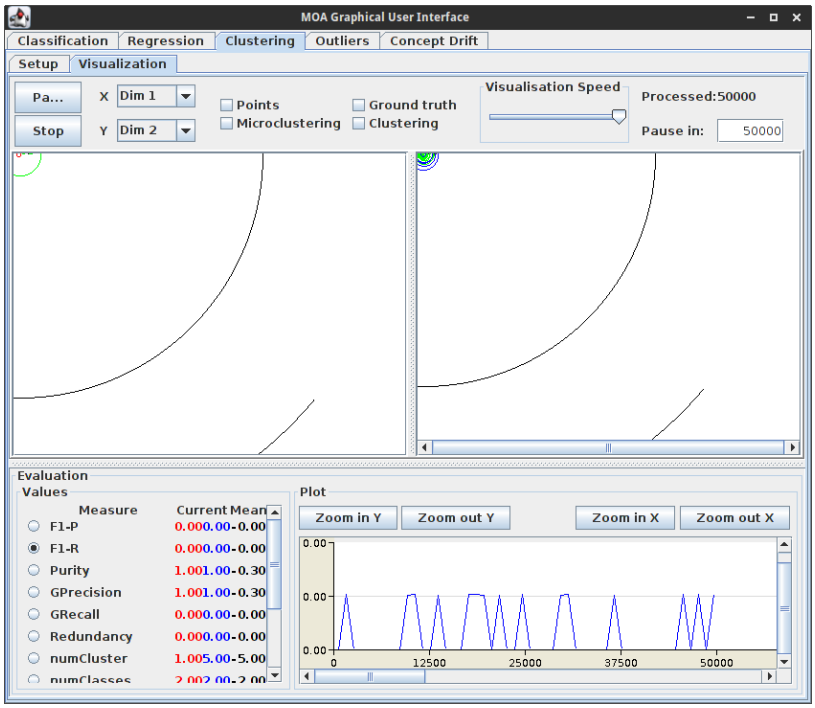


Figure A.20: MOA Results of DenStream on KDD2009 Data Set: F1-R

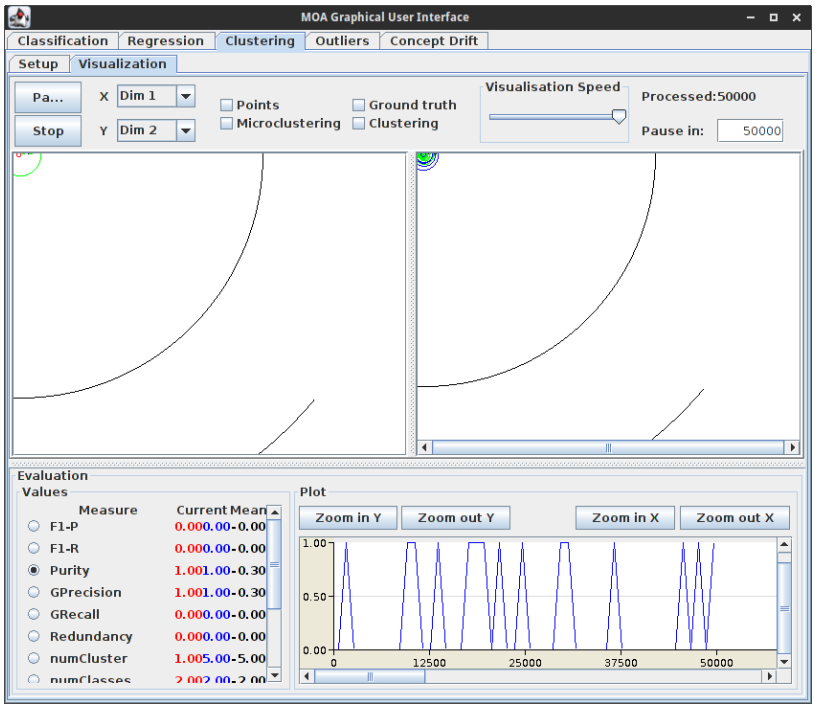


Figure A.21: MOA Results of DenStream on KDD2009 Data Set: Purity

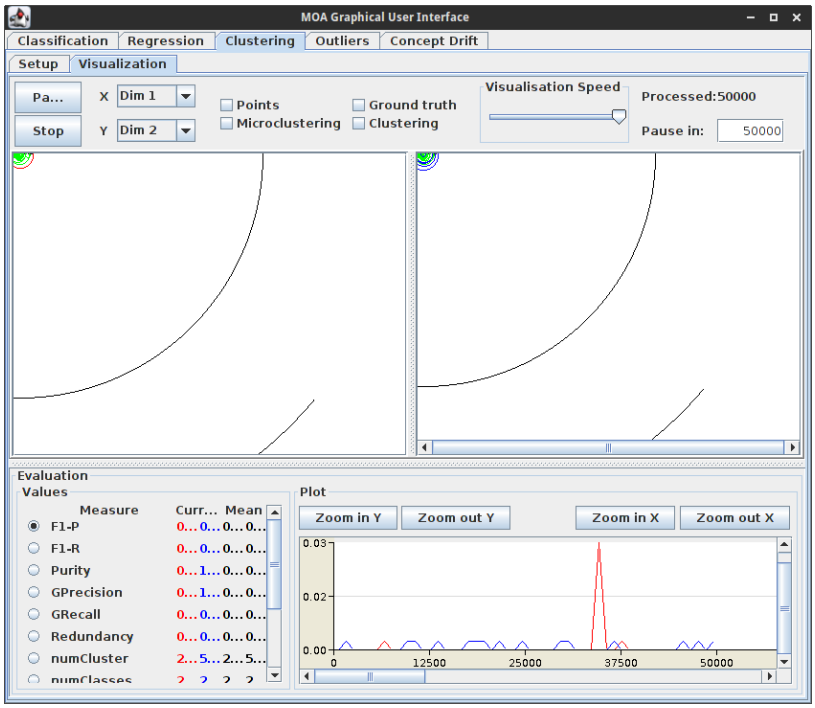


Figure A.22: MOA Results of CluStream on KDD2009 Data Set: F1-P

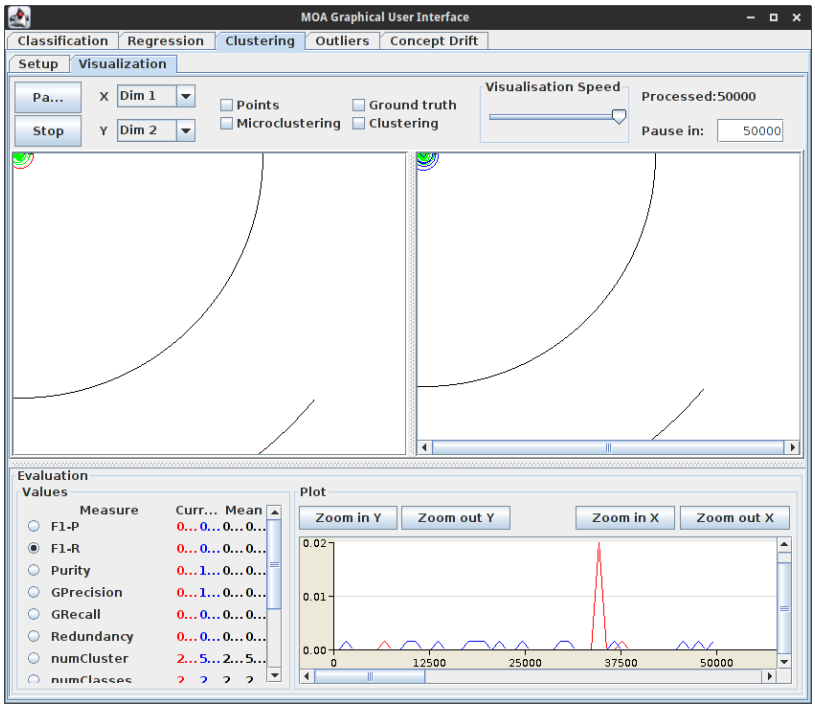


Figure A.23: MOA Results of CluStream on KDD2009 Data Set: F1-R

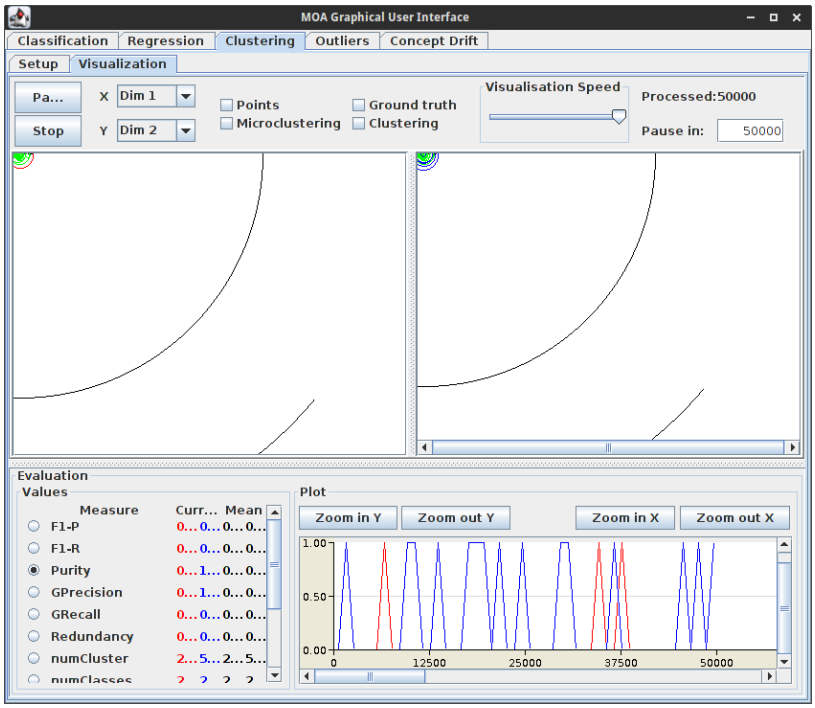


Figure A.24: MOA Results of CluStream on KDD2009 Data Set: Purity

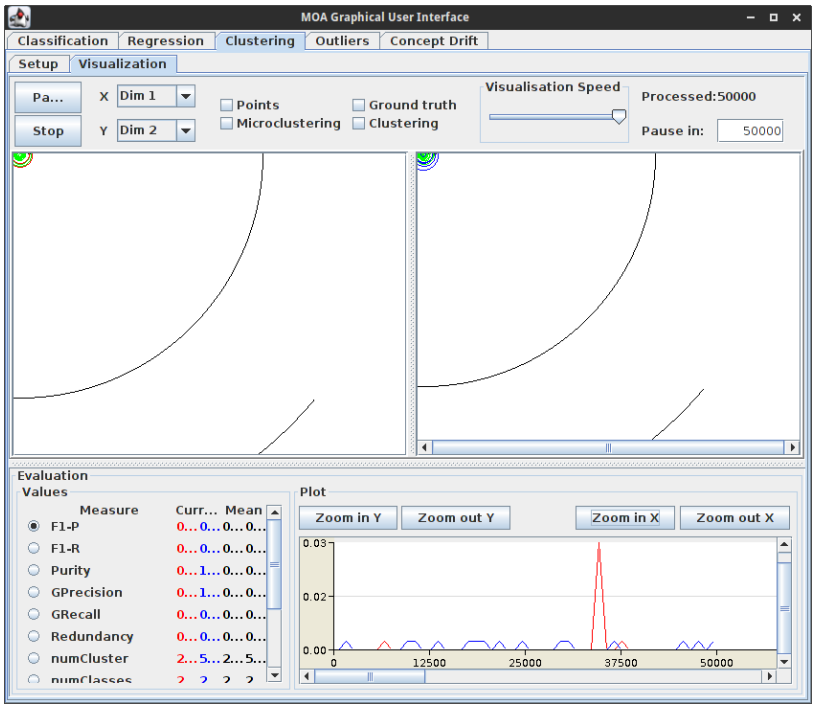


Figure A.25: MOA Results of ClusTree on KDD2009 Data Set: F1-P

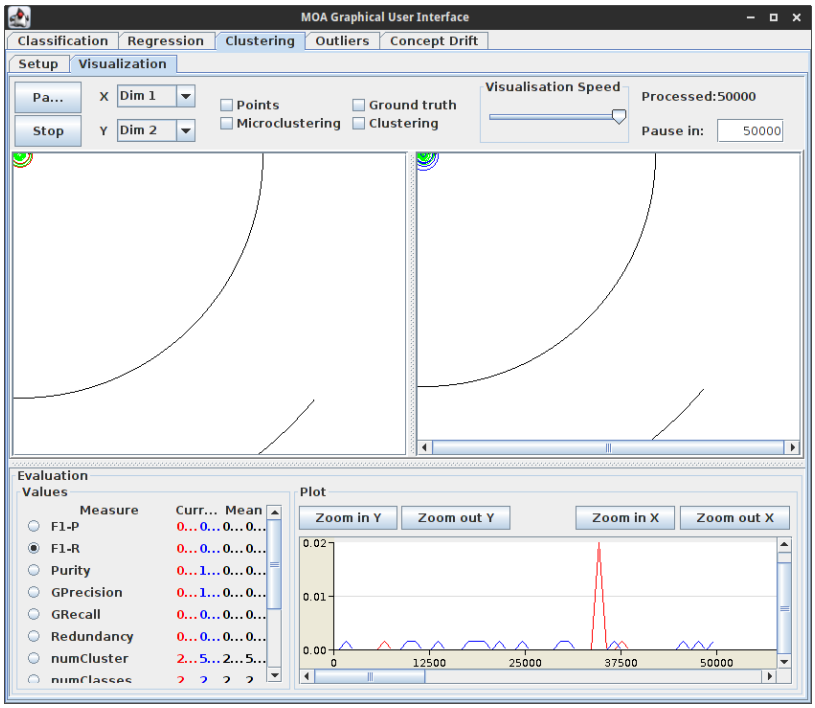


Figure A.26: MOA Results of ClusTree on KDD2009 Data Set: F1-R

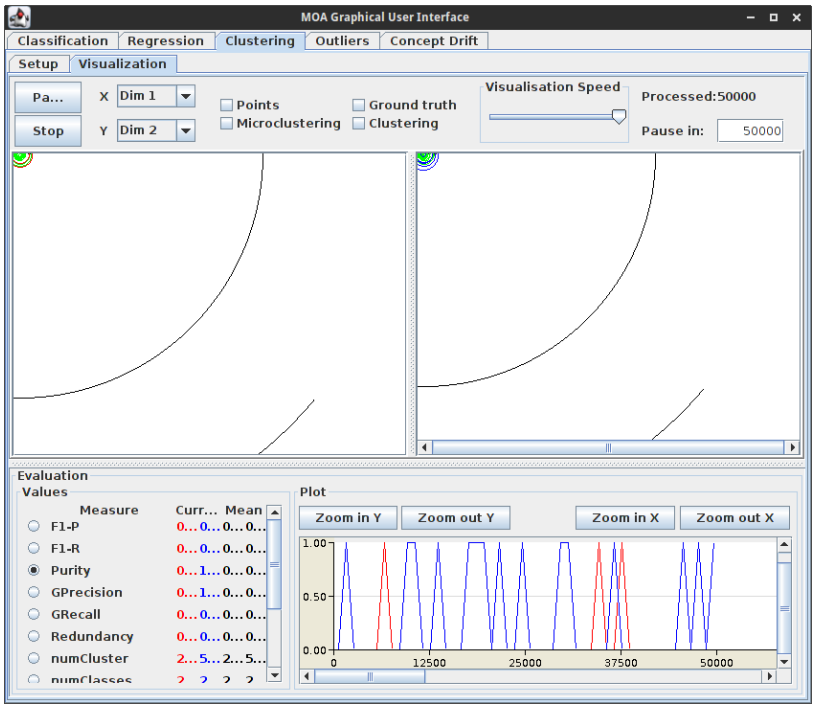


Figure A.27: MOA Results of ClusTree on KDD2009 Data Set: Purity

## A.4 Commercial-Small

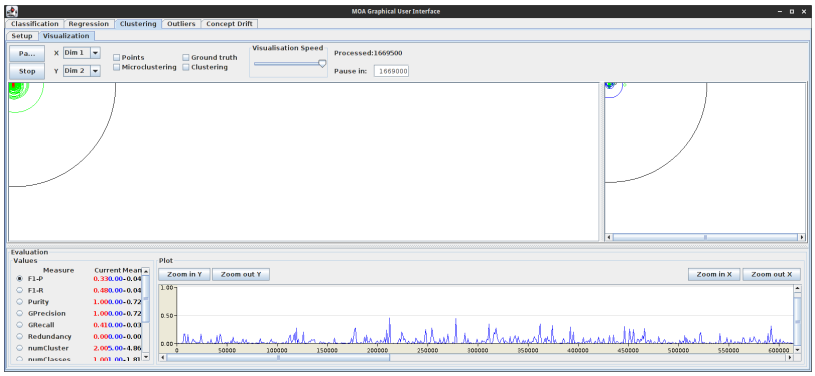


Figure A.28: MOA Results of DenStream on Commercial-Small Data Set: F1-P

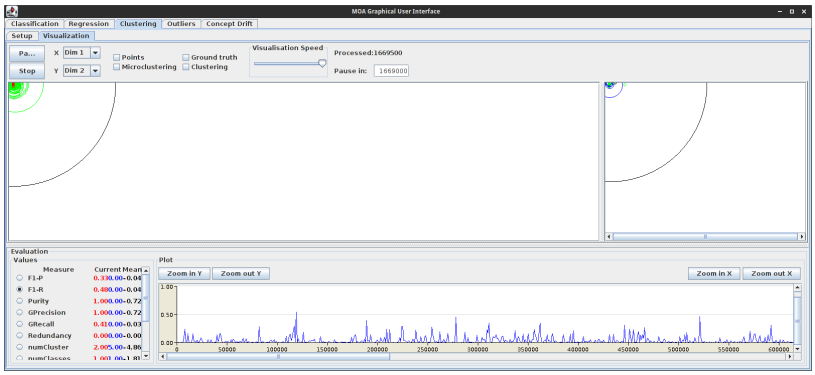


Figure A.29: MOA Results of DenStream on Commercial-Small Data Set: F1-R

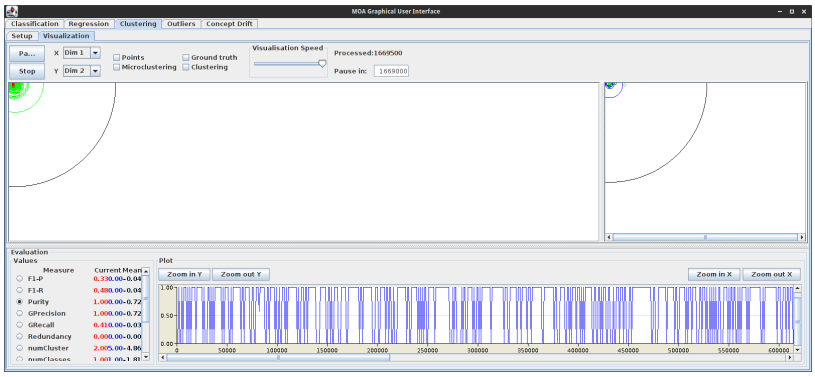


Figure A.30: MOA Results of DenStream on Commercial-Small Data Set: Purity

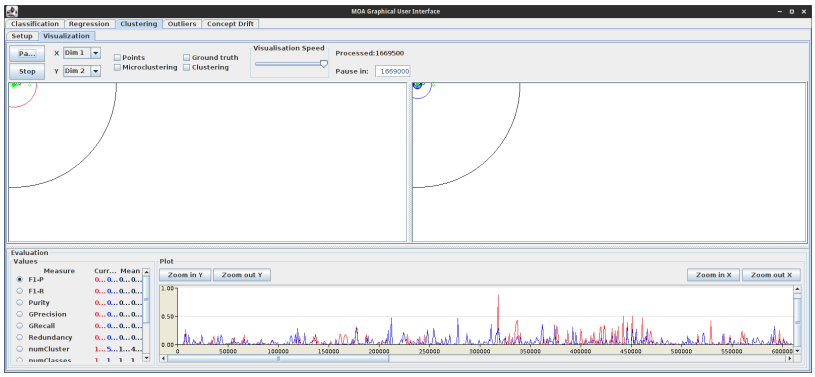


Figure A.31: MOA Results of CluStream on Commercial-Small Data Set: F1-P

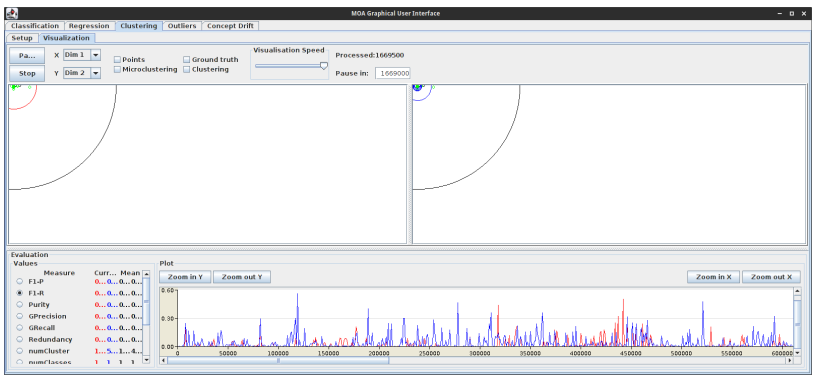


Figure A.32: MOA Results of CluStream on Commercial-Small Data Set: F1-R

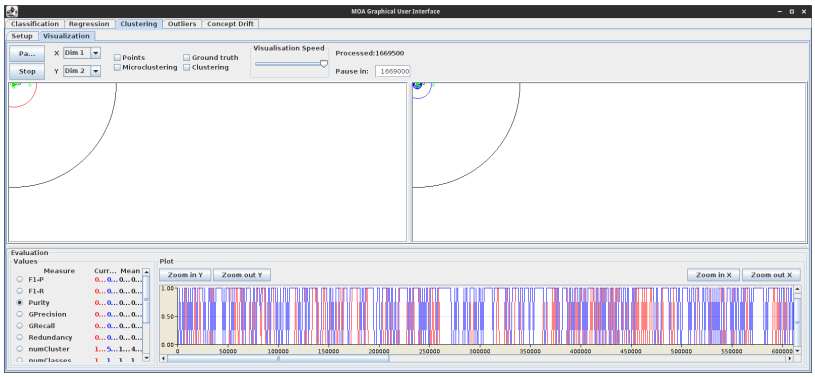


Figure A.33: MOA Results of CluStream on Commercial-Small Data Set: Purity

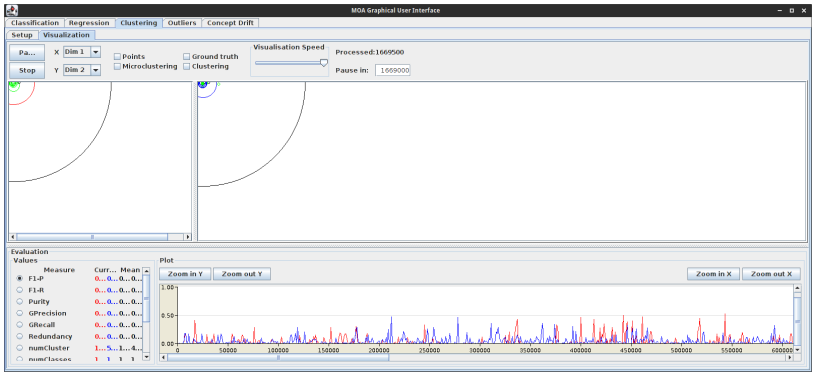


Figure A.34: MOA Results of ClusTree on Commercial-Small Data Set: F1-P

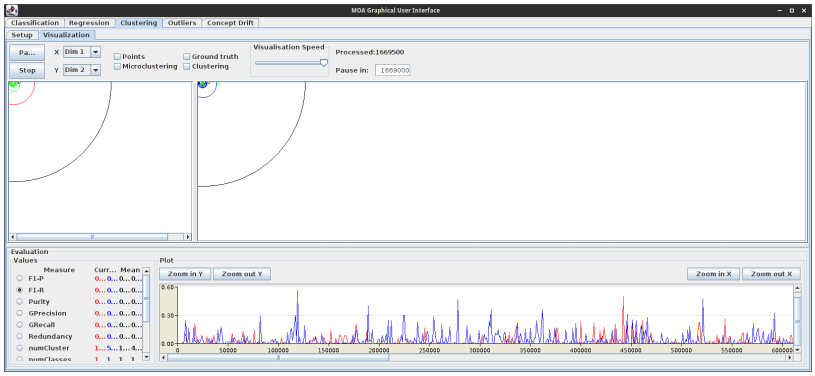


Figure A.35: MOA Results of ClusTree on Commercial-Small Data Set: F1-R

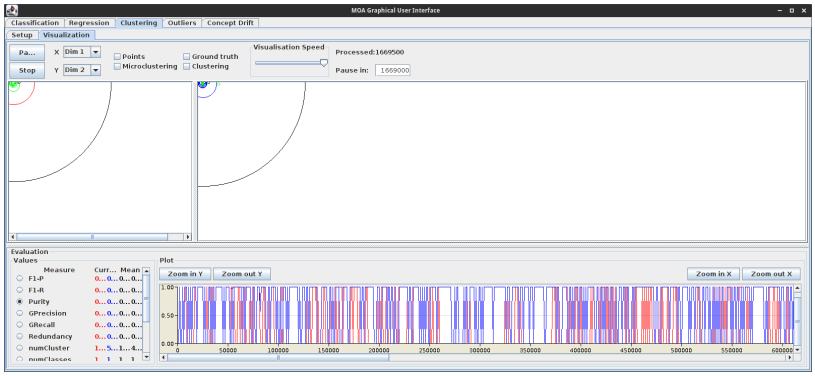


Figure A.36: MOA Results of ClusTree on Commercial-Small Data Set: Purity

## A.5 Commercial-Big

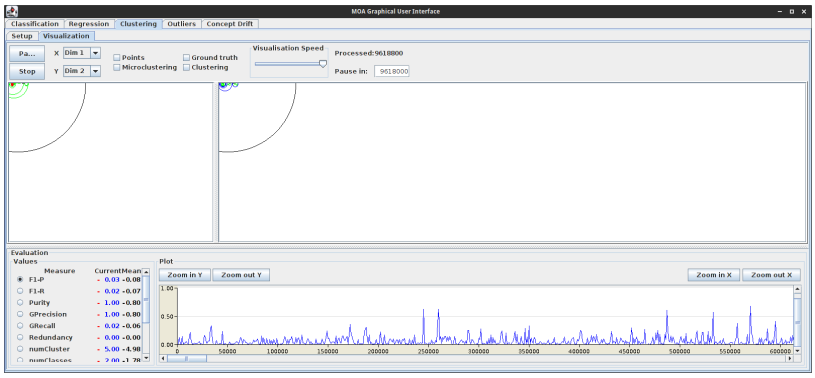


Figure A.37: MOA Results of DenStream on Commercial-Big Data Set: F1-P

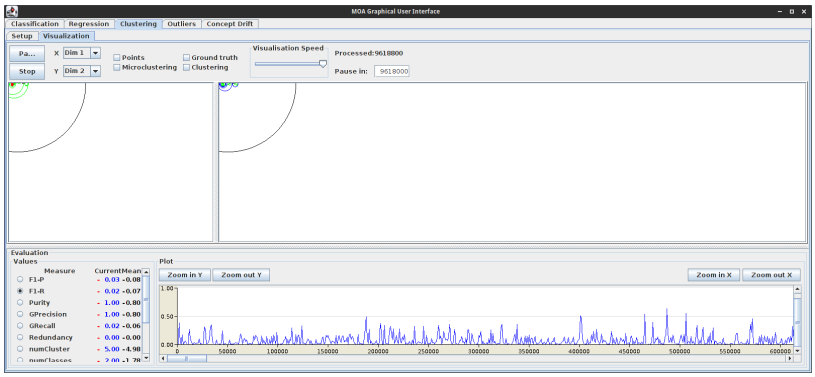


Figure A.38: MOA Results of DenStream on Commercial-Big Data Set: F1-R

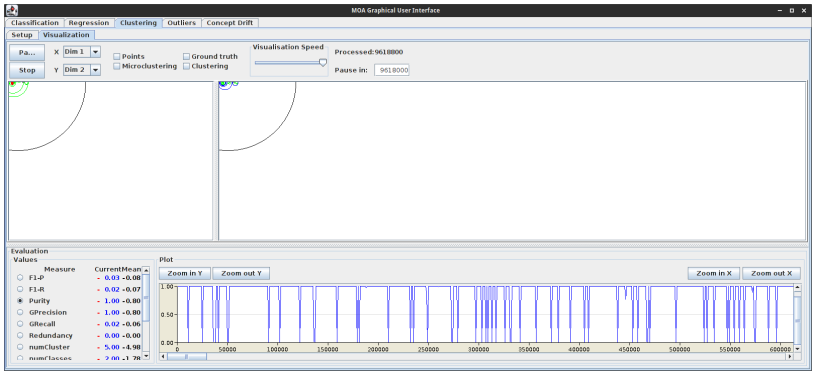


Figure A.39: MOA Results of DenStream on Commercial-Big Data Set: Purity

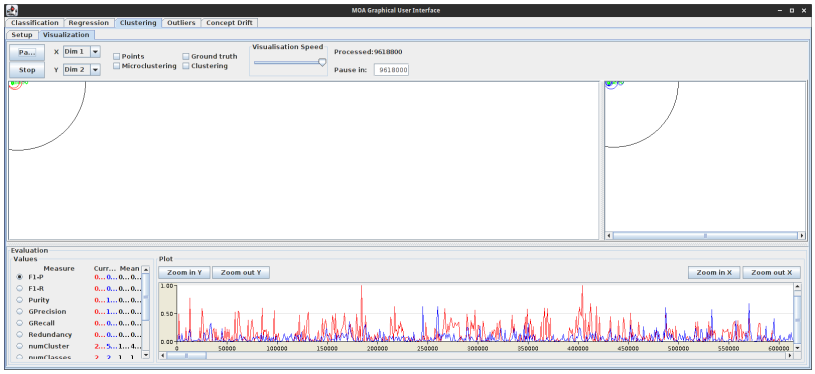


Figure A.40: MOA Results of CluStream on Commercial-Big Data Set: F1-P

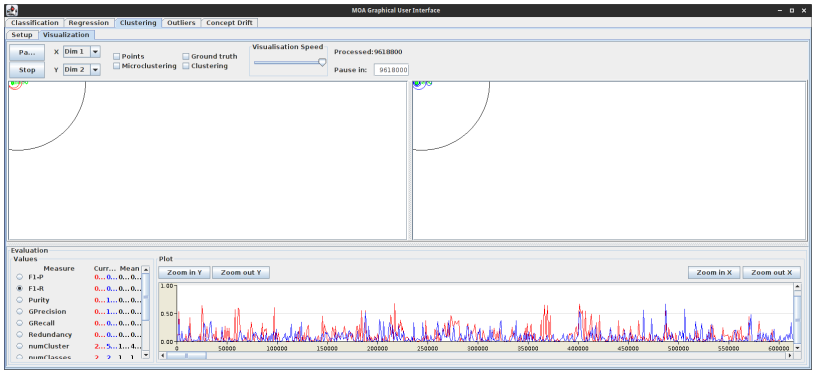


Figure A.41: MOA Results of CluStream on Commercial-Big Data Set: F1-R

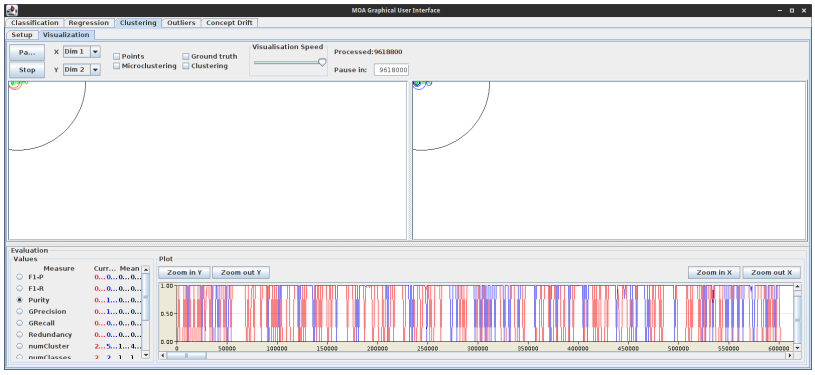


Figure A.42: MOA Results of CluStream on Commercial-Big Data Set: Purity

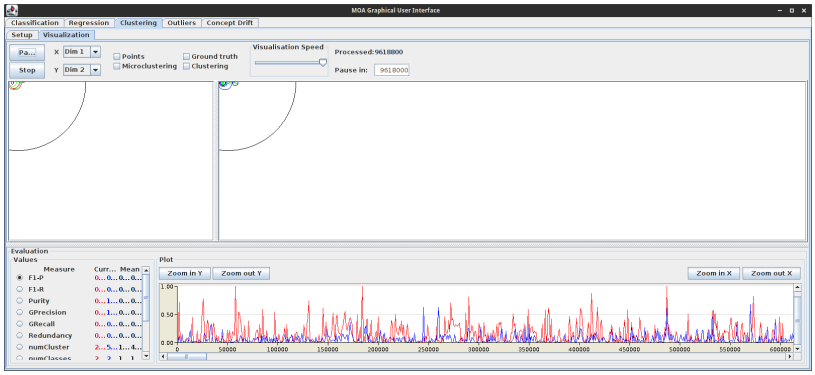


Figure A.43: MOA Results of ClusTree on Commercial-Big Data Set: F1-P

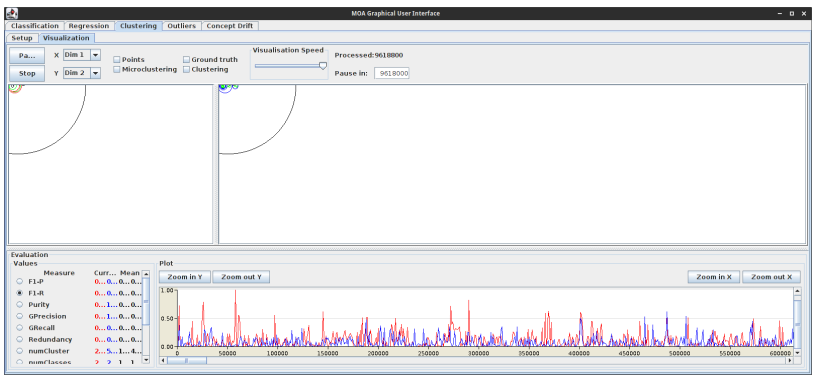


Figure A.44: MOA Results of ClusTree on Commercial-Big Data Set: F1-R

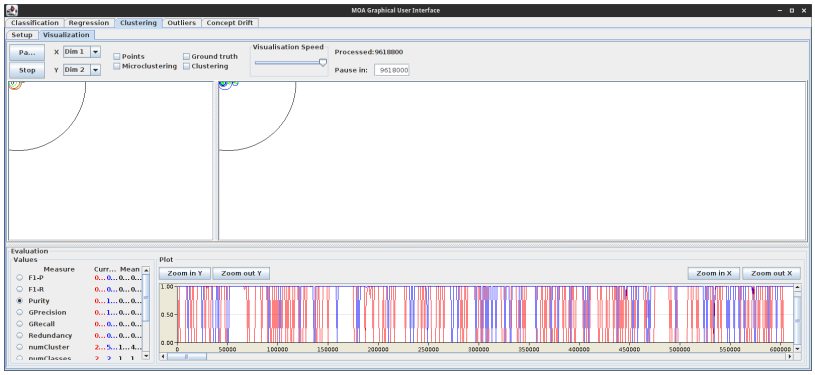


Figure A.45: MOA Results of ClusTree on Commercial-Big Data Set: Purity