University of Louisville

ThinkIR: The University of Louisville's Institutional Repository

---

Electronic Theses and Dissertations

5-2013

Stream-dashboard : a big data stream clustering framework with applications to social media streams.

Basheer Hawwash 1984-
University of Louisville

Follow this and additional works at: <https://ir.library.louisville.edu/etd>

Recommended Citation

Hawwash, Basheer 1984-, "Stream-dashboard : a big data stream clustering framework with applications to social media streams." (2013). *Electronic Theses and Dissertations*. Paper 587.
<https://doi.org/10.18297/etd/587>

This Doctoral Dissertation is brought to you for free and open access by ThinkIR: The University of Louisville's Institutional Repository. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of ThinkIR: The University of Louisville's Institutional Repository. This title appears here courtesy of the author, who has retained all other copyrights. For more information, please contact thinkir@louisville.edu.

STREAM-DASHBOARD: A BIG DATA STREAM CLUSTERING
FRAMEWORK WITH APPLICATIONS TO SOCIAL MEDIA
STREAMS

By

Basheer Hawwash

B.A., Jordan University of Science and Technology, 2006

M.S., University of Louisville, 2008

A Dissertation

Submitted to the Faculty of

Speed School of Engineering of the University of Louisville

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

Department of Computer Engineering and Computer Science

University of Louisville

Louisville, KY

May, 2013

STREAM-DASHBOARD: A BIG DATA STREAM CLUSTERING FRAMEWORK WITH APPLICATIONS TO SOCIAL MEDIA STREAMS

By

Basheer Hawwash
B.A., Jordan University of Science and Technology, 2006
M.S., University of Louisville, 2008

A Dissertation Approved on

April 22, 2013

by the following Dissertation Committee

Dissertation Director: Dr Olfa Nasraoui

Dr Ibrahim Imam

Dr. Hichem Frigui

Dr. Ming Ouyang

Dr. Ayman El-Baz

DEDICATION

This dissertation is dedicated to my parents

Mr. Zuhair Hawwash

and

Mrs. Aida Hawwash

for their continuous support, love and guidance in every step of my life.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr Olfa Nasraoui, for her inspirational guidance and support, and from whom I gained invaluable knowledge over the years, that would help me better build my career at both personal and academic levels. I would also like to express my deepest thanks to my wife, Aziza, for her support and patience in all the long and endless nights. I would not have finished my dissertation without her encouragement. Also, many thanks to my friends and lab mates at the Knowledge Discovery and Web Mining Lab and to the members of my dissertation committee for their feedback on this work.

Finally, I am grateful that this work was supported by US National Science Foundation Grant IIS-0916489.

ABSTRACT

STREAM-DASHBOARD: A BIG DATA STREAM CLUSTERING FRAMEWORK WITH APPLICATIONS TO SOCIAL MEDIA STREAMS

Basheer Hawwash

April 22, 2013

Data mining is concerned with detecting patterns of data in raw datasets, which are then used to unearth knowledge that might not have been discovered using conventional querying or statistical methods. This discovered knowledge has been used to empower decision makers in countless applications spanning across many multi-disciplinary areas including business, education, astronomy, security and Information Retrieval to name a few. Many applications generate massive amounts of data continuously and at an increasing rate. This is the case for user activity over social networks such as Facebook and Twitter. This flow of data has been termed, appropriately, a *Data Stream*, and it introduced a set of new challenges to discover its evolving patterns using data mining techniques. Data stream clustering is concerned with detecting evolving patterns in a data stream using only the similarities between the data points as they arrive without the use of any external information (i.e. unsupervised learning).

In this dissertation, we propose a complete and generic framework to simultaneously *mine*, *track* and *validate* clusters in a big data stream (Stream-Dashboard). The proposed framework consists of three main components: an online data stream clustering algorithm, a component for tracking and validation of pattern behavior using regression analysis, and a component that uses the behavioral information about the detected patterns to improve the quality of the clustering algorithm. As a first component, we propose RINO-Streams, an online clustering algorithm that incrementally updates the clustering model using robust statistics and incremental optimization. The second component is

a methodology that we call TRACER, which continuously performs a set of statistical tests using *regression analysis* to track the evolution of the detected clusters, their characteristics and quality metrics. For the last component, we propose a method to build some *behavioral* profiles for the clustering model over time, that can be used to improve the performance of the online clustering algorithm, such as adapting the initial values of the input parameters.

The performance and effectiveness of the proposed framework were validated using extensive experiments, and its use was demonstrated on a challenging real word application, specifically unsupervised mining of evolving cluster stories in one pass from the Twitter social media streams.

TABLE OF CONTENTS

	Page
ABSTRACT	viii
DEDICATION	viii
ACKNOWLEDGMENTS	viii
LIST OF TABLES	ix
LIST OF FIGURES	xi
1 INTRODUCTION AND MOTIVATION	1
1.1 Motivation	2
1.2 Problem Statement	4
1.3 Research Contributions	7
1.4 Organization of this Document	8
2 BACKGROUND AND RELATED WORK	10
2.1 Clustering Overview	11
2.2 Stream Data Mining	28
2.3 Tracking Cluster Evolution	43
2.4 Robust Statistics	51
2.5 Linear Regression Models	57
2.6 Topic Modeling	60
2.7 Summary and Conclusions	63

3	STREAM-DASHBOARD: A NEW FRAMEWORK TO MINE, TRACK AND VALIDATE EVOLVING DATA STREAM CLUSTERS	65
3.1	The RINO-Streams Algorithm	66
3.2	The TRACER Algorithm	85
3.3	Configuration Adaptation	100
3.4	Stream Genealogy Graph	100
3.5	Complete Generic Framework for Stream Cluster Tracking and Validation	103
3.6	Visualization Dashboard	105
3.7	Summary and Conclusions	109
4	EXPERIMENTAL RESULTS	110
4.1	Evaluation of Component 1: RINO-Streams	112
4.2	Evaluation of Component 2: TRACER	160
4.3	Application: Mining Twitter Data Streams	182
4.4	Summary and Conclusions	191
5	CONCLUSIONS AND FUTURE WORK	194
5.1	Summary	194
5.2	Future Work	196
	REFERENCES	196
	Appendix A	206
	CURRICULUM VITA	211

LIST OF TABLES

	Page
2.1 Sample Contingency Table	26
2.2 Similarity Contingency Table	27
2.3 Common M-estimators and W-estimators (Ricardo A. Maronna, 2006)	54
3.1 Comparison between RINO-Streams and other stream clustering algorithms	86
3.2 Stream Clustering Algorithm Metrics for Cluster C_i	88
3.3 Transition Conditions and Symbols	93
3.4 Transition Characterization Rules (conjunction $\wedge$, disjunction $\vee$, negation $\neg$), sorted by the order in which they are applied	93
3.5 Comparison between TRACER and local change detection techniques	101
4.1 Roadmap to the Experiments	111
4.2 RBF Data Stream Generator Parameters (Bifet <i>et al.</i> , 2010).	113
4.3 Real Text and Network Intrusion Detection Data Set Descriptions	114
4.4 RINO-Streams Parameter Values	115
4.5 TRAC-Streams Parameters' values	116
4.6 CluStream Parameters	116
4.7 Growing K-Means Parameters	116
4.8 DBSCAN's Optimal Parameters Values	118
4.9 The X-axis index values used in Figures (4.1-4.12)	119
4.10 The meaning of the X-axis index values used in Figures (4.13-4.16)	119
4.11 RINO-Streams vs CluStream vs Growing K-Means: Synthetic Data (Significance) .	136
4.12 RINO-Streams vs CluStream vs Growing K-Means for TREC text data (p-values)	145

4.13	Parameter configurations for cluster splitting and merging	147
4.14	RBF Parameters	149
4.15	ANOVA Table - RINO-Streams versus RBF Generator (Davies-Bouldin)	150
4.16	ANOVA Table - RINO-Streams versus RBF Generator (Silhouette-Index)	150
4.17	ANOVA Table - RINO-Streams versus RBF Generator (Normalized Mutual Infor- mation)	151
4.18	ANOVA Table - RINO-Streams Parameters (Davies-Bouldin)	154
4.19	ANOVA Table - RINO-Streams Parameters (Silhouette Index)	154
4.20	ANOVA Table - RINO-Streams Parameters (Normalized Mutual Information)	155
4.21	RINO-Streams Pareto Frontier	160
4.22	RINO-Streams Default Parameter Values	160
4.23	Dataset Properties & Component 1's Clustering Algorithms used in our experiments	162
4.24	ANOVA Parameters and their values	162
4.25	TRACER: N-Way ANOVA Results	163
4.26	TRACER: 1-Way ANOVA Results	164
4.27	TRACER: ANOVA Parameters Best Values	164
4.28	Unfiltered Twitter Dataset Properties	184
4.29	Twitter Stories: Sugar Bowl 2013 Topic Cluster Evolution Properties	189
4.30	Twitter Stories: Charlie Strong Topic Cluster Evolution Properties	190
4.31	Twitter Stories: Kevin Ware Injury Topic Cluster Evolution Properties	191
4.32	Twitter Stories: NCAA tweet discussions (Note: slang **)	192

LIST OF FIGURES

	Page
1.1 Motivational Example: Tracking Twitter Trending Topics	3
2.1 LDA Generative Process (Blei <i>et al.</i> , 2003)	63
3.1 Stream Dashboard Flowchart	66
3.2 Consecutive Time Periods (T1& T2)	90
3.3 Cluster Changes: Typical Causes and Effects	91
3.4 Internal & External Transitions	92
3.5 TRACER Example (Source and final output)	97
3.6 TRACER Example (T1 & T2)	97
3.7 TRACER Example (T3 & T4)	98
3.8 An example of density regression models of 8 clusters	106
3.9 An example of density stability plots for 8 clusters	107
3.10 Stream-Genealogy Graph	108
4.1 RINO-Streams vs TRAC-Streams: DS8 (difference in number of clusters detected)	120
4.2 RINO-Streams vs TRAC-Streams: DS8 (average centroid error)	121
4.3 RINO-Streams vs TRAC-Streams: DS8 (average scale error)	122
4.4 RINO-Streams vs TRAC-Streams: DS8 (difference in estimated noise)	123
4.5 RINO-Streams vs TRAC-Streams: DS8 (Davies-Bouldin Index)	124
4.6 RINO-Streams vs TRAC-Streams: DS8 (Silhouette Index)	125
4.7 RINO-Streams vs TRAC-Streams: DS16 (difference in number of clusters detected)	126
4.8 RINO-Streams vs TRAC-Streams: DS16 (average centroid error)	127

4.9	RINO-Streams vs TRAC-Streams: DS16 (average scale error)	128
4.10	RINO-Streams vs TRAC-Streams: DS16 (difference in estimated noise)	129
4.11	RINO-Streams vs TRAC-Streams: DS16 (Davies-Bouldin Index)	130
4.12	RINO-Streams vs TRAC-Streams: DS16 (Silhouette Index)	131
4.13	RINO-Streams vs IncDBSCAN: DS8 & DS16 (relative difference in number of clusters detected)	132
4.14	RINO-Streams vs IncDBSCAN : DS8 & DS16 (difference in estimated noise) . . .	133
4.15	RINO-Streams vs IncDBSCAN: DS8 & DS16 (Davies-Bouldin Index)	133
4.16	RINO-Streams vs IncDBSCAN: DS8 & DS16 (Silhouette Index)	134
4.17	RINO-Streams vs TRAC-Streams & IncDBSCAN: DS8 (final output for experi- ment C_8RN_{24})	135
4.18	RINO-Streams vs TRAC-Streams & IncDBSCAN: DS8 (similarity matrices for ex- periment C_8RN_{24})	135
4.19	RINO-Streams vs TRAC-Streams & IncDBSCAN: DS16 (final output for experi- ment $C_{16}RN_{18}$)	135
4.20	RINO-Streams vs TRAC-Streams & IncDBSCAN: DS16 (similarity matrices for experiment $C_{16}RN_{18}$)	136
4.21	RINO-Streams vs CluStream vs Growing K-Means: Synthetic Data (Overall Per- formance)	137
4.22	RINO-Streams vs CluStream vs Growing K-Means: Synthetic Data (Number of Clusters)	138
4.23	RINO-Streams vs CluStream vs Growing K-Means: Synthetic Data (Number of Dimensions)	139
4.24	RINO-Streams vs CluStream vs Growing K-Means: Synthetic Data (Noise Level) .	140
4.25	RINO-Streams vs CluStream vs Growing K-Means: Synthetic Data (Stream Length) 141	
4.26	RINO-Streams vs CluStream vs Growing K-Means: Synthetic Data (Time Com- plexity)	141
4.27	RINO-Streams vs CluStream vs Growing K-Means: Synthetic Data (Time Com- plexity with Respect With Data Stream Properties)	142
4.28	RINO-Streams vs CluStream : TREC (Normalized Mutual Information)	143

4.29 RINO-Streams vs CluStream : TREC (Cluster Purity)	144
4.30 RINO-Streams vs CluStream : TREC (Recall)	144
4.31 RINO-Streams vs CluStream : TREC (F1 Score)	144
4.32 RINO-Streams vs CluStream : TREC (Davies-Bouldin Index)	145
4.33 RINO-Streams vs CluStream : TREC (Time Complexity)	146
4.34 RINO-Streams vs CluStream : KDD CUP 99 (External Validity Metrics)	146
4.35 RINO-Streams vs CluStream : KDD CUP 99 (Time Complexity)	147
4.36 A cluster that gradually splits into three clusters over time	147
4.37 RINO-Streams: Three clusters that gradually merge into one cluster over time . . .	148
4.38 RINO-Streams: Time Complexity on Big Data Streams	148
4.39 The effect of the data stream properties on RINO-Streams output (Part 1)	152
4.40 The effect of the data stream properties on RINO-Streams output (Part 2)	153
4.41 RINO-Stream's parameter effect on quality of the clusters: Number of Clusters . .	156
4.42 RINO-Stream's parameter effect on quality of the clusters: Initial Scale	156
4.43 RINO-Stream's parameter effect on quality of the clusters: Forgetting Factor	157
4.44 RINO-Stream's parameter effect on quality of the clusters: Chebyshev Constant for outliers	157
4.45 RINO-Stream's parameter effect on quality of the clusters: Chebyshev constant for merging	157
4.46 RINO-Stream's parameter effect on quality of the clusters: Minimum Sum of Weights	158
4.47 RINO-Stream's parameter effect on quality of the clusters: Maturity Age	158
4.48 RINO-Streams Pareto Frontier	159
4.49 DS6: Cardinality vs. Time and its Summary Regression Models	165
4.50 Re0: Density vs. Time and its Summary Regression Model	165
4.51 KDD Cup 99 Network Activity	167
4.52 Cardinality regression models versus the groundtruth cardinality for three main net- work activities	167
4.53 DS1A: Online Validation of the Tracking and Summary of the Cluster Scale Evo- lution using Regression Models against their Ground-truth for two different online clustering algorithms used in Component 1.	168

4.54 DS1: Ground Truth	170
4.55 DS1: Cardinality & Scale Regression Models	170
4.56 DS2A: Dataset Evolution and Final Clustering	171
4.57 DS2A: Validating the Detection and Tracking of a Splitting Transition	171
4.58 DS5: Validating and Tracking the Outputs of Two Different Online Algorithms using Density Regression Models	172
4.59 DS2B: MONIC vs Stream-Dashboard External Changes	174
4.60 DS6: TRACER Compared against Baseline	175
4.61 Effect of the TRACER parameters on milestone detection and regression model quality	176
4.62 Time & Memory Complexity of TRACER	177
4.63 DS6: Stream Genealogy Layout with different filters	178
4.64 DS2B: behavioral trend of a merged cluster and its ancestors	178
4.65 DS10: Density Over time Vs. Regression Models (for experiment $C_{10}ON_{10}$, ordered data arrival)	179
4.66 DS10: Density Over time Vs. Regression Models (for experiment $C_{10}RN_{10}$, random data arrival)	180
4.67 DS10: Density Stability Over Time	181
4.68 DS10: Interactive Cluster Ancestry Tree	182
4.69 Twitter Properties	185
4.70 Twitter: Popular Hashtags Per Month	186
4.71 Twitter: Detected Trending Topics for Several Days	187
4.72 Louisville Tweets	188
4.73 Louisville tweets density regression models	188
4.74 Twitter Stories: Sugar Bowl 2013 Topic Cluster Evolution	189
4.75 Twitter Stories: Charlie Strong Topic Evolution	190
4.76 Twitter Stories: Kevin Ware Injury Topic Cluster Evolution	191
4.77 Twitter Stories: NCAA topic centroids	192

CHAPTER 1

INTRODUCTION AND MOTIVATION

In recent years, there has been a proliferation of applications that generate massive amounts of data continuously and at an increasing rate. This is due, primarily, to the faster and lower cost hardware, the emergence of new paradigms that thrive on user-generated data such as social networks, and the recognition of the importance of utilizing raw data (which was previously useless) in obtaining new information that are vital to deal with a variety of real life issues. These massive amounts of data have been called data streams, and are mainly characterized by a huge throughput and a dynamic nature. Data streams can be found in many applications, such as sensor networks, web logs, computer network traffic, and the activities on mobile phone networks.

To discover useful information out of data streams, classical data mining techniques have been modified and applied on data streams, which caused the emergence of the new discipline of Stream Data Mining (Babcock *et al.*, 2002; Guha *et al.*, 2003; Stonebraker *et al.*, 2005; Gaber *et al.*, 2005). Stream data mining is concerned with extracting knowledge represented as a set of patterns in a continuous stream of data, and it is a much more challenging task, than traditional data mining, due to the constraints imposed by the nature of data streams. First, the memory constraints are very tight since storing the whole data stream is not feasible or even possible. Second, as new data arrives, it needs to be processed fast enough to minimize any delays. These constraints on the computational and memory complexity add more challenges in designing stream data mining algorithms. Many researchers have developed stream data mining algorithms to meet these challenges (Aggarwal *et al.*, 2003; Cao *et al.*, 2006; Charikar *et al.*, 2003; Chen & Tu, 2007; Nasraoui & Rojas, 2006).

Stream data mining techniques can be grouped, as in classical data mining techniques, into three main groups: supervised (classification), unsupervised (clustering) and semi-supervised learning.

The focus of this work is on unsupervised stream data mining, and more specifically, we propose a complete and generic framework to mine, track and validate clusters in big data streams on the fly, which highlights the importance of monitoring the behavior of the evolving clusters rather than simply detecting them.

The remainder of this chapter discusses the motivations behind this work (Section 1.1), presents a statement of the problems to be solved relating to mining and monitoring data stream clusters, the challenges, and the questions that will be answered by solving the stated problems (Section 1.2). This is followed by a list of the contributions of this dissertation (Section 1.3), and finally the organization of this document (Section 1.4).

1.1 Motivation

Most of the research in Stream Data Clustering has focused primarily on detecting the clusters embedded in the data stream and using this knowledge in higher-level applications (e.g. in visualization (Leuski & Allan, 1997) or in recommender systems (Sarwar *et al.*, 2002)). However, only a few have focused on tracking the actual evolution of these clusters over time, especially in data streams, where new data arrives with a huge throughput that makes conventional clustering algorithms useless. For example, it is estimated that users generate about 500 Million tweets every day¹.

Detecting clusters in a data stream, while simultaneously monitoring and tracking their evolution, adds a more challenging but valuable perspective for Stream Data Mining. The discovered clusters provide the tools to help decision makers make a knowledgeable decision. If this task is done periodically (e.g. detecting trending topics in Twitter), then extracting the clusters every time becomes expensive and possibly redundant (e.g. Twitter trending topics do not change, typically, every hour). Hence, the decision makers become familiar with the clusters over time, and become more interested in identifying the changes that took place instead. These changes might occur to the discovered clusters internally (i.e. the characteristics of the cluster) or externally (i.e. interaction of clusters with each other). For example, in customer relationship management, it may be valuable to know if an emerging pattern of behavior is the result of (*i*) a completely new group of users

---

¹<http://www.telegraph.co.uk/technology/twitter/9945505/Twitter-in-numbers.html>

Figure 1.1: Motivational Example: Tracking Twitter Trending Topics

(i.e. emergence of a new pattern), **(ii)** a shift in a previous group's behavior (i.e. internal pattern change), or if **(iii)** it is a mergence of two groups of users that used to have different interests? (i.e. external pattern changes (Nasraoui *et al.*, 2008)). Answering such questions can provide valuable information for planning marketing strategies.

As a motivational example, Figure 1.1 shows how the proposed framework could be applied to a stream of tweets. As an input, a stream of tweets, regarding the revolutions that started in the middle east at the end of 2010 and beginning of 2011, is collected and processed using an online clustering algorithm to find the trending topics (3 topics are shown). The topic metrics, that characterize the topics, are tracked over time. For example, the density (ratio between number of tweets and the variance of the topic) reflects the popularity of the topic (i.e. as the topic becomes popular, its density increases). Some of the critical events (i.e. milestones) during the time periods are annotated using circles and are defined in the legend. It can be seen that the topics became viral at major events, which reflects the twitter community's reaction to those events. Tracking the evolution of the topics, and detecting the major events automatically, provides important insights about these topics and how society reacts to them.

State of the art approaches, which will be discussed in Section 2.3, try to track the evolution of clusters by finding the differences or deviations between two clustering models at two different time periods. However, this implies that re-clustering the data is needed at each time stamp, which would

increase the time complexity and is not scalable enough to handle data streams. Moreover, only few of those approaches can detect internal and external changes that might take place at the clusters. Another approach to track clusters' evolution, could be to keep a snapshot of the clustering model at every time stamp or at certain time stamps. However, this approach would increase the memory complexity and defy the scalability requirement in stream data mining. Moreover, selecting the time stamps, at which a snapshot of the clustering model is stored, is a challenging problem.

1.2 Problem Statement

The problems that we attempt to solve in this work are: *mining*, *validating* and *tracking* the evolving clusters that are hidden in a continuous data stream in a single pass over the data. These problems are discussed below.

1.2.1 Detecting clusters in a data stream

Given a data stream X , where a new data point x_i arrives at time i , we are interested in detecting a set of evolving clusters ζ (i.e. a clustering model) that reflects the *evolving behavior* of the data stream. Each cluster represents a portion of a the data stream seen so far, where the data points in this portion are more similar to each other than data points in other clusters. Each cluster consists of a set of metrics/properties that distinguish it from other clusters. Such metrics include a cluster representative (i.e. a cluster's centroid) and scale (i.e. the size of a cluster's influence area). Data streams are characterized by a dynamic nature, where new clusters emerge, old ones may undergo changes in their metrics (i.e. internal changes), merge together if they become very similar, or split if they become too general (i.e. external changes). Hence, the classical definition of a cluster needs to be modified to capture the evolving clusters in a data stream.

For intuition, a typical example of stream data mining is in the domain of web usage mining, where the webmaster is interested in detecting all the usage/browsing patterns that take place on an e-commerce website that sells different products. Each of the detected usage patterns is considered one cluster, and it may represent a set of products that are related based on the users' activities and interests, and they provide invaluable information for the webmaster. In fact, this knowledge might help in recognizing which parts of the website are more related to each other, and in turn

help improve the structure of the website. Another obvious usage is for strategic marketing, where advertisements could be tailored to individual users or groups of users based on their usage patterns. The browsing patterns tend to be dynamic, in the sense that groups of users might change their interests for various internal or external reasons.

1.2.2 Validating evolving data stream clusters

After detecting the clusters, the second problem deals with *validating* their quality. Traditional validation methods (Halkidi *et al.*, 2002a,b; Tan *et al.*, 2005) evaluate the clustering results by either (i) using some external knowledge about what the clustering model should be and comparing it against the detected clusters, or (ii) by assessing how well the data stream conforms to the detected clusters, or else by judging the clusters' quality without referring to the data (i.e. internal metrics). For example, the detected clusters should have low similarity with each other (i.e. they are distinct). However, in the evolving data stream scenario, these validation methods are hard to apply because of the evolution of the clusters that can change their properties anytime. For example, if two products are known to be similar through external knowledge (e.g. both products are books about data mining), but over time users show different interests in them for different reasons (e.g. one book receives very low ratings), then these products might be classified in two different clusters. By classical external evaluation metrics, the new two clusters will be misjudged of being of low quality, although they might be truly of high quality and really reflect the needs of the users.

1.2.3 Tracking the evolution of the detected data stream clusters

The final part of the problem deals with *tracking* the evolution of the detected clusters. This task is important and helps in validating the clusters, since it provides the means to explore the internal and external quality of each cluster during its entire lifetime. For example, a group of users who are interested in a specific product at some point in time might change their interests and show more interest in a different product. However, both usage clusters may still be completely valid and useful. Moreover, tracking the evolution of clusters helps in detecting the major changes that take place, and possibly support finding the reasons driving these changes. Back to the web usage mining example, introducing a new product or offering a discount on a product might cause major changes in the users' behavior. Tracking and recording the clusters' evolution can increase the

memory overhead needed for storing this enriched cluster model. Hence, we propose a method that is capable of tracking the clusters' evolution with low memory cost by storing the evolution behavior of clusters only at special moments corresponding to when major changes, that we call *milestones*, take place.

1.2.4 Challenges

Besides the main problems that we are trying to solve in this dissertation, our proposed solutions must adhere to the following requirements of mining data streams:

- The huge data throughput makes it hard, if not impossible, to store every single data point and process it, therefore each new data point should be processed, *on the fly*, only once.
- There is no control or expectation on the order of the data arrival (for example, data points from the same cluster do not have to arrive sequentially one after the other).
- Data streams are unbounded in size, thus imposing severe restrictions on storage capacity.
- Outlier detection is more challenging, since a data point that is flagged as an outlier at the beginning of the data stream might turn out to be part of a cluster that emerges later in the data stream lifetime.
- Validating the clustering model is hard since the input data points are not kept, and only a summary of the data may be maintained. In fact, even if data points were stored, the data stream is expected to continue evolving, hence data points might be assigned to different clusters at different time periods.
- There is no notion of a static cluster model or output. Instead, at any given time or point in the data stream, there is a different cluster model. Hence, which cluster model should be reported?

1.2.5 Questions to be answered through solving the stated problems

By proposing solutions to the stated problems, this dissertation answers the following questions:

1. Whether a single cluster model is a sufficient output of the data stream clustering process?

2. If not, which cluster model should be reported or stored?
3. If every point in time, throughout the data stream, generates a distinct cluster model, then there is a possibility that the cluster model outputs from clustering will generate their own data stream that will add even more overhead to the entire stream data mining task. To avoid this problem, is there a simple way to summarize the dynamic clustering output for a fast moving data stream?

1.3 Research Contributions

A generic framework

The main contribution of this work is a complete framework to simultaneously *mine*, *track* and *validate* the detected clusters in big data streams. The proposed framework, termed *Stream-Dashboard*, makes an emphasis on characterizing and tracking the behavior of the detected clusters' description and validity metrics in a data stream over time, rather than just detecting the clusters, and it provides the means to investigate the clusters' evolution characteristics at any point in time. Stream-Dashboard consists of three main components: an online data stream clustering algorithm (Section 3.1), a component for tracking and validation of cluster behavior using regression analysis (Section 3.2), and a component that can exploit the observed clusters' behavior to improve the quality of the online clustering component (Section 3.3). The framework is generic in the sense that *any* online clustering algorithm can be used in the first component, in order to detect the clusters. The only requirement is that this online clustering algorithm must be able to quantify the characteristics of the detected clusters via cluster model parameters or metrics.

An online robust stream clustering algorithm

For the first component, we propose the *Robust clustering of data streams using INcremental Optimization algorithm* or *RINO-Streams* (Section 3.1), which is an online clustering algorithm that incrementally updates the clustering model using robust statistics. RINO-Streams complies with all the requirements of data stream clustering discussed in Section 3.1.10, and is robust to noise thanks to the use of robust statistics, combined with using distribution-independent Chebyshev bounds to

detect outliers and to detect similar clusters to be merged.

An algorithm for tracking the clustering model behavior using regression analysis and detecting milestones

For the second component, we propose an algorithm for *TR*Acking and validating *Cluster Evolution* using *Regression analysis* or *TRACER* (Section 3.2), which is a method that uses regression analysis to keep track of the evolution of the detected clusters and their metrics through time, and that stores only *milestones* corresponding to the occurrence of significant changes in the clusters' behavior. Instead of storing an infinite number of summaries of the stream (at each instant or an arbitrary sample of summaries), only *temporally salient* synopsis snapshots of the stream will be stored to disk when significant changes are detected, together with a model of this change in between consecutive salient snapshots.

Improving the performance of the online clustering algorithm via feedback

Tracking the behavior of the clustering model over time can eventually help in building *behavioral profiles* of 'good' and 'bad' clusters. These profiles could be used as feedback to improve the performance of the online clustering model in two aspects: *(i)* reducing the sensitivity associated with using some of the threshold parameters needed to judge the quality of the clusters, and *(ii)* offering better means to initialize the input parameters used within the online clustering algorithm (Section 3.3).

Visualization dashboard

Tracking the evolution of the data stream is presented using a *dashboard* that enables the user to control the input parameters of the framework, displays the changes of the cluster metrics over time, identifies the detected evolution milestones (Section 3.2.3), and creates a genealogy graph of all the clusters detected (Section 3.4).

1.4 Organization of this Document

The rest of this dissertation is organized as follows: Chapter 2 reviews important research background related to the problems at hand, including state of the art clustering algorithms, clustering validation methods, the issues of stream data mining along with state of the art stream clustering algorithms, robust statistics, approaches to track the clusters' evolution, and ending with an overview of linear regression models and topic modeling. Chapter 3 provides a detailed analysis of the proposed framework and its components. Chapter 4 presents the experiments conducted to prove the functionality and effectiveness of the proposed work. Finally, Chapter 5 concludes this dissertation and discusses possible future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we will review the background and related work that is related to this dissertation. Section 2.1 provides an overview of unsupervised learning, where we present some of the existing clustering algorithms: Partition-based algorithms (Section 2.1.1), Hierarchical algorithms (Section 2.1.2), Density-based algorithms (Section 2.1.3) and Grid-base algorithms (Section 2.1.4). Section 2.1.5 discusses the issues related to clustering in high dimensions, and Section 2.1.6 lists some of the clustering validation methods used to evaluate the quality of the clustering model. Section 2.2 provides an overview about Stream Data Mining, and it starts by listing the challenges imposed by the stream model and the requirements for algorithms to deal with these challenges (Section 2.2.1). Then we present some state of the art data stream clustering algorithms from two families: One-pass algorithms (Section 2.2.2.1) and Evolving algorithms (Section 2.2.2.2). Section 2.2.3 lists the challenges and approaches to evaluate the detected clusters in data streams.

Section 2.3 discusses the issues related to the problem of tracking the evolution of clusters detected in the data stream, as well as analyzing some of the current research in that field. Section 2.4 reviews some of the essential concepts from the field of Robust Statistics (Ricardo A. Maronna, 2006; Huber, 1981), which are used in developing the online clustering algorithm RINO-Streams (Section 3.1). Section 2.5 reviews Linear Regression Modeling (Chatterjee & Hadi, 2006), which is used in the proposed cluster evolution tracking algorithm, TRACER (Section 3.2). Section 2.6 provides a brief overview about Topic Modeling and presents two of the state-of-the art approaches. Topic Modeling can be used as a pre-processing step to reduce the very high dimensionality of the text data streams, such as Twitter data, which will be used as an application domain in this work. Finally, Section 2.7 presents the summary and conclusions found in this chapter.

2.1 Clustering Overview

Clustering is the process of grouping data objects based only on the information found in the data that describes the data objects and their relationships (?). Clustering is also called “unsupervised” learning as opposed to “supervised” learning. The latter, also known as classification, corresponds to the case when class prediction models need to be built using a training set of data objects whose label is known (Tan *et al.*, 2005). The goal of clustering is to generate groups of data objects where the data objects in the same group are similar to each other, but are different from the data objects in other groups. Clustering has been studied intensively, with several surveys in (?Berkhin, 2006; Jain, 2010) and has numerous applications in many fields: information retrieval (grouping search results), biology (creating a taxonomy of living things), network security (learning usage groups for detecting anomalous attacks), and in business (building customer profiles) to name a few.

The following sections review several well known clustering algorithms, classified into four main families: partition-based, hierarchical-based, density-based and grid-based algorithms.

2.1.1 Partition-based algorithms

Partition-based clustering techniques divide the data objects into non-overlapping subsets (clusters) such that each data object belongs to one cluster. They achieve this goal by assigning the data objects to the clusters in such a way that it optimizes a certain objective function defined in advance.

K-Means

A well known technique in this category is K-Means (MacQueen, 1967), which starts by first selecting K initial centroids, where each centroid is a representative of a cluster and whose feature values are equal to the mean of the corresponding features of the data points in that cluster. Each data point is assigned to the closest cluster using some similarity or dissimilarity measure function (typically the Euclidean or L_2 distance for numerical data in a Euclidean space), and then each centroid is updated using the points assigned to its cluster. This process of assigning data points to centroids and updating the centroids is repeated until the centroids converge (i.e. until the objective function converges to an optimum). The K-Means algorithm is listed in Algorithm 1. The time complexity of K-means is linear with the number of data points n , and is equal to $O(I \times K \times n \times m)$ where K is

---

Algorithm 1 K-Means Algorithm

---

1. Select K arbitrary initial centroids
  2. Assign each data point to its closest centroid
  3. Compute the centroids of the new clusters using (2.2)
  4. Repeat steps 2 and 3 until the centroids don't change or the change is below a specified threshold value
-

the number of clusters, I is the number of iterations and m is the number of attributes.

The objective function that is optimized is the sum of squared distances (SSE) between each data point and its assigned cluster representative. The objective function is as follows:

$$SSE = \sum_{i=1}^K \sum_{x \in C_i} dist(c_i, x)^2 \quad (2.1)$$

where $dist$ is the distance between two objects and c_i is the centroid of cluster C_i . The centroid that minimizes SSE can be found as:

$$c_i = \frac{\sum_{j=1}^n x_j}{n} \quad (2.2)$$

where n is the cardinality of the cluster C_i (i.e. number of points assigned to C_i).

K-Means is intuitive and easy to implement, but it has serious drawbacks, including the need for selecting the number of clusters K , as well as the initial K centroids. However, an even harder problem lies at the core of the objective function. The minimization is based on squared distances which magnifies the differences for data points that are located far away from the centroids (i.e. outliers). Thus, there is a bias to make the centroids close to these outlying points, regardless of how representative they are. Hence, K-Means is not robust to noise and outliers. Being distance-based, K-means also forcibly seeks clusters of equal size, scale or width in feature space regardless of any difference in sizes.

Fuzzy C-Means (FCM)

The assignment of each data point to only one cluster, in K-means, was later relaxed in the Fuzzy C-Means (FCM) algorithm developed by Dunn (Dunn, 1973), and later improved by Bezdek (Bezdek,

---

Algorithm 2 FCM Algorithm

---

1. Assign initial values to the memberships of all data points
  2. Compute the centroids using (2.4)
  3. Recompute the membership values using (2.5)
  4. Repeat steps 2 and 3 until the centroids don't change or the change is below a specified threshold value
-

1981). In FCM, every data point is allowed to have a degree of membership in more than one cluster, and these memberships are constrained to be in the interval $[0, 1]$, and to sum to 1. The memberships give information about the relative closeness between the data point and all the clusters, which is very useful in case of overlapping clusters as in the case of documents (i.e. a document can be in categorized in two categories such as politics and economics). FCM is listed in Algorithm 2.

The objective function for FCM is a modification of K-means' objective function (2.1) as follows:

$$SSE = \sum_{i=1}^K \sum_{j=1}^n \mu_{ij}^m \text{dist}(c_i, x_j)^2 \quad (2.3)$$

where K is the number of clusters, n is the number of data points in each cluster, μ_{ij}^m is the membership of the data point x_j in the cluster C_i , and m is an exponent that controls the influence of the weights and has a value between 1 and ∞ . Note that setting the exponent m to 0 yields the K-means' objective function. The centroid that minimizes SSE is given by:

$$c_i = \frac{\sum_{j=1}^n x_j \mu_{ij}^m}{\sum_{j=1}^n \mu_{ij}^m} \quad (2.4)$$

The memberships are updated after the centroids are updated, and the formula to update the memberships can be derived from (2.3) as follows:

$$\mu_{ij} = \frac{1}{\sum_{q=1}^K \left(\frac{\text{dist}(c_i, x_j)}{\text{dist}(c_q, x_j)} \right)^{\frac{2}{m-1}}} \quad (2.5)$$

FCM still suffers from most of the problems suffered by K-means. However the use of a fuzzy partition smoothes the search space, thus making optimization easier and therefore better results,

especially in recovering from bad initializations.

Expectation-Maximization (EM)

The Expectation-Maximization algorithm (EM) (Dempster *et al.*, 1977), follows an approach close in spirit to K-Means, which can be shown to be a special case of EM (Casella & Berger, 2001). Modeling the dataset as a mixture of data points generated by K distributions with known form, such as Gaussian, EM tries to determine the model parameters, θ_j , using posterior probabilities to maximize the likelihood of the data under these estimated model parameters. If the j^{th} distribution has parameters θ_j , then $prob(x_i|\theta_j)$ is the probability of the i^{th} data point coming from the j^{th} distribution. Each distribution has a weight w_j which reflects the probability of being chosen to generate a data point, and the weights for all distributions sum to 1. If Θ is the set of all parameters, then the probability of the i^{th} object is given by:

$$prob(x_i|\Theta) = \sum_{j=1}^K w_j prob(x_i|\theta_j) \quad (2.6)$$

If the objects are assumed to be identically generated, then the probability of the data set X (or the *likelihood* function) is the product of the probabilities of each data point:

$$prob(X|\Theta) = \prod_{i=1}^N \sum_{j=1}^K w_j prob(x_i|\theta_j) \quad (2.7)$$

where N is the number of data points.

The posterior probabilities can be viewed as memberships as in FCM, but keeping in mind that, in contrast to FCM, each data point belongs to one cluster only. The step E (Expectation) estimates the posterior probabilities, while the step M (Maximization) updates the parameters Θ , in an iterative process that finishes when no significant change occurs. EM is listed in Algorithm 3.

The EM algorithm provides a more general representation of data using mixture models, which allows the detection of clusters with different sizes and shapes. Clusters are easier to characterize since they can be described by a small number of parameters. However, EM has a high computational complexity, does not perform well when clusters have low cardinality, and requires estimating the number of models or clusters in advance.

Algorithm 3 EM Algorithm

---

1. Select an initial set of model parameters (Θ)
  2. **Expectation Step:** Find the probability that each data point belongs to each distribution
  3. **Maximization Step:** Use the probabilities found in the E step to find new estimate of the model parameters (Θ) that maximize the likelihood (2.7)
  4. Repeat steps 2 and 3 until the parameters' change is below a specified threshold value
-

2.1.2 Hierarchical algorithms

Hierarchical clustering algorithms (Ward, 1963) generate a 'taxonomy' of clusters (also known as a dendrogram or a cluster tree), which allows exploring data at different levels of clustering granularity. There are two main families of hierarchical clustering algorithms: agglomerative (bottom-up) and divisive (top-down). The agglomerative scheme starts with each data point being a cluster, then recursively merges two or more clusters (based on some similarity measure) until a stopping criterion is met. The divisive approach works inversely (top-down), thus beginning with the whole dataset considered as one cluster, and recursively splits appropriate clusters (based on some criterion of cluster quality). In both cases, each step (merging or splitting) represents one stage or level in the hierarchy. The key step in the hierarchical algorithms is the splitting (divisive approach) or the merging (agglomerative approach) of the clusters. The advantage of hierarchical clustering includes the flexibility regarding the level of granularity and the ease of handling any measure of similarity. However, they suffer from some vagueness of termination criteria, and clusters can not be improved once they are constructed (i.e. once a cluster is split or two clusters are merged they cannot be modified). Measures of closeness among clusters are the single linkage (minimum distance between any two points from the clusters), the complete linkage (maximum distance between any two points from the clusters), and variations along these lines, such as the mean or the median of distances. The agglomerative algorithm chooses the two closest groups (according with the above measure) and merges them, whereas the divisive algorithm chooses the cluster to be split that is the biggest in size (or lowest in quality).

Algorithm 4 CURE Algorithm

---

1. Draw a sample from the dataset
  2. Partition the sample into p groups
  3. Cluster each partition into $\frac{m}{pq}$ clusters, which means a total number of clusters equal to $\frac{m}{q}$
  4. Cluster the found $\frac{m}{q}$ clusters into K clusters
  5. Eliminate outliers
  6. Assign all (unsampled) remaining data points to the nearest cluster
-

CURE

(Guha *et al.*, 1998) introduced the hierarchical agglomerative clustering algorithm CURE (Clustering Using REpresentatives). This algorithm achieves scalability by using two devices: *(i)* using a certain number of data points, instead of all of them, to determine the closeness between clusters (data sampling), and *(ii)* partitioning the data in p groups, so that fine granularity clusters are constructed in partitions first. A major feature of CURE is that it represents a cluster by a fixed number of points (medoids) that are well scattered around it instead of using only one point such as the centroid, which makes it possible to detect non-spherical shapes. The distance between two clusters used in the agglomerative process is equal to the minimum of distances between two scattered representatives. Single and average link closeness is replaced by the representative medoids' aggregate closeness. CURE employs one additional device: the originally selected scattered points are shrunk to the geometric centroid of the cluster by a user-specified factor, which suppresses the effect of outliers, since outliers happen to be located further from the cluster centroid than the other scattered representatives. CURE is listed in Algorithm 4, where K is the number of desired clusters, m is the number of data points, p is the number of partitions, and q controls the desired number of points in each partition

CHAMELEON

The hierarchical agglomerative algorithm CHAMELEON (Karypis & Kumar, 1999) utilizes dynamic modeling in cluster aggregation. The first step in CHAMELEON is to represent data items using a k-nearest neighbor graph: each node represents a data item, with an edge between every two

---

Algorithm 5 CHAMELEON Algorithm

---

1. Build a k-nearest neighbor graph
  2. Partition the graph using a graph partitioning algorithm (Karypis & Kumar, 1998)
  3. Merge the small sub-clusters using measures of relative inter-connectivity while preserving cluster self-similarity
  4. Repeat step 3 until no more clusters can be merged
-

nodes if one of them is among the k-most similar points to the other one. The similarity between each pair of clusters is determined using their relative inter-connectivity (the sum of the weights of the edges that connect nodes in both clusters) and their relative closeness (the average similarity between the connected nodes in the clusters). CHAMELEON consists of two stages: the first stage uses a graph partitioning algorithm to partition the k-nearest neighbor graph into a large number of small sub-clusters that minimize the relationship among data points in different sub-clusters. In the second stage, an agglomerative process is performed that uses measures of relative inter-connectivity and relative closeness. CHAMELEON is listed in Algorithm 5. The algorithm does not depend on assumptions about the data model, and it was proven to find clusters of different shapes, densities, and sizes in 2D (two-dimensional) space. However, it has problems if the partitioning process does not produce valid sub-clusters (i.e. most of the data points in the sub-cluster belong to a true cluster), which is often the case for high dimensional data.

PDDP

(Boley, 1997) proposed the divisive hierarchical algorithm PDDP (Principal Direction Divisive Partitioning). To split a cluster, the eigenvector with the highest eigenvalue of the covariance matrix is calculated, and this is called the principal direction. Then, all the data points in that cluster are projected on this principal direction, and based on the sign of their projection, they are assigned either to the left or right child (i.e. if the sign is negative then the data point is assigned to the left child, and to the right child otherwise). To reduce the time complexity of finding the eigenvectors, PDDP uses Singular Value Decomposition (SVD) (Golub & Kahan, 1965). To choose which cluster to split next, PDDP uses a scatter value that measures the non-cohesiveness of the cluster, and selects the one with the highest scatter value. The authors used the Frobenius norm of the data points

---

Algorithm 6 DBSCAN Algorithm

---

1. Label all the data points as core, border, or noise points
  2. Remove noise points
  3. Add an edge between all core points that are within ϵ distance from each other
  4. Each group of connected core points is considered a cluster
  5. Assign each border point to the cluster of its closest core point
-

matrix as the scatter value.

2.1.3 Density-based algorithms

For density-based algorithms, the notion of cluster is identified by crowded regions, with a nearest-neighborhood flavor. This notion of dense regions results in discovering clusters with arbitrary shape as opposed to a certain shape (e.g. a hypersphere in the Gaussian model).

DBSCAN

A well known density based algorithm is Density Based Spatial Clustering of Applications with Noise, DBSCAN, (Ester *et al.*, 1996). DBSCAN searches for the regions that have at least a certain number of data point (MinPts), each one no further from the others than a certain distance (ϵ -neighborhood). Using these user-defined parameters (MinPts and ϵ), each point is labeled as core, border, or noise point based on the distance of the points in its ϵ -neighborhood. An incremental version of DBSCAN was proposed in (Ester *et al.*, 1998), where the same technique of detecting dense regions was applied on chunks of data at a time. DBSCAN can detect arbitrary-shaped clusters, however, it is very sensitive to the choice of its parameters (MinPts and ϵ) since a small value of MinPts and ϵ may result in mislabeling a set of noise points that are close to each other, as a valid cluster. Moreover, it does not perform well when the clusters vary widely in their densities. Also, its complexity is high, $O(N^2)$, where N is the number of data points. DBSCAN is listed in Algorithm 6.

Algorithm 7 DENCLUE Algorithm

---

1. Define the density functions of the data points
  2. Find the density attractors
  3. Associate each data point with a density attractor which would increase their density
  4. Define clusters such that each set of points associated to the same density attractor are considered one cluster
  5. Discard clusters whose density attractor has a low density (i.e. less than a threshold value)
  6. Combine clusters which are connected via a path of high density points
-

DENCLUE

DENsity based CLUstEring, DENCLUE (Hinneburg & Keim, 1998) models the density at any point in space in terms of individual influence functions, each defined around a specific data point. The overall density of the data at any point in space is defined as the sum of the influence functions of all data points, and the clusters can be determined mathematically as density attractors (local maxima of this overall density function). Determining the density-attractors is done using a hill-climbing procedure guided by the gradient of the overall density function. The first step in DENCLUE is to partition the data set into high-dimensional hypercubes to speed up the calculation of the density function needed in the second step. The next step is the actual clustering, where only the highly populated cubes (and cubes connected to them) are considered to estimate the density function for the data points, and then find the density-attractors (clusters). The attractors determine the actual clusters as the regions that have density greater than a certain threshold. DENCLUE can detect clusters of different sizes and shapes and can handle noise. However, DENCLUE shares some of the limitations of DBSCAN, more specifically it is sensitive to the parameter values, cannot handle clusters with different densities, and has high complexity: $O(N \log m + m^2)$ where N is the number of data points and m is the number of the highly populated cubes. DENCLUE is listed in Algorithm 7.

2.1.4 Grid-based algorithms

Another way to deal with the clustering problem is by inheriting the topology from the underlying attribute space, and shifting out attention to space partitioning rather than data partitioning, hence grid-based clustering algorithms are sometimes called *spatial* clustering algorithms.

STING

The Statistical Information Grid approach to spatial data mining, STING (Wang *et al.*, 1997) constructs data summaries into rectangular cells and forms a hierarchical tree. Each cell contains statistical information of the points (and child cells) that comprises: number of data points, minimum, maximum, mean, standard deviation, and type of distribution. After constructing the tree, the actual clustering is done in an SQL-style query applied on the tree. The 'WHERE' section of the clause specifies what conditions the data has to meet. STING starts with the root of the tree and descends one layer at a time. At each layer, it finds all the cells that are relevant to the query with some confidence. For all the relevant cells, it proceeds to their children and does the same process again until the leaves are reached. The clusters are then formed as the regions which are relevant to the query. STING has low complexity, $O(K)$ where K is the number of grid cells at the lowest level, and is easy to parallelize. However, it can only detect horizontal or vertical cluster boundaries and not those that are diagonal.

WaveCluster

(Sheikholeslami *et al.*, 2000) proposed WaveCluster, a grid-based clustering algorithm based on the wavelet transformation used in signal processing. The authors consider the multidimensional spatial data as a multidimensional signal and they apply wavelet transformations to convert the data in the frequency domain. Convolution of the data in the frequency domain with an appropriate kernel function results in detecting the dense regions which form the clusters. WaveCluster can detect clusters with arbitrary shapes, can handle outliers, and provides a multi-resolution view of the data which is an embedded property in wavelet transformations. However, WaveCluster can only be applied on low dimensional data.

2.1.5 Clustering High Dimensional Data

Many real applications nowadays generate data with a large number of attributes (i.e. high dimensional data), which imposes harder challenges on the clustering algorithms. Most high-dimensional data are generated from Power Law distribution (Zipf distribution) and can therefore be represented as a sparse data matrix A , where every row i represents a data object, every column j represents an attribute and every entry (i, j) represents the weight of attribute j in the data object i . For example, a set of documents can be represented as a sparse matrix where each row i represents one document and each column j represents one word (of all the possible words in all documents) and the entry (i, j) is the frequency of the word j in the document i . In this light, it can be seen that most of the entries would be empty, hence the matrix is sparse. This sparsity means that typically, only a few attributes are relevant or present in any given data record, resulting in non-zero attribute values, while the others (the majority) are not, resulting in zero-values. Such sparse data are therefore characterized by very low density or sparse data matrices.

2.1.5.1 Curse of Dimensionality

Clustering in very high dimensional spaces can present tremendous difficulties. First, under any definition of similarity, the presence of irrelevant attributes can eliminate any hope of clustering tendency, since trying to detect clusters where there are no clusters is useless. Second, high dimensional data causes a lack of data separation which is known as the curse of dimensionality (E.Bellman, 1961). Most clustering techniques depend critically on the measure of distance or similarity, since the definition of a cluster is a set points that are more similar to each other than to data points in other clusters. However, with the presence of too many attributes, the distance to the nearest neighbor becomes indistinguishable from the distance to the majority of the points. In fact, the smallest and largest distances become very close (Tan *et al.*, 2005). Third, for most of the clustering algorithms, finding the similarity matrix for a high dimensional data would increase the complexity tremendously. And finally, visualizing the data in low dimensional datasets helps in gaining a feel of the data, however visualizing the data accurately becomes impossible as dimensionality increases.

2.1.5.2 Dimensionality Reduction using Singular Value Decomposition and Probabilistic Latent Semantic Analysis

One way to deal with high dimensionality is by using fewer but more relevant dimensions that comprise most of the information in the dataset. Singular Value Decomposition (SVD) (Golub & Kahan, 1965) has been widely used to approximate (i.e. reduce dimensions) the sparse matrix A of size $m \times n$ into the product of three matrices $A = U\Sigma V^t$, where U is the $m \times m$ orthogonal matrix of left-singular vectors and its columns are known as principal components, Σ is the $m \times n$ diagonal matrix of positive singular values arranged in decreasing order of their magnitude, and V is the $n \times n$ orthogonal matrix of right-singular vectors.

Principal Component Analysis (PCA) (Pearson, 1901; Shlens, 2005) can also be used to reduce the dimensionality, as well as finding the most important dimensions. The first step in PCA is to make the mean of the data equal to zero by subtracting the mean from each dimension, then the eigenvectors and eigenvalues of the covariance matrix are calculated. Then the eigenvectors with the highest eigenvalues are chosen as the most relevant factors (principal components), and the number of the chosen eigenvectors represents the number of dimensions kept. Multiplying the principal components by the data matrix results in a new matrix solely in terms of the principal components. Since the principal components represent the most influential vectors in the data set, they can be considered as clusters. However, those clusters are hard to interpret.

SVD has also been used in a popular method in text mining known as Latent Semantic Indexing or Analysis (LSI or LSA) (Dumais *et al.*, 1995). LSI has a probabilistic-based analogue known as Probabilistic Latent Semantic Analysis (Hofmann, 1999) (PLSI), which is also known as the Aspect Model, and is used for Topic Modeling (discussed in Section 2.6) and dimensionality reduction. PLSI is very similar to LSI in that it computes a factorization of the matrix. However, instead of factoring the original term-document data matrix as in LSI, PLSI factors a matrix of probabilities into probability matrices.

The only difference between PCA and LSI is that while PCA performs the SVD to factor the covariance matrix of the centroid data, i.e. $A = (X - \bar{X})(X - \bar{X})^t$, the LSI performs the SVD of the original matrix X .

2.1.6 Cluster Validation

Cluster validation (Halkidi *et al.*, 2002a,b; Tan *et al.*, 2005) is an important step in knowledge discovery, since it can assess the quality of the clustering model output of a clustering algorithm. Cluster validation deals with several important issues; first, it tries to determine the clustering tendency of the data (i.e. whether the data follows some non-random structure, such as groups, or it is completely random) since any clustering algorithm will find some clusters in any set of data. Validation can be done by using some statistical tests for spatial randomness, such as the Hopkins Statistic (Banerjee & Dave, 2004). Another way to visually detect clustering tendency (and approximating the number of real clusters) was presented in (Hathaway *et al.*, 2006) where they ordered the similarity matrix using an algorithm related to Prim's algorithm for the minimal spanning tree. The second validation issue is concerned with determining the correct number of clusters present in the data, which would help tremendously in most of the clustering algorithms. This can be done by plotting the values of validation metrics such as the silhouette index (Rousseeuw, 1987) across different numbers of clusters, and look for a knee, peak, or dip that might indicate the number of natural clusters. However, this approach does not work all the time especially when the clusters are nested or overlapping. The third validation issue is concerned with evaluating how well the clustering model fits the data using two kinds of metrics: *(i)* internal validity metrics that depend only on the data and the clustering model, and *(ii)* external validity metrics which compare the clustering model against external information (e.g. class labels). The final cluster validation issue is to compare two or more clustering models, using the internal and/or external metrics, and determine which model better.

2.1.6.1 Internal Cluster Validity Metrics

These are measures of the goodness of the clustering model based only on the data and the clustering model without the use of any external information, hence they are sometimes called unsupervised metrics. Each of the internal measures reflects the cluster cohesion, separation or some combination of these quantities. The cluster cohesion (compactness or tightness) determines how similar the data objects in the same cluster are, while the cluster separation (isolation) determines how well-separated a cluster is from the other clusters. Usually, a good clustering model should show high

cohesion (i.e. data objects in the same cluster are similar) and high separation (i.e. clusters are well-separated). Some of the internal metrics are discussed below:

- **Sum of Squared Errors (SSE)** (Mood, 1950): SSE measures the average cohesion of all the clusters, and is equal to the average distance between each data point and its cluster centroid, hence a low value is desirable. SSE is defined in (2.8), where m_i is the size, in number of points, of cluster c_i , K is the number of clusters, and n is the total number of data points. Because this metric tends to decrease with the number of clusters, K , it tends to be biased by K .

$$SSE = \frac{1}{n} \sum_{i=1}^K \sum_{x \in c_i} m_i \text{dist}(c_i, x)^2 \quad (2.8)$$

- **Sum of Squared Errors Between Clusters (SSB)** (Mood, 1950): SSB measures the separation between the clusters, and is equal to the sum of the squared distances of each cluster centroid c_i to the overall data mean c . The higher the value of SSB, the more separated the clusters are. SSB is defined in (2.9).

$$SSB = \sum_{i=1}^K m_i \text{dist}(c_i, c)^2 \quad (2.9)$$

- **Davies-Bouldin Index** (Davies & Bouldin, 1979): DB-index is the ratio of the sum of within-cluster scatter (SSE) or cohesion to the between-cluster separation (i.e. distance between centroids) as shown in (2.10). Hence a lower value is desirable since it means that the within-cluster scatter is small (i.e. clusters are compact) while the clusters are well separated from each other.

$$DB = \frac{1}{K} \sum_{i=1}^K \max_{i \neq j} \left\{ \frac{SSE(c_i) + SSE(c_j)}{\text{dist}(c_i, c_j)} \right\} \quad (2.10)$$

- **Silhouette Index** (Rousseeuw, 1987): The Silhouette coefficient for the i^{th} data point is defined in (2.11), where $a(x_i)$ is the average distance of the i^{th} data point to all other data points in the *same* cluster, and $b(x_i)$ is the minimum of the average distances between the i^{th} data point and all data points in the *other* clusters. The Silhouette coefficient value for a data point is between -1 and 1, where a high positive value means that the data point was well clustered, whereas a zero value means that the data point lies at an equal distance from two or more clusters, and so could be assigned to any of them, and a low negative value means that it was

misclassified. The overall reported Silhouette index is obtained by averaging the coefficients over all data points. Moreover, the silhouette index can be examined visually by sorting the data points according to their cluster and then plotting the silhouette coefficients for all the data points.

$$S(x_i) = \frac{b(x_i) - a(x_i)}{\max\{b(x_i), a(x_i)\}} \quad (2.11)$$

- **Similarity Matrix** (Ling, 1973): The similarity matrix is found by computing the similarity between every two data points, then ordering the similarity values with respect to the cluster labels, and inspecting the resulting matrix visually. If high similarities are mapped to a darker gray level color, then in an ideal clustering result, we would expect to see darker blocks along the diagonal of the matrix (high intra-cluster similarity values), and lighter blocks off the diagonal corresponding to inter-cluster similarities (low values).

2.1.6.2 External Cluster Validity Metrics

External validity metrics compare the clustering model against ideal external information (ground truth). The purpose of these metrics is to evaluate the extent to which a manual classification (ground truth) process can be automatically discovered in an unsupervised manner using clustering. We will consider three kinds of external validity metrics: classification oriented, similarity oriented and clustering model oriented.

Classification Oriented External Cluster Validity Metrics

These validity metrics are commonly used to measure the degree to which a cluster contains data objects from the same class.

- **Contingency Table** (Pearson, 1904): For each cluster, $Cluster_i$, we find the number of data points that belong to each class $Class_j$, n_{ij} , where n_i is the number of points in $Cluster_i$ and n_j is the number of points in $Class_j$. This table is not directly used to evaluate the clustering model, however, most of the external evaluation metrics are extracted from it. A sample contingency table is shown in Table 2.1.

- **Precision:** For cluster i ($Cluster_i$) and class j ($Class_j$), precision is the probability that a data point from cluster i belongs to class j as $p_{ij} = \frac{n_{ij}}{n_i}$, which can be computed from the contingency table.
- **Entropy** (Ihara, 1993): Using the precision, p_{ij} , the entropy of cluster i is equal to, $E_i = -\sum_{j=1}^L p_{ij} \log_2 p_{ij}$ where L is the number of classes. A low entropy means that the data points in the same cluster belong to the same class. The total entropy is the sum of the entropies of all the clusters weighted by the size of each cluster, $E_{clusters} = \sum_{i=1}^K \frac{n_i}{N} E_i$ where N is the total number of data points and K is the number of clusters.
- **Purity** (van Rijsbergen, 1977): The purity of cluster i is given by $P_i = \max_j p_{ij}$, and the total purity is equal to $P = \sum_{i=1}^K \frac{n_i}{N} P_i$. A high purity means that the cluster is pure (i.e contains objects from the same class).
- **Recall** (van Rijsbergen, 1977): Measures the extent to which a cluster i contains data objects from a specific class j as follows:

$$recall(i, j) = \frac{n_{ij}}{n_j} \quad (2.12)$$

- **F-measure** (van Rijsbergen, 1977): Measures the extent to which a cluster i contains only data points from class j by combining the recall and precision as follows:

$$F(i, j) = \frac{2 \times precision(i, j) \times recall(i, j)}{precision(i, j) + recall(i, j)} \quad (2.13)$$

- **Normalized Mutual Information** (Coombs *et al.*, 1970): Measures the information shared by cluster $Cluster_i$ and class $Class_j$. If $Cluster_i$ and $Class_j$ are completely independent, then their mutual information is zero (i.e. knowing $Cluster_i$ does not give any information about $Class_j$). On the other extreme, if they are identical, then their mutual information is equal to one. The mutual information I is given by:

$$I = \sum_{i=1}^K \sum_{j=1}^L p_{ij} \log \left(\frac{p_{ij}}{p_i p_j} \right) \quad (2.14)$$

$$I = \sum_{i=1}^K \sum_{j=1}^L \frac{n_{i,j}}{N} \log \frac{N \times n_{i,j}}{n_i \times n_j}$$

where N is the total number of points. The mutual information is normalized by the average

Table 2.1: Sample Contingency Table

	Class 1	Class 2	...	Class L
Cluster 1	$n_{1,1}$	$n_{1,2}$		
Cluster 2	$n_{2,1}$	$n_{2,2}$		
$\vdots$				
Cluster K				$n_{K,L}$

of the entropy of the clusters ($E_{clusters}$) and the classes ($E_{classes}$) as follows:

$$\begin{aligned}
 NMI &= \frac{I}{[E_{clusters} + E_{classes}]/2} \\
 E_{clusters} &= \sum_{i=1}^K \frac{n_i}{N} E_i \\
 E_{classes} &= \sum_{j=1}^L \frac{n_j}{N} E_j
 \end{aligned} \tag{2.15}$$

Similarity Oriented External Metrics

These external metrics measure the extent to which two data points in the same cluster should belong to the same class, by using two similarity matrices: the cluster similarity matrix which contains 1 in the $(i, j)^{th}$ cell if data points i and j belong to the same cluster and 0 otherwise, and the class similarity matrix which contains 1 in the $(i, j)^{th}$ cell if data points i and j belong to the same class and 0 otherwise. From those two matrices the Table 2.2 is formed, then four quantities are computed as:

f_{00} = number of pairs of data points that have different class labels and are in different clusters

f_{01} = number of pairs of data points that have different class labels and are in the same cluster

f_{10} = number of pairs of data points that have the same class label and are in different clusters

f_{11} = number of pairs of data points that have the same class label and are in the same cluster

Using these quantities, two matching coefficients are calculated:

- **Rand Index** (Rand, 1971): measures the fraction of the total number of pairs that are either in the same cluster and in the same class, or in different clusters and in different classes. Its value lies between 0 and 1, and a value close to 1 indicates high agreement between the clustering model and the classes. The Rand coefficient is defined as follows,

$$Rand\ Index = \frac{f_{00} + f_{11}}{f_{00} + f_{01} + f_{10} + f_{11}} \tag{2.16}$$

Table 2.2: Similarity Contingency Table

	different cluster	same cluster
different class	f_{00}	f_{01}
same class	f_{10}	f_{11}

- **Jaccard Index** (Jaccard, 1901): measures the proportion of data point pairs that are in the same cluster and in the same class to those that are either in the same cluster or in the same class. Its value lies between 0 and 1, and a value close to 1 indicates high agreement between the clustering model and the classes. The Jaccard coefficient is defined as follows,

$$Jaccard\ Index = \frac{f_{11}}{f_{01} + f_{10} + f_{11}} \quad (2.17)$$

Clustering Model Oriented External Metrics

This set of metrics compares the detected cluster properties (e.g. centroid) against the groundtruth cluster properties when available, and they reflect the accuracy of the estimated cluster properties. Some of these metrics are defined below:

- **The Relative Difference in Number of Clusters:** is the difference between the number of clusters found and the actual number of clusters divided by the number of true clusters in the ground truth.
- **Average of Scale Errors:** the average difference between the found cluster scales and the actual scales of corresponding clusters.
- **Average of Centroid Errors:** the average difference between the found cluster centroids and the actual centroids of corresponding clusters.
- **Difference in Estimated Noise:** the difference in the percentage of detected noise compared to the actual noise percentage.

2.2 Stream Data Mining

In recent years, Stream Data Mining (Babcock *et al.*, 2002; Guha *et al.*, 2003; Stonebraker *et al.*, 2005; Gaber *et al.*, 2005) has emerged as a new discipline in knowledge discovery due to the in-

creasing number of applications that generate continuous streams of data and increasing awareness about the importance of utilizing such data. For example, the activity on a network generates massive amounts of packets of data, and applying data mining techniques on such raw data could help distinguish between valid and invalid activity. More specifically, stream data mining is concerned with extracting knowledge represented as a set of models in a continuous stream of data, and it is a much more challenging task due to the constraints imposed by the nature of data streams. This work is concerned only with Unsupervised Stream Mining, hence, we will focus our discussion on unsupervised algorithms.

We present some of the issues concerning the data stream model and the requirements needed to deal with these issues in Section 2.2.1, then we analyze some of the existing attempts in clustering data streams in Section 2.2.2, and finally discuss the issues concerned with evaluating the clustering model of a data stream in Section 2.2.3.

2.2.1 The Data Stream Model

In the data stream model, the data arrives continuously with a huge throughput. This model differs from other conventional data models in several ways:

- The huge data throughput makes it hard if not impossible to store every single data point and process it, therefore each new data point should be processed, *on the fly*, only once.
- There is no control or expectation on the order of the data arrival (for example, data points from the same cluster do not have to arrive sequentially one after the other).
- Data streams are unbounded in size, thus imposing severe restrictions on storage capacity.
- Outlier detection is more challenging, since a data point that is flagged as an outlier at the beginning of the data stream might turn out to be part of a cluster that emerges later in the data stream lifetime.
- Validating the clustering model is hard since the input data points are not kept, and only a summary of the data may be maintained. And even if data points were stored, the data stream is evolving, hence data points might be assigned to different clusters at different time periods.

- There is no notion of a static cluster model or output. Instead, at any given time or point in the data stream, there is a different cluster model. Hence, which cluster model should be reported?

To meet the challenges of mining data streams, stream data clustering algorithms should poss the following characteristics (Barbará, 2002; Stonebraker *et al.*, 2005):

1. **Compactness of representation:** The descriptions of the clusters discovered should be compact, since new data is continuously available - in real time - which would exhaust the main memory resources. So the cluster representation should be compact and should not grow appreciably with the number of data points processed.
2. **Fast processing of new data points :** The online nature of stream data mining requires that processing new data points should be fast and incremental. Moreover, the function that finds which cluster or clusters the new point belongs to, should not require a comparison with all data points processed in the past (since they do not exist anymore), and it should exhibit good complexity. Trying to store the data would add an unnecessary latency, which is neither acceptable nor necessary in many stream processing applications.
3. **Fast handling of stream outliers:** The outliers are new data points that do not fit well under the clustering model used so far. Dealing with outliers is application dependent. If a sufficient number of outliers is found, we may want to abandon old clusters for new ones (e.g. in weather data), or the boundaries of current clusters may need to be redefined (e.g. the case when outliers indicate that a disease had spread in new areas).
4. **Integration of offline and online data:** The algorithm should be able to efficiently store, access and modify information about the state of the data stream encountered so far, and combine this information with the streaming data.
5. **Presenting the discovered clusters instantly:** The algorithm should be able to present or visualize the clusters discovered so far in an optimized and fast manner.
6. **Making no assumption on the number of clusters:** The number of clusters in a data stream are often unknown, and this number keeps changing as the data evolves.

7. **Handling evolution:** A typical data stream evolves and is infinite by definition. Hence, the clustering model should give more emphasis on the newer data points and gradually *forget* older ones.

As a matter of fact, those requirements can be summarized as follows:

1. “keep it short”
2. “keep it moving”
3. “keep it tidy”
4. “keep it current”

2.2.2 Unsupervised Learning in Data Streams

Several clustering algorithms have been designed to deal with the challenging nature of data streams by incrementally updating the clustering model or by processing data points in small batches. These algorithms can be grouped into two groups based on the assumptions made by each method: (i) *one-pass*, and (ii) *evolving* methods (Cao *et al.*, 2006).

2.2.2.1 One-pass methods

One-pass methods assume that the data stream follows only one model throughout its lifetime, and hence they cannot handle evolving data distributions which is one of the main characteristics of data streams (i.e. they fail to comply with the 7th requirement in Section 2.2.1). These algorithms usually adopt a divide-and-conquer strategy, where they apply a local search routine every time a new chunk of data, with a predefined size, is available to generate the cluster centers of that chunk. Then an offline routine is performed, at the user’s request, that generates a new set of centers based on the data chunk centers. The one-pass family of clustering algorithms includes mainly the first generation of *scalable* clustering algorithms, summarized below.

BIRCH

A classical one-pass algorithm is BIRCH (Zhang *et al.*, 1996). BIRCH’s distinctive properties can be summarized in that: (i) it makes the time and memory constraints explicit in the clustering

problem, (ii) it is local in that it does not scan all the data points to make each clustering decision, (iii) it uses the notion of outliers (i.e. not every data point is equally important for clustering), (iv) it uses all memory resources to ensure accuracy while being efficient in terms of I/O, and finally (v) it is an incremental method. BIRCH utilizes a set of statistical measures to capture the closeness of data points, and updates these measures incrementally. These measures include cluster properties (centroid, radius and diameter) and distance measurements between clusters (centroid Euclidean distance, centroid Manhattan distance, average inter-cluster distance, average intra-cluster distance, and variance increase distance).

BIRCH introduced two important concepts: Clustering Feature (CF) and CF tree.

1. A Clustering Feature is a triple that summarizes the cluster: $CF(N, LS, SS)$ where N is the number of data points in the cluster, LS is the linear sum of the data points, and SS is the square sum of the data points in that cluster.
2. The CF tree is a height-balanced tree where: each non-leaf node has at most B (branching factor) entries of the format $[CF_i, child_i]$, and each leaf node contains at most L entries in the form $[CF_i]$ each of which must satisfy the threshold requirement which is that the cluster diameter has to be less than T (the diameter is the radius of the leaf nodes). Moreover, each leaf node has two pointers “prev” and “next” to chain all leaf nodes together.

BIRCH consists of four main phases:

1. The first phase is mandatory and sufficient for good clustering, and the rest are for improving the clustering results. Phase 1 scans the data once and builds an initial CF tree using the available memory and removes all the outliers (using the threshold value T).
2. Phase 2 rebuilds a smaller CF tree while removing more outliers and grouping crowded sub-clusters into larger ones. Although this phase is optional, it is needed if phases 3 and 4 are to be executed.
3. In Phase 3, global clustering methods, such as K-means, are used to generate better clusters. These methods need different input size than Phase 1’s output, so this is why Phase 2 is needed.

4. Phase 4 is needed to further refine the clusters, and is done by using the centroids of the clusters as seeds and redistributing the data points to the closest seed which might result in new clusters.

Clustering using n^ϵ memory

The authors in (Guha *et al.*, 2000) analyzed a set of algorithms to cluster data streams with a guarantee of a constant-factor approximation using n^ϵ memory where n is the size of the data stream and $\epsilon < 1$. The optimal solution is the one that minimizes the sum of squared distances measure (SSE), which is similar to the objective function in K-Means (MacQueen, 1967). By minimizing SSE, the algorithm tends to find hyper-spheres as clusters, and hence cannot detect arbitrary-shaped clusters. The data stream is processed in batches, and for each batch, it generates a set of weighted cluster centers. Then the set of cluster centers for all batches are clustered. The clustering in each batch and at the end of all batches is done using some local-search algorithm that runs in asymptotic linear time. However, it is expected to fail to handle or detect outliers because it is based on minimizing the sum of squared distances which is very sensitive to extreme outliers (Huber, 1981).

Randomized k-Median

(Charikar *et al.*, 2003) presented a randomized streaming algorithm for the k-Median problem, where clustering is done in phases, such that in each phase, a further segment of the stream is processed. In each phase, a subroutine is called to find the medians on a modified version of the data stream, which consists of the medians found from the previous phase, concatenated with the new batch of points. The outliers are detected by excluding, with some probability, a constant percentage of the data points which are the furthest from their closest median.

Clustering with fractals

(Barbará & Chen, 2000) proposed the use of fractals to cluster large datasets incrementally. Fractals are structures that consist of parts that are self-similar (each part is approximately a reduced-size copy of the whole). For a set of n points with d -dimensions, the space is divided in grid cells of size r , and a set of grid cells represents a fractal. Initialization is done by processing a sample of the data, and creating clusters using points that are close to each other (i.e. for which, the distance is less

than a threshold). Then every new data point is tentatively added to each of the clusters and the new fractal dimension is computed. Only the cluster that showed the minimal fractal impact (i.e. change in fractal dimension) is considered to be compatible with the new point, and if that impact is greater than a threshold value, then the point is considered an outlier, otherwise it is added to the cluster. Since the number of clusters might (and most likely will) change over time, they considered only splitting the current clusters. A good indication that a cluster may need reshaping is given by how much the fractal dimension of the cluster has changed since its inception during the initialization step. A large change may indicate that the points inside the cluster do not belong together. To split a cluster, a sample of the points in the cluster go through the initialization step again to find out the number of clusters needed to represent this cluster. Then the points in the original cluster go through the incremental step to be assigned to this new set of clusters.

Gaussian Model Generator

Another one-pass clustering algorithm was proposed in (Karkkainen & Franti, 2007), where the proposed algorithm generated a Gaussian mixture model from a large dataset. The algorithm consists of two steps: *(i)* generating a cluster model by one pass over the data (online step), and *(ii)* simplifying this model as a post-processing (offline) step. Generating the model is done by comparing each new data point with the current cluster model, and if it fits (with quality that is higher than an acceptance threshold) then using this data to update the model, otherwise the data point is added it to a fixed-size buffer. Updating the model is done by updating the weight, mean and covariance matrix of each component in proportion to the new point. Once the buffer is full, all the data points in the buffer are compared again against the model, and the ones that fit are used to update the model. This is done because the acceptance threshold value is continuously updated, and thus improving, as new components are created. The rest of the data points in the buffer, that still do not fit the model, are clustered into new K clusters by minimizing the distance between the points in the same cluster. The Gaussian Model Generator is listed in Algorithm 8. A Large buffer produces more compact components, while a small buffer and a large K may produce components that join two clusters. Since the model obtained after the first step, tends to over-fit the data, a post-processing step is applied on the model. This step is simply done by applying an adapted clustering algorithm (they used a modified K-means algorithm) on the model to get the final clusters. Since the input to

---

Algorithm 8 Gaussian Model Generator

Initialize the data buffer and create an empty model
WHILE the data buffer is not empty
    Read a data point
    IF the point fits into the model
        Update the model
    ELSE
        Add point to the buffer
    IF the buffer is full or no data is left in the input stream
        IF some buffer points fit into the model
            Update the model
        ELSE
            Create K-clusters of the remaining points
END WHILE

---

this last step is small (only the model resulting from the previous steps), the procedure is expected to be fast.

2.2.2.2 Evolving methods

The second family of stream clustering algorithms, known as evolving methods, view the data stream as evolving over time, where new clusters might emerge and old ones might fade. These methods introduce a forgetting factor on the data points based on their arrival time, thus giving more importance to newer data points.

TECNO-Streams

TECNO-Streams (Nasraoui *et al.*, 2003) is a stream clustering algorithm that works in one pass and under restricted space limits, by continuously computing a limited-size synopsis of cluster representatives that can serve as an evolving summary of the data stream. Inspired by the dynamics of learning in the immune system, the input data is seen as foreign agents, carrying antigens such as viruses or microbes, while clusters represent B-cells that adapt to recognize previously encountered antigens. The algorithm relies on incremental processing of the input data for learning an evolving model of the clusters in a data stream. Each data point x_j is assigned a weight with respect to each cluster C_i at time J , $w_{ij,J}$, where the time reflects the order of the point's arrival. This weight decays over time with a forgetting factor of $e^{-\frac{1}{\tau}}$, where τ is a constant. The algorithm tries to evolve the optimal centroids (c_i) by evolutionary immune operators, in particular high rates of mutation and

aggressive cloning each time that a recognized antigen is encountered. As in the immune system, cloning is set to depend on the strength of affinity between the clusters (B-cells) and the incoming data (antigens). One particular feature of the immune system based cluster model is that it is structured as a network or graph, where clusters continue to excite each other into survival even after the antigens have disappeared, thus maintaining a slowly decaying memory of the past data stream. TECNO-STREAMS adheres to all the requirements of clustering data streams: compactness of representation, fast incremental processing of new data points, and clear and fast identification of outliers

TRAC-Streams

In (Nasraoui & Rojas, 2006), an algorithm for clustering data streams, known as TRAC-Streams, was presented. It relied on incremental processing and robust statistics for learning an evolving model of the clusters in a data stream. Each data point x_j is assigned a weight with respect to each cluster C_i at time J , $w_{ij,J}$, where the time reflects the order of the point's arrival. This weight decays over time with a factor of $e^{-\frac{1}{\tau}}$, where τ is a constant. The algorithm tries to find the optimal centroids (c_i) and scales (σ_i) that minimize the following objective function, which is a modified incremental version of an earlier batch algorithm known the Maximal Density Estimator (Nasraoui & Krishnapuram, 1996), :

$$\min_{c_{i,J}, \sigma_{i,J}} \left\{ \Gamma_{i,J} = \sum_{j=1}^J w_{ij,J} \frac{d_{ij}^2}{\sigma_{i,J}^2} - \alpha \sum_{j=1}^J w_{ij,J} \right\}, i = 1, \dots, C, \quad (2.18)$$

where $c_{i,J}$ and $\sigma_{i,J}$ are the centroid and scale of the cluster C_i at time J respectively, and d_{ij} is the Euclidean distance between the data point x_j and the cluster C_i , and α is a constant. The first term tries to minimize the scaled distances of the good points to the optimal cluster locations to achieve robustness, while the second term tries to maximize a soft estimate of the cluster cardinality (sum of weights) which ensures that as many good points as possible are used in the estimation process. However, the constant factor α is used to balance the two terms, and has to be chosen carefully, generally depending on the data's dimensionality.

With the arrival of a new point, TRAC-Streams finds the point's distance with respect to all existing clusters, then all the clusters to which the point belongs (based on Chebyshev bounds) are

---

Algorithm 9 TRAC-Streams

WHILE data points still exist

    Read a data point x_j

    Find the distances (d_{ij}^2) and weights (w_{ij}) of x_j with respect to all existing clusters

    Update all clusters in which x_j is not an outlier

    IF x_j is an outlier with respect to all clusters

        Create a new cluster with x_j as its centroid

    Remove clusters whose density is less than a threshold

    Merge compatible clusters

END WHILE

---

updated. If the point is found to be an outlier with respect to all existing clusters, then a new cluster is created. Moreover, a test on the quality of the clusters is used to ensure that only clusters with a density ($\frac{\sum_{j=1}^n w_{ij}}{\sigma_i^2}$) higher than a threshold value, are kept. TRAC-Streams tests the compatibility between clusters, and merges clusters whose compatibility is less than a threshold (based on Chebyshev bounds). TRAC-Streams is listed in Algorithm 9.

Stream Fuzzy K-means

The streaming variant of the Fuzzy K-means (Hore *et al.*, 2007) presented a study of the trade-off between responding to an evolving distribution and summarizing the data seen so far by varying past history usage in clustering streaming data. The algorithm works by clustering n data points that are present at time instance t_i and represents them by K cluster centroids. Each centroid has a weight that is calculated as the sum of all the points' memberships in that cluster. In the next time instance t_{i+1} , only the centroids from the last (or previous p instances) will be used as initial centroids for clustering the new points. The number of previous time instances used determines how much history is retained, and this is application dependent.

Sequential Leader Algorithm

The Sequential Leader Clustering algorithm is a classical algorithm that was presented in (Hartigan, 1975). It is a straightforward adaptation of K-Means (MacQueen, 1967) to data stream clustering. It starts by reading a new data point from the data stream. Then it finds the winning cluster (i.e. the closest cluster), and if its distance is less than a threshold then the winning cluster is updated (i.e. its centroid is updated to include the new data point). Otherwise, a new cluster is created with the new data point being its centroid. The main difference is that new clusters could emerge if none of the

---

Algorithm 10 Growing K-Means

```
WHILE points still exists in data stream
  Read new data point
  Find closest cluster
  IF the distance to closest cluster < threshold
    Update the cluster parameters with the new data points using (2.19)
  ELSE
    Merge closest two clusters
    Make a new cluster whose center is the new data point
  END IF
END WHILE
```

---

existing clusters is close to the new data points.

Growing K-Means

The authors in (Lowette & Laerhoven, 2004) presented several variants of online clustering algorithms. One of them is the Growing K-Means which is an extension of the Sequential Leader Algorithm. The main difference is that the Growing K-Means constraints the maximum number of clusters, which ensures that only a predefined number of clusters are maintained. Both the Sequential Leader Algorithm and Growing K-Means were not originally designed for stream data. For example, they do not handle the existing of outliers or keep track of the evolution over time. However, they can be adapted to stream clustering with simple modifications.

First, before creating a new cluster, we merge the closest two clusters to keep the number of clusters limited to a maximum value. Second, we update the clusters' centroid and scale incrementally as follows:

$$\begin{aligned} c_{i,t+1} &= c_{i,t} + \frac{x_t - c_{i,t}}{|C_i| + 1} \\ \sigma_{i,t+1}^2 &= \frac{\text{distance}(c_{i,t}, x_t) + |C_i| * \sigma_{i,t}^2 - \text{distance}(c_{i,t+1}, x_t)^2 * (|C_i| + 1)}{|C_i| + 1} \\ |C_i| &= |C_i| + 1 \end{aligned} \tag{2.19}$$

where x_t is the new data point (i.e. at time t), $c_{i,t}$ and $\sigma_{i,t}$ are the previous centroid and scale of the winning cluster respectively, while $c_{i,t+1}$ and $\sigma_{i,t+1}$ are the updated centroid and scale of the winning cluster respectively, and $|C_i|$ is the cardinality of the winning cluster (i.e. the number of points that are represented by this cluster). Growing K-Means is listed in Algorithm 10.

CluStream

(Aggarwal *et al.*, 2003) presented a framework, called CluStream, that consists of online and offline components.

(i) Online Component: In the online component, the framework maintains BIRCH-type statistical information at a high level of (temporal and spatial) granularity about the data locality in terms of micro-clusters. For each micro-cluster i , it incrementally maintains the sum of the squares of the data values $\overline{CF2^x}$, the sum of the data values $\overline{CF1^x}$, the sum of the squares of the time stamps $CF2^t$, the sum of the time stamps $CF1^t$, and the number of points n . Micro-clusters are stored at particular moments in the stream (snapshots) and these moments are selected based on a pyramidal time frame. Each of the snapshots has a different order ranging from 1 to $\log(T)$, where T is the clock time elapsed since the beginning of the stream. This order defines the level of granularity in time at which the snapshots are maintained. When a new point arrives, it will be assigned to the closest micro-cluster if the point is in the clusters' maximal boundary factor, which is defined as a factor of a constant of the root mean square deviation of the data points in the micro-cluster from the centroid. If the data point does not belong to any of the micro clusters, then a new micro-cluster is created. To create a new cluster, one of the clusters should be deleted or merged with another cluster. A cluster is deleted if its recency is less than a threshold. The online process is listed in Algorithm 11.

(ii) Offline Component: The offline process in CluStream generates a set of macro-clusters, and it consists of first having the user pick a time-horizon and the number of macro-clusters needed, then the micro-clusters at the current time are subtracted from the micro-clusters at horizon offset of the current time (i.e. at the current time-horizon). These micro-clusters are then clustered using a variant of the K-means algorithm (MacQueen, 1967).

CluStream provides a straight-forward method to summarize data streams, however it maintains a constant number of micro-clusters, hence it might consider some outliers as real clusters, or even split some of the good clusters (since it uses a K-means variant to get the final results).

Algorithm 11 CluStream Online Process

Input: Number of micro-clusters, snapshot order constant (α), maximal boundary factor (t), recency threshold (δ)

Output: Micro-clusters represented using $\overline{CF2^x}$, $\overline{CF1^x}$, $CF2^t$, $CF1^t$, and the number of points n .

Initialize the clusters by running K-Means (MacQueen, 1967) on an initial batch of data points

WHILE there are points in the data stream

IF current time is divisible by α^i , where i is an integer and refer to the pyramidal time frame

        Store the clustering model

END

    Read a new data point x

IF x is within the maximum boundary of the closest clusters

        Add x to the closest cluster by updating the clusters' statistical information

ELSE

IF a cluster has a recency less than a threshold

            Delete this cluster

ELSE

            Merge the closest two clusters

END IF

    Create a new cluster using x

END IF

END WHILE

DenStream

Inspired by DBSCAN (Ester *et al.*, 1996) and CluStream (Aggarwal *et al.*, 2003), (Cao *et al.*, 2006) presented DenStream, an evolving clustering algorithm that makes no assumption on the number of clusters, discovers clusters with arbitrary shape, and can handle outliers. The evolution of the data stream is captured by assigning to each data point a weight function $f(t) = 2^{-\lambda \cdot t}$ where t is the current time and $\lambda > 0$ is a constant that reflects how much importance is given to more recent data.

DenStream consists of two components:

(i) an online component that incrementally maintains a set of potential-micro-clusters (p-micro-cluster) and outlier-micro-clusters (o-micro-cluster), and

(ii) an offline component that generates the higher-level clusters by applying DBSCAN (Ester *et al.*, 1996) on the set of p-micro-clusters maintained by the online component.

The structure of the p-micro-clusters and the o-micro-clusters is similar to the micro-clusters defined in (Aggarwal *et al.*, 2003). They consist of the sum of the squares of the data values $\overline{CF2^x}$, the sum of the data values $\overline{CF1^x}$ and the weight $w = \sum_{j=1}^n f(t - T_j)$, where T_j is the time-stamp of the j -th data point. The influence area around each micro-cluster is defined as the radius $r =$

Algorithm 12 DenStream

Initialize the clusters by running DBSCAN (Ester *et al.*, 1996) on an initial batch of data points

WHILE there are data points in the data stream

    Read a new data point x

IF adding x to the closets p-micro-cluster yields a radius (r) less than the threshold (ϵ)

        Add x to the closest p-micro-cluster by updating the clusters' statistical information

ELSEIF adding x to the closets o-micro-cluster yields a radius (r) less than the threshold (ϵ)

        Add x to the closest o-micro-cluster by updating the clusters' statistical information

IF the new weight of the updated o-micro-clusters is greater than $\beta\mu$

            Convert the updated o-micro-cluster to a p-micro-clusters

END IF

ELSE

        Create a new o-micro-cluster using x

END IF

IF the current time is a multiple of a constant value

        Perform the pruning strategy

END IF

END WHILE

$\sqrt{\frac{|CF2|}{w} - (\frac{|CF1|}{w})^2}$ and it is less than or equal to a threshold value ϵ . The only difference between p-micro-clusters and o-micro-clusters is the weight threshold value. For the p-micro-cluster, the weight $w \geq \beta \cdot \mu$, while for the o-micro-cluster, the weight $w < \beta \cdot \mu$.

DenStream algorithm starts with a set of initial p-micro-clusters generated using DBSCAN on the first segment of the data stream. Then, for each new data point p , it finds its nearest p-micro-cluster, and checks if the new radius (after trying to merge p) is less than ϵ , then merges p with this cluster. If p was not merged with any of the p-micro-clusters, then using the same process, the algorithm tries to merge it with its nearest o-micro-cluster. If it did not merge with any of the clusters, then a new o-micro-cluster is created using p . Moreover, a pruning strategy is adopted to bound the size of memory needed and to detect outliers. To reduce memory requirements, a p-micro-cluster is deleted if its weight is less than the threshold $\beta \cdot \mu$, and an o-micro-cluster is deleted if its weight is less than a threshold value called the lower limit of the weight, which depends on the current time and the cluster creation time. The pruning steps are crucial to detect real outliers. DenStream is listed in Algorithm 12.

D-Stream

In (Chen & Tu, 2007), the authors proposed a new grid-based and density-based clustering algorithm, called D-Stream. D-Stream has similar goals to DenStream (Cao *et al.*, 2006), which also does not require specifying the number of clusters in advance, detects clusters with arbitrary shapes, and can detect outliers. D-Stream consists of two components:

- (i) an online component which continuously reads new data points from the data stream and assigns them into the points' corresponding density grids, and
- (ii) an offline component that periodically adjusts the clusters.

Each of the spaces in the multidimensional space S is divided into P partitions, and a set of partitions on all spaces is called a density grid. Each data point x is assigned a density coefficient which decays over time: $D(x, t) = \lambda^{t-t_c}$, where t is the current time and t_c is the arrival time of x . The overall density of the density grid is the sum of the density coefficients for all the points that belong to that grid, and this density is used to check (against a threshold value) to determine if the grid is dense.

In D-Stream, each density grid has a characteristic vector which is updated periodically, that includes the last time the grid was updated, the last time the grid was removed as a sporadic grid (outlier), the overall density, and the label of the grid. Moreover, the grid is categorized as dense, sparse or transitional based on some threshold value that is user-determined. The clusters are created using a similarity measure between density grids. Two density grids are neighboring grids if there exists at least one dimension k such that the difference between the two partitions in the k -th dimension is equal to one and the rest of the partitions are the same in both grids. A sequence of neighboring grids results in a grid group, where they are considered as inside or outside grids. An inside grid is a grid that has neighboring grids in every dimension, otherwise it is an outside grid. A grid group G is a grid cluster if every inside grid in G is dense and every outside grid is either dense or transitional.

The offline component runs periodically, with the time period determined theoretically as the minimum between the minimum time needed for a dense grid to become sparse and the minimum time need for a sparse grid to become dense. The offline component removes sporadic grids using a threshold value to reduce memory requirements and increase speed; this threshold differentiates

between outliers (sporadic grids) and old grids that are simply not receiving new points and hence their density is decreasing with time. Moreover, the offline component updates the grids by checking whether a grid changes its type depending on its density.

Evolutionary Clustering

(Chakrabarti *et al.*, 2006) presented the evolutionary clustering framework, which performs clustering at different time periods by simultaneously optimizing two objectives:

- (i) the clustering at each time period should be of high quality (high snapshot quality),
- (ii) and the clustering should not dramatically change from the previous clustering (thus preferring a low history cost).

The authors argue that evolutionary clustering benefits include: consistency between clustering models at different time periods, robustness against noise due to the high quality and consistent clustering, a smooth view of the data stream evolution and the ability to put clusters within their historical context. The framework is generic and requires only an online clustering algorithm, and the authors presented the framework using K-Means and agglomerative hierarchical clustering algorithms. At each timestep t , the clustering algorithm produces a clustering model for all the objects seen so far by optimizing the trade-off between having a high snapshot quality and low history cost. The clustering is done using an inter-object similarity matrix that incorporates two kinds of similarities: the local similarity (such as the cosine similarity) and the temporal similarity (correlation between the number of occurrences of data objects at different timesteps).

Section 3.1.11 will compare the proposed stream clustering algorithm (RINO-Streams) against some of the evolving clustering algorithms presented above.

2.2.3 Evaluating Data Stream Clustering Results

Continuously validating the clustering model in live data streams is much more challenging compared to traditional clustering models for several reasons. First, as discussed in Section 2.2.1, the data points are viewed only once and then discarded, and the clustering model is just a summary of those data points. Hence, trying to use traditional validation metrics (e.g. purity) would be impossible. Second, the clustering model is evolving over time, so a cluster at time t_1 might shift to a new

location at time t_2 . And finally, the clustering model cannot be predicted because the data stream is infinite, hence no realistic ground truth can be created.

For these reasons, in this work, we will follow two approaches to evaluate a data stream:

- (i) we find the evaluation metrics at predefined periods of time, and
- (ii) we propose performing the validation hand in hand with detecting the clusters, and constructing behavioral profiles of clusters over time, that could help guide the clustering configuration as well as provide some form of online real-time validation.

2.3 Tracking Cluster Evolution

A typical use scenario of data mining tools is one where the application expert periodically invokes the data mining tool on the most recent data available to extract clusters. Each set of clusters provides new insights on the application domain, enriching the expert's domain knowledge. If this task is done periodically (e.g. data stream), then extracting the clusters each time from scratch becomes an expensive, and possibly redundant, operation. More specifically it suffers from the following drawbacks: (i) because data keeps growing at a fast pace, application experts are sometimes forced to use a sample of the data which risks losing some important clusters. (ii) Most data mining techniques return many models, and most of these models might be redundant (i.e. the application experts might have seen them before). As a result, more time is needed to find those that corresponds to interesting new models. Hence, as the expert becomes more familiar with the clusters extracted, he or she becomes more interested in the *changes* rather than just the clusters. These changes represent the evolution of clusters and add a more challenging but valuable perspective to data mining.

Furthermore, these changes might occur to the discovered clusters internally (i.e. the characteristics of the cluster) or externally (i.e. the clusters' interaction with each other). Tracking and monitoring those changes is more crucial in the context of stream data mining, since the structure within the data changes quickly due to the changes in the data content itself. For example, in customer relationship management, it may be valuable to know if an emerging cluster of user behavior is a completely new group of users, a shift in a previous group's behavior, or if it is the result of the merging of two groups of users that used to have different interests. Answering such a question can

provide valuable information for planning marketing and other business strategies.

In the literature, there has been some research that showed an increasing interest in the evolution of the clusters (Böttcher *et al.*, 2008). For small datasets, detecting the changes in clusters can be done by examination of the clusters by a domain expert, however, when the datasets are big and evolving quickly, as can be the case in data streams, it can become impractical or even impossible to compare clusters manually. Hence, it becomes crucial to develop tools that help in detecting and characterizing change automatically in an application independent manner.

In this section, we will present some of the change detection algorithms that have been proposed. We review the two main approaches, namely the local approach in Section 2.3.1, and the global approach in Section 2.3.2. We also provide a brief review of Topic Detection and Tracking in Section 2.3.3, which are related to tracking cluster evolution, however, they are specific to text data and have a much narrower goals than our proposed approach. .

2.3.1 The Local Cluster Tracking Approach

In the local change detection approach, the time axis is divided into a specific number of intervals of fixed width. Then clustering is performed at each time period. Finally, the mapping between clustering models at consecutive time periods is performed. The clusters at each time period can be found independently from each other (e.g. MONIC (Spiliopoulou *et al.*, 2006)), or by using some information from previous clusterings as heuristics to improve the clustering results of future clusterings (e.g. (Günemann *et al.*, 2011)). The mappings are usually many-to-many mappings (i.e. one or more clusters at time T_i can be mapped to one or more clusters at time T_{i+1}), and they are used to deduce the transitions that took place between consecutive time periods. The main difference between the algorithms in this approach is in the similarity measure used to quantify the changes between clusters at different time periods. The methods described below are examples of this local approach.

PAM

The most common way of automatically detecting changes in clusters is done by comparing the clusters obtained at different timestamps, measuring the changes, and then drawing conclusions based on these changes. PAM (Baron *et al.*, 2003) is an automated cluster monitor that can identify

changes in usage behavior. PAM focuses primarily on association rules, where they are modeled as temporal objects combined with some statistics. The first level of change detection is done on the cluster statistics from one time period to another, and it is done using a null hypothesis test, on the data obtained between the two time periods, to test if the value of a statistic is the same or if it has changed significantly. The second level of change detection is done by using some heuristics on the values of statistics starting at the time point at which the clusters emerged. For example, they proposed a heuristic to reflect the stability of the cluster based on the number of times the cluster is present. The last level of change detection is done on the actual content of the clusters to determine whether the cluster content is shared by more than one cluster, hence it needs to be removed to reduce the memory used.

MONIC

MONIC (Spiliopoulou *et al.*, 2006) is a framework that is built on PAM (Baron *et al.*, 2003) but is more generic. MONIC takes as an input an evolving data stream, finds the clusters (using some clustering algorithm) at consecutive time points and monitors their evolution using a categorization of cluster transitions. MONIC assumes re-clustering rather than cluster adaptation or incremental clustering at each time period, and it separates the cluster construction from the aging function that assigns weights to the data points. To detect a cluster transition/change, MONIC compares the clusters found at two time periods and determines if there is an overlap or match between every pair of clusters from the two time periods. For the cluster $C_{i,t1}$ detected at time $t1$ and cluster $C_{j,t2}$ detected at $t2$, the overlap is the normalized sum of weights of the data points in their set intersection, and they are considered to be a match if that overlap is the maximum and is larger than a threshold value. MONIC introduced a transition detection algorithm that uses the overlap and match between clusters at different time periods to categorize those transitions into: *internal changes* (changes in cluster size, compactness or location) and *external changes* (the cluster survived, was split, absorbed, disappeared or emerged). It also calculated some statistics about those changes, such as the number of timepoints during which a cluster has survived (lifetime of the cluster), and the survival ratio of a clustering as a portion of its clusters that survived.

MONIC introduced a simple and generic framework to detect internal and external changes to clusters. However, similar to PAM (Baron *et al.*, 2003), it requires some restrictions on the

underlying clustering algorithm used; each batch of data is clustered from scratch which increases the complexity and memory requirements especially in stream data mining. It also assumes that the outliers had been removed by a pre-processing step, which is a strict assumption since the outliers detected at time t_i might be the beginning of a new cluster at time t_{i+1} . Moreover, MONIC is only concerned with the changes between the clustering models at two different time periods, and not the changes that took place over many time periods.

Tracking Evolving Subspace Clusters

The authors in (Günemann *et al.*, 2011) proposed a generic framework for tracing evolving subspace clusters in data streams. This framework finds subspace clusters at different time periods, and then finds the mapping graph between clusters at consecutive time periods. The mappings are many-to-many (i.e. one or more clusters can be mapped to one or more clusters), and are determined using a novel distance function. The temporal transitions are categorized using the mapping graph as follows: a cluster disappears if its outdegree is 0, a cluster appears if its indegree is 0, a cluster diverges if its outdegree >1 , and different clusters converge to a single cluster if the indegree >1 .

For the clusters C_t and C_{t+1} at consecutive time periods t and $t+1$, the distance function is based on the shared subspace between the clusters as well as the values of those dimensions (i.e. it does not depend on the data points identities). The subspace distance is found using a trade-off between the number of shared dimensions and the number of new dimensions in cluster C_{t+1} . The statistical distance is found by first finding the optimal core dimensions among the shared dimensions between C_t and C_{t+1} , and then representing each dimension by a normal distribution and finding the distance between the distributions using the Kullback-Leibler divergence. Furthermore, they penalize the dimensions that are not included in the core. The final distance function is a combination between the subspace distance as well as the core-based value distance.

FOCUS

Other frameworks for detecting changes in clusters include FOCUS (Ganti *et al.*, 1998), which quantifies the difference, called *deviation*, between two datasets based on the models they induce. FOCUS provides a generic framework for measuring changes that they applied on decision trees, clustering and frequent itemsets. To find the deviation between datasets, they first find the models for

each one; the models consist of the structural component (e.g. clusters) and the measure component (e.g. cluster centroids). The structural components are first extended to a set of regions called greatest common refinement that are common between both datasets. Then the deviation is found using a difference function applied on those common regions as the amount of work required to transform one model into the other. The measure component summarizes the subset of the data points mapped to each of the regions. Moreover, a more focused deviation is computed on only parts of the common regions (e.g. the deviation between the clusters with highest quality). FOCUS is a general framework that quantifies the deviation between two datasets even if their model structure is different, and how significant that difference is. However, it does not provide any insights on the actual evolution of the new model and it is not suitable for data streams (since it requires rescanning all the past datasets to analyze deviations).

STREAM-DETECT

(Gaber & Yu, 2006) proposed STREAM-DETECT, a framework to identify changes in data streams by measuring online clustering results' deviation over time. Most data streams follow a stable data distribution within a domain in the normal situation, and any change in the distribution and/or domain represents an event or phenomenon that has already occurred or will occur. STREAM-DETECT starts by running some online clustering algorithm, and then for each new set of data objects, the online clustering algorithm is applied again and measurements about the clusters are collected (mean of cluster centroids, standard deviation of cluster centroids, mean size of clusters and the maximum and minimum cluster centroids). The deviation between the old and new measurements is calculated, and this deviation is stored in two cases: the deviation exceeds a pre-specified threshold, or an event or phenomenon has been encountered. If the deviation did not meet any of those two cases, then only the data characteristics are stored. Otherwise, the calculated change is stored. The deviation could be on the clustering characteristics or the data distribution. The clustering deviation calculates the deviation among two cluster characteristics (i.e. mean of cluster centroids, standard deviation of cluster centroids and/or mean of cluster sizes) using the absolute value of the difference between each two consecutive runs normalized by the older one. For the domain deviation detection, the deviation is calculated by discovering the change of the maximum and minimum centers normalized by the distance between the old maximum and minimum centers.

Algorithm 13 STREAM-DETECT

Find the clusters in an initial set of data points using an online clustering algorithm

Measure clustering characteristics (mean of cluster centroids...etc)

WHILE more data points are in the data stream

    Read a new set of data points X

    Find clusters in X

    Measure clustering characteristics of X

    Measure clustering deviation

IF deviation > threshold Or an event has occurred

        Store clustering deviation

ELSE

        Store only the clusters from X

END IF

END WHILE

STREAM-DETECT is followed by a process of offline classification CHANGE-CLASS, that uses the data produced from STREAM-DETECT to run a voting-based classification algorithm over the changed attributes. The event that gets the highest vote would be the classification result, i.e. the event that has attracted the majority of change attributes. STREAM-DETECT is listed in Algorithm 13.

Disadvantages of the Existing Local Cluster Tracking Methods

The main disadvantages of the methods in the local cluster tracking approach are *(i)* their assumption of re-clustering the data instead of adaptation, which increases the time complexity due to performing the clustering from scratch, at each time period, and then having to find the mappings, *(ii)* the need to keep all the data points and their identities to compute the similarity measures between different clusters, which increases the memory and time complexity, and *(iii)* the arbitrary division of the time domain into fixed interval lengths which makes their results vulnerable to the choice of that fixed length, and hence, increasing the risk of missing transitions. These disadvantages are crucial and they can be prohibitive especially in the context of adhering to the strict requirements of stream data mining.

Our proposed approach, Stream-Dashboard, overcomes these disadvantages thanks to an online processing that does not store the data nor data identifiers while eliminating the need to re-cluster the whole data in each time window. It is also less sensitive to the choice of window length thanks to an automated discovery of milestones of cluster trend changes over time.

2.3.2 The Global Cluster Tracking Approach

The global change detection approach tracks the evolution on a global level by finding the deviation between the distribution of the data at consecutive time periods. The two methods described below are examples of such an approach.

Diagnosing changes using kernel density estimation

(Aggarwal, 2003) proposed a framework to diagnose changes in multidimensional data streams with the use of a concept called velocity density estimation, which measures the rate of change of data concentration at a given spatial location over some time horizon. The data concentration/density at a spatial location is the sum of smoothed values of kernel functions associated with each point in the dataset, where the kernel function is used to replace the discrete point with a continuous function that peaks at that point. The velocity density estimation is equal to the difference between two sets of estimates: the forward time slice density estimate (density estimate of data points arrived in the past) and the reverse time slice density estimate (density estimate of data points that will arrive in the future). A positive velocity density indicates that a greater number of points that are closer to the spatial location have arrived at the end of the interval. On the other hand, a negative value indicates that those points arrived at the beginning of the interval. The global overview of the rate of changes of densities at different data points is referred to as the temporal velocity profile. The second type of profiles proposed is the spatial velocity profile, which reflects the directions of movement of data at spatial locations, and is done by finding the direction in which the density gradient is the largest.

Moreover, batch-processing methods were proposed to identify which combinations of dimensions in a high-dimensional data stream have more effect on the evolution of the data stream, hence only those combinations need to be examined more carefully. This framework provided nice visualizations of the data stream, however it did not specify how it can be used to track the cluster evolution. CluStream (Aggarwal *et al.*, 2003) which had previously been proposed by the same authors, and was discussed in Section 2.2.2.2, provides some tracking of the evolution by finding which clusters were created, deleted, or changed based on their ID. However, it does not characterize the real changes (internally and externally) that took place on these clusters.

Information Theory-based distance method

The authors in (Kifer *et al.*, 2004) assume that data points are generated independently but otherwise make no assumptions about the generating distribution (i.e. the techniques are nonparametric). They give provable guarantees that the change that is detected is not noise but statistically significant, and they allow describing the change to a user in a succinct way. The method is based on finding the distance between two sets of points, with each having a different distribution. They start with the first set as the reference set, and start changing the other set by sliding a window whenever new points are available. Once the distance is greater than a threshold, a change is detected. The authors propose a new notion of variation, called relativized discrepancy, between the probability distributions at different time periods and use it to quantify the change. Their distance is based on considering the maximum difference between the empirical weight of two sets of data points. Moreover, they provide statistical proofs of the significance of their distance measure, as well as strict bounds on the threshold values.

Disadvantages of the Existing Global Cluster Tracking Methods

The main disadvantage of the global cluster tracking approach is that it focuses on the global change in *data* at different time periods rather than on the clusters' evolution itself, hence, it fails to capture the internal and external changes that take place at the cluster level. In contrast to this approach, our proposed approach, Stream-Dashboard, tracks the evolution on the clusters' level and captures both internal and external changes.

2.3.3 Topic Detection and Tracking

Another approach, that is similar to the proposed cluster tracking framework, is topic detection and tracking (TDT) (Allan *et al.*, 1998), which is concerned with detecting the appearance of new topics in text data and tracking the reappearance of existing topics and their evolution. TDT consists of three main tasks:

- (i) First, it segments the stream of text into separate stories;
- (ii) second, it detects the events being discussed or identifies new events that have not been discussed before; and

(iii) finally, it tracks the evolution of events by associating new stories with existing events known to the system.

The main differences between TDT and our proposed framework is that:

(i) the tracking task of TDT is done after the events had been detected such that it simply finds which new stories are associated with them, thus it is a supervised learning task. On the other hand, Stream-Dashboard tracks the evolution of the clusters as a module that can wrap around any online clustering algorithm to detect and model the cluster evolution patterns on the fly while the unknown clusters are being discovered;

(ii) TDT does not quantify the evolution, rather it only finds which event is being more discussed. On the other hand, Stream-Dashboard quantifies the evolution in detail using rich behavioral profiles (including evolution trend modeling and summarization), and detects any deviations in that behavior (Section 3.2.3);

(iii) Our framework can detect a richer spectrum of evolution trends or deviations (e.g. internal and external transitions) and not only new topic detection or deviation.

Section 3.2.9 will summarize the various attempts to detect and characterize the changes of clusters in a data stream, discussed above, and compares them to the proposed framework Stream-Dashboard.

2.4 Robust Statistics

In this section, we review the basic concepts of robust statistical estimation, since our proposed clustering algorithm will rely on robust estimation via the use of robust weights.

Classical statistics guarantee optimality in their estimates given that the model assumptions are correct (Huber, 1981), however these assumptions are rarely met in practice. Rather, they are used only for mathematical convenience. For example, if we assume that a sample of data follows a normal distribution, then the optimal estimate of the expected value is the mean of the data points. However, the presence of outliers, which is common in real data, can have an extreme influence on the mean value. Robust statistics (Huber, 1981; Ricardo A. Maronna, 2006) seek to obtain a robust estimation of the parameters of a parametric model while not being too affected by outliers or small deviations from the assumed model.

For example, the ordinary Least Squares (LS) method estimates the parameters of a distribution by minimizing the squared residuals, i.e. its objective function is given by

$$\min \sum_{j=1}^N r_j^2 \quad (2.20)$$

where $r_j = x_j - \theta$ is the residual between the j^{th} data point x_j and its assumed model θ , and N is the number of data points. LS is not robust since extreme outliers with arbitrarily large residuals can have a large influence on the resulting estimate.

In this section, we will first describe the distribution assumptions in robust statistics (Section 2.4.1). Then we will describe the M-Estimators and W-Estimators (Section 2.4.2) and their application to robust location estimation (Section 2.4.3). Finally, we will present some of the properties of M-Estimators (Section 2.4.4).

2.4.1 Robustness

The assumption in classical statistics is that the data points $x_1, \dots, x_n$ are *i.i.d* (independent and identically distributed) (Kruskal, 1988) and follow a common distribution function F . On the other hand, robust statistics assumes that a proportion of the data sample of size $(1 - \varepsilon)$ follows the distribution F , while the remaining ε data points are generated by an unknown mechanism with an unknown distribution H . Hence the overall distribution, G , of the data sample is assumed to be:

$$G = (1 - \varepsilon)F + \varepsilon H \quad (2.21)$$

where $F = N(\mu, \sigma^2)$, H might be any distribution and is called the *contamination distribution* and G is called the *mixture* of F and H . The aim of robust statistics is to find nearly-optimal estimates of the parameters of G , when G is exactly or approximately normal.

2.4.2 M-Estimators and W-Estimators

An M-estimator attempts to limit the influence of outliers by replacing the square of residuals with a less rapidly increasing loss function. The M-estimate $\Theta(x_1, x_2, \dots, x_N)$ estimates the parameter θ by minimizing the loss function ρ as follows

$$\min_{\theta} \left\{ J = \sum_{j=1}^N \rho(x_j; \theta) \right\}. \quad (2.22)$$

The optimal estimate of the parameter is found by setting the derivative of the loss function to zero as follows

$$\frac{\partial J}{\partial \theta} = \sum_{j=1}^N \frac{\partial \rho(x_j; \theta)}{\partial \theta} = \sum_{j=1}^N \psi(x_j; \theta). \quad (2.23)$$

When the M-estimator is shift equivariant, i.e. $\Theta(x_1 + c, \dots, x_N + c) = \Theta(x_1, \dots, x_N) + c$ for any constant c , the loss function ρ and its derivative ψ can be written in terms of the residuals $r = x - \theta$. Moreover, a scale estimate S is used to obtain a scaled residual $r = \frac{x - \theta}{S}$. Hence, the objective function can be written as

$$\min_{\theta} \left\{ J = \sum_{j=1}^N \rho\left(\frac{x_j - \theta}{S}\right) \right\}. \quad (2.24)$$

W-estimators are an alternative to M-estimators, obtained by introducing a robust weight function $w(x)$ that represents the importance of each data sample x in estimating the parameter θ . Its relation to the M-estimator is given by

$$\psi(r) = w(r)r.$$

The optimal estimate of the parameter θ is found by solving

$$\sum_{j=1}^N w\left(\frac{x_j - \theta}{S}\right) \frac{x_j - \theta}{S} = 0 \quad (2.25)$$

The ρ , ψ and w functions for some familiar M-estimators and W-estimators (Ricardo A. Maronna, 2006; Rousseeuw, 1987; Huber, 1981) are listed in Table 2.3. M-estimators and W-estimators rely on an estimate of scale and a constant tuning c . Most estimators use a multiple of the Median of Absolute Deviations (MAD) as a scale estimate, which assumes that the contamination rate is 50%. MAD is defined as follows:

$$MAD(x_i) = \text{med}_i \{ |x_i - \text{med}_j(x_j)| \} \quad (2.26)$$

Table 2.3: Common M-estimators and W-estimators (Ricardo A. Maronna, 2006)

Type	$\rho(r)$	$\psi(r)$	$w(r)$	Range of r	Scale
L_2 (mean)	$\frac{1}{2}r^2$	r	1	$\mathbb{R}$	none
L_1 (median)	$ r $	$\text{sign}(r)$	$\frac{\text{sign}(r)}{r}$	$\mathbb{R}$	none
Huber	$\frac{1}{2}r^2$	r	1	$ r \leq k$	MAD
	$k r - \frac{1}{2}k^2$	$k \text{sign}(r)$	$\frac{k \text{sign}(r)}{r}$	$ r > k$	
Cauchy	$\frac{c^2}{2} \log \left[1 + \left(\frac{r}{c} \right)^2 \right]$	$\frac{r}{1 + (\frac{r}{c})^2}$	$\frac{1}{1 + (\frac{r}{c})^2}$	$\mathbb{R}$	MAD
Tukey	$\frac{1}{6} [1 - (1 - r^2)^3]$	$r(1 - r^2)^2$	$(1 - r^2)^2$	$ r \leq 1$	$c \times MAD$
	$\frac{1}{6}$	0	0	$ r > 1$	
Andrews	$\frac{1}{\pi^2} [1 - \cos(\pi r)]$	$\frac{1}{\pi} \sin(\pi r)$	$\frac{1}{r\pi} \sin(\pi r)$	$ r \leq 1$	$c \times MAD$
	$\frac{2}{\pi^2}$	0	0	$ r > 1$	
Welsch	$\frac{c^2}{2} [1 - \exp(-(\frac{r}{c})^2)]$	$r \exp(-(\frac{r}{c})^2)$	$\exp(-(\frac{r}{c})^2)$	$\mathbb{R}$	MAD

2.4.3 The Location Model

For the data sample $x_1, \dots, x_n$, we assume that each data point x_i depends on the true value μ of an unknown location parameter and some random error u_i , where the errors act in an additive manner. The location model can be represented as follows:

$$x_i = \mu + u_i \quad (i = 1, \dots, n) \quad (2.27)$$

Given this location model, the goal is to find a good estimate of the true value μ (i.e. the sample mean) as a function of the data points: $\hat{\mu} = \hat{\mu}(x_1, \dots, x_n)$, where the estimate should be as close as possible to the true value. One of the classical ways to measure the approximation is by using the mean squared error (MSE):

$$MSE(\hat{\mu}) = E(\hat{\mu} - \mu)^2 \quad (2.28)$$

The estimated mean ($\hat{\mu}$) has the property of *shift equivariance*, which means that if a constant value c is added to the data sample, then the estimate will adapt automatically to reflect that change, as in the following:

$$\hat{\mu}(x_1 + c, \dots, x_n + c) = \hat{\mu}(x_1, \dots, x_n) + c \quad (2.29)$$

Given that random errors $u_i (i = 1, \dots, n)$ in the location model follows the distribution F with

density function f , then the likelihood function of the data sample is given by:

$$L(x_1, \dots, x_n; \mu) = \prod_{i=1}^n f(x_i - \mu) \quad (2.30)$$

And the best estimate of the sample mean is the value $\hat{\mu}$ that maximizes this likelihood function, which is known as the maximum likelihood estimate (MLE) of μ . If the distribution F is exactly known (i.e. has no outliers) then MLE is optimal since it results in finding the value of μ with the lowest asymptotic variance. The MLE of μ is as follows:

$$\hat{\mu} = \arg \max_{\mu} L(x_1, \dots, x_n; \mu) \quad (2.31)$$

The estimate of μ can be written using the logarithm function, since it an increasing function (assuming that the density function f is everywhere positive), as follows:

$$\hat{\mu} = \arg \min_{\mu} \sum_{i=1}^n \rho(x_i - \mu) \quad (2.32)$$

$$\rho = -\log f \quad (2.33)$$

If ρ is differentiable, then $\hat{\mu}$ can be estimated by finding the derivative of ρ , $\psi = \rho'$, and setting it equal to zero as in (2.34). This is called the M-estimate of location.

$$\sum_{i=1}^n \psi(x_i - \hat{\mu}) = 0 \quad (2.34)$$

The M-estimate of location can also be considered as a weighted mean, where each data point has a weight relative to the estimate such that the weight is high when the error u is low. If a weight function W is defined as follows:

$$W(x) = \begin{cases} \frac{\psi(x)}{x} & \text{if } x \neq 0 \\ \psi'(x) & \text{if } x = 0 \end{cases} \quad (2.35)$$

then (2.34) can be written as

$$\sum_{i=1}^n W(x_i - \hat{\mu})(x_i - \hat{\mu}) = 0 \quad (2.36)$$

or as

$$\hat{\mu} = \frac{\sum_{i=1}^n w_i x_i}{\sum_{i=1}^n w_i} \quad (2.37)$$

where $w_i = W(x_i - \hat{\mu})$. For an outlier data point x_i , the difference $(x_i - \hat{\mu})$ would be high and hence when substituted into the weight function in (2.35), its weight value would be small since the denominator would be high.

2.4.4 Properties of M-estimators

If the derivative of ρ , i.e. ψ in (2.34), is monotone non-decreasing (i.e. if $a < b$ then $\psi(a) < \psi(b)$), then (2.34) (and hence (2.32)) has a solution. If ψ is continuous and increasing then the solution $\hat{\mu}$ is unique, otherwise the set of solutions is either a point or an interval. This can be proved by the following theorem (Ricardo A. Maronna, 2006):

Theorem 2.1: *Given the M-estimator $g(\theta) = \sum_{i=1}^n \Psi(x_i, \theta)$ to estimate the parameter θ , where for the location $\Psi(x_i, \theta) = \psi(x_i - \theta)$ and $\theta = \mu$. If ψ is monotone and non-decreasing, and hence Ψ is non-increasing in θ and*

$$\lim_{\theta \rightarrow \theta_1} \Psi(x, \theta) > 0 > \lim_{\theta \rightarrow \theta_2} \Psi(x, \theta) \quad (2.38)$$

where for the location $\theta_2 = -\theta_1 = \infty$.

Then:

1. *There is at least one solution $\hat{\theta} = \hat{\theta}(x_1, \dots, x_n)$ at which g changes sign, i.e., $g(\theta) \geq 0$ for $\theta < \hat{\theta}$ and $g(\theta) \leq 0$ for $\theta > \hat{\theta}$.*
2. *The set of solutions is an interval*
3. *If Ψ is continuous in θ , then $g(\hat{\theta}) = 0$.*
4. *If Ψ is decreasing, then the solution is unique.*

The resistance of the M-estimator reflects the extent to which a small number of outliers can only have a limited effect on the estimate, and it can be quantified using the following properties:

1. **The Influence Curve** shows how the estimate is affected by adding a contaminated data point, where its location is varied. This usually takes the form of the ψ function.
2. **The Breakdown Point**: Reflects the proportion of outlying data points that can be added to the data set without changing the estimation drastically. For example, the mean has a breakdown point of 0, since adding an extreme outlier (i.e. $x = \infty$) would cause the mean to tend to infinity. On the other hand, the median is more robust and has a breakdown point of 50%, since the median value depends only on the two values in the middle after sorting the data points.
3. **The Rejection Point** is the point beyond which the influence curve becomes zero (i.e. the data point makes no contribution to the estimate). An M-estimator with a finite rejection point is called re-descending, and it completely ignores extreme outliers. However, a finite rejection point affects the efficiency of the estimator, since only a few data samples will be used to find the estimate (after completely ignoring the data points near the tails of the distribution).

2.5 Linear Regression Models

Regression analysis (Chatterjee & Hadi, 2006) is a simple statistical method to investigate and quantify relationships between two or more variables. The relationship is represented by an equation and a set of coefficients which summarizes the interaction between the variables, and allows the prediction of future values.

In this section, we will provide details about linear regression models, how to estimate (Section 2.5.1) and evaluate them (Section 2.5.2), and finally how to find the difference or deviation between two linear regression models (Section 2.5.3).

2.5.1 Building regression models

In this section, we review the linear regression models, because they will form the modeling basis for our stream model tracking component, called TRACER (Section 3.2).

A *response* or *dependent* variable ($Y = (y_1, y_2, \dots, y_n)$) is the variable that is being investigated, and the regression model aims at finding the right equation to predict its value using the *predictor* or *independent* variables ($X_1, X_2, \dots, X_P$), where P is the number of predictor variables. The relationship between a dependent variable (Y) and an independent variable ($X = (x_1, x_2, \dots, x_n)$) can be represented as the linear model:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, i = 1, \dots, n \quad (2.39)$$

where β_0 and β_1 are constants called the *model regression coefficients*, ε_i represents the *error* or *discrepancy* in approximating y_i and n is the number of points. The coefficient β_1 is called the *slope* and represents the change in Y for a unit change in X , and the coefficient β_0 is called the *intercept* and represents the predicted value of Y when X is 0.

The goal of regression analysis is to use some of the available data to estimate the parameters β_0 and β_1 , which means finding a straight line that best fits the available data. The Least squares method is typically used to do that by minimizing the sum of the squares of the vertical distances (i.e. errors or deviations) of each point with respect to the straight line, which can be written as:

$$S(\beta_1, \beta_0) = \sum_{i=1}^n \varepsilon_i^2 = \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2 \quad (2.40)$$

The estimated values of the coefficients that minimize the vertical distances can be found using the following two equations, where $\bar{x}$ and $\bar{y}$ are the means of the X and Y values respectively:

$$\begin{aligned} \hat{\beta}_1 &= \frac{\sum_{i=1}^n (y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^n (x_i - \bar{x})^2} \\ \hat{\beta}_0 &= \bar{y} - \hat{\beta}_1 \bar{x} \end{aligned} \quad (2.41)$$

and now the least squares regression line equation becomes:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X \quad (2.42)$$

The difference between the real value y_i and the estimated value $\hat{y}_i$ represents the error in the approximation e_i which is called the *residual*. One of the assumptions in regression analysis is that the ε 's are independent random quantities that are normally distributed with zero mean and variance

σ^2 . In this case, the estimates in (2.41) are unbiased¹ with variances:

$$\begin{aligned} \text{Var}(\hat{\beta}_1) &= \sigma^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \right] \\ \text{Var}(\hat{\beta}_0) &= \frac{\sigma^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \end{aligned} \quad (2.43)$$

and an unbiased estimate of the unknown parameter σ^2 is given by:

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^n e_i^2}{n-2} = \frac{SSE}{n-2} \quad (2.44)$$

where SSE is the sum of the squared residuals (i.e. $\sum_{i=1}^n (y_i - \hat{y}_i)^2$) and $n-2$ is the degrees of freedom which is equal to the number of observations minus the number of estimated regression coefficients (β_1 and β_0). Replacing this estimated value in (2.43) will result in an unbiased estimation of the variances, and an estimate of the standard error can be found as:

$$\begin{aligned} s.e.(\hat{\beta}_1) &= \hat{\sigma} \sqrt{\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^n (x_i - \bar{x})^2}} \\ s.e.(\hat{\beta}_0) &= \frac{\hat{\sigma}}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2}} \end{aligned} \quad (2.45)$$

where these standard errors represent how accurately the coefficients were estimated. Using these standard errors, the *confidence intervals* (CI) of the estimation, with a confidence equal to $(1 - \alpha) * 100\%$, can be found as follows:

$$\begin{aligned} \hat{\beta}_0 \pm t_{(n-2, \alpha/2)} \times s.e.(\hat{\beta}_0) \\ \hat{\beta}_1 \pm t_{(n-2, \alpha/2)} \times s.e.(\hat{\beta}_1) \end{aligned} \quad (2.46)$$

where $t_{(n-2, \alpha/2)}$ is the $(1 - \alpha/2)$ percentile of a t distribution with $(n-2)$ degrees of freedom.

In TRACER (Section 3.2) we use a linear regression model with one independent parameter, which is the time index. However, this can be easily extended to more than one variable.

2.5.2 Evaluating Regression Models

After finding the regression line as in (2.42), we are interested in measuring the quality of the model to the data and not just the existence of such a model. One way of doing that is by first computing the following values:

---

¹An estimate $\hat{\beta}$ is said to be an unbiased estimate of β if the expected value of $\hat{\beta}$ is equal to β

$$\begin{aligned}
SST &= \sum_{i=1}^n (y_i - \bar{y})^2 \\
SSE &= \sum_{i=1}^n (y_i - \hat{y}_i)^2
\end{aligned}
\tag{2.47}$$

where SST stands for the total sum of squared deviations in Y from the mean $\bar{y}$, and SSE stands for the sum of squared residuals. Using these values, a test for the goodness of fit can be found as:

$$R^2 = 1 - \frac{SSE}{SST} \tag{2.48}$$

where R^2 is called the *goodness-of-fitness index* or *coefficient of determination* and it reflects the proportion of the total variation in Y that is accounted for by the predictor variable X . R^2 is in the interval between 0 and 1, and the closer it is to 1, the better the model is, because this means that it accounts for a large variation in Y .

2.5.3 Finding Deviation Between Regression Models

Regression analysis is usually used to analyze the relationships between multiple variables. For example, each stock is represented by one or more regression models in the stock market. As some of these relationships might become similar, it is useful to estimate the difference or deviation between two or more regression models.

One way of finding the deviation is by estimating the angle between the regression models, and if this angle is less than a threshold, then they could be considered similar. The angle θ between models L_i with coefficients $\{\beta_{0,i}, \beta_{1,i}\}$ and L_j with coefficients $\{\beta_{0,j}, \beta_{1,j}\}$ can be estimated by finding the tangent of the angle as follows:

$$\tan(\theta) = \frac{\max(\beta_{1,i}, \beta_{1,j}) - \min(\beta_{1,i}, \beta_{1,j})}{1 + \beta_{1,i}\beta_{1,j}} \tag{2.49}$$

2.6 Topic Modeling

Topic modeling (Wallach, 2006) is a set of algorithms that aim at discovering the latent topics (i.e. hidden topics) among a set of documents. Topic modeling is an unsupervised method, that analyzes the words in their original form and discovers the latent topics, their interaction and how they change over time. For example, a document can belong to many topics such as politics and economics.

In this section, we will provide a brief overview of two of the main approaches for topic modeling: Probabilistic Latent Semantic Indexing (PLSI) in Section 2.6.1 and Latent Dirichlet Allocation (LDA) in Section 2.6.2.

2.6.1 Probabilistic Latent Semantic Indexing

PLSI was proposed by Hofmann (Hofmann, 1999), and uses the same conceptual assumptions as in LSI (Dumais *et al.*, 1995) but follows a probabilistic approach. The first step in finding the latent models is to convert the dataset into a bag-of-words (BOW) representation. BOW is basically a matrix that captures the frequency (or the existence) of each feature or word in each document. For a set of D documents and W unique words, the BOW matrix X would be of size $|D| \times |W|$ where each row represents a document and each column represents a unique word.

PLSI assumes that each word $w \in W$ is generated into a document $d \in D$ that belongs to topic $i \in K$ following this probabilistic approach:

- First, a document d is sampled following a multinomial distribution of documents $p(d)$
- Second, a topic i is sampled based on the topic distribution with respect to the selected document $\theta_{di} = p(z = i|d)$.
- Finally, a term v is sampled based on the multinomial distribution of the selected topic $\Phi_{iv} = p(v = w|z = i)$

PLSI aims at associating a topic variable z with each term v in each document d . This joint probability can be expressed as:

$$p(v, d) = p(d) \times p(v|d) \quad (2.50)$$

$$\text{where } p(v|d) = \sum_i^K p(v|z = i) \times p(z = i|d)$$

The Expectation Maximization algorithm (Dempster *et al.*, 1977) can be used to find these probabilities by maximizing the log-likelihood given by:

$$\mathcal{L} = \sum_{d=1}^{|D|} \sum_{v=1}^{|W|} x_{vd} \log p(w = v, d) \quad (2.51)$$

$$\mathcal{L} = \sum_{d=1}^{|D|} \sum_{v=1}^{|W|} x_{vd} \log \sum_{i=1}^K p(w = v|z = i) \times p(z = i|d) \times p(d)$$

where x_{vd} is the frequency of the term v in document d .

2.6.2 Latent Dirichlet Allocation

PLSI generates a large number of parameters (i.e. a parameter for each document), hence, it might suffer from over-fitting. Also it cannot find the probability of completely new documents. LDA was proposed by (Blei *et al.*, 2003) as an alternative that solves the problems of PLSI.

LDA follows a similar probabilistic approach to PLSI to generate the data. The generative process can be summarized as follows:

- The term distribution Φ_i for topic i is a multinomial distribution that follows a symmetric Dirichlet distribution with parameter β , where $\Gamma()$ denotes the Gamma function.

$$p(\Phi_i|\beta) = \frac{\Gamma(W\beta)}{\Gamma(\beta)^W} \prod_{v=1}^W \phi_{iv}^{\beta-1} \quad (2.52)$$

- The topic distribution θ_d for document d is a multinomial distribution that follows a Dirichlet distribution with parameters in the vector α

$$p(\theta_d|\alpha) = \frac{\Gamma(\sum_i \alpha_i)}{\prod_{i=1}^K \Gamma(\alpha_i)} \prod_{i=1}^K \theta_{di}^{\alpha_i-1} \quad (2.53)$$

- The topic z_{dn} for each token indexed by n in document d is sampled from document topic distribution θ_d

$$p(z_{dn} = i|\theta_d) = \theta_{di} \quad (2.54)$$

- Each token w is sampled from the distribution associated with the selected topic

$$p(w_{dn} = v|z_{dn} = i, \phi_i) = \phi_{iv} \quad (2.55)$$

The generative process is shown in Figure 2.1. Basically, LDA provides a mechanism to find patterns of co-occurrence between the terms and using these patterns to find coherent topics. Hence, LDA finds topics in which the most probable terms frequently co-occur together in the documents, thus helping with polysemy. This is a departure from more naive independent term assumptions. Another advantage is that in the topic-specific term distributions $p(\Phi_i|\beta)$, the Dirichlet prior provides smoothing that assigns non-zero probabilities even to unseen terms in a document.

Figure 2.1: LDA Generative Process (Blei *et al.*, 2003)

Finding the parameters of LDA (i.e. β and α) can be accomplished by maximizing the likelihood function:

$$\mathcal{L} = \prod_{d=1}^M \prod_{n=1}^N p(w_{dn}|z_{dn}, \Phi) \times p(z_{dn}|\theta_d) \times p(\theta_d|\alpha) \times p(\Phi|\beta) \quad (2.56)$$

where M is the number of documents and N is the number of tokens.

Optimizing the likelihood directly proved to be hard since the topic assignments z_{dn} are not given. As a result, approximation methods are used such as the Collapsed Gibbs Sampling (Liu, 1994).

An online version of LDA was proposed by (Hoffman *et al.*, 2010), which is based on online variational Bayes optimization. Online LDA aims at handling massive collections of documents such as a data stream, where each document is only examined once and then discarded.

2.7 Summary and Conclusions

In this chapter, we reviewed some of the concepts and methods related to the proposed cluster mining and tracking framework. We presented some of the clustering algorithms as well as their limitations and strengths (Section 2.1). Then we presented the data stream model and its constraints, which motivated the development of stream data clustering algorithms (Section 2.2). Section 2.3 reviewed some of the approaches to track clusters evolution and their limitations, mainly their assumption of re-clustering instead of adaptation, the need to keep all the data points to find transitions and their arbitrary division of the time domain. This has motivated the use of linear regression mod-

els (Section 2.5) in developing the second component of the proposed framework (TRACER) which will be discussed in the next chapter (Section 3.2). Robust Statistics (Section 2.4) are used in developing the first component of the proposed framework (RINO-Streams) which will be discussed in the next chapter (Section 3.1). Finally, we presented a brief overview of Topic Modeling (Section 2.6) which is used as a pre-processing step to reduce the dimensionality of some of the data streams, especially Twitter.

CHAPTER 3

STREAM-DASHBOARD: A NEW FRAMEWORK TO MINE, TRACK AND VALIDATE EVOLVING DATA STREAM CLUSTERS

In this chapter, we propose a framework, called **Stream Dashboard** (Hawwash & Nasraoui, 2012), to *mine*, *track* and *validate* evolving data stream clusters simultaneously. Stream Dashboard consists of three main components:

1. an *online clustering* component,
2. a *tracking and validation* component, and
3. a *configuration adaptation* component.

The online clustering component incrementally maintains a clustering model of the data stream. The clustering model is represented as a set of properties or metrics for each of the clusters, such as the centroids and scales. The clustering model can be used as an input to a higher level application. For example, in the web usage mining domain, it can be used as an input to a recommender system, while in a network security context, it could be used as input to an anomaly detection or forensic analysis system.

The tracking and validation component monitors the characteristics of the clustering model (i.e. the properties of the clusters) and builds and maintains *regression models* for them. Thus this component can be used toward the following goals: *(i)* It can serve as an input for an interactive dynamic visualization component that would give a deep insight into the evolution of clusters over the data stream lifetime.

Figure 3.1: Stream Dashboard Flowchart

Framework for Mining, Tracking and Validating Evolving Data Stream Clusters (*Stream-Dashboard*)

(ii) It can serve as an input to another application such as an online recommender system, where it can be used to improve the quality of the recommendations.

The configuration adaptation component uses the regression models maintained to fine-tune the configuration of the online clustering algorithm (i.e. the input parameters such as threshold values). The framework flowchart is shown in Figure 3.1.

The remainder of this chapter is organized as follows: the online clustering component is discussed in Section 3.1, the tracking and validation component in Section 3.2, and the configuration adaption component in Section 3.3. Section 3.4 presents the genealogy graph which is used to store the history of the data stream. Section 3.5 presents the complete generic framework, and Section 3.6 describes the visualization features. Finally, Section 3.7 presents the summary and conclusions found in this chapter.

3.1 The RINO-Streams Algorithm

The generic online clustering algorithm component can be *any* clustering algorithm that adheres to the following criteria:

(i) it satisfies typical stream clustering algorithm requirements (Barbará, 2002) that were reviewed in Section 2.2.1,

(ii) it incrementally updates the clustering model, and

(iii) it quantifies the clusters' characteristics using a set of properties.

Examples of such algorithms include (Aggarwal *et al.*, 2003; Chen & Tu, 2007; Lowette & Laerhoven, 2004; Zhong, 2005). The above requirements are the only requirements for the online clustering component, because the regression analysis requires real discrete values to generate the evaluation models.

As an online clustering algorithm, we propose RINO-Streams (**R**obust clustering of data streams using **I**ncremental **O**ptimization) (Hawwash & Nasraoui, 2010), an incremental clustering algorithm inspired by TRAC-Streams (Nasraoui & Rojas, 2006). Both algorithms extract evolving clusters from a massive data stream in one pass, while being able to resist and detect the presence of outliers. They incrementally update the clustering model using an estimation of centroids and scales rooted in robust statistics (Rousseeuw, 1987; Huber, 1981; Ricardo A. Maronna, 2006). Moreover, they detect outliers and merge clusters using a robust distribution-independent statistical test, called the Chebyshev test (Marshall & Olkin, 1960), which ensures robustness to outliers and cluster compactness. The main distinction of the proposed algorithm RINO-Streams from TRAC-Streams is in its simpler density-based objective function and a better merging technique.

The rest of this section is organized as follows: some definitions are first presented in Section 3.1.1, then the equations for incremental updates of the centroid and scale are presented in Section 3.1.3 and the relation between the proposed objective function and M-estimators is discussed in Section 3.1.4. Section 3.1.5 discusses the use of Chebyshev bounds to detect outliers, while merging and splitting clusters are discussed in Section 3.1.6. Detecting arbitrary-shaped clusters is discussed in Section 3.1.7, and the complete RINO-Streams algorithm pseudo code is listed in Section 3.1.8. The time and memory complexity of RINO-Streams are discussed in Section 3.1.9, while Section 3.1.10 shows how RINO-Streams complies with the requirements of online stream clustering algorithms. Finally, Section 3.1.11 analyzes the differences between RINO-Streams and the most competitive stream clustering algorithms from the literature.

3.1.1 Definitions

The data stream X consists of a set of data points that are indexed based on the order of their arrival, and presented as: $x_1, x_2, \dots, x_N$ where N is the size of the data stream and x_i is an d dimension data point (i.e. $x_i = (x_i^1, \dots, x_i^d)$). Each cluster i at time n (i.e. after receiving n points) is defined as follows:

Definition 3.1.1. *Cluster : The i^{th} cluster at time n ($C_{i,n}$) is defined using two spatial parameters: a centroid $c_{i,n}$ and a scale $\sigma_{i,n}^2$. Two additional measures are used to keep track of stream-specific properties, namely the soft cardinality $W_{i,n}$ (the sum of the weights of the data points) and the age of the cluster, a_i .*

The centroid represents the location of the cluster center at any time, while the scale represents the size of an influence zone around the centroid, where data from the stream has landed in the past. Both centroid and scale are affected by a robust weight function that is defined for each data point, relative to each cluster, and that decreases with the distance from the data instance to the cluster centroid, and also decreases with the timestamp of arrival the data in the stream. Hence, newer data points would have more impact on the model than older ones, which allows capturing the evolution of clusters over time. The soft cardinality W_i is the sum of the robust weights of each data point belonging to the cluster, and is one indicator of the quality of the cluster: a high cardinality means that the cluster represents a large portion of the data stream. The age a_i is defined as the difference between the current data arrival's timestamp (n) and the time when the cluster was first created. In addition to keeping track of the cluster creation time, the age is used to compute a grace period a_{mature} for the cluster before it becomes eligible for testing its quality based on the minimum density threshold δ_{min} , which is an important criterion for cluster survival. Shielding new clusters from premature deletion serves to prevents deleting clusters that are still in an infancy stage.

Definition 3.1.2. *Adaptive Robust Weight: At any given timestamp n (i.e. after the arrival of data points $x_1, x_2, \dots, x_n$ in the stream), the robust weight of the j^{th} data point, arriving at timestamp $j, 1 \leq j \leq n$, is defined as follows, for the i^{th} cluster, $C_i, i = 1, \dots, K$:*

$$w_{ij,n} = e^{-\left(\frac{d_{ij}^2}{\kappa \sigma_{i,n}^2} + \frac{n-j}{\tau}\right)} \quad (3.1)$$

where τ is an optional application-dependent parameter that controls the time decay or forgetting rate ($e^{-1/\tau}$) of the old data points (when needed, i.e. $\tau < \infty$), and how much importance is given to newer data points, d_{ij}^2 is the Euclidean distance from the j^{th} data point to the i^{th} cluster's centroid c_i , $\sigma_{i,n}^2$ is the scale of cluster i at timestamp n , and κ is a constant used to normalize the distances.

The robust weight $w_{ij,n}$ is essentially a Welsh estimator (Table 2.3) as we will show in Section 3.1.4. W-estimators are usually used to optimize the scaled residuals (i.e. $\frac{r^2}{\sigma}$), hence, the distances are divided by the scale of the distances as shown in (2.24). Moreover, in most robust estimators, a tuning constant is usually used along with the scale, as described in Section 2.4.2, thus the total scale is typically set to a constant multiplier of MAD (Rousseeuw, 1987), i.e. $constant \times MAD$, where MAD is the median of the absolute deviations of distances (2.26). However, in the context of data streams, the data point is seen only once and then discarded, hence, neither the data nor their distances to the clusters are kept. We solve this problem by exploiting the fact that scaled distances follow a Chi-Square distribution which has a variance equal to $2d$, where d is the number of dimensions. Hence, the distances can be normalized by the standard deviation of a Chi-Square distribution, which is equal to $\sigma_{\chi^2} = \sqrt{2d}$ (i.e. $\kappa = \sqrt{2d}$).

The second term ($e^{-\frac{(n-j)}{\tau}}$) represents a forgetting factor, that causes the weight of the data point j to decrease geometrically by the value of $n - j$. Hence a new data (with $j = n$) would have a higher weight since the forgetting factor would be close to 1, while an older data point (with $j \ll n$) would have a lower value for the forgetting factor (i.e. approaching 0 as n increases), which results in a smaller weight.

Assuming that the parameters of the model do not change significantly with every new point, then each old data point's weight, and thus its influence on the cluster parameter estimates, can be easily shown to decrease, after the arrival of each additional new data point, as follows:

$$w_{ij,n} = e^{-\frac{1}{\tau}} w_{ij,n-1}. \quad (3.2)$$

Thus the time forgetting factor controls the speed of forgetting older data. As $\tau \rightarrow \infty$, the time decay rate $1/\tau \rightarrow 0$, resulting in a maximal forgetting factor ($e^{-\frac{1}{\tau}} \rightarrow 1$), meaning that no forgetting occurs, and both the oldest and the most recent data would contribute equally to the parameter estimation.

Definition 3.1.3. *Sum Of Weights :* For the i^{th} cluster, $C_i, i = 1, \dots, K$, the sum of the robust weights of the data points $W_{i,n}$ at time n is defined as follows:

$$W_{i,n} = \sum_{j=1}^n w_{ij}. \quad (3.3)$$

Given that the robust weights decrease with both the distance from the cluster centroid and the time since the arrival timestamp of each data point, the sum of weights will decrease for an older cluster, if no new data arrives to land in its influence zone.

Definition 3.1.4. *The Clustering Model :* The clustering model at time n is defined as follows:

$$\zeta_n = C_1 \cup C_2 \cdots \cup C_K, \text{ where } C_i = (c_{i,n}, \sigma_{i,n}^2, a_i, W_{i,n}) \quad (3.4)$$

Definition 3.1.5. *Density Optimization Function :* After encountering n data points, we search for a maximum of K cluster centroids $c_{i,n}$ and scales $\sigma_{i,n}$ by optimizing the density objective function $\delta_{i,n}$ as follows:

$$\delta_{i,n} = \sum_{j=1}^n \frac{w_{ij,n}}{\sigma_{i,n}^2}, \quad (3.5)$$

$$\max_{c_{i,n}, \sigma_{i,n}} \left\{ \sum_{i=1}^K \delta_{i,n} \right\} i = 1, \dots, K, \sigma_{i,n}^2 > 0 \quad (3.6)$$

The robust weight $w_{ij,n}$ can be considered as the degree of membership of the point j in the cluster i after encountering n data points, hence the sum of the weights $W_{i,n}$ for each cluster represents the soft cardinality of that cluster. A high cardinality is desirable because it means that this cluster represents enough points to justify its existence.

The scale $\sigma_{i,n}$ is related to the size of the influence zone of the cluster (i.e. all points inside that zone are considered part of the cluster). Hence, a small scale means that it is a good and compact cluster.

The density of the cluster, $\delta_{i,n}$ in (3.5), combines the previous two metrics of the cluster, and hence it increases as the soft cardinality increases and the scale decreases. The advantage of optimizing the density, which combines the two metrics, is that judging the quality of the cluster using

only the sum of weights (the numerator) is not enough, because a cluster with a large number of points is not desirable from the point of view of density, unless these data points are confined within a small influence zone .

Note that we added an inequality constraint on the scale in (3.6) to make sure that it remains greater than zero. Without this constraint, the optimization of the density with respect to the scale can lead to the scale shrinking to zero. This is because a zero scale would result in a singleton cluster which is a global optimum of the objective function.

3.1.2 Using Chebyshev Bounds to Delimit the Cluster's Influence Zone

There exist some upper tail bounds in statistics, that bound the total probability that some random variable is in the tail of the distribution with some significant value (i.e. far from the mean). One of these bounds is the Chebyshev bound (Marshall & Olkin, 1960), which is a tight bound that, unlike bounds such as the Chernoff bounds for example, relies on no assumptions about the distribution of the data. The only assumption is that a reliable scale is available, which is available using RINO-Streams by virtue of the robust density optimization.

Definition 3.1.6. *Chebyshev Bounds : The Chebyshev bound for a random variable Y in a distribution with mean μ and standard deviation σ for any real number $t > 0$, is given by*

$$Pr\{|Y - \mu| \geq t\sigma\} \leq \frac{1}{t^2} \quad (3.7)$$

The Chebyshev inequality can be rearranged as follows:

$$Pr\{|Y - \mu|^2 \geq t^2\sigma^2\} \leq \frac{1}{t^2} \quad (3.8)$$

For data with d dimensions, the bound can be written as

$$Pr\left\{\sum_{m=1}^d |Y_m - \mu_m|^2 \geq t^2 \sum_{m=1}^d \sigma_m^2\right\} \leq \frac{1}{t^2} \quad (3.9)$$

which, in the simple case of $\sigma_m = \sigma$, becomes

$$Pr \left\{ \sum_{m=1}^d |Y_m - \mu_m|^2 \geq t^2 \cdot d \cdot \sigma^2 \right\} \leq \frac{1}{t^2} \quad (3.10)$$

3.1.3 Incremental Optimization of the Cluster Density Criterion

Optimizing the density objective function is done using alternating optimization, where finding each parameter is done by fixing all the other parameters, and the same process is repeated for each parameter in alternating fashion. This is the same optimization technique that is used in the Expectation Maximization (EM) algorithm (Dempster *et al.*, 1977).

The first step is to form the objective function in (3.6) for each cluster C_i as a Lagrangian to include the constraints as follows.

$$\mathcal{L} = \delta_i - \lambda \sigma_i^2 = \sum_{j=1}^n \frac{w_{ij,n}}{\sigma_{i,n}^2} - \lambda \sigma_i^2 \quad (3.11)$$

Now we find the optimal values by first setting the derivative of the Lagrangian with respect to the centroid to zero while fixing the scale. Then we set the derivative with respect to scale to zero while fixing the centroid. We need also to check for the Karush-Kuhn-Tucker conditions to handle the inequality condition.

Theorem 3.1.1. *Optimal Incremental Centroid Update : Given the previous centroids, $c_{i,n-1}$, and assuming that the scales do not change much relative to the scale that resulted from the previous iteration, the new centroid that optimizes (3.11) after the arrival of the n^{th} data point is given by*

$$c_{i,n} = \frac{e^{\frac{-1}{\tau}} c_{i,n-1} W_{i,n-1} + w_{in,n} x_n}{e^{\frac{-1}{\tau}} W_{i,n-1} + w_{in,n}} \quad (3.12)$$

Proof. Since the time dependency has been absorbed into the weight function, and by fixing the previous centroid $c_{i,n-1}$, scale $\sigma_{i,n-1}$ and weight sums $W_{i,n-1}$, the equations for the center updates are found by finding the derivative of the density $\delta_{i,n}$ with respect to the centroid $c_{i,n}$, while all the other parameters are held constant as follows:

$$\frac{\partial \mathcal{L}}{\partial c_{i,n}} = \frac{1}{\sigma_{i,n}^2} \times \sum_{j=1}^n w_{ij,n} \frac{\partial d_{ij}^2}{\partial c_{i,n}} - \frac{\partial (\lambda \sigma_{i,n}^2)}{\partial c_{i,n}} = \mathbf{0}$$

In case an inner norm inducing metric is used such as $d_{ij}^2 = (x_j - c_{i,n})^t A (x_j - c_{i,n})$, where A is a positive semi-definite matrix (A is the identity matrix for the Euclidean norm), it is easy to show that $\frac{\partial d_{ij}^2}{\partial c_{i,n}} = -2A(x_j - c_{i,n})$. For the case of $A = I$ (identity matrix), we have

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial c_{i,n}} &= \frac{1}{\sigma_{i,n}^2} \times \sum_{j=1}^n \frac{\partial w_{ij,n}}{\partial c_{i,n}} - \frac{\partial (\lambda \sigma_{i,n}^2)}{\partial c_{i,n}} \\
\frac{\partial \mathcal{L}}{\partial c_{i,n}} &= \frac{1}{\sigma_{i,n}^2} \times \sum_{j=1}^n w_{ij,n} \times \frac{\partial \left(\frac{-(x_j - c_{i,n})^t (x_j - c_{i,n})}{\kappa \sigma_{i,n}^2} \right)}{\partial c_{i,n}} - 0 \\
\frac{\partial \mathcal{L}}{\partial c_{i,n}} &= \frac{1}{\sigma_{i,n}^2} \times \sum_{j=1}^n w_{ij,n} \times \left\{ \frac{\frac{\partial (-(x_j - c_{i,n})^t (x_j - c_{i,n}))}{\partial c_{i,n}} \times \kappa \sigma_{i,n}^2 - \frac{\partial (\kappa \sigma_{i,n}^2)}{\partial c_{i,n}} \times -(x_j - c_{i,n})^t (x_j - c_{i,n})}{(\kappa \sigma_{i,n}^2)^2} \right\} \\
\frac{\partial \mathcal{L}}{\partial c_{i,n}} &= \frac{1}{\sigma_{i,n}^2} \times \sum_{j=1}^n w_{ij,n} \times \left\{ \frac{-2(x_j - c_{i,n}) \times \kappa \sigma_{i,n}^2 - 0 \times -(x_j - c_{i,n})^t (x_j - c_{i,n})}{\kappa^2 \sigma_{i,n}^4} \right\} \\
\frac{\partial \mathcal{L}}{\partial c_{i,n}} &= \frac{1}{\sigma_{i,n}^2} \times \sum_{j=1}^n \frac{-2w_{ij,n}(x_j - c_{i,n})}{\kappa \sigma_{i,n}^2} \\
\frac{\partial \mathcal{L}}{\partial c_{i,n}} &= \frac{2}{\kappa \sigma_{i,n}^4} \times \left\{ \sum_{j=1}^n w_{ij,n} c_{i,n} - \sum_{j=1}^n w_{ij,n} x_j \right\} = 0,
\end{aligned}$$

Therefore,

$$\begin{aligned}
\sum_{j=1}^n w_{ij,n} c_{i,n} &= \sum_{j=1}^n w_{ij,n} x_j \\
c_{i,n} &= \frac{\sum_{j=1}^n w_{ij,n} x_j}{\sum_{j=1}^n w_{ij,n}}
\end{aligned}$$

Given the previous centroids, $c_{i,n-1}$, and assuming that the scales do not change much relative to the scale that resulted from the previous point, the new centroid that optimizes (3.6) after the arrival

of the n^{th} data point, and by penalizing the previous information as in (3.2), is given by:

$$\begin{aligned}
c_{i,n} &= \frac{\sum_{j=1}^n w_{ij,n} x_j}{\sum_{j=1}^n w_{ij,n}} \\
c_{i,n} &= \frac{\sum_{j=1}^{n-1} w_{ij,n} x_j + w_{in,n} x_n}{\sum_{j=1}^n w_{ij,n} + w_{in,n}} \\
c_{i,n} &= \frac{\sum_{j=1}^{n-1} (e^{-\frac{1}{\tau}} w_{ij,n-1}) x_j + w_{in,n} x_n}{\sum_{j=1}^n (e^{-\frac{1}{\tau}} w_{ij,n-1}) + w_{in,n}} \\
c_{i,n} &= \frac{e^{-\frac{1}{\tau}} \sum_{j=1}^{n-1} w_{ij,n-1} x_j + w_{in,n} x_n}{e^{-\frac{1}{\tau}} \sum_{j=1}^n w_{ij,n-1} + w_{in,n}} \\
c_{i,n} &= \frac{e^{-\frac{1}{\tau}} (c_{i,n-1} \sum_{j=1}^n w_{ij,n-1}) + w_{in,n} x_n}{e^{-\frac{1}{\tau}} (\sum_{j=1}^n w_{ij,n-1}) + w_{in,n}} \\
c_{i,n} &= \frac{e^{-\frac{1}{\tau}} c_{i,n-1} W_{i,n-1} + w_{in,n} x_n}{e^{-\frac{1}{\tau}} W_{i,n-1} + w_{in,n}}
\end{aligned}$$

□

The first term in the numerator (and the denominator) represents the previous knowledge about the location of the centroid obtained from the points $(x_1, \dots, x_{n-1})$, and this term is multiplied by the forgetting factor $(e^{-\frac{1}{\tau}})$ to reduce its effect on the new updated centroid and give more emphasis to the new data point x_n . This comes directly from the fact that the weight of each point is reduced by $e^{-\frac{1}{\tau}}$ as given in (3.2). The second term in the numerator (and denominator) represents the new information obtained from the new data point x_n .

Theorem 3.1.2. *Optimal Incremental Scale Update: Given the previous scale $\sigma_{i,n-1}^2$, the new scale that optimizes (3.6) after the arrival of the n^{th} data point is given by*

$$\sigma_{i,n}^2 = \frac{\kappa e^{-\frac{1}{\tau}} (\sigma_{i,n-1}^2 W_{i,n-1}) + w_{in,n} d_{ij}^2}{\kappa (e^{-\frac{1}{\tau}} W_{i,n-1} + w_{in,n})} \quad (3.13)$$

Proof. For the cluster C_i at time n , we find the derivative of the density $\delta_{i,n}$ with respect to the centroid $\sigma_{i,n}^2$, while all other parameters are held constant, giving

$$\begin{aligned}
\frac{\partial \delta_{i,n}}{\partial \sigma_{i,n}^2} &= \sum_{j=1}^n \frac{\frac{\partial w_{ij,n}}{\partial \sigma_{i,n}^2} \times \sigma_{i,n}^2 - w_{ij,n} \times \frac{\partial (\sigma_{i,n}^2)}{\partial \sigma_{i,n}^2}}{(\sigma_{i,n}^2)^2} - \frac{\partial (\lambda \sigma_{i,n}^2)}{\partial \sigma_{i,n}^2} \\
\frac{\partial \delta_{i,n}}{\partial \sigma_{i,n}^2} &= \sum_{j=1}^n \frac{w_{ij,n} \times \frac{\partial (\frac{-d_{ij}^2}{\kappa \sigma_{i,n}^2})}{\partial \sigma_{i,n}^2} \times \sigma_{i,n}^2 - w_{ij,n} \times 1}{\sigma_{i,n}^4} - \lambda \\
\frac{\partial \delta_{i,n}}{\partial \sigma_{i,n}^2} &= \sum_{j=1}^n \frac{w_{ij,n} \times \left\{ \frac{\frac{\partial (-d_{ij}^2)}{\partial \sigma_{i,n}^2} \times \kappa \sigma_{i,n}^2 - \frac{\partial (\kappa \sigma_{i,n}^2)}{\partial \sigma_{i,n}^2} \times -d_{ij}^2}{(\kappa \sigma_{i,n}^2)^2} \right\} \times \sigma_{i,n}^2 - w_{ij,n}}{\sigma_{i,n}^4} - \lambda \\
\frac{\partial \delta_{i,n}}{\partial \sigma_{i,n}^2} &= \sum_{j=1}^n \frac{w_{ij,n} \times \left\{ \frac{0 \times \kappa \sigma_{i,n}^2 - \kappa \times -d_{ij}^2}{\kappa^2 \sigma_{i,n}^4} \right\} \times \sigma_{i,n}^2 - w_{ij,n}}{\sigma_{i,n}^4} - \lambda \\
\frac{\partial \delta_{i,n}}{\partial \sigma_{i,n}^2} &= \sum_{j=1}^n \frac{\frac{w_{ij,n} d_{ij}^2}{\kappa \sigma_{i,n}^2} - w_{ij,n}}{\sigma_{i,n}^4} - \lambda = 0
\end{aligned}$$

The Karush-Kuhn-Tucker conditions are necessary (but not sufficient) for the scale to be maximum. The conditions are:

$$\lambda \geq 0 \text{ \& } \lambda \sigma_{i,n}^2 = 0 \quad (3.14)$$

Which means that we have two cases. In the first case, $\lambda = 0$, and the scale can be found by setting the gradient

$$\sum_{j=1}^n \frac{\frac{w_{ij,n} d_{ij}^2}{\kappa \sigma_{i,n}^2} - w_{ij,n}}{\sigma_{i,n}^4} - \lambda = \sum_{j=1}^n \frac{\frac{w_{ij,n} d_{ij}^2}{\kappa \sigma_{i,n}^2} - w_{ij,n}}{\sigma_{i,n}^4} = 0 \quad (3.15)$$

Thus

$$\begin{aligned}
\sum_{j=1}^n \frac{w_{ij,n} d_{ij}^2}{\kappa \sigma_{i,n}^2} &= \sum_{j=1}^n w_{ij,n} \\
\sum_{j=1}^n w_{ij,n} d_{ij}^2 &= \kappa \sigma_{i,n}^2 \sum_{j=1}^n w_{ij,n} \\
\sigma_{i,n}^2 &= \frac{\sum_{j=1}^n w_{ij,n} d_{ij}^2}{\kappa \sum_{j=1}^n w_{ij,n}}
\end{aligned} \quad (3.16)$$

In the second case, $\lambda > 0$, thus $\sigma_{i,n}^2 = 0$. Hence, we need to have a test to make sure that the scale does not become zero. This can be done by checking whether the value of λ is non zero, in

this case, we set the scale to the initial value (i.e. $\sigma_{i,n}^2 = \sigma_0^2$). The value of λ is derived from

$$\sum_{j=1}^n \frac{\frac{w_{ij,n}d_{ij}^2}{\kappa\sigma_{i,n}^2} - w_{ij,n}}{\sigma_i^4} - \lambda = 0 \quad (3.17)$$

Thus

$$\lambda = \sum_{j=1}^n \frac{\frac{w_{ij,n}d_{ij}^2}{\kappa\sigma_{i,n}^2} - w_{ij,n}}{\sigma_i^4} \quad (3.18)$$

Given the previous scales, $\sigma_{i,n-1}^2$, the new scale that optimizes (3.6) after the arrival of the n^{th} data point, can also be rewritten as follows, which explicitly shows the penalizing effect of the forgetting mechanism on the previous information via the weight decay expression in (3.2),

$$\begin{aligned} \sigma_{i,n}^2 &= \frac{\sum_{j=1}^n w_{ij,n}d_{ij}^2}{\kappa \sum_{j=1}^n w_{ij,n}} \\ \sigma_{i,n}^2 &= \frac{\sum_{j=1}^{n-1} w_{ij,n}d_{ij}^2 + w_{in,n}d_{ij}^2}{\kappa \left(\sum_{j=1}^{n-1} w_{ij,n} + w_{in,n} \right)} \\ \sigma_{i,n}^2 &= \frac{\sum_{j=1}^{n-1} (e^{-\frac{1}{\tau}} w_{ij,n-1})d_{ij}^2 + w_{in,n}d_{ij}^2}{\kappa \left(\sum_{j=1}^{n-1} (e^{-\frac{1}{\tau}} w_{ij,n-1}) + w_{in,n} \right)} \\ \sigma_{i,n}^2 &= \frac{e^{-\frac{1}{\tau}} \left(\sum_{j=1}^{n-1} w_{ij,n-1}d_{ij}^2 \right) + w_{in,n}d_{ij}^2}{\kappa \left(e^{-\frac{1}{\tau}} \left(\sum_{j=1}^{n-1} w_{ij,n-1} \right) + w_{in,n} \right)} \\ \sigma_{i,n}^2 &= \frac{e^{-\frac{1}{\tau}} \left(2\sigma_{i,n-1}^2 \sum_{j=1}^{n-1} w_{ij,n-1} \right) + w_{in,n}d_{ij}^2}{\kappa \left(e^{-\frac{1}{\tau}} \left(\sum_{j=1}^{n-1} w_{ij,n-1} \right) + w_{in,n} \right)} \\ \sigma_{i,n}^2 &= \frac{\kappa e^{-\frac{1}{\tau}} \left(\sigma_{i,n-1}^2 W_{i,n-1} \right) + w_{in,n}d_{ij}^2}{\kappa \left(e^{-\frac{1}{\tau}} W_{i,n-1} + w_{in,n} \right)} \end{aligned}$$

□

Similar to the centroid update equation, the first term in the numerator (and denominator) represents the sum of the contributions of all the previous data points $(x_1, \dots, x_{n-1})$, and this contribution is penalized by the forgetting factor. The second term represents the new information obtained from

the new data point x_n .

3.1.4 The objective function as an M-estimator

The objective function in 3.6 can be written as an M-estimator (See (2.22)), as follows

$$\max_{c_i, \sigma_i^2} \left\{ J = \sum_{j=1}^N \sum_{i=1}^K \rho(r_{ij}^2; c_i, \sigma_i^2) \right\} \quad (3.19)$$

where

$$r_{ij}^2 = \frac{-d_{ij}^2}{\kappa \sigma_i^2} = \frac{-(x_j - c_i)^t (x_j - c_i)}{\kappa \sigma_i^2} \quad (3.20)$$

$$\rho(r_{ij}^2; c_i, \sigma_i^2) = \frac{e^{\frac{-d_{ij}^2}{\kappa \sigma_i^2}}}{\sigma_i^2} \quad (3.21)$$

The parameters can be estimated by setting the derivative of the ρ function to zero as follows

$$\frac{\partial J}{\partial c_i} = \sum_{j=1}^N \psi(r_{ij}^2, c_i) = 0 \quad (3.22)$$

$$(3.23)$$

where

$$\begin{aligned} \psi(r_{ij}^2, c_i) &= \frac{\partial \rho(r_{ij}^2, c_i)}{\partial c_i} \\ &= \frac{\partial \rho(r_{ij}^2, c_i)}{\partial r_{ij}} \times \frac{\partial r_{ij}}{\partial c_i} \\ &= \frac{-2r_{ij}}{\kappa \sigma_i^2} \times \frac{e^{-r_{ij}^2}}{\sigma_i^2} \times \frac{-1}{\sigma_i^2} \\ &= \text{constant} \times r_{ij} \times e^{-r_{ij}^2} \end{aligned} \quad (3.24)$$

To interpret the objective function as a W-estimator, the robust weight can be obtained by

$$\begin{aligned} w(r_{ij}) &= \frac{\psi(r_{ij})}{r_{ij}} \\ &= \text{constant} \times e^{-r_{ij}^2} \end{aligned} \quad (3.25)$$

which is a Welsh estimator (see Table 2.3) (Rousseeuw, 1987). Having established that the

clustering process performs robust estimation through the stream on multiple clusters, we conclude that the cluster centroid estimation benefits from the same advantages as the Welsch estimator in terms of its resistance to outliers.

3.1.5 Detecting outliers using Chebyshev bounds

Outliers are a common nuisance in raw data sets, and they can be due to many reasons such as Human error, machine error, or the randomness of a few data points that follow no cluster. Detecting outliers is a very challenging task in data mining, and is even more challenging in mining data streams. This is because in stream data mining, the data points are processed only once, and since there is no control over the flow of data, a data point that is flagged as an outlier at the beginning of the data stream might turn out to be part of a cluster that evolves later in the data stream lifetime.

The proposed algorithm, RINO-Streams (Hawwash & Nasraoui, 2010), is resistant to outliers because its objective function is rooted in robust statistics (by virtue of using an objective function that resists outliers using robust weights) as shown in Section 3.1.4, and not in standard non-robust estimation methods that make rigid assumptions about the distribution of the data. An outlier is defined as a data point that does not belong to any of the existing clusters (i.e. not in their influence zone) and that does not form any cluster with other points. If the point is determined to be an outlier with respect to all existing clusters, then it will create a new cluster with the point itself being the centroid. This newly created cluster will be allowed a grace period, a_{mature} , and if after this threshold, it is still weak (it has a density less than a threshold δ_{min}), then it will be considered an outlying cluster and will be deleted.

The Chebyshev bound, discussed in Section 3.1.1, will allow us to design an outlyingness test for any new data point with respect to cluster C_i with significance probability $1/t^2$. The rearranged Chebyshev inequality in (3.8) can be applied directly on the robust weight as follows:

$$Pr \left\{ e^{\frac{-|Y-\mu|^2}{\kappa\sigma^2}} \leq e^{\frac{-t^2}{\kappa}} \right\} \leq \frac{1}{t^2} \quad or \quad Pr \left\{ w_{ij} \leq e^{\frac{-t^2}{\kappa}} \right\} \leq \frac{1}{t^2}, \quad (3.26)$$

which means that if the robust weight w_{ij} of data point j with respect to cluster C_i is less than the constant value of $e^{\frac{-t^2}{\kappa}}$, then point j is considered an outlier with respect to cluster C_i with a

significance probability of $\frac{1}{t^2}$. This is formalized in the following definition:

Definition 3.1.7. *Outlier with Chebyshev probability $\frac{1}{t^2}$: the data point x_j is an outlier with respect to cluster C_i at time n with a significance probability of $\frac{1}{t^2}$ if:*

$$w_{ij,n} < e^{\frac{-t^2}{k}} \quad (3.27)$$

3.1.6 Cluster merging and splitting

The detected clusters in a real data stream typically evolve over time, thus besides giving more importance to newer data points, the online clustering algorithm should detect when two or more clusters become more similar to each other in order to merge them. Similarly, a cluster can become too diffuse and split into one or more sub-clusters.

To handle the merging of two clusters, RINO-Streams uses the Chebyshev bound to design a compatibility test for merging clusters C_i and C_k . This is done by checking their mutual Chebyshev bounds (i.e. testing if each cluster's centroid can be considered as an outlier with respect to the other cluster) with significance probability $\frac{1}{t^2}$: Given the distance d_{ik} between the centroids c_i and c_k , then using (3.8), the clusters are merged if none of them is found to be an outlier with respect to the other cluster, i.e.,

$$d_{ik}^2 < t^2 \sigma_i^2 \text{ \& } d_{ik}^2 < t^2 \sigma_k^2 \quad (3.28)$$

which means that the centroid c_i is not an outlier, and thus is inside the influence zone of cluster C_j with significance probability equal to $1 - \frac{1}{t^2}$. The same condition applies to centroid c_j with respect to cluster C_i .

When clusters C_i and C_k are merged, the centroid of the new cluster becomes a weighted average of the two centroids as follows,

$$c_{new,n} = \frac{c_{i,n} \sum_{j=1}^n w_{ij,n} + c_{k,n} \sum_{j=1}^n w_{kj,n}}{\sum_{j=1}^n w_{ij,n} + \sum_{j=1}^n w_{kj,n}}, \quad (3.29)$$

and the new scale is also a weighted average as follows,

$$\sigma_{new,n}^2 = \frac{\sigma_{i,n}^2 \sum_{j=1}^n w_{ij,n} + \sigma_{k,n}^2 \sum_{j=1}^n w_{kj,n}}{\sum_{j=1}^n w_{ij,n} + \sum_{j=1}^n w_{kj,n}} \quad (3.30)$$

Equations (3.29) and (3.30) preserve the optimal equations for the centroid and scale, i.e. (3.12) and (3.13) respectively, given the combination of points that contributed to each cluster before merging them, with only one assumption: that the old weights of the data points in the respective clusters are not very different from the new weights in the new merged clusters, an assumption that is reasonable given the similarity between the two clusters.

The new age, a_{new} , for the new cluster is set as the maximum of the ages, a_i and a_k , of the merged clusters, i.e. $a_{new} = \max(a_i, a_k)$, while the new sum of weights (soft cardinality) is simply the sum of the old sum of weights of the two merged clusters C_i and C_k , i.e. $W_{new} = W_i + W_k$.

Splitting clusters in RINO-Streams occurs naturally and does not require any special treatment. A cluster split occurs when points from one cluster bifurcate by evolving in two or more different directions, and hence their weights with respect to the original centroid would start decreasing to the point where they start being considered outliers, which continues until they form their own new clusters.

3.1.7 Handling arbitrary-shaped clusters

RINO-Streams uses the Euclidean distance in its objective function, and hence, results in hyper-spherical clusters. To detect clusters with arbitrary shapes, RINO-Streams uses an offline step that is invoked upon the request of the user, and which recursively tests for the compatibility between every pair of clusters using a more relaxed version of the Chebyshev test in (3.26). Clusters that are found to be “neighbors” are linked together, and a set of “linked” clusters are considered to be one real cluster. The relaxed version of this Chebyshev test is defined as follows:

$$d_{ik}^2 < \alpha t^2 \sigma_i^2 \text{ \& } d_{ik}^2 < \alpha t^2 \sigma_k^2 \quad (3.31)$$

where α is a constant greater than 1.

3.1.8 The complete RINO-Streams algorithm

Following the update equations (3.13) and (3.12), RINO-Streams updates the cluster parameters with the arrival of a new non-outlying data point incrementally, and keeps as a summary of the data stream only the centroid ($c_{i,n}$), scale ($\sigma_{i,n}^2$), sum of weights ($W_{i,n} = \sum_{j=1}^n w_{ij,n}$) and the age a_i for each cluster. Moreover, a test is added to eliminate weak clusters whose density falls below a minimum threshold (δ_{min}) and are mature enough (i.e. $a_i > a_{mature}$). The complete steps of RINO-Streams are listed in Algorithm 14.

The input parameters to RINO-Streams include the maximum number of clusters K_{max} which is a higher bound on the allowed number of clusters and is needed to control the memory space used to store the cluster models, the initial scale σ_0 which is assigned to the newly created cluster, the density threshold δ_{min} which is used to ensure that only good clusters with high density are kept, the maturity age a_{mature} which provides a newly created cluster with a grace period before testing its density quality, the time decay τ which sets the forgetting factor $e^{\frac{-1}{\tau}}$ that controls the decay of the data points' weights over time and the Chebyshev bound constant t that is used in (3.26) and (3.31) to set the significance probabilities of the test.

Algorithm 14 RINO-Streams

Input: Maximum number of clusters (K_{max}), Initial scale (σ_0), density threshold (δ_{min}), maturity age (a_{mature}), forgetting factor (τ), Chebyshev Bound constant (t)

Output: Cluster model after n points $\zeta = C_1 \cup C_2 \dots \cup C_K$, where $C_i = (c_{i,n}, \sigma_{i,n}^2, a_i, W_{i,n})$

$K = 0$

FOR $n = 1$ TO N **DO**

    //single pass over the data stream of size N

    Compute the distances, d_{in}^2 , and robust weights $w_{in,n}$ (Definition 3.1.2) between x_n and clusters $C_i, \forall i = 1, \dots, K$

IF $K < K_{max}$ And x_n is an outlier with respect to all clusters in ζ (Definition 3.1.4)

        // Create a new cluster centered on x_n

$K = K + 1$

$c_K = x_n$ //centroid

$\sigma_K = \sigma_0$ //initial scale

$a_K = 0$ //initial age

$W_K = 1$ //initial sum of robust weights

$\delta_K = \frac{1}{\sigma_0^2}$ // initial density

END IF

FOR each cluster $C_i, \forall i = 1, \dots, K$

        //Update the compatible clusters when x_n is not an outlier

IF x_n is NOT an outlier with respect to cluster i

            Update $c_{i,n}$ using (3.12)

            Update $\sigma_{i,n}$ using (3.13)

            Update sum of weights: $W_{i,n} = e^{-\frac{1}{\tau}} W_{i,n-1} + w_{in}$

            Update density $\delta_{i,n} = \frac{W_{i,n}}{\sigma_{i,n}^2}$

END IF

        Update age $a_i = a_i + 1$

END FOR

FOR each pair of clusters $C_i \& C_k, \forall i, k = 1, \dots, K$

IF C_i and C_k are Chebyshev-compatible using equation (3.31)

            Merge clusters C_i and C_k using equations (3.29) and (3.30)

END IF

END FOR

FOR each cluster $C_i, \forall i = 1, \dots, K$

IF ($a_i > a_{mature}$) & ($\delta_i < \delta_{min}$) //remove mature clusters that have low density

$\zeta = \zeta - C_i$

$K = K - 1$

END IF

END FOR

END FOR

3.1.9 Complexity

For each new data point, RINO-Streams computes the distance and the weights with respect to all the clusters in ζ , which is done in linear steps. Since the clustering model is updated incrementally,

nothing is recomputed from scratch, and hence the computational complexity of RINO-Streams is $O(NK^2)$ where N is the size of the data stream and K is the upper bound on the number of clusters (which is a very small value compared to the size of the data stream, N). Note that the K^2 term is due to the pairwise-cluster compatibility tests for merging, and could be reduced to K by performing these pairwise tests only after every K data points have arrived instead of after each data point. Another way to reduce the complexity is by checking the pairwise compatibility only within local neighborhoods confined to surroundings of the cluster in which the current data point has landed. Moreover, the memory requirements of RINO-Streams are linear with the number of clusters, because at any point in time, only the cluster model properties $(c_i, \sigma_i, W_i, a_i)$ are kept in addition to the most recent data point. The memory requirements at time n (i.e. after the arrival of n data points in the stream) can be written as

$$M(n) = (3 + d) \times B \times K_n, \quad (3.32)$$

where B is the number of bytes needed to store one value (for simplicity, we assume that all cluster model properties are stored using the same number of bytes), d is the number of dimensions in the data, and K_n is the number of clusters at time n . The first term consists of the three scalar values (σ_i, W_i, a_i) and the d dimensions of each centroid vector c_i .

3.1.10 Compliance with data stream clustering requirements

Below, we show how RINO-Streams meets all the requirements of data stream clustering algorithms, that were discussed in Section 2.2.1.

1. Requirement 1: Compactness of representation:

Each cluster is represented using four components: The centroid c_i which is a vector of size equal to the dimensionality of data, the scale σ_i , the sum of weights W_i and the age a_i . Hence, the total memory requirement of the clustering model is given in (3.32), which is a very compact representation compared to the original data stream.

2. Requirement 2: Fast processing of new data points:

The second requirement is also met since each new data point is compared against the existing clusters in a linear time $O(K)$, and is thereafter discarded.

3. Requirement 3: Fast handling of outliers:

If the data point was determined to be an outlier (see Definition 3.1.7), then it is used to create a new cluster and this verification is also done in linear time with the number of clusters.

4. Requirement 4: Integration of offline and online data:

The fourth requirement suggests the ability to store the clustering model offline and to access it easily, which is also met in RINO-Streams because the clustering model is very compact and can be stored in main memory. However, if needed, the clustering model at different time steps can also be stored offline in secondary memory, and can be easily accessed.

5. Requirement 5: Presenting the discovered clusters instantly:

The cluster representatives can be used directly to plot (in the case of 2 or 3-D) the clustering model as a set of hyper-spheres centered at the cluster centroid and with influence area equal to the cluster scale parameter. For high dimensional data, alternative formats are typically used, such as the term or item frequencies within a cluster, for the case of text or transactional data.

6. Requirement 6 :Making no assumptions about the number of clusters:

RINO-Streams does not assume a number of clusters in advance. Instead, it only requires a maximum number of clusters which is also used to control the memory space used.

7. Requirement 7: Handling evolution:

RINO-Streams handles the evolution of data by the very definition of its “dynamic” robust weight in (3.1), which uses a forgetting mechanism to give more emphasis to the newer data points, thus allowing adaptation to the changing nature of the data stream.

3.1.11 Comparison with related work

Table 3.1a compares the proposed online clustering algorithm, RINO-Streams, against some of the competing algorithms (which were discussed in Section 2.2.2.2), while Table 3.2b explains the meaning of the symbols used in Table 3.1a. The following conclusions can be drawn from the comparison.

- All algorithms provide an explicit way to detect outliers with the exception of CluStream,

since it maintains a constant number of micro-clusters. Hence, it might mislabel outliers as valid micro-clusters.

- Both RINO-Streams and TRAC-Streams have a low complexity (which depends on the number of points (N) as well the number of clusters (K)). On the other hand, the remaining algorithms have a complexity that includes the number of data points (N) as well as the number of micro-clusters (M_C), which is typically a much higher value than the number of regular clusters (K).
- RINO-Streams, DenStream and D-Stream are the only algorithms that can detect clusters with arbitrary shape, . Handling such clusters is done in a very simple step in RINO-Streams, as discussed in Section 3.1.7. This step adds a complexity
- CluStream is the only algorithm that requires specifying the number of clusters in advance, in contrast to the sixth requirement of stream clustering algorithms (Section 2.2.1).
- The size of each cluster in RINO-Streams and TRAC-Streams is small ($1d + 3$) compared to the rest of the algorithms. Moreover, the number of cluster they maintain (C) is much smaller than the number of micro-clusters (M_C) which is maintained by the rest of the algorithms. Hence, the memory requirement for RINO-Streams and TRAC-Streams is small.
- RINO-Streams and TRAC-Streams are less sensitive to the model assumption and data distributions, thanks to the use of robust statistics in their objective functions via the robust weights.

3.2 The TRACER Algorithm

The second component of the proposed framework, called TRACER (**TR**acking and validating **Cluster Evolution** using **R**egression analysis), aims at tracking the detected clusters' evolution over time by building and maintaining a summarizing regression model for each cluster property (e.g. scale). In the case of RINO-Streams, the output at each instant is a set of clusters ζ , each characterized by a set of parameters (c, σ, W, δ) . Thus for each one of these parameters, a linear regression model is built and maintained incrementally over a time window consisting of the last Δ_{Reg} values

Table 3.1: Comparison between RINO-Streams and other stream clustering algorithms

(a) Comparison

Property	RINO-Streams	DenStream	CluStream	D-Stream	TRAC-Streams
Detecting Outliers	yes	yes	no	yes	yes
Complexity of online clustering	$O(NK^2)$	$O(NM_C)$	$O(NM_C^2)$	$O(NM_C K)$	$O(NK^2)$
Complexity of offline clustering	none	$O(M_C^2)$	$O(KM_C d)$	$O(M_C)$	none
Detecting arbitrary-shaped clusters	no	yes	no	yes	no
Requiring specification of No. Clusters	no	no	yes	no	no
Requiring an offline step	no	yes (upon request)	yes (upon request)	yes (run periodically)	no
Memory requirements	$(1d + 3) * K$	$(2d + 1) * M_C$	$S * M_C * (2d + 3)$	$(1d + 4) * M_C$	$(1d + 3) * K$
Needing special Initialization (typically Batch clustering)	no	Run DBSCAN on initial set of pts.	Run K-Means on initial set of pts.	no	no
Built-in forgetting mechanism for old data	yes	yes	no	no	yes
Built-in robustness to noise	yes	no	no	no	yes
Reference	(Hawwash & Nasraoui, 2010)	(Cao <i>et al.</i> , 2006)	(Aggarwal <i>et al.</i> , 2003)	(Chen & Tu, 2007)	(Nasraoui & Rojas, 2006)

Symbol	Meaning
K	No. of clusters
N	No. of data points
M_C	No. of micro-clusters (or grids in the case of D-Stream) ($M_C \gg K$)
S	No. of snapshots
d	No. of dimensions

(b) Symbol meanings

(i.e. every time Δ_{Reg} data points arrive, new regression models are built or old ones are updated). These linear regression models can be used to reconstruct the clusters' evolution discovered over the entire lifetime of a data stream as well as build some *behavioral* profiles of clusters, without having to store the entire history of the cluster models, corresponding to each timestamp of the data stream.

The rest of this section is organized as follows: Section 3.2.1 describes the types of metrics that are tracked, then Section 3.2.2 discusses how the regression models are built for each one of these metrics. Section 3.2.3 shows how major deviations in the behavior of the tracked metrics are detected (thus corresponding to milestones of evolution in the data stream). Section 3.2.4 presents how the cluster behavior is quantified, and the method of detecting the internal and external cluster transitions is explained in Section 3.2.5. The full algorithm for TRACER is listed in Section 3.2.6 and its computational and memory complexities are discussed in Section 3.2.7. Section 3.2.8 provides an example of using TRACER. Finally, Section 3.2.9 analyzes the differences between TRACER and other change detection approaches for the clustering problem.

3.2.1 Tracked Cluster Metrics

TRACER aims at analyzing the behavior of stream clusters over time by observing a set of metrics associated with each cluster. These metrics could be generic (i.e. computed using the results of the clustering algorithm) or algorithm-specific, and they must be incrementally maintained. A metric could be (i) a cluster descriptor or (ii) a validation measure. In the proposed framework, we use two generic cluster descriptors (cardinality and scale) and one validity metric (density) that can be generated from most stream clustering algorithms, and are defined as follows.

Definition 3.2.1. *Cardinality* : given a cluster C_i and the membership or weight w_{ij} of the point x_j into cluster C_i , the soft cardinality W_i can be computed as the sum of memberships of all points $X : x_1, x_2, \dots, x_N$ with respect to C_i , i.e. $W_i = \sum_{j=1}^N w_{ij}$ where N is the number of data points seen so far. The memberships can be soft in $[0, 1]$ as in 3.1 or hard in $\{0, 1\}$ as in (Aggarwal et al., 2003).

Definition 3.2.2. *Scale*: given a cluster C_i , the scale σ_i relates to the size of the influence area of C_i within the space of the data, i.e. data points that belong to the cluster are enclosed within an area of size proportional to its scale.

Table 3.2: Stream Clustering Algorithm Metrics for Cluster C_i

Algorithm	CluStream (Statistics based) (Aggarwal <i>et al.</i> , 2003)	D-Stream (Grid based) (Chen & Tu, 2007)
Cardinality	number of points n_i	sum of density coefficients D_i
Scale	$\sqrt{\frac{CFI_i}{n_i} - \left(\frac{CFI_i}{n_i}\right)^2}$	No. of neighboring density grids G

For example, RINO-Streams estimates the scale using (3.13). Other ways to compute the scale are shown in Table 3.2.

Definition 3.2.3. *Density:* given the scale σ_i and cardinality W_i of cluster C_i defined above, the density δ_i can be defined as the ratio between the cardinality and the squared scale, i.e. $\delta_i = \frac{W_i}{\sigma_i^2}$. A larger density value is desired since it means a higher cardinality (i.e. more data points) within a smaller scale (i.e. compact cluster).

The metrics described above can be obtained directly from most of the algorithms that satisfy our requirements (Section 3.1) or by simple operations as shown in Table 3.2 for two stream clustering algorithms.

3.2.2 Building Cluster Regression Models

For each detected cluster, a set of regression models is built and maintained throughout the lifetime of the cluster (i.e. until it disappears or merges with other clusters). The independent variable X in (2.39) for all the regression models is the *timestamp* (i.e. the index of arrival of data points) and the dependent variable Y in (2.39) is one of the metrics used to describe the cluster. For example, RINO-Streams computes the *centroid* (c), *scale* (σ), *sum of robust weights* (W), and *density* (δ) for each cluster, and these can serve as the dependent variable Y being modeled. The regression models in (2.42) are found by estimating the regression coefficients ($\hat{\beta}_0$ and $\hat{\beta}_1$) using (2.41) at intervals of time called *Regression Windows* each having a width Δ_{Reg} which is domain-dependent, and reflects the desired details in the description of the evolution of clusters. The regression window should be relatively small to justify the use of a linear model instead of a more complex model for regression.

In real life scenarios, the metrics might not follow a linear model, hence the regression models could be extended to higher-order models. However, this would result in *(i)* increased time complexity (i.e. more complicated optimization methods), *(ii)* increased memory complexity (since

more than two coefficients would be needed for the model), and **(iii)** more complications in defining milestones (that represent major deviation in the metrics' behavior (Section 3.2.3)). It is important however, that the regression window be relatively small to provide an accurate modeling using a linear model instead of a more complex model for regression.

The outputs of regression analysis are the linear regression models, defined below, for each of the cluster metrics, which can be used to examine the behavior of the cluster during the lifetime of the data stream.

Definition 3.2.4. Metric Regression Model: For metric P_i ($P_i \in \{'Cardinality', 'Scale', 'Density'\}$) describing cluster C_i in the time period $[t, t+1]$ of size Δ_{Reg} data samples, the metric regression model is $\Xi_{P_i, [t, t+1]} = \{\beta_{0, P_i, [t, t+1]}, \beta_{1, P_i, [t, t+1]}\}$.

3.2.3 Detecting Milestones

As the regression models are built throughout the unraveling of the data stream, the stored regression coefficients are used to model or summarize the behavior of each cluster over time, and possibly to predict future cluster behavior or detect any *deviation* from that behavior. These deviations are detected automatically based on the angle between two consecutive linear regression models (Figure 3.2), and the times at which these deviations occur are called *milestones*.

Definition 3.2.5. Milestone: Given the regression models $\Xi_{P_i, T1}$ and $\Xi_{P_i, T2}$ at two consecutive time periods $T1 = [z_1 \Delta_{Reg} + 1, z_2 \Delta_{Reg}]$ and $T2 = [z_2 \Delta_{Reg} + 1, (z_2 + 1) \Delta_{Reg}]$ where $z_1, z_2 \in \mathbb{Z}$, $z_1 < z_2$; if the angle θ between $\Xi_{P_i, T1}$ and $\Xi_{P_i, T2}$ is larger than a threshold θ_{max} , then a milestone $M_{P_i[t]}$ is detected at time $t = z_2 \Delta_{Reg}$ (i.e. between $T1$ and $T2$).

The angle θ can be found using the tangents of the slopes of the models, i.e. $\beta_{1, P_i, T1}$ and $\beta_{1, P_i, T2}$, as follows:

$$\tan(\theta) = \frac{\max(\beta_{1, P_i, T1}, \beta_{1, P_i, T2}) - \min(\beta_{1, P_i, T1}, \beta_{1, P_i, T2})}{1 + \beta_{1, P_i, T1} \beta_{1, P_i, T2}} \quad (3.33)$$

If no milestone was detected between $T1$ and $T2$, then an updated *combined* model $\Xi_{P_i, T1+T2}$ can be obtained by simple calculations (the proof can be found in Appendix A). The only memory overhead is to store a single value $\bar{y}_{P_i, T1}$, which is the sum of the values of the tracked cluster metric from the previous time period $T1$, and this value is updated incrementally.

Figure 3.2: Consecutive Time Periods (T1& T2)

Milestones represent important phases in the lifetime of a cluster, because they represent stages when major changes took place either in the structure of the cluster (i.e. internal changes) or in its relationship with other clusters (i.e. external changes). Each time that a *milestone* is detected, a new regression model is built (since it reflects a new behavior that needs to be captured), otherwise the old regression model needs only be updated to reflect the changes that took place.

3.2.4 Monitoring the behavior of clusters

Monitoring the regression models can help build *behavioral* profiles for each cluster. The behavioral profile reflects how each of the cluster properties has evolved over time. More specifically, we are interested in whether the cluster's behavior is *changing* or *stabilizing*. To quantify this behavior, we will use the slope of the regression model, β_1 in (2.42) and its confidence interval $CI_{\beta_1, \alpha}$ in (2.46) (an observed interval of the reliability of estimating β_1 with $(1 - \alpha)\%$ confidence). Using the confidence intervals CI provides a more reliable and flexible test for stability, because we are only interested in a plateau-like regression line and not necessarily a strict plateau.

Definition 3.2.6. *Stability:* Given a metric P_i with regression model $\Xi_{P_i, [t, t+1]} = \{\beta_{0, P_i, [t, t+1]}, \beta_{1, P_i, [t, t+1]}\}$ and its α -level regression slope's confidence interval $(CI_{\beta_{1, P_i}, \alpha})$ over the time period between two consecutive milestones $[M_{P_i[t]}, M_{P_i[t+1]}]$, the metric is stable if $0 \in CI_{\beta_{1, P_i}, \alpha}$, and is expressed using the stability status indicator $P_{i, [M_{P_i[t]}, M_{P_i[t+1]}]}^*$.

Definition 3.2.7. *Instability:* Given an unstable metric P_i and its regression model's slope, β_{1, P_i} , over the time period between two consecutive milestones $[M_{P_i[t]}, M_{P_i[t+1]}]$, the metric is considered to be increasing if $\beta_{1, P_i} > 0$ and is expressed using the stability status indicator $P_{i, [M_{P_i[t]}, M_{P_i[t+1]}]}^+$. Otherwise it is decreasing and is expressed using the stability status indicator $P_{i, [M_{P_i[t]}, M_{P_i[t+1]}]}^-$.

Definition 3.2.8. *Behavioral Profile:* a behavioral profile $\mathcal{H}_i$ of a cluster C_i is the sequence of stability status indicators of all its metrics and their regression models over all time periods.

Figure 3.3: Cluster Changes: Typical Causes and Effects

A stable metric can be seen visually as a plateau-like regression line, whereas an unstable metric is seen as increasing or decreasing. This helps to visually analyze the behavior of the clusters over time. The stability measures of the cluster descriptors (e.g. cardinality) are used to (i) infer both the internal and external changes that took place as will be described in the following section, and (ii) help in evaluating the quality of clusters along with the stability measures of the evaluation metrics (e.g. density).

3.2.5 Detecting internal and external transitions

In any online clustering algorithm, there are several changes or causes that might affect the behavior of clusters, and that in turn result in internal and/or external transitions. Figure 3.3 shows the cause and effect relationships between different events and their typical effect on the cluster metrics (i.e. cardinality, scale and density). For example, if new data points are assigned to cluster C_i , and they are close to the centroid c_i , then this will cause C_i to increase its cardinality and decrease its scale, hence increasing the density. Moreover, if a forgetting factor is used to give more emphasis to newer data points as in RINO-Streams and TECNO-Streams (Nasraoui *et al.*, 2003), the existing data points in C_i will decrease their weight or influence on C_i . Hence, if those existing points are located at the periphery of the cluster, both the scale and cardinality would be decreased.

The internal and external transitions, which extend the simpler ones defined in (Spiliopoulou *et al.*, 2006), are illustrated in Figure 3.4 and are defined as follows.

Figure 3.4: Internal & External Transitions

Definition 3.2.9. *Internal Transition:* Given a cluster C_i , it is said that it went through an internal change if one of its cluster descriptor metrics P_i was unstable (i.e. $P_{i,[M_t, M_{t+1}]}^+$ or $P_{i,[M_t, M_{t+1}]}^-$).

Internal transitions refer to the changes in the cluster descriptors, and are detected using the stability measures described in Section 3.2.4. More specifically, there could be a cluster expansion or shrinkage.

Definition 3.2.10. *External Transition:* these transitions refer to the interaction between clusters which are inferred using the internal transitions, and can be categorized into five types: survival, disappearance, appearance, splitting, and mergal.

Table 3.3: Transition Conditions and Symbols

Symbol	Description
M_t and M_{t-1}	Current milestone and previous milestone
$W_{i,[M_{t-1},M_t]}^{\{*,+,-\}}$	Stable (*), increase (+) or decrease (-) in cardinality
$\sigma_{i,[M_{t-1},M_t]}^{\{*,+,-\}}$	Stable (*), increase (+) or decrease (-) in scale between consecutive milestones
$C_i(\text{neighbor}[M_t])$	A neighboring cluster to C_i was found at milestone M_t using a threshold on distance (depends on the clustering alg.)
$C_i[M_{t-1},M_t]$	C_i has been updated between the milestones M_t and M_{t-1}
t_{start} and t_{end}	Cluster creation and deletion times
t_{min}	Age grace period

Table 3.4: Transition Characterization Rules (conjunction $\wedge$, disjunction $\vee$, negation $\neg$), sorted by the order in which they are applied

Transition	Conditions
Mergal	$W_{i,[M_{t-1},M_t]}^+ \wedge \sigma_{i,[M_{t-1},M_t]}^+ \wedge (\neg C_i[M_{t-1},M_t])$
Splitting	$W_{i,[M_{t-1},M_t]}^- \wedge \sigma_{i,[M_{t-1},M_t]}^- \wedge C_i(\text{neighbor}[M_t])$
Appearance	$t_{start} \in [M_{t-1},M_t]$
Disappearance	$t_{end} \in [M_{t-1},M_t]$
Survival-Aging	$W_{i,[M_{t-1},M_t]}^-$
Survival-Gain At Periphery	$W_{i,[M_{t-1},M_t]}^+ \wedge \sigma_{i,[M_{t-1},M_t]}^+ \wedge C_i[M_{t-1},M_t]$
Survival-Gain at center	$W_{i,[M_{t-1},M_t]}^+ \wedge \sigma_{i,[M_{t-1},M_t]}^-$
Survival-Absorption	$\left(W_{i,[M_{t-1},M_t]}^+ \wedge \sigma_{i,[M_{t-1},M_t]}^* \right) \vee \left(W_{i,[M_{t-1},M_t]}^* \wedge \neg \sigma_{i,[M_{t-1},M_t]}^* \right)$

The external transitions, for a cluster C_i taking place at the time elapsed between two milestones $[M_{t-1}, M_t]$, are inferred using the transition characterization rules described in Table 3.4, where the conditions and their symbols are given in Table 3.3.

Note that survival encompasses several possible scenarios, including, in the order of conditions listed in Table 3.4: aging of points, gaining points at the border of the cluster, densification or addition of points near the center, and absorption.

For example, if both the cardinality and scale have increased, then if the cluster was updated between M_{t-1} and M_t , this would mean that the cluster gained new data points near its border. Otherwise, the cluster must have merged with a different cluster. A similar analysis is done on the rest of the transitions.

Algorithm 15 TRACER (At time period t)

Input: Cluster metric values P_i stored after receiving the last Δ_{Reg} data points

Output: Regression Models $\Xi_{P_i, [t-1, t]}$ (Def. 3.2.4) and Behavioral Profiles $\mathcal{H}_i$ (Def. 3.2.8)

```
1. FOR each cluster  $C_i, i = 1, \dots, K$ 
2.   FOR each metric  $P_i$ 
3.     IF a milestone is detected (Def. 3.2.5)
4.       Create a new regression model for  $P_i$  using (2.41)
5.     ELSE
6.       Update the regression model for  $P_i$ 
7.     END IF
8.   END FOR
9. Find the transitions using the rules in Table 3.4
10. Update the behavioral profile  $\mathcal{H}_i$ 
11. Update Stream Genealogy graph where transitions took place
12. END FOR
```

3.2.6 The TRACER Algorithm

TRACER is invoked every time Δ_{Reg} data points have been encountered, and as an input, requires the Δ_{Reg} values of the cluster metrics. These values are temporarily stored and then discarded once TRACER is completed. The complete steps of TRACER are listed in Algorithm 15. For each of the clusters, it detects whether a milestone exists (i.e. whether the regression model of any tracked metric is significantly different from the previous model); and if a milestone is detected, then a new regression model is created and the transition characterization rules (Table 3.4) are applied for the current milestone to compare it with the previous milestone,. Otherwise, the regression model is updated to absorb the behavior of the two consecutive time intervals as described in Section 3.2.3. If transitions were detected, then the Stream Genealogy graph (which will be discussed in Section 3.4) is updated to reflect those transitions.

3.2.7 Computational and Memory Complexities

We focus only on the complexity of the second component (TRACER) since the complexity of the first component (i.e. stream clustering algorithm) depends on the algorithm used (e.g. Section 3.1.9), and this was discussed for the case of RINO-Streams and several other related stream clustering algorithms in Sections 3.1.9 and 3.1.11 .

Time Complexity The frequency of invoking TRACER depends on the size of the regression window (Δ_{Reg}). More specifically, if the length of the stream is N , then TRACER is invoked a number of times equal to the smallest integer greater than $\frac{N}{\Delta_{Reg}}$, i.e. $\left\lceil \frac{N}{\Delta_{Reg}} \right\rceil$. The time complexity of TRACER (Algorithm 15) is linear and equal to $O(\bar{K} \times |P|)$ where $\bar{K}$ is the average number of clusters through time and $|P|$ is the number of metrics describing the cluster. Thus the total time complexity is:

$$T(N) = O\left(\bar{K} \times |P| \times \frac{N}{\Delta_{Reg}}\right) \quad (3.34)$$

Note that TRACER does not involve the data dimensionality in its calculations since it maintains the regression models of the cluster metrics which are scalar values. The transition characterization rules (Table 3.4) require finding the neighborhood of a cluster using a distance function (which would be affected by dimensionality), however, most stream clustering algorithms provide this as a by product at no additional cost (i.e. finding the closest cluster in RINO-Streams). For this reason, we do not consider this component to be part of TRACER's complexity.

Memory Complexity Capturing the evolution information of clusters may raise the concern of a memory complexity overhead, however using regression analysis will significantly reduce this overhead. If we do not use regression analysis, then in order to track the cluster evolution, the average number of values that need to be stored for each of the metrics would be the product of the length of the data stream (N) and the average number of clusters detected throughout the data stream ($\bar{K}$), i.e. $O(\bar{K} \times |P| \times N)$, where $|P|$ is the number of metrics describing the cluster. On the other hand, when using regression modeling, only two values (regression coefficients β_1 and β_0) are needed to be stored for each of the metrics at the end of each interval (Δ_{Reg}), which leads to a reduction of $\Delta_{Reg}/2$ in needed memory space. Furthermore, since new regression models are calculated only when a milestone is detected and are updated otherwise, as discussed in Section 3.2.3, and assuming that clusters do not go through radical deviations every time that regression analysis is performed, the memory needed to store the regression coefficients will be further reduced. The memory complexity of TRACER is given in (3.35).

$$Memory = O(NumOfMilestones \times \bar{K} \times |P|) \quad (3.35)$$

3.2.8 TRACER Example

Before going further with discussing other aspects of TRACER, we will present a motivational example that shows the main functionality of TRACER.

Figures 3.5 (a) and (b) show the input to TRACER and the final output for two metrics respectively. In this example, there are four time periods each of length $\Delta_{Reg} = 5$. In other words, Stream-Dashboard processes five points and stores the metric values and then calls TRACER. The final output is obtained as follows:

1. After T1, TRACER finds two regression models for each of the metrics as shown in Figure 3.6 (a). Since this is the first regression model, we skip the milestone detection. After finding the regression models, the metric values are discarded.
2. After T2, TRACER also finds two regression models for the metrics as shown in Figure 3.6 (b).
3. TRACER performs the milestone detection step between time periods T1 and T2, and it detects two milestones for each metric as shown in Figure 3.6 (c). Hence, it keeps both regression models.
4. After T3, TRACER finds the regression models as shown in Figure 3.7 (a). Then it performs the milestone detection step as shown in Figure 3.7 (b). No milestone was detected, in this case, since both regression models have similar angles (3.33). Hence, TRACER merges the two regression models into one combined model.
5. Finally, after T4, TRACER finds the regression models as shown in Figure 3.7 (c), and then finds that there are milestones. The final output is shown in Figure 3.5 (b).

Figure 3.5: TRACER Example (Source and final output)

Figure 3.6: TRACER Example (T1 & T2)

Figure 3.7: TRACER Example (T3 & T4)

(a) T3 (Regression Model)

(b) T3 (Milestone Detection)

(c) T4 (Regression Model)

As shown in this example, TRACER can track the behavior of the cluster metrics with only simple regression models. Besides finding the milestones, TRACER reduced the memory complexity needed to track the behavior. More specifically, each metric’s behavior is represented using only six values (two coefficients for each of the three models) versus keeping 20 values (five metric values for each time period), thus gaining 70% savings compared to no modeling.

Furthermore, monitoring the behavior of the final regression model in Figure 3.5 (b) , as discussed in Section 3.2.4, would result in the following observations:

1. Metric 1 first decreases, then it increases and finally stabilizes.
2. Metric 2 first increases, then stabilize and finally decreases.

3.2.9 Comparison with related work

Table 3.5 (a) compares the proposed second component, TRACER, against other approaches for detecting changes in clusters. More specifically, we compare against the methods following the local approach as described in Section 2.3.1 since TRACER is considered a local approach.

Table 3.5 (b) explains the meaning of the symbols used. Note that the time complexity (second row) refers to the complexity of running the method itself and not how many times it is called, while the memory requirements refer to the overall memory needed over the lifetime of the data stream.

The following conclusions can be drawn from the comparison:

- TRACER is the only approach that validates the detected cluster on the fly. This is done via the behavioral profiles built for each cluster’s metric.
- TRACER has the lowest complexity, which is linear with the number of clusters (K) and number of metrics that are tracked for each cluster (P).
- TRACER and MONIC can detect the internal and external clusters’ changes that take place over time, while the other algorithms can detect either the internal or external changes. However, TRACER provides more detailed transitions compared to MONIC.
- PAM is not generic, and this limits its applicability to only a few domains.

- TRACER requires much less memory than MONIC, FOCUS and the Evolving Subspace methods, which must all store the N_K data points (on average) belonging to the clusters being tracked. The reason for TRACER's lower memory complexity is that the number of tracked metrics in TRACER is much lower than the size of each cluster (i.e. $P \ll N_K$) and the number of detected milestones is also much lower than the number of snapshots, as discussed in Section 3.2.3 (i.e. $M \ll S$).
- MONIC requires pre-processing the data before being able to detect changes, which might not be possible in the context of data streams.
- TRACER is the only algorithm that *incrementally* tracks the changes. On the other hand, the rest of the algorithms require re-clustering the data at each timestamp, which limits the scalability of these approaches for data streams.

3.3 Configuration Adaptation

The behavioral profiles of the most stable clusters can eventually help build behavioral profiles for 'good' and 'bad' clusters which will help in two main aspects: first, it could reduce the sensitivity associated with using some threshold parameters to judge the quality of the clusters (e.g. δ_{min} in RINO-Streams) by adjusting those threshold parameters based on the behavioral profiles of 'good' clusters. Second, it could provide valuable feedback about the initial parameter values (when a new cluster emerges) and allow us to automatically adapt those parameters over the lifetime of the data stream instead of keeping them constant. Using the *behavioral* profiles to judge cluster quality and to update the initial parameters would help close the loop of the knowledge discovery process, leading to better adaptation.

3.4 Stream Genealogy Graph

Stream-Dashboard provides the means to track the behavior of stream clusters using low memory requirements. However, real streams of data are infinite by definition, hence, it is impossible to keep the rich information generated by Stream-Dashboard in the main memory. To tackle this problem,

Table 3.5: Comparison between TRACER and local change detection techniques

(a) Comparison between TRACER and local change detection methods

Property	TRACER	PAM (Baron <i>et al.</i> , 2003)	MONIC (Spiliopoulou <i>et al.</i> , 2006)	FOCUS (Ganti <i>et al.</i> , 1998)	Stream-Detect (Gaber & Yu, 2006)	Evolving Subspace Clustering (Günemann <i>et al.</i> , 2011)
Online unsupervised cluster validation	yes	no	no	no	no	no
Complexity of evolutionary tracking	$O(K * P)$	$O(K^2)$	$O(K^2)$	$O(K^2)$	$O(K^2)$	$O(2^d * K * N_K)$
Detecting internal & external changes	both	internal	both	internal	internal and domain	external
Requires an offline step	no	no	no	no	yes	no
Generic (can use any clustering algo.)	yes	no	yes	yes	yes	yes
Memory Requirements	$P * K * M$	$S * H$	$S * K * N_K$	$S * K * N_K$	$S * H$	$S * K * N_K$
Requires a pre-processing phase	no	no	yes	no	no	no
Incremental or Re-clustering of data	incremental	re-cluster	re-cluster	re-cluster	re-cluster	re-cluster

Symbol	Meaning
K	No. of clusters
M	No. Milestones
N_K	Size of each cluster
P	No. of tracked metrics for each cluster
S	No. of snapshots
d	No. of dimensions
H	No. of heuristics/values stored

(b) Symbol meanings

we propose to keep the entire history of the stream in a graph, called *the Stream Genealogy*, and to store it in secondary memory.

Definition 3.4.1. *Stream Genealogy: A directed acyclic graph $G(V, E)$ where each node $v_i \in V$ represents the cluster C_i and consists of a unique ID, C_i 's most recent metrics (i.e. P_i), activity status flag $I \in \{\text{active}, \text{inactive}, \text{outlier}\}$, starting age t_{start} , ending age t_{end} (the time when the cluster became inactive (i.e. $I = 0$)), and C_i 's behavioral profile $\mathcal{H}_i$, and $e(i, j) \in E$ is a directed edge from node v_i to node v_j*

Each node in the Stream Genealogy represents a unique cluster created during the lifetime of the stream. When two clusters are merged into one, a new node is created with the merged clusters becoming its children nodes. When a cluster is split, two new nodes are added with the split cluster becoming their parent. When a cluster is deleted, merged or split, it is flagged as inactive and it ceases to be updated. The Stream Genealogy is updated when major changes take place to reduce I/O operations. More specifically, it is updated in the case of creating a new cluster and in cluster merging or splitting.

We designed a special tool to visualize the Stream Genealogy in an interactive manner that shows the complete evolution of the stream over time. The tool plots the graph in a 2D plane where:

- the x-value of each node reflects its creation time,
- each node's color reflects its status (active vs. inactive/defunct cluster),
- the nodes can be re-sized based on their metrics (e.g. a bigger node means that its cardinality is larger than other nodes),
- the graph can be filtered based on the node status and starting age (thus we can opt to hide inactive clusters which were created before time t),
- double-clicking each node brings up the cluster properties as well as the historical behavioral profiles of the cluster and all its children,
- the active nodes are labeled with the number of clusters that they represent.

The Stream Genealogy provides valuable analytical insights that enables the user to: *(i)* track the *external and internal* changes (Section 3.2.5) that have taken place in the clustering model, *(ii)* track

the behavior of each cluster by analyzing the behavioral profiles of all the previous/intermediate stage clusters that have evolved into the current cluster (e.g. if a cluster representing a trending topic on Twitter, was a result of a merging, then we can examine the cardinality changes - i.e. popularity - of the original trending topics that merged together if they had different behavior), (iii) evaluate the quality of the clustering algorithm when using different parameters (e.g. the use of a forgetting factor changes the number of inactive clusters in RINO-Streams).

3.5 Complete Generic Framework for Stream Cluster Tracking and Validation

One of the main advantages of the proposed framework is its *generic nature* in the sense that *any* online clustering algorithm can be used in the first component. The only requirement is that the online clustering algorithm generates a set of cluster properties such as centroid and scale, and not just the assignment of points to the clusters. Having a general framework makes it domain-independent and can be easily applied to any knowledge discovery application. This framework thus introduces a new perspective of knowledge discovery which focuses on the evolution and interaction of the clusters through time, rather than just discovering those clusters and assigning data points to them.

3.5.1 Stream-Dashboard Pseudo-Code

Algorithm 16 lists the major steps in Stream-Dashboard. The first component of the framework, the online clustering algorithm (e.g. RINO-Streams), is invoked when a new data point or batch of points are available and it updates the clustering model. If a new cluster is created, then it is added to the Stream Genealogy. Periodically, after the arrival of increments corresponding to multiples of Δ_{Reg} points, Stream-Dashboard invokes the second component, TRACER described in Section 3.2, which builds and maintains the regression models of the clusters' behavior. It should be noted here that in the case where the clustering algorithm does not explicitly merge or split clusters (e.g. (Ester *et al.*, 1998)), merging can be detected using the Transition Characterization Rules in Table 3.4.

Algorithm 16 Stream-Dashboard Pseudo-Code

Input: Data Stream $X = \{x_j = (x_j^1, \dots, x_j^d), \forall j = 1, \dots, N\}$

Output: Clustering Model $\zeta = \{C_i, \forall i = 1, \dots, K\}$, Regression Models Ξ_P and Behavioral Profiles ($\mathcal{H}$)

```
1. FOR  $j = 1$  to  $N$  // Loop through the data stream  $X$ 
2.    $OnlineClustering(x_j)$  // Call an online clustering algorithm such as RINO-Streams
3.   Update clustering model  $\zeta$ 
4.   IF a new cluster is created
5.     Update Stream Genealogy (Section 3.4)
6.   END IF
7.   IF  $mod(j, \Delta_{Reg}) = 0$  // Whenever a multiple of  $\Delta_{Reg}$  (the size of the Regression Window) data
   points have been encountered
8.      $TRACER(P_\zeta)$ ; //  $\Delta_{Reg}$  metric values for each cluster in  $\zeta$  (Algorithm 15)
9.   END IF
10. END FOR
```

3.5.2 Time Complexity

The time complexity of Stream-Dashboard depends on the complexity of the first two components (i.e. online clustering algorithm and TRACER). The configuration adaptation component consists of ad-hoc rules that take place in constant time as well as updating the Stream-Genealogy. Hence, the time complexity is equal to the time complexity of TRACER (Section 3.2.7) and the time complexity of the clustering algorithm.

If RINO-Streams is used as the online clustering component within Stream-Dashboard, then the time complexity is:

$$\begin{aligned} T(N) &= O(N \times \bar{K}) + O\left(\bar{K} \times |P| \times \frac{N}{\Delta_{Reg}}\right) \\ &= O(\bar{K} \times |P| \times N) \end{aligned} \quad (3.36)$$

where N is the stream length, $\bar{K}$ is the average number of clusters and $|P|$ is the number of tracked metrics per cluster.

3.5.3 Memory Complexity

The memory complexity of Stream-Dashboard also depends only on the first two components, since the Stream-Genealogy graph is stored offline.

If RINO-Streams was used in Stream-Dashboard, then the running memory complexity is:

$$\begin{aligned}
M(N) &= O((3+d) \times B \times \bar{K}) + O(\text{NumOfMilestones} \times \bar{K} \times |P|) \\
&= O(\text{NumOfMilestones} \times \bar{K} \times |P|)
\end{aligned} \tag{3.37}$$

where d is the dimensionality and B is the number of bytes needed to store a single value. Note that d and B are constants, hence, they do not affect the overall memory complexity.

3.6 Visualization Dashboard

The proposed framework provides the user with a visualization dashboard, hence the name Stream-Dashboard, that provides an analytical and visual view of the evolution of the detected clusters in the data stream over time. Using visualization aids the user in making decisions and conclusions about the behavior of the data stream. Currently, we provide these visual graphs at the end of the knowledge discovery, however in the future, we plan to present the graphs in real time during the knowledge discovery process and to construct videos of the clusters' evolution through time.

3.6.1 Visualizing Regression Models, Evolution Milestones and Cluster Merging

For each of the tracked cluster metrics, we plot the corresponding regression models over the lifetime of the data stream. Moreover, the evolution milestones, discussed in Section 3.2.3, are marked using a circle on the regression line, and each merging of two clusters, discussed in Section 3.1.6, is marked using an 'x'. These plots show how each clusters' metric is changing over time, and identifies the significant phases of the cluster evolution as evolution milestones or as cluster merging. Figure 3.8 shows an example of plotting the density regression models of a synthetic dataset with eight clusters, where the data points were presented one cluster at a time. It can be seen that the density starts increasing, then it stabilizes after some time, and the milestones were only detected during the periods of increase in density (i.e. true change).

3.6.2 Stability

Detecting the behavior of the cluster metrics was discussed in Section 3.2.4, where the behavior was quantified using the confidence interval of the regression coefficients. To visualize the quantified

Figure 3.8: An example of density regression models of 8 clusters

behavior, we plot the stability of the property for each cluster over the lifetime of the data stream where an increase is plotted as the value 1, a stable behavior as the value 0, and a decrease as the value -1. Figure 3.9 shows an example of plotting the stability of the density of the same dataset from Figure 3.8. Comparing the two Figures shows that the stability plot can accurately detect the behavior of density over time.

3.6.3 Stream Genealogy

The Stream Genealogy graph, discussed in Section 3.4, is stored offline to keep a history of the external changes that have occurred as a result of the interaction between clusters. Figure 3.10 shows all the nodes of the Stream Genealogy where the size of the nodes is based on the density of the clusters. The x-axis value for each node represents the creation time of each cluster, and the edges represent the merging transitions between clusters. Furthermore, the nodes are colored based on their status: a red node is an outlier, a green node is active while a blue node is inactive. The graph can be filtered, for example, to hide the outliers or to change the size of the nodes based on their cardinality or scale. Moreover, double clicking each node results in displaying the cluster

Figure 3.9: An example of density stability plots for 8 clusters

metric values associated with that node at that time, as well as the behavioral profiles of its children.

The Stream Genealogy provides a compact view of the entire data stream thus supporting a deep analytical study of the interactions between the clusters throughout the stream's lifetime. For example, when analyzing Twitter datasets we could answer questions such as:

1. What are the current trending topics (green nodes)?
2. What is the popularity of each topic (re-sizing nodes based on density)?
3. Which topics were merged into bigger topics or split into more specific topics? and when?

3.6.4 Evaluation of the Visualization Dashboard

At this stage of development, the proposed visualization dashboard is not meant as a visualization tool in the artistic sense. In other words, it is not meant to simply visualize the datasets or the mined clustering model. Rather, it actually performs a second level of *mining* the *quantitative* metrics of the detected clusters. Consequently, the mined clusters can then be validated objectively in a quantitative well-controlled experimental fashion. This can be done by generating synthetic

Figure 3.10: Stream-Genealogy Graph

datasets (or adapting real datasets with known ground truth) with clusters that are generated to evolve according to well designed temporal scenarios. For example, a cluster can be designed such that it changes its metrics (e.g. centroid and scale) at certain timestamps. Given these controlled datasets, evaluating the visualization dashboard could be done as follows:

1. Calculating the number of correctly detected milestones, as well as incorrect or missed ones. This accuracy could be quantified using classical validation metrics such as recall and precision.
2. Finding the accuracy of the estimated clusters' metrics at each milestone. This can be done using the evaluation metrics discussed in Section 2.1.6.
3. Finding the efficiency of summarizing the clusters' evolution behavior using the proposed framework (i.e. using milestones and regression models) in terms of memory usage.

3.7 Summary and Conclusions

In this chapter, we presented the a new framework, called Stream-Dashboard, for clustering and tracking evolution in data streams. First, we presented the first component, RINO-Streams (Section 3.1), which is an online clustering algorithm that incrementally updates a clustering model by optimizing the density of clusters. Section 3.2 then presented the second component, TRACER, which consists of a set of statistical tests and rules to detect cluster transitions, and uses regression analysis to build compact linear regression models of the clusters' metrics evolution over time. TRACER allows the analysis of the evolution of clusters at any point in time, while reducing the memory complexity, and it can detect important times, called *milestones*, when the clusters undergo major changes. The third component, discussed in Section 3.3, exploits the regression models, built in the second component, to improve the quality of the online clustering algorithm. Section 3.4 presented the Stream-Genealogy graph, which is a compact acyclic directed graph that stores the entire history of the evolution of the data stream offline. Section 3.5 presented the complete Stream-Dashboard framework with a particular emphasis on one of the main contributions, which is the fact that it is generic in the sense that any online clustering algorithm can be used in the first component. Section 3.6 presented the visual capabilities of the proposed framework to track and validate the evolution of clusters over time. Next, in Chapter 4, we will discuss the experiments conducted to validate the quality of the clustering models obtained using RINO-Streams, as well as the effectiveness of tracking the evolution of the clusters over time using TRACER.

CHAPTER 4

EXPERIMENTAL RESULTS

In this chapter, we will present some of the empirical results that have been obtained from the proposed framework. Section 4.1 performs a thorough evaluation of the online clustering component, RINO-Streams, against four other competing online clustering algorithms: TRAC-Streams (Nasraoui & Rojas, 2006), IncDBSCAN (Ester *et al.*, 1998), CluStream (Aggarwal *et al.*, 2003) and Growing K-Means (Lowette & Laerhoven, 2004). Section 4.2 evaluates the performance of the second component, TRACER. Section 4.3 presents an application of the proposed framework to a real world data stream in the context of the Twitter social media site; and finally, Section 4.4 presents the summary and conclusions of the experiments.

Because we propose a generic framework consisting of two main components, namely a component for online clustering and a component for tracking and validating evolving clusters, our experiments span a variety of formats and settings. In particular, we performed extensive experiments to test each component on its own and both components in combination. As part of component 1, we performed an extensive evaluation of the proposed clustering algorithm (RINO-Streams). Because clustering is inherently unsupervised (no external labels) and the results being possibly hard to evaluate, we performed several categories of experiments, such that some of them serve to both evaluate and illustrate what the clustering algorithm is doing with the data. For that purpose, many of the data sets had to be 2-dimensional to allow visual inspection of the data and results without ambiguity. For extensive evaluation on varying data set properties (dimensionality, noise, number of clusters, cluster densities, etc) and for varying streaming scenarios, we had to resort to several validity metrics and statistical analysis of the results. Finally, we illustrate the use of Stream-Dashboard in the context of discovering and tracking story clusters from twitter, a real life data stream application.

Table 4.1 summarizes the experiments according to their different categories and purposes.

4.1 Evaluation of Component 1: RINO-Streams

In this section, we will evaluate the performance of the first component of Stream-Dashboard, which is the online clustering algorithm RINO-Streams (Section 3.1). We will start by presenting some of the preliminary experiments conducted when comparing against TRAC-Streams (Nasraoui & Rojas, 2006) and IncDBSCAN (Ester *et al.*, 1998). These experiments aim at analyzing the performance using a small set of synthetic datasets. Then we perform thorough comparisons between RINO-Streams, CluStream (Aggarwal *et al.*, 2003) and Growing K-Means (Lowette & Laerhoven, 2004) on synthetic and real datasets in Sections 4.1.3 and 4.1.4, respectively. The effectiveness of handling cluster splitting and merging is tested in Section 4.1.5. The scalability of RINO-Streams is demonstrated in Section 4.1.6 on big data streams. Finally in Section 4.1.7, we present a sensitivity analysis of the performance of RINO-Streams with respect to the data stream properties as well as the RINO-Streams parameter values.

4.1.1 Experimental Settings

4.1.1.1 Datasets

Synthetic Data Streams To emulate an infinite data stream in a typical real world scenario, we used the random Radial Basis Function (RBF) data stream generator provided as part of the MOA stream data benchmarking framework ¹ (Bifet *et al.*, 2010). Massive Online Analysis (MOA) is an open source framework specifically designed to analyze massive streams of data. MOA includes a collection of stream classification and clustering algorithms, as well as a collection of synthetic data stream generators. Using the data stream generators allows us to control all the aspects of a data stream to mimic realistic data streams. For example, we can control the frequency of merging clusters. Moreover, we added a functionality to control the order of the data. For example, we can generate data points from one cluster at a time or randomly from all clusters. The parameters that control the RBF generator and their descriptions are listed in Table 4.2.

---

¹<http://moa.cms.waikato.ac.nz/>

Table 4.1: Roadmap to the Experiments

Section	Main algorithm tested/Purpose	Comparison baseline	Data sets	Evaluation Metrics
4.1.2.1	RINO-Streams	TRAC-Streams	Synthetic 2D: DS5, DS8, DS16	Silhouette, DB Index, Similarity matrix, error in: # of clusters, centroids, scales, detected noise proportion
4.1.2.2		IncDBSCAN		
4.1.2.3		TRAC-Streams& IncDBSCAN		
4.1.3		ClusStreams, Growing K-Means	8640 MOA Synthetic Datasets (2 to 10D, up to 1B records): varied data set properties + stream properties	Silhouette, DB Index, NMI, Purity, Recall, F1, Time
4.1.4.1			6 Real Text Datasets (5K to 10K Dimensions): varied	
4.1.4.2			Real Network Intrusion Dataset (KDD Cup 99)	
4.1.5			Synthetic 2D data with controlled merging and splitting	
4.1.6	RINO-Streams: ANOVA & Pareto Front		MOA Synthetic Datasets: 100K - 1TB	Testing Splitting and Merging
4.1.7.1			8640 MOA Synthetic Datasets (2 to 10D, up to 1B records): varied data set properties + stream properties	Testing scalability of online clustering
4.1.7.2				Sensitivity Analysis & ANOVA - Effect of Algorithm Parameters on: Silhouette, DB Index, NMI, Purity, Recall, F1
4.2.1.3	TRACER: ANOVA		Synthetic Benchmark generated by interpolation of linear regression models	Sensitivity Analysis & ANOVA - Effect of Stream Properties on: Silhouette, DB Index, NMI, Purity, Recall, F1
4.2.2	TRACER		Synthetic 2D (DS6), Reuters news text (Re0), KDD Cup 99 (intrusion stream)	Pareto Efficiency based on Silhouette, DB Index, NMI
4.2.3.1			Synthetic 2D (DS1A)	R^2 -statistic for regression modeling, Milestone Detection Quality: Precision, Recall, F1
4.2.3.2			Synthetic 2D (DS1, DS2B)	Track Cluster Behavior
4.2.4			Synthetic 2D (DS5)	Detecting Internal Transitions
4.2.5		MONIC	Synthetic 2D (DS2A)	Detecting External Transitions
4.2.6		Baseline (Equi-width)	Synthetic 2D (DS6)	Validate by observing density regression models
4.2.7	TRACER: Sensitivity and Complexity Analysis		Synthetic Benchmark generated by interpolation of linear regression models - varied TRACER parameters (Regression window size & Angle Threshold)	Validating Detection of External Transitions
4.2.8	Stream-Dashboard Final Visualizations		Synthetic 2D (DS6, DS2B, DS10, DS2)	Memory Savings for outputs of evolution summaries and R^2 for regression quality
4.3	Stream-Dashboard Application to Twitter		Twitter Streams	R^2 -statistic for regression models, Milestone Detection Quality: Precision, Recall, F1, Time and Memory Complexity
				TRACER evolution summary outputs: regression models & milestones, Stream Genealogy & ClusTree, Stability plots.
				Illustrating the entire system at work
				TRACER evolution summary outputs: Cluster Tag Clouds, regression models & milestones, Stream Genealogy

Table 4.2: RBF Data Stream Generator Parameters (Bifet *et al.*, 2010).

Parameter	Description
Stream Length	Number of data points generated in the data stream
No. Clusters	Number of random centroids
No. Dimensions	Number of Dimensions
No. Clusters Range	Deviation of the number of centroids in the model
Radii	The average radii of the centroids in the model
Density Range	Offset of the average weight a cluster has. A value of 0 means all clusters contain the same amount of points
Speed	Kernels move a predefined distance of 0.01 every X points (X is the speed)
Noise	Noise level
Event Frequency	Frequency of events taking place (i.e. merging/splitting or emerging/disappearance)

The RBF generator works as follows: first, a fixed number of random centroids are generated where each centroid has a random position, standard deviation and weight. Second, data points are generated by selecting a random centroid (while taking the weights into consideration so that higher weight centroids are more likely to be chosen). Third, the attribute values of the data point are displaced from the centroid using randomly generated offsets (i.e. using a Gaussian distribution based on the standard deviation of the centroid). Following this approach, RBF generates a continuous data stream following a normally distributed hypersphere.

Real Datasets We will use several real text datasets and one network activity dataset. The real text datasets are taken from the CLUTO toolkit ² (Karypis, 2002) and are derived from the TREC collection ³. The KDD Cup 99 dataset⁴ represents network activity traces, collected over a period of nine weeks of normal activity interspersed with various attacks and intrusions simulated in a military network environment . We used the training dataset with the 33 continuous features. There are a total of 23 different attacks that fall into four main categories.

The properties of the real datasets are listed in Table 4.3. The balance is the ratio of the smallest cluster (in terms of number of points) to the largest cluster.

²<http://glaros.dtc.umn.edu/gkhome/fetch/sw/cluto>

³<http://trec.nist.gov>

⁴<http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

Table 4.3: Real Text and Network Intrusion Detection Data Set Descriptions

Dataset	Source	Num. Points	Num. Dimensions	Num. Classes	Mean Class Size	Balance
tr11	TREC	414	6424	9	46	0.0455
tr12	TREC	313	5799	8	39	0.0968
tr23	TREC	204	5831	6	34	0.0659
tr31	TREC	927	10127	7	132	0.0057
tr41	TREC	878	7453	10	87	0.037
tr45	TREC	690	8261	10	69	0.0875
KDD CUP 99	KDD CUP 99	494021	33	23	21479	0.000007

4.1.1.2 Evaluation Metrics

As discussed in Section 2.2.3, evaluating data stream clusters is done at predefined periods of time since the data stream keeps evolving over time, hence, the final clustering model does not necessarily represent earlier data points. We performed the evaluation every 10% of the stream length, and then we computed the average, minimum and maximum metric values.

We compared the results using the following three internal validity metrics described in Section 2.1.6.1:

- Silhouette index,
- Davies-Bouldin index,
- Similarity matrix.

In addition to the internal validity metrics, since class or cluster labels are provided with the data sets, we computed the four external validity metrics, described in Section 2.1.6.2, namely,

- Normalized Mutual Information,
- Purity,
- Recall
- F1 Score.

In addition we computed the following four external evaluation metrics which are clustering model-oriented, hence comparing the output clusters to the ground-truth generating cluster parameters:

Table 4.4: RINO-Streams Parameter Values

Parameter	Description	Values
$\frac{K_{max}}{K_G}$	The maximum number of clusters allowed (K_{max}) as a percentage of the real number of clusters in the ground truth (K_G).	50%, 100%, 150%
σ_0	Initial Scale	0.05, 0.1
$\frac{\tau}{ X }$	Optional forgetting lifetime (τ) as a percentage of the data stream length ($ X $)	0% , 10%, 20%
$\frac{1}{t_{outlier}}$	Chebyshev constant for outlier detection	0.05, 0.1
$\frac{1}{t_{merging}}$	Chebyshev constant for the cluster merging test	0.05, 0.1
W_{min}	The minimum sum of weights (W_{min}) which affects the minimum density threshold value ($\delta_{min} = \frac{W_{min}}{\sigma_0}$)	5, 10
$\frac{a_{mature}}{ X }$	The maturity age (a_{mature}) as a percentage of the data stream length ($ X $)	1%, 2%

- the relative error of the number of detected clusters,
- the average error of the estimated centroids,
- the average error of the estimated scales,
- and the error in the detected noise percentage.

Finally, we compared the average time taken by each experiment.

4.1.1.3 Experimental Setup

For each of the algorithms, we varied the parameter values and found the best results for each dataset, then we calculated the average performance over all the datasets. To ensure a fair and realistic comparison, we initialized the algorithms (with the exception of RINO-Streams and TRAC-Streams since it is not required) with the first data points, i.e. the centroids of the first K clusters were set to the values of the first K data points.

RINO-Streams The parameters, along with their descriptions and values, are shown in Table 4.4. There is a total of 192 parameter settings.

TRAC-Streams TRAC-Streams (Nasraoui & Rojas, 2006) follows a similar approach to RINO-Streams, and has similar parameters. These parameters, along with their description and values, are shown in Table 4.5. Note that setting the forgetting factor to ∞ means that there is no forgetting.

Table 4.5: TRAC-Streams Parameters' values

Parameter	Description	Values
$\frac{K_{max}}{K_G}$	The maximum number of clusters allowed (K_{max}) as a percentage of the real number of clusters in the ground truth (K_G).	50%, 100%, 200%
σ_0	Initial Scale	0.05
$\frac{\tau}{ X }$	Optional forgetting lifetime (τ) as a percentage of the data stream length ($ X $)	15%, 40%, ∞
$\frac{1}{t_{outlier}}$	Chebyshev constant for outlier detection	0.075
$\frac{1}{t_{merging}}$	Chebyshev constant for the cluster merging test	0.075
W_{min}	The minimum sum of weights (W_{min}) which affects the minimum density threshold value ($\delta_{min} = \frac{W_{min}}{\sigma_0}$)	5, 20, 50
a_{mature}	The maturity age (a_{mature})	100

Table 4.6: CluStream Parameters

Parameter	Description	Values
$\frac{K_{macro}}{K_G}$	The number of macro clusters (K_{macro}) as a percentage of the real number of clusters in the ground-truth (K_G).	50%, 100%, 150%
$\frac{K_{micro} \times K_{macro}}{K_G}$	The number of micro clusters (K_{micro}) as a percentage of the number of macro clusters (K_{macro})	50%, 100%
$t_{recency}$	Threshold used to delete micro clusters when a new micro cluster is created	10, 20

CluStream The implementation of CluStream, provided by the MOA framework (Bifet *et al.*, 2010), was used in these experiments. CluStream (described in Section 2.2.2.2) incrementally updates a set of micro-clusters and generates the final clusters (i.e. macro-clusters) using K-means. The number of micro-clusters is usually higher than the final number of generated macro-clusters. In contrast, RINO-Streams incrementally maintains the final clusters, which are considered the equivalent of CluStream's resulting macro clusters. Hence, to make a fair comparison, the evaluation metrics for CluStream are generated from the macro-clusters that are found using K-means at the end of each evaluation time period. The CluStream parameters, along with their description and values, are shown in Table 4.6

Growing K-Means We implemented the Growing K-Means algorithm as described in Algorithm 10. Growing K-Means requires three input parameters: the number of clusters, distance threshold and an initial scale. These parameters, along with their description and values, are shown in Table 4.7.

Table 4.7: Growing K-Means Parameters

Parameter	Description	Values
$\frac{K}{K_G}$	The number of clusters (K) as a percentage of the real number of clusters in the ground truth (K_G).	50%, 100%, 150%
σ_0	The initial scale value used when creating a new cluster	0.05, 0.1, 0.2
$t_{distance}$	Threshold used to determine if a data point belongs to a cluster	0.1, 0.2, 0.3

IncDBSCAN IncDBSCAN (Ester *et al.*, 1998) is another density-based online clustering algorithm that requires two parameters: *(i)* the minimum number of points in the neighborhood and *(ii)* the distance threshold ε .

4.1.1.4 ANOVA Sensitivity Analysis

ANOVA tests the null hypothesis that the means of the observations in each group (i.e. having the same value for the parameter) is the same for each parameter, versus the hypothesis that they are not. If the p-value of the F-measure is less than α (we set $\alpha = 10\%$) then the null hypothesis is rejected and the observation means are different (i.e. the parameter has a significant affect).

ANOVA can be done in two ways: 1-Way ANOVA and N-Way ANOVA. 1-way ANOVA is done when only one factor is considered while other factors are considered constant, whereas N-Way ANOVA is done when all the factors are changing. The results with 1-way and N-Way might generate different F-statistics, which is the ratio between the variance between items and the variance within items. The reason is that the number of degrees of freedom for the error decreases (since there are more factors to consider), hence, the variance between items increases causing the F-statistic to increase.

4.1.2 Initial Experiments

In this section, we present some of the preliminary results obtained from comparing RINO-Streams against TRAC-Streams and IncDBSCAN .

We used three synthetic datasets generated by our own synthetic RBF generator: **DS5** has five clusters, **DS8** has eight clusters and **DS16** has 16 clusters. For each dataset, nine variations were created by adding different percentages of noise and by changing the order of data arrival. We will be using a brief code that describes the experiments obtained with these dataset variations. The

first part of the code is the name of the algorithm used, the second letter reflects the number of true clusters, C_x where x is the number of true clusters, then followed by the order of the points arrival (O : ordered one cluster at a time then followed by the noise if any, R_2 : random points from two clusters at a time followed by noise, R : completely random from all clusters and noise). The final part of the code describes the percentage of noise added as N_y where y is the percentage of noise. For example $RINO - C_5RN_{20}$ denotes an experiment where we ran RINO-Streams on the data set that consists of 5 clusters and 20% of random noise added, with all points presented in random order. $TRAC - C_5RN_{20}$ denotes an experiment where we ran TRAC-Streams (Nasraoui & Rojas, 2006) on the same data set, and $DBSCAN - C_5RN_{20}$ denotes an experiment where we ran IncDBSCAN (Ester *et al.*, 1998) on this same data set.

Finding the external evaluation metrics for both RINO-Streams and TRAC-Streams requires assigning each data point to one of the clusters, and to do that, we took the clustering model at the end of the clustering process and used the Chebyshev test in (3.26) to assign data points to the clusters: all the points that pass the Chebyshev test are considered part of the cluster with a significance probability of $1 - \frac{1}{t^2}$.

IncDBSCAN does not have the notion of centroids or scales, instead, a cluster is represented as a set of the data points that are connected via their density relations. Hence, we cannot use some of the internal validity measures that use the centroids or scales such as the Davies-Bouldin index.

Since both RINO-Streams and TRAC-Streams have similar parameters, we used the TRAC-Streams parameter values shown in Table 4.5. When comparing against IncDBSCAN, we picked the best results for both methods and compared them using different evaluation metrics, because both algorithms do not share similar parameters. To find the best performance of IncDBSCAN, we tried different values for $MinPts$ from 1 to 10, and chose the best value, then we plotted the sorted k-dist graph for each run and chose the best Eps value as recommended in (Ester *et al.*, 1996). Table 4.8 shows the values of $MinPts$ and Eps for the different datasets. To improve the visibility of the figures, Table 4.9 defines the x-axis indexes used in Figures 4.1-4.12, that present the different values for the parameters $(\frac{K_{max}}{K_G}, \frac{\tau}{|X|}, W_{min})$ when evaluating the performance of RINO-Streams against TRAC-Streams. Table 4.10 explains the x-axis index values used in Figures 4.13-4.16 that show the results when comparing RINO-Streams against IncDBSCAN.

Table 4.8: DBSCAN's Optimal Parameters Values

	MinPts	Eps
DS5	6	0.029
DS8	3	0.02
DS16	4	0.01

Table 4.9: The X-axis index values used in Figures (4.1-4.12)

X-axis Index	1	2	3	4	5	6
$(\frac{K_{max}}{K_G}, \frac{\tau}{ X }, W_{min})$	50,40,20	50,40,50	50,Inf,20	50,Inf,50	100,40,20	100,40,50
	7	8	9	10	11	12
	100,Inf,20	100,Inf,50	200,40,20	200,40,50	200,Inf,20	200,Inf,50

Table 4.10: The meaning of the X-axis index values used in Figures (4.13-4.16)

X-axis Index	1	2	3	4	5	6	7	8	9
DS8 Variations	C_8ON_0	C_8ON_{12}	C_8ON_{24}	$C_8R_2N_0$	$C_8R_2N_{12}$	$C_8R_2N_{24}$	C_8RN_0	C_8RN_{12}	C_8RN_{24}
DS16 Variations	$C_{16}ON_0$	$C_{16}ON_9$	$C_{16}ON_{18}$	$C_{16}R_2N_0$	$C_{16}R_2N_9$	$C_{16}R_2N_{18}$	$C_{16}RN_0$	$C_{16}RN_9$	$C_{16}RN_{18}$

4.1.2.1 RINO-Streams vs TRAC-Streams

To compare the proposed algorithm against TRAC-Streams using the synthetic datasets, we plotted for each dataset (DS5, DS8 and DS16) the output of the evaluation metric values (except the similarity matrix) for all variations of the dataset (i.e. with different noise percentage and order of data point arrival). To make the figures more visible, we are going to show the results using three values for K_{max} (50%, 100% and 200%), only two values for τ (40% and ∞) and two values for W_{min} (20 and 50). Moreover, we divided the results for each evaluation metric into three sub-figures based on the order of the data points arrival:

1. the first sub-figure reflects the results when the data arrived in order (i.e. one cluster at a time),
2. the second sub-figure reflects a random arrival of two clusters at time, and
3. the third sub-figure reflects a complete random arrival of the data points.

We will present only the results of processing DS8 and DS16, noting that similar results were obtained for DS5, which was actually less challenging because of the better separation of its clusters.

Figures (4.1-4.5) show the results for DS8 where the x-axis represents the different configurations $(\frac{K_{max}}{K_G} \%, \frac{\tau}{|X|} \%, W_{min})$ and its values can be found in Table 4.9. Figure 4.1 shows the difference

in the number of correctly detected clusters as a ratio of the true number of clusters. When K_{max} was less than the actual number of clusters, some clusters were missed which is normal. However RINO-Streams was able to detect most of the clusters for other configurations except when noise was present and the W_{min} was too strict which is seen the Figure 4.1(c). In contrast, TRAC-STREAMS missed more clusters except when data arrived in order as in Figure 4.1(a). This proves that the proposed algorithm is more resistant to the presence of outliers and is less sensitive to the order of data arrival.

Figure 4.1: RINO-Streams vs TRAC-Streams: DS8 (difference in number of clusters detected)

Figure 4.2 shows the error in the estimated centroid, and it can be seen that both methods fare well here even though some clusters were missed, which proves that at least those clusters that were found were accurate and the order of arrival did not affect the accuracy of estimation. The error in

the estimated scale in Figure 4.3 shows that RINO-Streams results in scales that are smaller than those of TRAC-Streams (we are showing the average value of the absolute difference), because RINO-Streams is more resistant to noise. Having a lower Chebyshev bound significance threshold is expected to result in a better (larger) estimate of the scale, but at the risk of causing very close clusters to be merged, hence there is a trade-off between having better scale estimation and risking to merge close clusters. One remedy to investigate is using two different Chebyshev significance probabilities: one for the outlier detection test, and another for the merging test.

Figure 4.2: RINO-Streams vs TRAC-Streams: DS8 (average centroid error)

Figure 4.3: RINO-Streams vs TRAC-Streams: DS8 (average scale error)

The error in estimated noise in Figure 4.4 shows that RINO-Streams provides a much better characterization of noise, and is less sensitive to the order of data arrival. Although the first four values show an over-estimated noise rate, we remind the reader that this is due to setting the maximal allowed number of clusters, K_{max} , to a value smaller than the true number of clusters. Hence, all the data points belonging to these missed clusters end up being considered as noise.

Figure 4.4: RINO-Streams vs TRAC-Streams: DS8 (difference in estimated noise)

Figures 4.5 and 4.6 show the Davies-Bouldin index and Silhouette coefficient respectively. Both algorithms perform well in terms of the quality of the partition, with RINO-Streams performing better in all cases, which proves that the discovered clusters are compact and well-separated. An overall conclusion can be drawn that RINO-Streams is more resistant to noise and less sensitive to the different parameter configurations, compared to TRAC-Streams .

Figure 4.5: RINO-Streams vs TRAC-Streams: DS8 (Davies-Bouldin Index)

Figure 4.6: RINO-Streams vs TRAC-Streams: DS8 (Silhouette Index)

The results for the more challenging dataset DS16 are shown in Figures 4.7-4.12. Similar analysis and conclusions could be made as in the results for DS8, where RINO-Stream's performance is at least as good as TRAC-Streams and even better in most cases. Figure 4.7 shows that RINO-Streams missed one or two clusters when the order of the data changes, whereas TRAC-Streams missed much more clusters due to the order of data and the contamination rate.

(a) Data (ordered arrival)

(b) Data (random arrival, 2 clusters)

(c) Data (random arrival)

127

Figure 4.8: RINO-Streams vs TRAC-Streams: DS16 (average centroid error)

Figure 4.9: RINO-Streams vs TRAC-Streams: DS16 (average scale error)

Figure 4.10 further shows that RINO-Streams is better in detecting noise, while Figures 4.11 and 4.12 reflect the good quality of the clusters for both algorithms by showing the Davies-Bouldin index and Silhouette index, respectively.

Figure 4.10: RINO-Streams vs TRAC-Streams: DS16 (difference in estimated noise)

Figure 4.11: RINO-Streams vs TRAC-Streams: DS16 (Davies-Bouldin Index)

Figure 4.12: RINO-Streams vs TRAC-Streams: DS16 (Silhouette Index)

4.1.2.2 RINO-Streams vs IncDBSCAN

In this Section, we will compare the quality of clustering of the proposed algorithm with that of IncDBSCAN. The IncDBSCAN parameter values are listed in Table 4.8, and they were chosen because they resulted in the best performance. For the proposed algorithm, we set the value of $K_{max} = 200\% * K_G$, $\tau = 40\% * |X|$ and $W_{min} = 20$.

Figures 4.13-4.16 show the results of processing DS8 and DS16, where the x-axis corresponds to the different dataset variations (i.e. different noise percentages and order of data arrival) as described in Table 4.10. Figure 4.13 shows the error in the number of clusters detected, where it can be seen that RINO-Streams outperforms IncDBSCAN in the presence of noise, which further proves the robustness of RINO-Streams to outliers and the effective use of the notion of scale in

Figure 4.13: RINO-Streams vs IncDBSCAN: DS8 & DS16 (relative difference in number of clusters detected)

Note: The absence of a bar means that the correct number of clusters was found

RINO-Streams. Figure 4.14 reflects the difference in the noise detected, both algorithms seem to be good in detecting noise, however by looking back at Figure 4.13, we see that IncDBSCAN always overestimates the number of clusters in the presence of noise, which means that IncDBSCAN mistakenly assumed that most of the real outliers are valid clusters.

To evaluate the quality of the detected clusters, Figures 4.15 and 4.16 represent the average Davies-Bouldin index and Silhouette index of the clustering models respectively. And for both indexes, RINO-Streams outperforms IncDBSCAN in all cases, which means that the clusters detected using RINO-Streams are of higher quality. An overall conclusion can be drawn that RINO-Streams outperforms IncDBSCAN due to the use of an automated and robust estimation scale and using it to find a robust weight of the data points, which insures that outliers are detected accurately. Moreover, RINO-Streams dynamically estimates the location and scale of the clusters by optimizing the density. On the other hand, IncDBSCAN lacks the notion of location or scale of clusters, which makes it harder to evaluate the quality of the clusters, and it depends on optimizing the density at a local scale by the means of its input parameters (i.e. if a data point has *MinPts* in its *Eps* neighborhood, then it is considered dense).

Figure 4.14: RINO-Streams vs IncDBSCAN : DS8 & DS16 (difference in estimated noise)

Figure 4.15: RINO-Streams vs IncDBSCAN: DS8 & DS16 (Davies-Bouldin Index)

Figure 4.16: RINO-Streams vs IncDBSCAN: DS8 & DS16 (Silhouette Index)

4.1.2.3 RINO-Streams vs TRAC-Streams and IncDBSCAN

In this Section we will present some of the results when applying all three algorithms on DS8 and DS16. Figures 4.17 and 4.18 show the final clustering model and the similarity matrices when applying all three algorithms on the dataset C_8RN_{24} (i.e. dataset with eight clusters and 24% added noise and the data points arrive randomly) respectively, while Figures 4.19 and 4.20 show the results on the $C_{16}RN_{18}$ dataset. RINO-Streams had detected all the right clusters for DS8 and all the clusters except one for DS16, and the similarity matrices further prove that the clusters detected are of good quality (since the diagonal blocks are darker than the off-diagonal blocks). IncDBSCAN detected all the clusters in both datasets, however it also incorrectly assumed that a lot of the noise is valid clusters which is due to the lack of the notion of scale, which can be also seen as very small (fragmented) dark blocks in the similarity matrices in the bottom right. TRAC-Streams has detected some of the clusters in both datasets, however it missed many of them, and this shows that it is more sensitive to the order of the data points' arrival.

Figure 4.17: RINO-Streams vs TRAC-Streams & IncDBSCAN: DS8 (final output for experiment C_8RN_{24})

Figure 4.18: RINO-Streams vs TRAC-Streams & IncDBSCAN: DS8 (similarity matrices for experiment C_8RN_{24})

Figure 4.19: RINO-Streams vs TRAC-Streams & IncDBSCAN: DS16 (final output for experiment $C_{16}RN_{18}$)

Figure 4.20: RINO-Streams vs TRAC-Streams & IncDBSCAN: DS16 (similarity matrices for experiment $C_{16}RN_{18}$)

4.1.3 RINO-Streams vs CluStream vs Growing K-Means (Synthetic Data Streams)

In this section, we present the results obtained from comparing RINO-Streams against CluStream and Growing K-Means on the synthetic datasets generated by the RBF stream generator of the MOA framework.

4.1.3.1 Overall Performance

Figures 4.21 (a) and (b) show the best and average results of the internal and external validity metrics respectively. These results are obtained by first finding the best performance for each of the datasets, and then finding the best and average performance over all the datasets. Moreover, the statistical significance of the results is validated by finding the p-value as shown in Table 4.11.

The results show that RINO-Streams significantly outperforms Growing K-Means (since all p-values in Table 4.11 are less than 0.05), and slightly outperforms CluStream. Likewise, the p-values when comparing RINO-Streams and CluStream suggest that the difference is significant, with the former clearly outperforming the latter.

Table 4.11: RINO-Streams vs CluStream vs Growing K-Means: Synthetic Data (Significance)

	Purity	F1	Recall	NMI	Davies-Bouldin	Silhouette Index
RINO-Streams vs CluStream	1 e-68	1 e-5	5 e-4	1 e-5	1 e-5	5 e-56
RINO-Streams vs Growing K-Means	0	2 e-273	3 e-209	3 e-186	8 e-90	3 e-6

Figure 4.21: RINO-Streams vs CluStream vs Growing K-Means: Synthetic Data (Overall Performance)

4.1.3.2 Performance with respect to Stream Properties

We also compared the performance of RINO-Streams, CluStream and Growing K-Means with respect to some of the synthetic datasets properties (Table 4.2). Figures 4.22-4.25 show the validity metric values of the algorithms when varying the number of clusters, number of dimensions, noise level and stream length of the generated datasets respectively. For each figure, we show two internal validity metric values (Silhouette Index and Davies-Bouldin) and two external validity metric values (F1 score and Normalized Mutual Information).

Figure 4.22 shows that the performance of the all algorithms, generally, declines as the the number of clusters increases (especially based on the Silhouette Index in Figure 4.22(a)). However, RINO-Streams performance decreases only slightly with respect to the other validity metrics. Figure 4.23 shows the performance of the algorithms slightly increases as the number of dimensions increases. Figure 4.24 shows that RINO-Streams is very robust in the presence of outliers. Also Figure 4.25 shows that RINO-Streams performance is not affected much as the number of points increases.

4.1.3.3 Time Complexity

We compared the average time taken for each of the algorithms in Figure 4.26. It can be seen that RINO-Streams runs much faster than the other algorithms.

Figure 4.22: RINO-Streams vs CluStream vs Growing K-Means: Synthetic Data (Number of Clusters)

Moreover, we analyzed how the time complexity changes with respect to the data stream properties. Figures 4.27 (a), (b) and (c) show the time complexity of the algorithms when changing the number of clusters, the number of dimensions, and the stream length of the synthetic data streams. RINO-Streams maintains a stable run time regardless of the properties of the data stream, which further highlights its scalability to handle massive data streams. On the other hand, both CluStream and Growing K-Means suffer in terms of time complexity when there is an increase in the number of clusters, the dimensionality or/and the size of the data stream.

Figure 4.23: RINO-Streams vs CluStream vs Growing K-Means: Synthetic Data (Number of Dimensions)

Figure 4.24: RINO-Streams vs CluStream vs Growing K-Means: Synthetic Data (Noise Level)

Figure 4.25: RINO-Streams vs CluStream vs Growing K-Means: Synthetic Data (Stream Length)

Figure 4.26: RINO-Streams vs CluStream vs Growing K-Means: Synthetic Data (Time Complexity)

Figure 4.27: RINO-Streams vs CluStream vs Growing K-Means: Synthetic Data (Time Complexity with Respect With Data Stream Properties)

Figure 4.28: RINO-Streams vs CluStream : TREC (Normalized Mutual Information)

4.1.4 RINO-Streams vs CluStream vs Growing K-Means (Real Data Text and Intrusion Detection Data Sets)

In this section, we compare the performance of RINO-Streams against CluStream and Growing K-Means using the text datasets (TREC) and the network activity dataset (KDD CUP 99). For each of the experiments, we will show the best results obtained from varying the parameter values for each algorithm as well as the significance of the difference using their p-values. The validation metrics are the same as the ones used in the previous section for the MOA synthetic stream data.

4.1.4.1 Results for Text Datasets (TREC)

Figures 4.28-4.32 show the best validity metric values, obtained from the various parameter configurations for the normalized mutual information, cluster purity, cluster recall, F1 score and the Davies-Bouldin index, respectively. The p-values are shown in Table 4.12.

The results show that RINO-Streams significantly outperforms CluStream and Growing K-Means for all the datasets based on the normalized mutual information and the F1 score. RINO-Streams also outperforms CluStream based on F1 score and Recall. Growing K-Means slightly, but not significantly (based on p-values), outperform RINO-Stream based on the normalized mutual information. CluStream shows the best purity, however, this is due to the fact that it uses a large number of micro-clusters.

Moreover, we compared the time complexity of RINO-Streams against CluStream and Grow-

Figure 4.29: RINO-Streams vs CluStream : TREC (Cluster Purity)

Figure 4.30: RINO-Streams vs CluStream : TREC (Recall)

Figure 4.31: RINO-Streams vs CluStream : TREC (F1 Score)

Figure 4.32: RINO-Streams vs CluStream : TREC (Davies-Bouldin Index)

Table 4.12: RINO-Streams vs CluStream vs Growing K-Means for TREC text data (p-values)

	Purity	F1	Recall	NMI	Davies-Bouldin
RINO-Streams vs CluStream	0	0	0.004	0	0.122
RINO-Streams vs Growing K-Means	0.002	0	0.004	0.395	0.041

ing K-Means in Figure 4.33, and it can be shown that RINO-Streams is much faster than both algorithms, while CluStream is the slowest. This is due to the fact that CluStream maintains a high number of micro-clusters and requires running K-Means to find the macro-clusters after computing the micro-clusters.

4.1.4.2 Results for KDD CUP 99 Network Intrusion Data

Figure 4.34 shows the external validity metric results for RINO-Streams, CluStream and Growing K-Means. RINO-Streams performs better than CluStream and Growing K-Means based on the normalized mutual information, and fares well based on the other validity metrics. Figure 4.35 compares the time complexity, showing that RINO-Streams is much faster than the other algorithms.

4.1.5 Validating Cluster splitting and merging

To illustrate how clusters merge and split in RINO-Streams, we designed two experiments where one cluster evolves into three different clusters to show cluster splitting, and one where three clusters evolve into one cluster to show cluster merging. Table 4.13 lists the configuration parameter values. Figure 4.36 shows the cluster output evolution at five different time periods, where time is measured

Figure 4.33: RINO-Streams vs CluStream : TREC (Time Complexity)

Figure 4.34: RINO-Streams vs CluStream : KDD CUP 99 (External Validity Metrics)

Figure 4.35: RINO-Streams vs CluStream : KDD CUP 99 (Time Complexity)

in terms of the number of data points that arrived relative to the data stream size ($|X|$). It can be seen that one cluster (cluster number 1) is detected at the beginning, and then, as the cluster splits, two more clusters are detected. Figure 4.37 illustrates the gradual merging of three different clusters, over five different time periods, into one cluster.

Table 4.13: Parameter configurations for cluster splitting and merging

Parameter	K_{max}	τ	σ_0	$\frac{1}{t^2}$	W_{min}	a_{mature}
Merging	10	5% of $ X $	0.1	0.075	20	50
Splitting	10	2% of $ X $	0.1	0.075	20	50

Figure 4.36: A cluster that gradually splits into three clusters over time

Note: Points in turquoise are old points (their time of arrival $> \tau$)

Figure 4.37: RINO-Streams: Three clusters that gradually merge into one cluster over time

Note: Points in turquoise are old points (their time of arrival $> \tau$)

Figure 4.38: RINO-Streams: Time Complexity on Big Data Streams

4.1.6 Scalability For Big Data Streams

In this section, we will demonstrate the scalability of RINO-Streams for big data streams. We generated several data streams using the MOA RBF synthetic generator, and we used RINO-Streams to find the evolving clusters. We used the same parameter values for all the data streams, since we are mainly concerned with the time complexity with respect to the size of the data stream.

Figure 4.38 shows the time in minutes for several big data streams, where it can be seen that the complexity is linear with the number of points.

Table 4.14: RBF Parameters

Parameter	Range of values
Stream Length	1000, 5000, 10000
No. Clusters	20, 60, 80, 100
No. Dimensions	2, 5, 10
No. Clusters Range	5
Radii	0.05, 0.1
Density Range	0 , 0.1
Speed	500
Noise	0, 0.1
Event Frequency	100, 500, 1000

4.1.7 Sensitivity Analysis

In this section, we will analyze RINO-Streams' performance sensitivity with respect to the properties of the datasets as well as the parameter inputs to RINO-Streams. We will use the synthetic datasets generated by the RBF generator. The values of the different parameters controlled by the RBF generator are shown in Table 4.14. There are a total of 864 different experimental settings, and for each one of them we generated 10 datasets. Hence, we have 8640 different datasets. We used the various values in Table 4.4 for RINO-Streams which resulted in 192 different settings. For each of the datasets we found the average over all 192 different RINO-Stream settings, and then we found the average for every 10 datasets that have the same RBF settings. To evaluate the quality of the clusters, we calculated the Davies-Bouldin index, Silhouette Index and Normalized Mutual Information.

4.1.7.1 Sensitivity based on the data stream properties

In this section, we will evaluate how RINO-Streams behaves under different data stream conditions. We analyzed the results using an analysis of variance test (ANOVA) with the hypothesis that the RBF parameters do not affect the quality of RINO-Streams. If the p-value is less than α (i.e. 0.05) then the factor (i.e. one of the RBF parameters) has a significant affect on the quality of the clusters generated by RINO-Streams.

The results of performing ANOVA with $\alpha = 0.05$ are shown in Tables 4.15-4.17 for Davies-Bouldin, Silhouette index and normalized mutual information, respectively. The results show that the density range and the event frequency do not have a significant effect on the performance of

Table 4.15: ANOVA Table - RINO-Streams versus RBF Generator (Davies-Bouldin)

Source	Sum Sq. Errors.	Degrees of Freedom	Mean Sq.	F-statistic	p-value
StreamLength	143.3546	2	71.6773	24.2834	0
NoClusters	162.2716	3	54.0905	18.3252	0
KernelRadii	5.8142	1	5.8142	1.9698	0.16084
DensityRange	0.036462	1	0.036462	0.012353	0.91153
NoiseLevel	75.5654	1	75.5654	25.6007	0
EventFreq	8.7085	2	4.3543	1.4752	0.22932
NoDim	16.7089	2	8.3545	2.8304	0.059545
Error	2511.8961	851	2.9517		
Total	2924.3559	863			

Table 4.16: ANOVA Table - RINO-Streams versus RBF Generator (Silhouette-Index)

Source	Sum Sq. Errors.	Degrees of Freedom	Mean Sq.	F-statistic	p-value
StreamLength	17.0166	2	8.5083	398.3991	0
NoClusters	20.8372	3	6.9457	325.2324	0
KernelRadii	6.8067	1	6.8067	318.7203	0
DensityRange	4.0874e-005	1	4.0874e-005	0.0019139	0.96512
NoiseLevel	3.0044	1	3.0044	140.6816	0
EventFreq	0.0058908	2	0.0029454	0.13792	0.87119
NoDim	7.6515	2	3.8258	179.1409	0
Error	18.1741	851	0.021356		
Total	73.4964	863			

RINO-Streams based on all three quality measure. Moreover, the radii of the clusters and the dimensionality of the data stream do not affect the Davies-Bouldin index. All the other RBF parameters have a significant effect on RINO-Streams.

To further analyze the effect of the data stream properties on the performance of RINO-Streams, we plotted the cluster quality, reflected by the Davies-Bouldin, silhouette index and normalized mutual information, versus the different values for seven of the parameters controlling the generation of the continuous RBF stream in Figures 4.39 and 4.40.

Analyzing the figures shows similar results to the ANOVA. However, these figures further show

Table 4.17: ANOVA Table - RINO-Streams versus RBF Generator (Normalized Mutual Information)

Source	Sum Sq. Errors.	Degrees of Freedom	Mean Sq.	F-statistic	p-value
StreamLength	2.431	2	1.2155	256.3937	0
NoClusters	2.0354	3	0.67846	143.1138	0
KernelRadii	2.6302	1	2.6302	554.8129	0
DensityRange	0.00035837	1	0.00035837	0.075595	0.78342
NoiseLevel	0.66782	1	0.66782	140.8695	0
EventFreq	0.00036842	2	0.00018421	0.038857	0.96189
NoDim	8.0653	2	4.0327	850.6501	0
Error	4.0343	851	0.0047407		
Total	19.8647	863			

how the quality of RINO-Streams clusters behave with respect to different characteristics of the data stream (i.e. increase or decrease). The results of these figures can be summarized as follows:

- Density Range and Event Frequency (Figures 4.40 (a) and (c)) do not significantly affect the quality of the clusters.
- The dimensionality and the cluster radius (Figures 4.39 (c) and (d)) do not significantly affect the Davies-Bouldin Index, while they exert a significant effect on the other metrics.
- The rest of the RBF parameters do affect the quality of the clusters generated by RINO-Streams, albeit not very significantly.

4.1.7.2 Sensitivity based on RINO-Streams parameters

In this section we will analyze the effect of using different parameter values for RINO-Streams on the quality of the clustering models.

ANOVA We analyzed the results using the analysis of variance test (ANOVA) with the hypothesis that RINO-Streams' parameter settings do not affect the quality of the clustering model. If the p-value is less than α (i.e. 0.05) then the value assigned to the corresponding RINO-Streams parameter has a significant effect on the quality of the clusters.

Figure 4.39: The effect of the data stream properties on RINO-Streams output (Part 1)

Figure 4.40: The effect of the data stream properties on RINO-Streams output (Part 2)

Table 4.18: ANOVA Table - RINO-Streams Parameters (Davies-Bouldin)

Source	Sum Sq. Errors.	Degrees of Freedom	Mean Sq.	F-statistic	p-value
$\frac{K_{max}}{K_G}$	98.6104	2	49.3052	43.6668	0
σ_0	306.986	1	306.986	271.8802	0
$\frac{\tau}{ X }$	121.5327	1	121.5327	107.6347	0
$t_{outlier}$	312.3281	1	312.3281	276.6115	0
t_{merge}	0.031976	1	0.031976	0.028319	0.86655
W_{min}	43.3285	1	43.3285	38.3736	0
a_{mature}	56.2499	1	56.2499	49.8174	0
Error	206.6294	183	1.1291		
Total	1145.697	191			

Table 4.19: ANOVA Table - RINO-Streams Parameters (Silhouette Index)

Source	Sum Sq. Errors.	Degrees of Freedom	Mean Sq.	F-statistic	p-value
$\frac{K_{max}}{K_G}$	0.31897	2	0.15949	184.2572	0
σ_0	0.82385	1	0.82385	951.8135	0
$\frac{\tau}{ X }$	0.24606	1	0.24606	284.2805	0
$t_{outlier}$	0.23088	1	0.23088	266.7367	0
t_{merge}	0.021273	1	0.021273	24.5772	0
W_{min}	0.56652	1	0.56652	654.5159	0
a_{mature}	0.0017032	1	0.0017032	1.9677	0.16238
Error	0.1584	183	0.00086556		
Total	2.3677	191	0		

The results of performing ANOVA with $\alpha = 0.05$ are shown in Tables 4.18-4.20 for Davies-Bouldin, Silhouette index and normalized mutual information respectively. The results show that the value assigned to most RINO-Streams' parameters have a significant effect on the quality of the clustering model. Only a few of the parameters show no significant effect on some of the quality metrics, such as the effect of t_{merge} on Davies-Bouldin.

To further analyze the effect of RINO-Streams parameters values on the performance of RINO-Streams, we plotted the cluster quality, reflected by the Davies-Bouldin, silhouette index and normalized mutual information, versus the different values for RINO-Stream parameters in Figures

Table 4.20: ANOVA Table - RINO-Streams Parameters (Normalized Mutual Information)

Source	Sum Sq. Errors.	Degrees of Freedom	Mean Sq.	F-statistic	p-value
$\frac{K_{max}}{K_G}$	0.060877	2	0.030439	112.9932	0
σ_0	0.000724	1	0.000724	2.6876	0.10285
$\frac{\tau}{ X }$	0.045126	1	0.045126	167.5141	0
$t_{outlier}$	1.88E-06	1	1.88E-06	0.0069972	0.93343
t_{merge}	0.015527	1	0.015527	57.6388	0
W_{min}	0.15797	1	0.15797	586.401	0
a_{mature}	0.09486	1	0.09486	352.1372	0
Error	0.049297	183	0.00026938		
Total	0.42438	191	0		

4.41-4.47. Analyzing the figures yields similar conclusions as the ANOVA. However, these figures further show how the quality of RINO-Streams clusters behaves with respect to different characteristics of the data stream (i.e. increase or decrease). The results of these figures can be summarized as follows:

- The Chebyshev constant (t_{merge}), used for testing merging (Section 3.1.6), does not have a significant effect on the quality of the clusters.
- The Chebyshev constant ($t_{outlier}$), used to detect outliers (Section 3.1.5), and the minimum sum of weights (W_{min}) are inversely proportional to the quality of the clustering model.
- The forgetting factor (τ), which affects the speed of decay of the data point weight (Section 3.1.1), is directly proportional to the quality of the clustering model.
- The initial scale (σ_0) is directly proportional to the quality of the clustering model based on Davies-Bouldin and Silhouette Index. However, it does not have a significant effect on the quality based on the normalized mutual information,
- The maturity age (a_{mature}), which provides a grace period for outliers (Section 3.1.5), shows a different effect for each of the quality metrics: directly proportional to the normalized mutual information, directly proportional to the Davies-Bouldin Index, and no significant effect on the Silhouette index.
- The number of clusters used has a mixed effect on the quality of the clustering model: it is inversely proportional to the Davies-Bouldin index, while for the Silhouette index and

Figure 4.41: RINO-Stream's parameter effect on quality of the clusters: Number of Clusters

Figure 4.42: RINO-Stream's parameter effect on quality of the clusters: Initial Scale

normalized mutual information, it generates best results if it was closer to the real number of clusters.

Figure 4.43: RINO-Stream's parameter effect on quality of the clusters: Forgetting Factor

Figure 4.44: RINO-Stream's parameter effect on quality of the clusters: Chebyshev Constant for outliers

Figure 4.45: RINO-Stream's parameter effect on quality of the clusters: Chebyshev constant for merging

Figure 4.46: RINO-Stream's parameter effect on quality of the clusters: Minimum Sum of Weights

Figure 4.47: RINO-Stream's parameter effect on quality of the clusters: Maturity Age

Figure 4.48: RINO-Streams Pareto Frontier

Pareto Frontier An important and difficult problem in data mining in general, is to find the best parameter values to maximize the quality of the model. However, analyzing the effect of the RINO-Streams parameters on the quality of the clustering model showed that different values can improve or reduce the quality of the generated clusters. They can even have different effect on different measures of quality (e.g. Number of clusters). Hence, there are always trade-offs that we need to consider when choosing the parameter values.

To solve this problem, we use Pareto efficiency (Kung *et al.*, 1975). Pareto efficiency deals with the problem of trade-offs between multiple solutions to a problem and it selects a set of *efficient* solutions which can not be further improved. These are called the Pareto Frontier and are shown in Figure 4.48 as red points. The values of the Pareto Frontier points are shown in Table 4.21. Analyzing the Pareto Frontier points shows that some parameters generate the best results when they are set to a specific value (e.g. $\frac{K_{max}}{K_G}$) while other parameters need a specific combination with other parameters (e.g. σ_0).

Default Parameter Values Based on the ANOVA, sensitivity analysis and the Pareto Frontier results discussed above, we will use the values in Table 4.22, unless stated otherwise, for the RINO-Streams parameters in the next experiments, since they provide the best results.

Table 4.21: RINO-Streams Pareto Frontier

Parameter	Point 1	Point 2	Point 3	Point 4	Point 5	Point 6
$\frac{K_{max}}{K_G}$	100%	100%	100%	100%	100%	100%
σ_0	0.05	0.05	0.05	0.1	0.1	0.1
$\frac{\tau}{ X }$	20%	20%	20%	20%	20%	20%
$\frac{1}{t_{outlier}}$	0.05	0.05	0.1	0.05	0.05	0.1
$\frac{1}{t_{merge}}$	0.1	0.1	0.1	0.1	0.1	0.1
W_{min}	5	5	5	5	5	5
$\frac{a_{mature}}{ X }$	1%	2%	2%	1%	2%	2%

Table 4.22: RINO-Streams Default Parameter Values

Parameter	Default Value Used
$\frac{K_{max}}{K_G}$	100%
σ_0	0.1
$\frac{\tau}{ X }$	20%
$\frac{1}{t_{outlier}}$	0.05
$\frac{1}{t_{merge}}$	0.1
W_{min}	5
$\frac{a_{mature}}{ X }$	2%

4.2 Evaluation of Component 2: TRACER

In this section, we evaluate the quality of TRACER, the tracking component of Stream-Dashboard, whose output is a set of regression models that summarize the behavior of the cluster metrics over time as well as the milestones of change for these metrics. Hence, we evaluate (i) how well the regression models reflect the cluster behavior and (ii) the accuracy of the milestone detection. Since our tracking and validation framework is generic with respect to the choice of the first component, we illustrate our framework using two stream clustering algorithms: Our proposed algorithm RINO-Streams (Section 3.1) and Growing K-Means (GKM) (Lowette & Laerhoven, 2004) which was described in Section 2.2.2.2.

4.2.1 Experimental Setting

4.2.1.1 Creating Benchmark Data sets and Evolution Trends

In order to test against a ground-truth with known milestones, several synthetic datasets were generated using a random Gaussian generator, and the behavior of each metric was simulated by generating linear regression models and interpolating them together, and considering the points of interpolation as milestones. Each regression model was generated based on random coefficients. To make sure that a milestone exists, we draw the coefficients of the consecutive regression models from different intervals using a parameter (*Volatility*) that reflects the volatility of milestones. The volatility quantifies how much the behavior changes at the milestones. Hence, if the volatility is equal to 50 and the first regression model is drawn from the interval $[a : b]$, then the next regression model is drawn from the interval $[a \pm 50 : b \pm 50]$. Another parameter is the frequency of milestones in time (which affects the length of each regression model). The time of milestone occurrence is random, however, each two consecutive milestones are guaranteed to be separated by a parameter called (*MilestonesDistance*) which is a percentage of the stream length ($|X|$). For example, if *MilestonesDistance* = 1% and the size of the data stream is $|X| = 1000$, then the first milestone location is a random location between $[1 : 10]$ and the second between $[20 : 30]$. This is done to ensure that the milestones are better distributed along the stream, without however fixing their location. For experimental purposes we tried selecting milestones completely at random, as will be indicated by the parameter *RandomMilestones*, with 1 meaning that the milestones are random, and 0 otherwise (i.e. the milestones are generated using the parameter *MilestonesDistance*). The dataset properties are listed in Table 4.23.

In addition to the controlled experiments, we used two real datasets: KDD CUP 99 Network Intrusion ⁵ and Reuters text datasets (Lewis, 1997).

4.2.1.2 Evaluation Metrics

To evaluate the accuracy of finding the milestones, we compare the closest milestone detected by TRACER to the ground-truth milestones, and if the distance is less than a threshold (*DistanceThreshold*),

---

⁵<http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

Table 4.23: Dataset Properties & Component 1’s Clustering Algorithms used in our experiments

Dataset	No. Pts	No. Dim	No. Classes	Noise%	Description	Clustering Alg. Used
DS1	700	2	1	0	1 cluster that changes in cardinality	RINO & GKM
DS2A	4700	2	2	0	2 clusters that split	RINO
DS2B	3000	2	2	0	2 clusters that merge	RINO
DS6	3365	2	6	20	3 clusters arriving in the order of their cluster label. Later, 3 additional clusters arrive in a random order	RINO & GKM
DS1A	20000	2	1	0	1 cluster that changes in scale	RINO & GKM
DS5	2625	2	5	25	5 cluster with random arrival order	RINO & GKM
DS10	2750	2	10	10	random or ordered data arrival	RINO
Re0	1504	2886	13	0	Reuters News text data (Lewis, 1997)	RINO
KDD CUP 99	494021	33	24	0	Network Activity ⁶	GKM

as a percentage of the regression window size (Δ_{Reg}), we consider it to be correctly detected. Finally, we compute the number of correctly detected milestones, missed milestones and spurious milestones, and calculate the recall, precision and F1 measures for the detection. The quality of the regression models itself is assessed using the average coefficient of determination statistic (R^2) for all regression models.

4.2.1.3 Analysis of Variance

TRACER uses two principal parameters that may affect its performance: the regression window (Δ_{Reg}) and the angle threshold (θ_{max}). Moreover, the properties of the data stream may affect the results, hence, we performed an analysis of variance (ANOVA) on all these parameters. The parameters and their values are shown in Table 4.24. For each experiment, we found the average using 100 generated regression models.

For each evaluation metric, we first perform the N-way ANOVA, then for the factors whose F-statistic is low but is considered significant, we performed a One-Way ANOVA. This is needed to make sure that the significant factors are really significant and not due to the variations between the two types of ANOVA tests.

Table 4.24: ANOVA Parameters and their values

	Parameter	Values
TRACER	Regression Window Size (Δ_{Reg})	[50,100,...,500]
	Angle Threshold (θ_{max})	[10,15,...,30]
Stream	Stream Length ($ X $)	[1000,2000,...,5000]
	<i>NoMilestones</i>	[10,20,...,50]
	<i>Volatility</i>	[50,60,...,150]
	<i>MilestonesDistance</i>	[0.05% , 0.075% , 1%]
	<i>DistanceThreshold</i>	[33% , 66% , 100%]
	<i>RandomMilestones</i>	[0 , 1]

We performed a rigorous analysis on all the evaluation metrics and report only the final findings. Table 4.25 shows the F-statistic and p-value for all the parameters and evaluation metrics. A p-value that is less than $\alpha = 10\%$ (i.e. p-value < 0.01), indicates a parameter has a significant effect on the validation metric. The results show that all parameters have a significant effect, except for *Volatility* on F1 and *DistanceThreshold* on R^2 .

The F-statistic is directly correlated with how significant a parameter is. To further analyze the effect of the parameters with a low F-statistic, we performed 1-way ANOVA on some of the parameters as shown in Table 4.26. The results show that θ_{max} in fact does not have a significant effect on R^2 , neither does *Volatility* on the precision. This clearly shows the difference between using N-Way and 1-Way ANOVA.

After the ANOVA study, we found the parameter values that generated the best results for all the metrics. Since there is a trade-off between obtaining higher quality regression models and accurate milestone detection, we found the values that optimize all evaluation metrics. Those values are shown in Table 4.27 and were later used for the remaining experiments.

4.2.1.4 Using TRACER for Tracking Cluster Evolution

We set TRACER's parameter values based on Table 4.27. Note that we also performed a sensitivity analysis of these parameters as discussed in Section 4.2.7. These values will be used for all the experiments unless stated otherwise. The experiments aim at evaluating the performance and accuracy

Table 4.25: TRACER: N-Way ANOVA Results

Parameters	Milestones						Regression Models	
	F1		Precision		Recall		R^2	
	F-Statistic	p-value	F-Statistic	p-value	F-Statistic	p-value	F-Statistic	p-value
Regression Window Size (Δ_{Reg})	2062.65	0	15010.44	0	5168.4	0	41564.04	0
Angle Threshold (θ_{max})	245.58	0	24.37	0	455.2	0	5.15	0
Stream Length ($ X $)	6424.39	0	12740.84	0	10193.2	0	58149.81	0
<i>NoMilestones</i>	3661.12	0	20446.82	0	8604.01	0	80957.5	0
<i>Volatility</i>	4.24	0.0144	10.99	0	24.51	0	11712.42	0
<i>MilestonesDistance</i>	67.71	0	122.23	0	112.64	0	95.34	0
<i>DistanceThreshold</i>	2806.98	0	6541.22	0	3252.94	0	0	1
<i>RandomMilestones</i>	1360.86	0	2647.95	0	2048.5	0	2129.34	0

Table 4.26: TRACER: 1-Way ANOVA Results

	Milestones						Regression Models	
	F1		Precision		Recall		R^2	
	F-Statistic	p-value	F-Statistic	p-value	F-Statistic	p-value	F-Statistic	p-value
Angle Threshold (θ_{max})	124.16	0	4.68	0.0009	155.17	0	0.34	0.8513
<i>Volatility</i>	N/A	N/A	2.11	0.1212	8.28	0.003	N/A	N/A
<i>MilestonesDistance</i>	34.02	0	23.49	0	38.09	0	6.29	0.0019

Table 4.27: TRACER: ANOVA Parameters Best Values

	Parameter	Values
TRACER	Δ_{Reg}	100
	θ_{max}	20
Dataset	$ X $	5000
	<i>NoMilestones</i>	50
	<i>Volatility</i>	150
	<i>MilestonesDistance</i>	1%
	<i>DistanceThreshold</i>	100%
	<i>RandomMilestones</i>	0

of tracking the evolution of clusters over time. Hence, although the results depend on the quality of clusters generated by the online clustering algorithm, we are going to use the default parameter values for the online clustering algorithms (RINO-Streams (Hawwash & Nasraoui, 2010) and GKM (Lowette & Laerhoven, 2004)).

4.2.2 Tracking Cluster Behavior

4.2.2.1 DS6 Dataset

Figures 4.49(a) and (b) show the cardinality values of each cluster against time as well as its respective summary regression models for DS6 using RINO-Streams and GKM respectively. Moreover, a milestone point is indicated as a circle (o). It can be seen that the regression models accurately represent the evolution of the cluster cardinality over time, while using much less memory (as discussed in Section 3.2.7).

The first three clusters arrive in the order of their label. RINO-Streams detects these clusters (Clusters 1, 2 and 3) and their cardinality keep increasing until they are not updated anymore, then their cardinality start decreasing due to the use of a forgetting factor in the cluster updates. On the other hand, GKM detects these clusters (Cluster 1,2 and 3) and their cardinality increases at the beginning until they are not updated anymore; at that point, their cardinality stays stable with only a slight increase due the arrival of noise. The other three clusters' points arrive in random order, and hence, their cardinality keeps increasing using both algorithms. The regression models for the rest of the cluster metrics show a similar accuracy in reflecting the clusters' evolution.

4.2.2.2 Re0 (Reuters Text Dataset)

The experiments on this real text collection are meant to demonstrate TRACER's ability to track the clusters' evolution in real datasets. Figure 4.50 shows the density values of each cluster against time as well as its respective regression models. The figure shows that the regression models accurately reflect the actual changes in cluster densities, where some clusters keep improving with time (e.g. Cluster 6) and others increase then stabilize (e.g. Cluster 8).

Figure 4.49: DS6: Cardinality vs. Time and its Summary Regression Models

A milestone is indicated as a circle (o), regression models are solid lines and raw cardinality values are dotted lines

Figure 4.50: Re0: Density vs. Time and its Summary Regression Model

A milestone is indicated as a circle (o), regression models are solid lines and raw density values are dotted lines

4.2.2.3 KDD Cup 99 Dataset

The KDD Cup 99 dataset represents network activity traces, collected over a period of nine weeks of normal activity interspersed with various attacks and intrusions simulated in a military network environment⁷. We used the training dataset with the 33 continuous features. There are a total of 23 different attacks that fall into four main categories. Figure 4.51 shows the occurrence of each attack versus the order of data arrival of network packets. There are two attacks that appear consistently (“smurf” and “neptune”), while the remaining attacks appear very sparsely. Therefore, we will focus on these two attacks in addition to the “normal” activity since their behavior can be validated. We used Growing K-Means (Lowette & Laerhoven, 2004) to find the clusters and we set the maximum model capacity at any point to 30 clusters.

To evaluate the accuracy of tracking the clusters’ behavior, we compare the cardinality regression models versus the groundtruth cardinality of the attacks. The groundtruth cardinality was found by finding the cumulative sum of each network attack. Figure 4.52 shows the cardinality regression models versus the groundtruth cardinality of the three main activities (smurf, neptune and normal). It can be seen that the regression models accurately represent the behavior of the main three activities. Specifically, cluster 21 represents the smurf attack, cluster 6 represents the neptune attack and cluster 1 represents the normal activity. The remaining attacks are also detected but they form minor proportions of the entire data stream, thus their trends are shown toward the lower part of Figure 4.52. Note that the milestones (shown as circles in Figure 4.52) reflect when the activity appeared, re-appeared or changed behavior.

4.2.3 Detecting Internal and External Transitions

4.2.3.1 Tracking Internal Transitions

To evaluate the accuracy of tracking cluster changes, we used GKM and RINO-Streams in component 1 on dataset DS1A. Figure 4.53 shows the detected regression models for the scale versus its groundtruth. TRACER was able to detect the scale increase, stabilization and subsequent decrease as data arrived with gradual diffusion toward the outside of the cluster (away from the center), then again a densification near the center. The continuous validation of the clustering algorithm out-

---

⁷<http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

Figure 4.51: KDD Cup 99 Network Activity

Figure 4.52: Cardinality regression models versus the groundtruth cardinality for three main network activities

Figure 4.53: DS1A: Online Validation of the Tracking and Summary of the Cluster Scale Evolution using Regression Models against their Ground-truth for two different online clustering algorithms used in Component 1.

A circle (o) refers to a milestone

puts in component 1, shows that RINO-Streams achieves a better estimate of the scale compared to GKM. This is due to the fact that it uses a forgetting factor (i.e. giving more importance to newer data points), hence, once the new points start emerging at a smaller scale (closer to the centroid), they cause the scale to decrease (hence the higher number of detected milestones). On the other hand, GKM does not use a forgetting factor, hence allowing all the points to have the same influence on the scale estimation which in turn can only stabilize at a high value.

4.2.3.2 Detecting External Transitions

Detecting external transitions is done by analyzing the internal transitions over time and applying the rules in Table 3.4. We conducted two experiments using two stream clustering algorithms, RINO-Streams (Hawwash & Nasraoui, 2010) and GKM (Lowette & Laerhoven, 2004) (since it handles external survival transitions differently, namely it does not split clusters) to detect the clusters in evolving synthetic datasets to illustrate and validate the detection of survival and splitting transitions.

Survival Figures 4.54(a), (b) and (c) show a single cluster that undergoes three different kinds of survivals. The scale and cardinality regression models are shown in Figures 4.55 (a) and (b) using RINO-Streams and GKM respectively. The survival transitions are detected using the rules in Table 3.4 as follows:

- Absorption between time period 1-500: The scale, using both algorithms, increases slightly then stabilizes, while the cardinality keeps increasing.
- Gaining points at the periphery at time period 501-600 :
  - Using RINO-Streams: The scale increases while the cardinality decreases slightly then increases. The cardinality decreases at the beginning because the new points have less weight (since they are at the periphery) while the points that had high weight start losing weight with time, but as more and more new points keep arriving, the older points' weight loss is compensated by the gain resulting from the new points, thus the cardinality increases again.
  - Using GKM: Similar to RINO-Streams, the scale increases. However, the cardinality keeps increasing since GKM does not use a robust weight.
- Gaining points at the center at time period 601-700: The scale, using both algorithms, decreases while the cardinality increases. When using RINO-Streams, these points are closer to the centroid, hence, they have higher weights, and the cardinality increases fast as opposed to the previous time period. On the other hand, when using GKM, the cardinality keeps increasing regardless of its positions since no weight is used as in RINO-Streams.

Other than the ability to track internal changes of the stream clusters, this experiment further illustrates the ability of Stream-Dashboard to validate and analyze clusters. When comparing RINO-Streams and GKM results, we can conclude that they are both good at capturing the scale of the clusters. However, GKM does not do a good job in capturing the cardinality since it does not use a robust weight as in the case of RINO-Streams.

Splitting Figure 4.56 shows four snapshots of one cluster that splits into two clusters. The scale and cardinality regression models are shown in Figures 4.57 (a) and (b) respectively. Splitting in RINO-Streams takes place when the new points are considered outliers with respect to the current

Figure 4.54: DS1: Ground Truth

(a) Data arriving in times 1-500 (b) Data arriving in times 501-600(c) Data arriving in times 601-700

Figure 4.55: DS1: Cardinality & Scale Regression Models

cluster, which results in the creation of a new cluster while keeping the old cluster, hence, two regression models are shown. Splitting takes place after 1200 points have arrived, and it can be seen that as a result, cluster 1 experiences a sudden drop in its scale and cardinality starting from the milestone at $t=1200$. After the split, the two clusters gain new points which explains the increase in cardinality at time period (1200-1500). During the rest of the time period, the two clusters fluctuate in their scale and cardinality since they gradually obtain acquire data points while losing older ones.

4.2.4 Using TRACER for Continuous Cluster Validation

One of the contributions of Stream-Dashboard is the ability to validate the clustering model on the fly by observing the behavior of the detected clusters through time. For this purpose, we ran an experiment to compare two stream clustering algorithms used in Component 1: Growing K-Means (GKM) (Lowette & Laerhoven, 2004) and RINO-Streams (RINO). We ran both algorithms on DS5,

Figure 4.56: DS2A: Dataset Evolution and Final Clustering

(a) 10% of the pts (b) 30% of the pts (c) 60% of the pts (d) 100% of the pts

The data points in turquoise are forgotten (i.e. their weight is less than the forgetting factor), and the data points in black are the newer ones

Figure 4.57: DS2A: Validating the Detection and Tracking of a Splitting Transition

Figure 4.58: DS5: Validating and Tracking the Outputs of Two Different Online Algorithms using Density Regression Models

A circle (o) refers to a milestone, and (x) refers to a mergal

and we used the default parameter values. Moreover, we kept the bad clusters to observe their behavior. The size of the regression window (Δ_{Reg}) was set to 200 (i.e. we call TRACER every 200 points).

Figure 4.58 show the density regression models for GKM and RINO respectively. In Figure 4.58 (a), the density of six clusters seems to be increasing at a constant rate, while the other four clusters have lower density at a smaller rate of increase. Hence, we can conclude that six clusters are valid, while the other four are bad clusters (outliers). On the other hand, Figure 4.58 (b) shows that the density of only five clusters (which is the correct number of clusters) are increasing and hence are valid. The other four clusters have much lower density and their density is mostly stable. Hence, we can conclude that RINO-Streams outperforms GKM, since the density of the outlier clusters computed using the former can be easily detected and suppressed based solely on their tracked density trend.

4.2.5 Comparison with MONIC

In this section, we will compare Stream-Dashboard with MONIC (Spiliopoulou *et al.*, 2006), since MONIC is the closest to our proposed framework out of the algorithms discussed in Section 3.2.9. Both algorithms can detect internal and external changes and are generic. The major differences in MONIC are that (i) it assumes re-clustering rather than adaptation, (ii) it uses the support of

the data points instead of the spatial differences between clusters, and *(iii)* it assumes that pre-processing is done to remove outliers. Hence, to make a fair comparison between MONIC and Stream-Dashboard, we modified MONIC, to be useful within a stream context, so that it uses adaptation (i.e. it uses the previous clustering model at time T as the seeds to find the clustering model at $T + 1$), and we used the spatial differences between the centroids of the clusters as the overlap matrix used in MONIC. This is necessary, since all online stream clustering algorithms keep only a synopsis of the cluster (e.g. centroid and scale) and not all the data points and their cluster assignments.

To show the proposed framework’s accuracy in detecting external changes, we used the dataset DS2B. Figure 4.59(a) shows the final clustering model and the clusters’ movement over time. The external transitions found using MONIC and Stream-Dashboard are shown in Figures 4.59(b) and 4.59(c) respectively. We highlighted four areas of interests (A, B, C and D) for illustration purposes. Both frameworks were able to detect the survival transitions of two clusters before the mergal (A), and one cluster after the mergal (C). The mergal of the clusters was also detected by both frameworks (B), however, MONIC failed to detect the splitting that took place towards the end of the data stream (D in Figures 4.59(b) and 4.59(c)). Note that MONIC’s definition of absorptions refers to both *(i)* obtaining new points (called absorption in TRACER), and *(ii)* merging clusters (called mergals in TRACER), so it does not distinguish between the two cases. Moreover, the other external transitions can be attributed to the dynamic nature of the clustering process.

4.2.6 Evaluation and Comparison of the Cluster Trend Summary’s Size and Quality for Stream-Dashboard against a Fixed Interval Width Baseline

To evaluate the performance of TRACER against other methods, we compare its generated outputs against those produced by a common baseline model that stores the metric values at arbitrary periods of time, which are a percentage of the regression window size (Δ_{Reg}). For example, if the arbitrary window is 10% and $\Delta_{Reg} = 200$ then the baseline model saves the metric every 20 points. The memory savings that result from using TRACER on DS6 are shown in Figure 4.60(a), which shows the ratio of the size of TRACER’s regression models (in bytes) with respect to the size produced by

Figure 4.59: DS2B: MONIC vs Stream-Dashboard External Changes

(a) Dataset Evolution and Final Clustering Model

(b) MONIC

(c) Stream-Dashboard

Figure 4.60: DS6: TRACER Compared against Baseline

the baseline model for different percentages of arbitrary window sizes. It can be seen that TRACER saves more memory than the baseline model (by at least 50%) for all arbitrary window sizes. This is because TRACER only saves the regression models when a milestone is detected (Section 3.2.3). Moreover, as the arbitrary window size increases, the memory savings decrease since fewer values are stored in the baseline method.

In addition to the space savings, we also compared the quality of the summary regression models generated by TRACER, based on the R^2 statistic as shown in Figure 4.60(b). To find the R^2 for the baseline model, we used the metric values stored at arbitrary locations and generated a linear model using every two consecutive values, then we found the average R^2 over all intervals of size Δ_{Reg} . The results show that TRACER outperforms the baseline method on all arbitrary window sizes, albeit using much less memory. Note that the performance of the baseline model degrades as the arbitrary window size increases, since a smaller window results in more accurate regression models.

4.2.7 Sensitivity and Complexity Analysis

In this section, we study the effect of the regression window size and the angle threshold while fixing the other parameters as shown in Figures 4.61(a) and (b) respectively. The figures also show the variance of the evaluation metrics plotted as two small lines above and below the metric measurement. Figure 4.61(a) shows that as the regression window size increases, the quality of both the milestone detection (except for Precision) and the regression model decreases. This is expected since a wider regression window results in a smoother regression model, which might cause some

Figure 4.61: Effect of the TRACER parameters on milestone detection and regression model quality

milestones to be missed if they happen to occur within the same regression model. The Precision increases because it counts the ratio between the number of correct milestones and the number of detected milestones, which means that all the detected milestones are correct when the regression window size increases, however this happens at the cost of missing more milestones (hence affecting the recall). Figure 4.61(b) shows that the angle threshold has minimal effect on the quality of the models.

To further study the effect of TRACER's parameters on Stream-Dashboard's performance, we measured the time and memory complexity of TRACER using different regression window sizes (Figure 4.62 (a)) and different angle threshold values (Figure 4.62 (b)). The time complexity represents the number of times that new regression models are generated and how many times they were updated (i.e. steps 4 and 6 in Algorithm 15).

Figure 4.62 (a) shows that as the regression window size increases, the time and memory complexity decrease, as expected, since a wider regression window results in less invocations to the TRACER algorithm (i.e. less time complexity as shown in (3.34)), hence, fewer regression models need to be generated (thus less memory needed).

Figure 4.62 (b) shows that an increase in the angle threshold results in an increase in the time complexity and a decrease in the memory complexity. The angle threshold is used to determine when a milestone is present (Def. 3.2.5). Hence, a larger threshold results in fewer milestones, thus fewer regression models, which in turn means lower memory requirements. However, a lower number of milestones means that more regression models need to be updated (Step 6 in Algorithm

Figure 4.62: Time & Memory Complexity of TRACER

15), hence the increase in time complexity. On the other hand, a decrease in the angle threshold results in more milestones and more regression models (thus an increase in memory complexity), and fewer regression model updates (thus a lower time complexity).

4.2.8 Illustrating the Dashboard Visualization

4.2.8.1 Stream Genealogy

In this section we will illustrate some of the properties of the Stream Genealogy graph (Section 3.4). Figure 4.63 (a) shows all the nodes of the Stream Genealogy where the size of the nodes is based on the density of the clusters, while Figure 4.63 (b) shows the same Stream Genealogy after hiding the outliers, using the scale as the size of the nodes and hiding all clusters created before time $t = 1500$. Both figures visualize the interactions between the clusters as well as their quality. Using the Stream Genealogy helps in validating the quality of the clusters. For example, the active clusters (C1, C2..., C6) have higher density and smaller scale, which means they are of good quality. Note that the first three clusters (C1, C2 and C3) have less density since they are not updated after being detected and their cardinality decreases due to the use of a forgetting factor.

Figures 4.64 (a) and (b) show the cardinality and scale changes, respectively, of a merged cluster (cluster 6) and its ancestors. Analyzing both figures shows that cluster 6 inherited most of its behavior from cluster 2, which suggests that the other clusters are of low quality, hence, they have a smaller effect on the merged cluster. Also, we can see that cluster 5 is not a real cluster (i.e. it had short lifetime), rather it was an extension of cluster 6.

Figure 4.63: DS6: Stream Genealogy Layout with different filters

Figure 4.64: DS2B: behavioral trend of a merged cluster and its ancestors

4.2.8.2 DS10 Results

DS10 consists of ten clusters and 10% of noisy data points, and we will use two versions, where one had the data points arrive in order ($C_{10}ON_{10}$) and the second with a random arrival ($C_{10}RN_{10}$). Figures 4.65 and 4.66 show the density values of each cluster against time as well as its respective regression models for datasets $C_{10}ON_{10}$ and $C_{10}RN_{10}$, respectively. Moreover, a milestone point is indicated as a circle (o) and the merging of clusters as an (x). It can be seen that the regression models accurately represent the evolution of the cluster density over time for both variations of the dataset, while using much less memory space (as discussed in Section 3.2.7). Figure 4.66 shows that more milestones were detected since the density kept improving over time, however the memory needed to capture the evolution is still much less than the memory needed if this method was not used. On the other hand, Figure 4.65 shows only a few milestones being detected for each cluster since the densities stabilized after some time (because the points were presented one cluster at a time), and only the regression models at these few milestones needed to be stored. The regression models for the rest of the cluster parameters show a similar accuracy in reflecting the clusters' evolution.

Figure 4.65: DS10: Density Over time Vs. Regression Models (for experiment $C_{10}ON_{10}$, ordered data arrival)

Note: A milestones is indicated as a circle (o), and a merging is indicated as an (x)

Figure 4.66: DS10: Density Over time Vs. Regression Models (for experiment $C_{10}RN_{10}$, random data arrival)

Note: A milestones is indicated as a circle (o), and a merging is indicated as an (x)

The stability measures for the density of dataset DS10 which are derived from statistical tests as discussed in Section 3.2.4, are shown in Figure 4.67. An increasing density is indicated by a positive value, a stable (plateau) behavior by a zero, and a decreasing density by a negative one. When compared against the real regression values in Figures 4.65 and 4.66, the stability measures are found to reflect the behavior of the cluster density values, where all the clusters in $C_{10}RN_{10}$ are improving continuously with time due to the continuous (random) arrival of data; while clusters in $C_{10}ON_{10}$ first increase in density, then stabilize since the data was presented one cluster at a time. Tracking the stability measures over time promises to help in developing *behavioral* profiles for good and bad clusters. For example, the minimum density (δ_{min}) can now be set as the minimum of all cluster densities which have shown a stable behavior (i.e. plateau so far).

Figure 4.67: DS10: Density Stability Over Time

We have also developed an alternative simpler genealogy visualization, that we call *ClusTrees* that traces the ancestry in the form of vertical trees, and this is shown for DS10 in Figure 4.68. There are ten independent binary trees, each representing one cluster (the label of each cluster is at the root) with the root being the final cluster output, and the children of each node representing the clusters that were merged. The tree is interactive and each of the nodes can be explored (by clicking on it) to examine its cluster properties (as shown in the box to the left).

Figure 4.68: DS10: Interactive Cluster Ancestry Tree

4.3 Application: Mining Twitter Data Streams

The proposed framework, Stream-Dashboard, was created to help analyze, in a completely unsupervised way, a continuous stream of data through the detection of evolving patterns and tracking their behavior. In this section, we illustrate the use of Stream-Dashboard in a real world application. To this extent, we will observe the effectiveness and performance of Stream-Dashboard when applied to Twitter streams.

Twitter is a prominent social networking website that enables its users to share short messages (called “tweets”), to follow other users, and to reply to users in an interactive manner, via either the web browser or mobile SMS channels. Each tweet consists of a maximum of 140 characters, and it may include

- regular text,
- a reply to a specific user by including the @ character followed by the user id,
- re-tweet another tweet, which is identified by the keyword “RT”,
- one or more URL,
- one or more keywords or topics that are relevant to the Tweet, which are marked by using

the hashtag symbol (#). The hashtags are created organically by the users, and represent the trending topics or keywords that people are tweeting about.

Twitter generates a massive and continuous data stream of tweets with an estimated 500 million tweets generated per day ⁸. The real time, informal and spontaneous nature of the tweets have made Twitter attractive for decision makers to extract the interests and opinions of the users. (Jansen *et al.*, 2009; Bifet & Frank, 2010; Mendoza *et al.*, 2010).

4.3.1 Data Preparation

Twitter provides an Application Programming Interface (API)⁹ which allows collecting tweets by third party users. The Free API is limited to a 1% sample of the tweets. We have used this API to collect tweets starting from October 2011. We collect tweets for 15 minutes every hour and store them in a PostgreSQL database. We did not use any filtering keywords when querying the API, so it was a wild collection of random tweets. However, for some of the experiments shown below, we will use filtering from the collected set.

4.3.1.1 Pre-processing

After storing the raw tweets in the database, we perform a number of pre-processing steps as shown below.

1. Detecting the language of the tweet and keeping only the English-written tweets
2. Cleaning the tweets by:
  - (a) extracting the web links, user names and hash tags and storing them in the database,
  - (b) removing non-English characters and digits
3. Removing stop words
4. Lemmatization of words using an open source tool (MorphAdorner ¹⁰)
5. Removing any tweet that has less than 3 words after cleaning

---

⁸<http://www.telegraph.co.uk/technology/twitter/9945505/Twitter-in-numbers.html>

⁹<https://dev.twitter.com/>

¹⁰<http://morphadorner.northwestern.edu/>

Table 4.28: Unfiltered Twitter Dataset Properties

Statistic	Value
Number of unique users	3,421,720
Number of tweets	4,164,402
Number of unique tweets	3,602,550
Number of tweets using hashtags	524,719
Number of unique tags	266,739

6. Extracting the other properties of the tweet besides the text (e.g. the user name) and storing it in the database.

4.3.1.2 Dimensionality Reduction

The processed tweets are presented to the clustering algorithm as a bag of words for each tweet. So, if there were M tweets and N unique words, then each tweet is an N -dimensional vector whose elements store the frequencies of each word. Since each tweet is limited to 140 characters, the resulting vector is mostly empty (i.e. sparse). To handle this problem, we have explored using Topic Modeling (Section 2.6) for the unfiltered dataset experiments, to reduce the dimensionality and the synonymy problem. More specifically, we used the LDA with collapsed Gibbs Sampling (Liu, 1994) to find the latent topics, and used the probabilities of each tweet to each topic as the input to the clustering algorithm. Hence, if we found T topics, then the input to the clustering algorithm would be $M \times T$ where $T \ll N$.

4.3.1.3 Data Statistics

For the purpose of this work, we extracted one year-worth of tweets without specifying any filtering terms. More specifically, we collected 500 random tweets every hour starting from March 2012 and ending in March 2013. Some of the statistics of the collected data are listed in Table 4.28.

The number of tweets per user is shown in Figure 4.69 (a), and the number of hash tags per tweet is shown in Figure 4.69 (b). Both figures show that the distribution follows a power law, i.e. most users have a few tweets and most tweets have a few hashtags.

Figure 4.69: Twitter Properties

4.3.1.4 Popular hashtags per month

Since hash tags are automatically generated by the users, their frequency represents the topics of interests (i.e trending topics). To that extent, we found the frequency of the hashtags for each month and presented the top 20 hashtags for each month. Figure 4.70 shows the popular hashtags for several months in a *tag cloud* visualization format, where the size of the font of each hashtag is proportional to its frequency (i.e. the bigger the font, the more frequent the hashtag is). Some of the hashtags are always popular such as #TEAMFOLLOWBACK, hence, they are not very informative. On the other hand, some hashtags provide some insights about the trending topics at the time period. For example the #London2012 hashtag in August 2012 (Figure 4.70 (a)) refers to the Olympics taking place in London at that time, whereas the #VMA hashtag in September 2012 (Figure 4.70 (b)) refers to the MTV Video Music Award taking place during that month.

4.3.2 Detecting Trending Topics

The stream of tweets was processed by RINO-Streams and we took a snapshot of the centroids at the end of each day. Figure 4.71 shows some of the trending topics detected on different days. The keywords representing each topic in the figure are generated as follows: each detected cluster represents a trending topic, and for each cluster, we found the top 10 features in its centroids. These features are in fact a set of topics which were generated during the pre-processing step. Hence, we will represent each cluster using the top 10 features/topics. Since each topic (from the pre-

(a) August 2012

4.3.3 Twitter Case Study: Louisville Cardinals

4.3.3.1 Pre-processing

188

Figure 4.71: Twitter: Detected Trending Topics for Several Days

(a) Topic 1 (Shopping)

(b) Topic 2 (Smart Phones)

(c) Topic 3: Sports News

(d) Topic 4 (General)

(e) Topic 5 (Cities)

(f) Topic 6 (TV Shows)

(g) Topic 7 (Christmas)

(h) Topic 8 (General)

Figure 4.72: Louisville Tweets

Figure 4.73: Louisville tweets density regression models

4.3.3.2 Tracking Louisville Twitter Stories through Cluster Evolution

We used Stream-Dashboard to process the Louisville tweets, and it was able to detect several trending topics as well as their behavior changes over time as shown in the density regression models detected by TRACER in Figure 4.73. We will analyze three example topics that were discovered, and some of the interesting topics related to the NCAA tournament where Louisville won the national championship.

The Sugar Bowl 2013 Cluster In January 2013, the Louisville Cardinals won the Super Bowl final game against the Florida Gators¹¹. Stream-Dashboard was able to detect a trending topic related to the super bowl and track its changes over time. Table 4.29 lists the dates and the top

¹¹http://en.wikipedia.org/wiki/2013_Sugar_Bowl

Table 4.29: Twitter Stories: Sugar Bowl 2013 Topic Cluster Evolution Properties

Milestone	Dates	Top Tweet (before pre-processing)
1	11/25/12-12/7/12	louisville plays florida the sugar bowl lets get
2	12/8/12-1/2/13	who wins tonightlouisville florida sugar bowl time tonight
3	1/2/13-1/8/13	louisville cardinals sugar bowl champions
4	1/16/13	official congratulations the sugar bowl champion louisville cardinals

Figure 4.74: Twitter Stories: Sugar Bowl 2013 Topic Cluster Evolution

raw tweet of four milestones of the Sugar Bowl topic. Figures 4.74 (a), (b) and (c) show the topic centroid as a cloud of tags of the top terms, the cardinality regression model, and the similarity matrix, respectively.

The results show that the users started tweeting about the game at the first milestone, then the topic gained more popularity just before the game at the second milestone. The topic popularity spiked after the Louisville Cardinals won at the third milestone, and finally, it lost popularity after a couple of days. The quality of the topic can be validated by observing a dense dark block along the diagonal of the similarity matrix.

The Charlie Strong Contract Extension Cluster Another topic that was related to the Sugar Bowl, and also at around the same time, was the news about extending the contract of Charlie Strong, the football coach ¹². Table 4.30 lists the dates and the top raw tweet at five detected milestones for cluster representing the Charlie Strong topic. Figures 4.75 (a), (b) and (c) show the topic centroid as a cloud of tags of the top terms, the cardinality regression model, and the similarity matrix, respectively.

¹²<http://sportsillustrated.cnn.com/college-football/news/20130123/louisville-charlie-strong-contract-extension.ap/>

Table 4.30: Twitter Stories: Charlie Strong Topic Cluster Evolution Properties

Milestone	Dates	Top Tweet (before pre-processing)
1	11/25/12-12/7/12	vote for charlie strong university louisville for
2	11/28/12-12/6/12	sources louisville working strong extension louisville negotiating contract extension with charlie
3	12/10/12 – 1/2/13	vote charlie strong university louisville for
4	1/2/13-1/3/13	heartfelt congratulations charlie strong and louisville
5	1/23/13	louisville new contract for football coach charlie strong includes buyout the courierjournal

Figure 4.75: Twitter Stories: Charlie Strong Topic Evolution

The topic cluster was detected as people started tweeting to vote for Charlie Strong's contract extension. It spiked when there were more sources to confirm the extension at the second milestone, and it increased popularity again when the extension was approved at the fourth milestone. The quality of the topic cluster is validated in the similarity matrix, where a dark block can be observed on the diagonal.

The Kevin Ware Injury Cluster The gruesome injury of Louisville guard, Kevin Ware ¹³, shocked the fans, resulting in a lot of discussion on Twitter. Table 4.29 lists the dates and the top raw tweet of four milestones that were detected of this topic cluster. Figures 4.74 (a), (b) and (c) show the topic centroid as a cloud of tags of the top terms, the cardinality regression model, and the similarity matrix, respectively.

The cluster appeared right when the injury took place, and it maintained popularity during the second and third milestones, where the tweets were very sympathetic with Kevin Ware. The topic

¹³http://www.nytimes.com/2013/04/01/sports/ncaabasketball/kevin-ware-gruesome-injury-shakes-and-rallies-louisville.html?_r=0

Table 4.31: Twitter Stories: Kevin Ware Injury Topic Cluster Evolution Properties

Milestone	Dates	Top Tweet (before pre-processing)
1	3/29/13-3/31/13	praying for kevin warelouisville cardinals recovery
2	3/31/13	pray for kevin ware and the louisville team the worst ncaa injury ive ever seen for your respect
3	3/31/13	praying for ware after seeing the teams reaction firmly rooting for louisville now rivalry damned
4	4/1/13-4/4/13	kevin ware out the hospital and heading back louisville back brothers

Figure 4.76: Twitter Stories: Kevin Ware Injury Topic Cluster Evolution

spiked again at the fourth milestone, when Kevin Ware left the hospital and went to join his team again.

The NCAA Championship Topic The Louisville basketball team won the NCAA championship in April 2013¹⁴. Stream-Dashboard was able to detect several topics related to this event, and the centroids of some of these topics are shown in Figure 4.77. Some of the interesting discussions and their top raw tweets are shown in Table 4.32.

4.4 Summary and Conclusions

In this chapter, we presented some of the experiments to evaluate the performance of the proposed framework. More specifically, Section 4.1 compared the proposed online clustering algorithm, RINO-Streams, against four competing algorithms. The results showed better performance of RINO-Streams over the competing algorithms in terms of robustness to noise, invariance to order of data arrival, higher quality of clusters and time complexity. Section 4.2 presented some of the results obtained when tracking the evolution of clusters over time. TRACER proved its ability

¹⁴http://en.wikipedia.org/wiki/2013_NCAA_Men's_Division_I_Basketball_Tournament

Figure 4.77: Twitter Stories: NCAA topic centroids

Table 4.32: Twitter Stories: NCAA tweet discussions (Note: slang **)

Topic	Top Raw Tweets
2 stellar players: Albrecht (Michigan) and Hancock (Louisville)	louisville reminds centralia their red and white
	that wh**e boy killin louisville
	lil wh**e boy from louisville look like one the little rascals
	louisville has wh**e boy who can shoot too
Michigan surprised Louisville with Albrecht	louisville man who this wh**e boy from michigan
	louisville all like have wh**e guy nobody ever heard too
	albrecht unreal louisvilles pitino needs find answer
	albrecht your embarrassing louisville you werent even the scouting report
Fan Love/Anger: blaming sympathy for Louisville on Kevin Ware's Injury	yall didnt know louisville existed till that n**** broke his leg
	n**** said louisville sacrificed kevin ware leg for the championship
	the state kentucky going nuts last year louisville this year aint nobody f**** with basketball
	half these h**s louisville d*** now but was all uks last year

to accurately replicate the evolution of cluster metrics over time and detect the milestones, which reflect major changes in the clusters' evolution. Section 4.3 presented how the proposed framework can be used in a real world application, Twitter, to detect the trending topics over time.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Summary

This dissertation started with a quest to formulate a methodology for clustering data streams and tracking and validating the cluster evolution with time. For this purpose, we had to answer the question of whether a single cluster model is a sufficient output of the data stream clustering process and if not, which cluster model should be reported or stored? Since in a truly adaptive incremental stream mining approach, the cluster models are updated at every point of time, throughout the data stream’s life, we incur the risk that the clustering process itself would generate a distinct cluster model at each time step, meaning that the clustering process will generate its own data stream that will add even more overhead to the entire stream cluster mining, validation and tracking task. To avoid this problem, we first proposed a clustering algorithm to summarize the data stream by a robust cluster model. This contribution makes the online clustering component, RINO-Streams. Then we developed a methodology to summarize the “evolving clustering output” of the first component into a concise set of simple linear models and milestones of significant change for each cluster, by using a second component, called TRACER. Finally we developed a simple visualization graph (the Genealogy Graph) to display the relationships between the clusters throughout different milestones, thus making it easy to quickly inspect the clusters’ external transitions such as merging and splittings, as well as the emergence of new clusters and the provenance of evolved clusters relative to older ones.

This dissertation has accomplished the following contributions: *(i)* The main contribution of this work was a complete framework to simultaneously *mine*, *track* and *validate* the detected clusters in

big data streams. The proposed framework, termed Stream-Dashboard, made an emphasis on the behavior of the detected cluster's metrics in a data stream over time, rather than just detecting the clusters, and provided the means to investigate the clusters' evolution characteristics at any point in time. Stream-Dashboard consists of three main components: an online data stream clustering algorithm (Section 3.1), a component for tracking and validation of cluster behavior using regression analysis (Section 3.2), and a component that exploits the observed clusters' behavior as feedback to improve the quality of the online clustering component (Section 3.3). The framework is generic in the sense that *any* online clustering algorithm can be used in the first component, in order to detect the clusters. The only requirement: that this clustering algorithm quantifies the characteristics of the detected clusters. *(ii)* For the first component, we proposed the RINO-Streams algorithm (Section 3.1), an online clustering algorithm that incrementally updates the clustering model using robust statistics. RINO-Streams complies with all the requirements of data stream clustering discussed in Section 3.1.10, and is robust to noise thanks to the use of robust statistics, combined with using distribution-independent Chebyshev bounds to detect outliers and to detect similar clusters to be merged. *(iii)* For the second component, we proposed TRACER (Section 3.2), a method that uses *regression analysis* to keep track of the evolution of the detected clusters and their metrics through time, and that stores only *milestones* corresponding to the occurrence of significant changes in the clusters' behavior. Instead of storing an infinite number of summaries of the stream (at each instant or an arbitrary sample of summaries), only *temporally salient* synopsis snapshots of the stream will be stored to disk when significant changes are detected, together with a model of this change in between consecutive salient snapshots. *(iv)* Tracking the behavior of the clustering model over time can eventually help in building *behavioral profiles* of 'good' and 'bad' clusters. These profiles could be used as feedback to improve the performance of the online clustering model by reducing the sensitivity associated with using some of the threshold parameters needed to judge the quality of the clusters, and by offering a better means to initialize the input parameters used within the online clustering algorithm (Section 3.3). *(v)* Tracking the evolution of the data stream is presented to the user in the form of a *dashboard* that enables the user to control the input parameters of the framework, displays the changes of the cluster metrics over time, identifies the detected evolution milestones (Section 3.2.3), and creates an ancestry tree of all the clusters detected (Section 3.4).

We have performed exhaustive experimentation to evaluate the performance of the proposed

framework from different aspects, ranging from internal validation to measure the compactness and separation of the computed clusters, to external validation against available class labels when these were available, and finally time complexity. Based on exhaustive experiments with more than 8000 data streams with varying stream conditions, in addition to several real text data sets and network intrusion data sets, the proposed RINO-Streams algorithm was found to be superior in most aspects to several online clustering algorithms, such as CluStreams and Growing K-Means. A complete sensitivity analysis and ANOVA analysis was performed on both the online clustering component (RINO-Streams) and the tracking component (TRACER), revealing insights that can be used to help determine which parameters are most critical on the results of two components and hence need to be fine-tuned in the future.

We have concluded by illustrating a real life application of the proposed stream clustering methodology to mine topic cluster stories and characterize their evolution from continuous twitter streams. For this particular problem, we exploited the modularity of the dashboard visualization to add specialized visualizations using tag clouds to summarize each cluster in an appealing way.

5.2 Future Work

Clustering evolving data streams is a challenging problem that requires fine-tuning several parameters. Although our exhaustive sensitivity analysis and ANOVA study helped reveal insights regarding this challenge, more work can be done to automate the setting of at least part of the parameters depending on the data stream properties and online tracking outputs. While our planned configuration component planned to do just that, it was not fleshed out in all its details, and thus we consider its refinement as part of potential future work. The real application to twitter was probably the most challenging from the point of view of setting and validation, given the huge complexity of the term space and the outputs, and would therefore also benefit from more thorough work and experimentation. Finally, additional work can be done to explore different robust weight functions in the objective function of RINO-Streams to adapt to the specifics of different domains, and more work should be done to explore using the Stream-Dashboard in different domains where noisy evolving data streams abound.

REFERENCES

- Aggarwal, Charu C. 2003. A Framework for Diagnosing Changes in Evolving Data Streams. *Pages 575–586 of: ACM SIGMOD Conference*. San Diego, CA, USA: ACM.
- Aggarwal, Charu C., Han, Jiawei, Wang, Jianyong, & Yu, Philip S. 2003. A Framework for Clustering Evolving Data Streams. *Pages 81–92 of: In VLDB*. Berlin, Germany: VLDB Endowment.
- Allan, James, Carbonell, Jaime, Doddington, George, Yamron, Jonathan, Yang, Yiming, Amherst, Umass, & Umass, James Allan. 1998. Topic Detection and Tracking Pilot Study. *In: Proceedings of the Broadcast News Transcription and Understanding Workshop*.
- Babcock, Brian, Babu, Shivnath, Datar, Mayur, Motwani, Rajeev, & Widom, Jennifer. 2002. Models and issues in data stream systems. *Pages 1–16 of: PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. New York, NY, USA: ACM.
- Banerjee, Amit, & Dave, Rajesh N. 2004. Validating clustering using the Hopkins statistics. *Pages 149–153 of: Fuzzy Systems Proceedings. 2004 IEEE International Conference*, vol. 1. Budapest, Hungary: IEEE.
- Barbará, Daniel. 2002. Requirements for clustering data streams. *ACM SIGKDD Explorations Newsletter*, 3(2), 23–27.
- Barbará, Daniel, & Chen, Ping. 2000. Using the Fractal Dimension to Cluster Datasets. *Pages 260–264 of: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. Boston, MA, USA: ACM.
- Baron, Steffan, Spiliopoulou, Myra, & Günther, Oliver. 2003. Efficient Monitoring of Patterns in

- Data Mining Environments. *Pages 253–265 of: Advances in Databases and Information Systems*. Lecture Notes in Computer Science, vol. 2798. Heidelberg, Germany: Springer.
- Berkhin, Pavel. 2006. A survey of clustering data mining techniques. *Pages 25–71 of: Grouping multidimensional data*. Springer.
- Bezdek, James C. 1981. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers.
- Bifet, Albert, & Frank, Eibe. 2010. Sentiment knowledge discovery in twitter streaming data. *Pages 1–15 of: Discovery Science*. Springer.
- Bifet, Albert, Holmes, Geoff, Kirkby, Richard, & Pfahringer, Bernhard. 2010. MOA: Massive online analysis. *The Journal of Machine Learning Research*, **11**, 1601–1604.
- Blei, David M, Ng, Andrew Y, & Jordan, Michael I. 2003. Latent dirichlet allocation. *the Journal of machine Learning research*, **3**, 993–1022.
- Boley, Daniel. 1997. Principal Direction Divisive Partitioning. *Data Mining and Knowledge Discovery*, **2**, 325–344.
- Böttcher, Mirko, Höppner, Frank, & Spiliopoulou, Myra. 2008. On exploiting the power of time in data mining. *ACM SIGKDD Explorations Newsletter*, **10**(2), 3–11.
- Cao, Feng, Ester, Martin, Qian, Weining, & Zhou, Aoying. 2006. Density-based clustering over an evolving data stream with noise. *Pages 328–339 of: In 2006 SIAM Conference on Data Mining*. Bethesda, Maryland: SIAM.
- Casella, George, & Berger, Roger L. 2001. *Statistical Inference*. Duxbury Press.
- Chakrabarti, Deepayan, Kumar, Ravi, & Tomkins, Andrew. 2006. Evolutionary clustering. *Pages 554–560 of: KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM.
- Charikar, Moses, O’Callaghan, Liadan, & Panigrahy, Rina. 2003. Better Streaming Algorithms for Clustering Problems. *Pages 30–39 of: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*. San Diego, CA, USA: ACM Press.

- Chatterjee, Samprit, & Hadi, Ali S. 2006. *Regression Analysis by Example (Wiley Series in Probability and Statistics)*. 4 edn. Wiley-Interscience.
- Chen, Yixin, & Tu, Li. 2007. Density-based clustering for real-time stream data. *Pages 133–142 of: KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM.
- Coombs, Clyde Hamilton, Dawes, Robyn M, & Tversky, Amos. 1970. *Mathematical psychology: An elementary introduction*. Prentice-Hall Englewood Cliffs, NJ.
- Davies, David L, & Bouldin, Donald W. 1979. A cluster separation measure. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **2**(2), 224–227.
- Dempster, A. P., Laird, N. M., & Rubin, D. B. 1977. Maximum likelihood from incomplete data via the EM algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, **39**(1), 1–38.
- Dumais, Susan T, Furnas, GW, Landauer, TK, Deerwester, S, Deerwester, SC, *et al.* 1995. Latent semantic indexing. *Pages 219–230 of: Proceedings of the Text Retrieval Conference*. Gaithersburg, Maryland: NIST.
- Dunn, J. C. 1973. A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters. *Journal of Cybernetics*, **3**, 32–57.
- E.Bellman, Richard. 1961. *Adaptive control processes - A guided tour*. Princeton, New Jersey, U.S.A.: Princeton University Press.
- Ester, Martin, peter Kriegel, Hans, S, Jörg, & Xu, Xiaowei. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. *Pages 226–231 of: KDD*. Portland, Oregon, USA: AAAI Press.
- Ester, Martin, Kriegel, Hans-Peter, Sander, Jörg, Wimmer, Michael, & Xu, Xiaowei. 1998. Incremental Clustering for Mining in a Data Warehousing Environment. *Pages 323–333 of: Proceedings of the International Conference on Very Large Data Bases*. Institute of Electrical & Electronics Engineers, New York City, USA.

- Gaber, Mohamed M., Zaslavsky, Arkady, & Krishnaswamy, Shonali. 2005. Mining data streams: a review. *ACM Sigmod Record*, **34**(2), 18–26.
- Gaber, Mohamed Medhat, & Yu, Philip S. 2006. Classification of Changes in Evolving Data Streams using Online Clustering Result Deviation. *In: Proceedings Of Internatinal Workshop on Knowledge Discovery in Data Streams*. ACM, Pittsburgh, PA, USA.
- Ganti, Venkatesh, Gehrke, Johannes, Ramakrishnan, Raghu, & yin Loh, Wei. 1998. A Framework for Measuring Changes in Data Characteristics. *Pages 126–137 of: In PODS*. ACM Press.
- Golub, G., & Kahan, W. 1965. Calculating the Singular Values and Pseudo-Inverse of a Matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, **2**(2), 205–224.
- Guha, Sudipto, Rastogi, Rajeev, & Shim, Kyuseok. 1998. CURE: an efficient clustering algorithm for large databases. *Pages 73–84 of: SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM.
- Guha, Sudipto, Mishra, Nina, Motwani, Rajeev, & O'Callaghan, Liadan. 2000. Clustering data streams. *Pages 359–366 of: Foundations of computer science, 2000. proceedings. 41st annual symposium on*. Redondo Beach, CA, USA: IEEE.
- Guha, Sudipto, Meyerson, Adam, Mishra, Nina, & Motwani, Rajeev. 2003. Clustering data streams: Theory and practice. *IEEE TKDE*, **15**, 515–528.
- Günemann, Stephan, Kremer, Hardy, Laufkötter, Charlotte, & Seidl, Thomas. 2011. Tracing evolving clusters by subspace and value similarity. *Pages 444–456 of: Proceedings of the 15th Pacific-Asia conference on Advances in knowledge discovery and data mining - Volume Part II*. PAKDD'11. Berlin, Heidelberg: Springer-Verlag.
- Halkidi, Maria, Batistakis, Yannis, & Vazirgiannis, Michalis. 2002a. Cluster Validity Methods: Part I. *ACM SIGMOD Record*, **31**, 2002.
- Halkidi, Maria, Batistakis, Yannis, & Vazirgiannis, Michalis. 2002b. Clustering validity checking methods: part II. *ACM SIGMOD Record*, **31**(3), 19–27.

- Hartigan, John A. 1975. *Clustering algorithms*. John Wiley & Sons, Inc.
- Hathaway, Richard, Bezdek, James, & Huband, Jacalyn. 2006. Scalable visual assessment of cluster tendency for large data sets. *Pattern Recognition*, **39**(7), 1315–1324.
- Hawwash, Basheer, & Nasraoui, Olfa. 2010. Robust Clustering of Data Streams using Incremental Optimization. In: *The IVth Alberto Mendelzon International Workshop on Foundations of Data Management. AMW*. Buenos Aires, Argentina: CEUR-WS.org.
- Hawwash, Basheer, & Nasraoui, Olfa. 2012. Stream-dashboard: a framework for mining, tracking and validating clusters in a data stream. *Pages 109–117 of: Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*. Beijing, China: ACM, for ACM.
- Hinneburg, Alexander, & Keim, Daniel A. 1998. An Efficient Approach to Clustering in Large Multimedia Databases with Noise. *Pages 58–65 of: Knowledge Discovery and Data Mining*. New York City, New York, USA: ACM.
- Hoffman, Matthew D, Blei, David M, & Bach, Francis. 2010. Online learning for latent dirichlet allocation. *Advances in Neural Information Processing Systems*, **23**, 856–864.
- Hofmann, Thomas. 1999. Probabilistic latent semantic indexing. *Pages 50–57 of: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM.
- Hore, Prodip, Hall, Lawrence O., & Goldgof, Dmitry B. 2007. A fuzzy c means variant for clustering evolving data streams. *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, 360–365.
- Huber, Peter J. 1981. *Robust Statistics*. Wiley-Interscience.
- Ihara, Shunsuke. 1993. *Information theory for continuous system*. Vol. 2. World Scientific Publishing Company.
- Jaccard, Paul. 1901. *Etude comparative de la distribution florale dans une portion des Alpes et du Jura*. Impr. Corbaz.

- Jain, Anil K. 2010. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, **31**(8), 651–666.
- Jansen, Bernard J, Zhang, Mimi, Sobel, Kate, & Chowdury, Abdur. 2009. Twitter power: Tweets as electronic word of mouth. *Journal of the American society for information science and technology*, **60**(11), 2169–2188.
- Karkkainen, Ismo, & Franti, Pasi. 2007. Gradual model generator for single-pass clustering. *Pattern Recognition*, **40**(3), 784–795.
- Karypis, George. 2002. *CLUTO-a clustering toolkit*. Tech. rept. DTIC Document.
- Karypis, George, & Kumar, Vipin. 1998. *hMETIS 1.5: A hypergraph partitioning package*. Tech. rept. Technical report, Department of Computer Science, University of Minnesota, 1998. Available on the WWW at URL <http://www.cs.umn.edu/metis>.
- Karypis, George, & Kumar, Vipin. 1999. Chameleon: Hierarchical Clustering Using Dynamic Modeling. *Computer*, **32**(8), 68–75.
- Kifer, Daniel, Ben-David, Shai, & Gehrke, Johannes. 2004. Detecting change in data streams. *Pages 180–191 of: Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*. VLDB '04. VLDB Endowment.
- Kruskal, William. 1988. Miracles and Statistics: The Casual Assumption of Independence. *Journal of the American Statistical Association*, **83**, 929–940.
- Kung, Hsiang-Tsung, Luccio, Fabrizio, & Preparata, Franco P. 1975. On finding the maxima of a set of vectors. *Journal of the ACM*, **22**(4), 469–476.
- Leuski, Anton, & Allan, James. 1997. *Interactive cluster visualization for information retrieval*. Tech. rept. In Proceedings of ECDL'98.
- Lewis, D.D. 1997. *Reuters-21578 Text Categorization Test Collection*. <http://www.daviddlewis.com/resources/testcollections/reuters21578/>.
- Ling, Robert L. 1973. A computer generated aid for cluster analysis. *Communications of the ACM*, **16**(6), 355–361.

- Liu, Jun S. 1994. The collapsed Gibbs sampler in Bayesian computations with applications to a gene regulation problem. *Journal of the American Statistical Association*, **89**(427), 958–966.
- Lowette, S., & Laerhoven, K. V. 2004. <http://www.teco.edu/tea/unsup/overview.html>.
- MacQueen, J. B. 1967. Some Methods for Classification and Analysis of MultiVariate Observations. *Pages 281–297 of: Cam, L. M. Le, & Neyman, J. (eds), Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1. University of California Press.
- Marshall, Albert W., & Olkin, Ingram. 1960. Multivariate Chebyshev Inequalities. *The Annals of Mathematical Statistics*, **31**, 1001–1014.
- Mendoza, Marcelo, Poblete, Barbara, & Castillo, Carlos. 2010. Twitter Under Crisis: Can we trust what we RT? *Pages 71–79 of: Proceedings of the first workshop on social media analytics*. ACM.
- Mood, A McF. 1950. *Introduction to the Theory of Statistics*. McGraw-hill.
- Nasraoui, Olfa, & Krishnapuram, Raghu. 1996. A robust estimator based on density and scale optimization and its application to clustering. *Pages 1031–1035 of: Fuzzy Systems, 1996., Proceedings of the Fifth IEEE International Conference on*, vol. 2. IEEE.
- Nasraoui, Olfa, & Rojas, Carlos. 2006. Robust Clustering for Tracking Noisy Evolving Data Streams. *In: Proceedings of the SIAM Conference on Data Mining*. Bethesda, MD, USA: SIAM.
- Nasraoui, Olfa, Uribe, Cesar Cardona, Coronel, Carlos Rojas, & Gonzalez, Fabio. 2003. TECNOSTREAMS: Tracking Evolving Clusters in Noisy Data Streams with a Scalable Immune System Learning Model. *Pages 235–242 of: Proceedings of the Third IEEE International Conference on Data Mining*. ICDM '03. Washington, DC, USA: IEEE Computer Society.
- Nasraoui, Olfa, Soliman, Maha, Saka, Esin, Badia, Antonio, & Germain, Richard. 2008. A Web Usage Mining Framework for Mining Evolving User Profiles in Dynamic Web Sites. *IEEE Transactions on Knowledge and Data Engineering*, **20**, 202–215.
- Pearson, Karl. 1901. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, **2**(6), 559–572.

- Pearson, Karl. 1904. *On the theory of contingency and its relation to association and normal correlation; On the general theory of skew correlation and non-linear regression*. Cambridge University Press.
- Rand, William M. 1971. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, **66**(336), 846–850.
- Ricardo A. Maronna, Douglas R. Martin, Víctor J. Yohai. 2006. *Robust Statistics*. Wiley-Interscience.
- Rousseeuw, Peter J. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, **20**, 53–65.
- Sarwar, Badrul M., Karypis, George, Konstan, Joseph, & Riedl, John. 2002. *Recommender Systems for Large-scale E-Commerce: Scalable Neighborhood Formation Using Clustering*.
- Sheikholeslami, Gholamhosein, Chatterjee, Surojit, & Zhang, Aidong. 2000. WaveCluster: a wavelet-based clustering approach for spatial data in very large databases. *The VLDB Journal*, **8**(3-4), 289–304.
- Shlens, Jonathon. 2005. A tutorial on Principal Component Analysis. *In: Systems Neurobiology Laboratory, Salk Institute for Biological Studies*.
- Spiliopoulou, Myra, Ntoutsi, Irene, Theodoridis, Yannis, & Schult, Rene. 2006. MONIC: modeling and monitoring cluster transitions. *Pages 706–711 of: KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM.
- Stonebraker, Michael, Çetintemel, Uğur, & Zdonik, Stan. 2005. The 8 requirements of real-time stream processing. *SIGMOD Record*, **34**(4), 42–47.
- Tan, Pang-Ning, Steinbach, Michael, & Kumar, Vipin. 2005. *Introduction to Data Mining*. Addison Wesley.
- van Rijsbergen, Cornelis Joost. 1977. A theoretical basis for the use of co-occurrence data in information retrieval. *Journal of documentation*, **33**(2), 106–119.

- Wallach, Hanna M. 2006. Topic modeling: beyond bag-of-words. *Pages 977–984 of: Proceedings of the 23rd international conference on Machine learning*. ACM.
- Wang, Wei, Yang, Jiong, & Muntz, Richard. 1997. STING: A statistical information grid approach to spatial data mining. *Pages 186–195 of: Proceedings of the International Conference on Very Large Data Bases*. Athens, Greece: Morgan Kaufmann.
- Ward, Joe H. 1963. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, **58**(301), 236–244.
- Zhang, Tian, Ramakrishnan, Raghu, & Livny, Miron. 1996. BIRCH: an efficient data clustering method for very large databases. *Pages 103–114 of: ACM SIGMOD Record*, vol. 25. Montreal, Canada: ACM, for ACM.
- Zhong, Shi. 2005. Efficient online spherical k-means clustering. *Pages 3180–3185 of: Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, vol. 5. IEEE, Montreal, Quebec, Canada.

Appendix A

Combining two regression models from two consecutive time periods:

First, we will review some arithmetic series identities. For a simple arithmetic series, the sum can be found as:

$$\sum_{i=1}^n i = \frac{n(n+1)}{2} \quad (5.1)$$

$$\sum_{i=m}^n i = \frac{(n-m+1)(n+m)}{2} \quad (5.2)$$

For a power arithmetic series, the sum can be found as follows:

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6} \quad (5.3)$$

$$\begin{aligned} \sum_{i=m}^n i^2 &= \sum_{i=1}^n i^2 - \sum_{i=1}^{m-1} i^2 \\ &= \frac{n(n+1)(2n+1)}{6} - \frac{(m-1)(m)(2m-1)}{6} \\ &= \frac{n(n+1)(2n+1) - m(m-1)(2m-1)}{6} \end{aligned} \quad (5.4)$$

For the arithmetic series that represents the difference between each value and the mean, the sum can be found using (5.2) and (5.4) as follows:

$$\begin{aligned}
\sum_{i=m}^n (i - \bar{i})^2 &= \sum_{i=m}^n (i^2 - 2i\bar{i} + \bar{i}^2) \\
&= \sum_{i=m}^n i^2 - 2\bar{i} \sum_{i=m}^n i + (n-m+1)\bar{i}^2 \\
&= \sum_{i=m}^n i^2 - 2 \frac{\sum_{i=m}^n i}{(n-m+1)} \sum_{i=m}^n i + (n-m+1) \left(\frac{\sum_{i=m}^n i}{(n-m+1)} \right)^2 \\
&= \sum_{i=m}^n i^2 - 2 \frac{(\sum_{i=m}^n i)^2}{(n-m+1)} + \frac{(\sum_{i=m}^n i)^2}{(n-m+1)} \\
&= \sum_{i=m}^n i^2 - \frac{(\sum_{i=m}^n i)^2}{(n-m+1)} \\
&= \frac{n(n+1)(2n+1) - m(m-1)(2m-1)}{6} - \frac{\left(\frac{(n-m+1) \times (n+m)}{2} \right)^2}{(n-m+1)} \\
&= \frac{n(n+1)(2n+1) - m(m-1)(2m-1)}{6} - \frac{(n-m+1)(n+m)^2}{4} \\
&= \frac{2n(n+1)(2n+1) - 2m(m-1)(2m-1) - 3(n^3 + mn^2 + n^2 - m^2n + 2mn - m^3 + m^2)}{12} \\
&= \frac{2n(n+1)(2n+1) - 2m(m-1)(2m-1) - 3(n^2(n+1) - m^2(m-1) + mn(n-m+2))}{12} \\
&= \frac{n(n+1)(2(2n+1) - 3n) - m(m-1)(2(2m-1) - 3m) - 3mn(n-m+2)}{12} \\
&= \frac{n(n+1)(n+2) - m(m-1)(m-2) - 3mn(n-m+2)}{12}
\end{aligned} \tag{5.5}$$

Rearranging the slope β_1 of a regression model yields:

$$\begin{aligned}
\beta_1 &= \frac{\sum_{i=m}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=m}^n (x_i - \bar{x})^2} \\
&= \frac{\sum_{i=m}^n x_i y_i - \bar{y} \sum_{i=m}^n x_i - \bar{x} \sum_{i=m}^n y_i + (n-m+1)\bar{y}\bar{x}}{\sum_{i=m}^n (x_i - \bar{x})^2} \\
&= \frac{\sum_{i=m}^n x_i y_i - \bar{y} \sum_{i=m}^n x_i - \bar{x}(n-m+1)\bar{y} + (n-m+1)\bar{y}\bar{x}}{\sum_{i=m}^n (x_i - \bar{x})^2} \\
&= \frac{\sum_{i=m}^n x_i y_i - \bar{y} \sum_{i=m}^n x_i}{\sum_{i=m}^n (x_i - \bar{x})^2}
\end{aligned} \tag{5.6}$$

Therefore:

$$\sum_{i=m}^n x_i y_i = \beta_1 \sum_{i=m}^n (x_i - \bar{x})^2 + \bar{y} \sum_{i=m}^n x_i \tag{5.7}$$

For two consecutive time periods with lengths of multiplies of Δ_{Reg} : $T_1 = [z_1 \Delta_{Reg} + 1 : z_2 \Delta_{Reg}]$ and $T_2 = [(z_2 + 1) \Delta_{Reg} + 1 : (z_2 + 1) \Delta_{Reg}]$, where $z_1, z_2 \in \mathbb{Z}$, $z_1 < z_2$, these identities can be inferred using (5.2) and (5.5):

$$\begin{aligned}
\bar{y}_{T_1+T_2} &= \frac{\sum_{i=z_1 \Delta_{Reg}+1}^{(z_2+1)\Delta_{Reg}} y_i}{(z_2+1)\Delta_{Reg} - (z_1 \Delta_{Reg} + 1) + 1} \\
&= \frac{\sum_{i=z_1 \Delta_{Reg}+1}^{z_2 \Delta_{Reg}} y_i + \sum_{i=z_2 \Delta_{Reg}+1}^{(z_2+1)\Delta_{Reg}} y_i}{\Delta_{Reg} (z_2 - z_1 + 1)} \\
&= \frac{\Delta_{Reg} (z_2 - z_1) \bar{y}_{T_1} + \Delta_{Reg} \bar{y}_{T_2}}{\Delta_{Reg} (z_2 - z_1 + 1)} \\
&= \frac{(z_2 - z_1) \bar{y}_{T_1} + \bar{y}_{T_2}}{(z_2 - z_1 + 1)}
\end{aligned} \tag{5.8}$$

$$\begin{aligned}
\sum_{i=z_1 \Delta_{Reg}+1}^{z_2 \Delta_{Reg}} i &= \frac{(z_2 \Delta_{Reg} - z_1 \Delta_{Reg} - 1 + 1)(z_2 \Delta_{Reg} + z_1 \Delta_{Reg} + 1)}{2} \\
&= \frac{\Delta_{Reg} (z_2 - z_1)(z_2 \Delta_{Reg} + z_1 \Delta_{Reg} + 1)}{2}
\end{aligned} \tag{5.9}$$

$$\begin{aligned}
\sum_{i=z_2 \Delta_{Reg}+1}^{(z_2+1)\Delta_{Reg}} i &= \frac{((z_2+1)\Delta_{Reg} - z_2 \Delta_{Reg} - 1 + 1)((z_2+1)\Delta_{Reg} + z_2 \Delta_{Reg} + 1)}{2} \\
&= \frac{\Delta_{Reg} (2z_2 \Delta_{Reg} + \Delta_{Reg} + 1)}{2}
\end{aligned} \tag{5.10}$$

$$\begin{aligned}
\sum_{i=z_1 \Delta_{Reg}+1}^{(z_2+1)\Delta_{Reg}} i &= \frac{((z_2+1)\Delta_{Reg} - z_1 \Delta_{Reg} - 1 + 1)((z_2+1)\Delta_{Reg} + z_1 \Delta_{Reg} + 1)}{2} \\
&= \frac{\Delta_{Reg} (z_2 - z_1 + 1)(\Delta_{Reg} z_1 + \Delta_{Reg} (z_2 + 1) + 1)}{2}
\end{aligned} \tag{5.11}$$

$$\begin{aligned}
\sum_{i=z_1 \Delta_{Reg}+1}^{z_2 \Delta_{Reg}} (i - \bar{i}_{T_1})^2 &= \frac{z_2 \Delta_{Reg} (z_2 \Delta_{Reg} + 1)(z_2 \Delta_{Reg} + 2) - (z_1 \Delta_{Reg} + 1)(z_1 \Delta_{Reg} + 1 - 1)(z_1 \Delta_{Reg} + 1 - 2)}{12} \\
&\dots \frac{-3z_2 \Delta_{Reg} (z_1 \Delta_{Reg} + 1)(z_2 \Delta_{Reg} - (z_1 \Delta_{Reg} + 1) + 2)}{12} \\
&= \frac{\Delta_{Reg} (z_1 - z_2)(\Delta_{Reg} z_1 - \Delta_{Reg} z_2 + 1)(\Delta_{Reg} z_2 - \Delta_{Reg} z_1 + 1)}{12}
\end{aligned} \tag{5.12}$$

$$\begin{aligned}
\sum_{i=z_2 \Delta_{Reg}+1}^{(z_2+1)\Delta_{Reg}} (i - \bar{i}_{T_2})^2 &= \frac{(z_2+1)\Delta_{Reg}((z_2+1)\Delta_{Reg}+1)((z_2+1)\Delta_{Reg}+2) - (z_2 \Delta_{Reg} + 1)(z_2 \Delta_{Reg} + 1 - 1)(z_2 \Delta_{Reg} + 1 - 2)}{12} \\
&\dots \frac{-3(z_2+1)\Delta_{Reg}(z_1 \Delta_{Reg} + 1)((z_2+1)\Delta_{Reg} - (z_1 \Delta_{Reg} + 1) + 2)}{12} \\
&= \frac{\Delta_{Reg}^3 - \Delta_{Reg}}{12}
\end{aligned} \tag{5.13}$$

$$\begin{aligned}
\sum_{i=z_1 \Delta_{Reg}+1}^{(z_2+1)\Delta_{Reg}} (i - \bar{i}_{T_1, T_2})^2 &= \frac{(z_2+1)\Delta_{Reg}((z_2+1)\Delta_{Reg}+1)((z_2+1)\Delta_{Reg}+2) - (z_1 \Delta_{Reg} + 1)(z_1 \Delta_{Reg} + 1 - 1)(z_1 \Delta_{Reg} + 1 - 2)}{12} \\
&\dots \frac{-3(z_2+1)\Delta_{Reg}(z_1 \Delta_{Reg} + 1)((z_2+1)\Delta_{Reg} - (z_1 \Delta_{Reg} + 1) + 2)}{12} \\
&= \frac{\Delta_{Reg} (z_2 - z_1 + 1)(\Delta_{Reg} - \Delta_{Reg} z_1 + \Delta_{Reg} z_2 - 1)(\Delta_{Reg} - \Delta_{Reg} z_1 + \Delta_{Reg} z_2 + 1)}{12}
\end{aligned} \tag{5.14}$$

Using (5.7), the slopes for the regression models on time periods T_1 and T_2 can be rearranged as

follows:

$$\sum_{i=z_1\Delta_{Reg}+1}^{z_2\Delta_{Reg}} iy_i = \beta_{1,T_1} \sum_{i=z_1\Delta_{Reg}+1}^{z_2\Delta_{Reg}} (i - \bar{i}_{T_1})^2 + \bar{y}_{T_1} \sum_{i=z_1\Delta_{Reg}+1}^{z_2\Delta_{Reg}} i \quad (5.15)$$

$$\sum_{i=z_2\Delta_{Reg}+1}^{(z_2+1)\Delta_{Reg}} iy_i = \beta_{1,T_2} \sum_{i=z_2\Delta_{Reg}+1}^{(z_2+1)\Delta_{Reg}} (i - \bar{i}_{T_2})^2 + \bar{y}_{T_2} \sum_{i=z_2\Delta_{Reg}+1}^{(z_2+1)\Delta_{Reg}} i \quad (5.16)$$

And by substituting (5.9), (5.10), (5.12) and (5.13):

$$\begin{aligned} \sum_{i=z_1\Delta_{Reg}+1}^{z_2\Delta_{Reg}} iy_i &= \beta_{1,T_1} \frac{\Delta_{Reg}(z_1 - z_2)(\Delta_{Reg}z_1 - \Delta_{Reg}z_2 + 1)(\Delta_{Reg}z_2 - \Delta_{Reg}z_1 + 1)}{12} \\ &\dots + \bar{y}_{T_1} \frac{\Delta_{Reg}(z_2 - z_1)(z_2\Delta_{Reg} + z_1\Delta_{Reg} + 1)}{2} \end{aligned} \quad (5.17)$$

$$\sum_{i=z_2\Delta_{Reg}+1}^{(z_2+1)\Delta_{Reg}} iy_i = \beta_{1,T_2} \frac{\Delta_{Reg}^3 - \Delta_{Reg}}{12} + \bar{y}_{T_2} \frac{\Delta_{Reg}(2z_2\Delta_{Reg} + \Delta_{Reg} + 1)}{2} \quad (5.18)$$

Now, the new slope β_{1,T_1+T_2} of both regression models can be found using (5.6) as follows:

$$\begin{aligned} \beta_{1,T_1+T_2} &= \frac{\sum_{i=z_1\Delta_{Reg}+1}^{(z_2+1)\Delta_{Reg}} iy_i - \bar{y}_{T_1+T_2} \sum_{i=z_1\Delta_{Reg}+1}^{(z_2+1)\Delta_{Reg}} i}{\sum_{i=z_1\Delta_{Reg}+1}^{(z_2+1)\Delta_{Reg}} (i - \bar{i}_{T_1+T_2})^2} \\ &= \frac{\sum_{i=z_1\Delta_{Reg}+1}^{z_2\Delta_{Reg}} iy_i + \sum_{i=z_2\Delta_{Reg}+1}^{(z_2+1)\Delta_{Reg}} iy_i - \bar{y}_{T_1+T_2} \sum_{i=z_1\Delta_{Reg}+1}^{(z_2+1)\Delta_{Reg}} i}{\sum_{i=z_1\Delta_{Reg}+1}^{(z_2+1)\Delta_{Reg}} (i - \bar{i}_{T_1+T_2})^2} \end{aligned} \quad (5.19)$$

And by substituting (5.8), (5.11) and (5.14):

$$\beta_{1,T_1+T_2} = \frac{\sum_{i=z_1\Delta_{Reg}+1}^{z_2\Delta_{Reg}} iy_i + \sum_{i=z_2\Delta_{Reg}+1}^{(z_2+1)\Delta_{Reg}} iy_i - \left(\frac{(z_2 - z_1)\bar{y}_{T_1} + \bar{y}_{T_2}}{(z_2 - z_1 + 1)} \right) \left(\frac{\Delta_{Reg}(z_2 - z_1 + 1)(\Delta_{Reg}z_1 + \Delta_{Reg}(z_2 + 1) + 1)}{2} \right)}{\frac{\Delta_{Reg}(z_2 - z_1 + 1)(\Delta_{Reg} - \Delta_{Reg}z_1 + \Delta_{Reg}z_2 - 1)(\Delta_{Reg} - \Delta_{Reg}z_1 + \Delta_{Reg}z_2 + 1)}{12}} \quad (5.20)$$

And finally by substituting (5.17) and (5.18), the combined slope can found as follows:

$$\begin{aligned} \beta_{1,T_1+T_2} &= \frac{\Delta_{Reg}\beta_{1,T_2}(\Delta_{Reg}^2 - 1) + 6\Delta_{Reg}\bar{y}_{T_2}(\Delta_{Reg} + 2\Delta_{Reg}z_2 + 1)}{\Delta_{Reg}(z_2 - z_1 + 1)(\Delta_{Reg} - \Delta_{Reg}z_1 + \Delta_{Reg}z_2 - 1)(\Delta_{Reg} - \Delta_{Reg}z_1 + \Delta_{Reg}z_2 + 1)} \\ &\dots \frac{-6\Delta_{Reg}\bar{y}_{T_1}(z_1 - z_2)(\Delta_{Reg}z_1 + \Delta_{Reg}z_2 + 1)}{\Delta_{Reg}(z_2 - z_1 + 1)(\Delta_{Reg} - \Delta_{Reg}z_1 + \Delta_{Reg}z_2 - 1)(\Delta_{Reg} - \Delta_{Reg}z_1 + \Delta_{Reg}z_2 + 1)} \\ &\dots \frac{-\Delta_{Reg}\beta_{1,T_1}(z_1 - z_2)(\Delta_{Reg}^2z_1^2 - 2\Delta_{Reg}^2z_1z_2 + \Delta_{Reg}^2z_2^2 - 1)}{\Delta_{Reg}(z_2 - z_1 + 1)(\Delta_{Reg} - \Delta_{Reg}z_1 + \Delta_{Reg}z_2 - 1)(\Delta_{Reg} - \Delta_{Reg}z_1 + \Delta_{Reg}z_2 + 1)} \\ &\dots \frac{-6\Delta_{Reg}(\bar{y}_{T_2} - \bar{y}_{T_1}z_1 + \bar{y}_{T_1}z_2)(\Delta_{Reg} + \Delta_{Reg}z_1 + \Delta_{Reg}z_2 + 1)}{\Delta_{Reg}(z_2 - z_1 + 1)(\Delta_{Reg} - \Delta_{Reg}z_1 + \Delta_{Reg}z_2 - 1)(\Delta_{Reg} - \Delta_{Reg}z_1 + \Delta_{Reg}z_2 + 1)} \end{aligned} \quad (5.21)$$

And the new intercept, β_{0,T_1+T_2} , can be found using (5.11) and (5.8) as follows:

$$\begin{aligned}
\beta_{0,T_1+T_2} &= \bar{y}_{T_1+T_2} - \beta_{1,T_2+T_2} \bar{i}_{T_1+T_2} \\
&= \frac{(z_2-z_1)\bar{y}_{T_1}+\bar{y}_{T_2}}{(z_2-z_1-1)} - \beta_{1,T_1+T_2} \frac{\frac{\Delta_{Reg}(z_2-z_1+1)(\Delta_{Reg}z_1+\Delta_{Reg}(z_2+1)+1)}{2}}{\Delta_{Reg}(z_2-z_1+1)} \\
&= \frac{(z_2-z_1)\bar{y}_{T_1}+\bar{y}_{T_2}}{(z_2-z_1-1)} - \beta_{1,T_1+T_2} \frac{(\Delta_{Reg}z_1+\Delta_{Reg}(z_2+1)+1)}{2}
\end{aligned} \tag{5.22}$$

CURRICULUM VITA

NAME	Basheer Hawwash
ADDRESS	Department of Computer Engineering and Computer Science University Of Louisville Louisville, KY, 40293
DOB	Beitsahour, Palestine, April 30, 1984
EDUCATION	B.S. Computer Information Systems Jordan University of Science and Technology 2002-2006 M.S. Computer Science University of Louisville 2006-2008 Ph.D Computer Science and Engineering University of Louisville 2008-2013
AWARDS	CECS Arthur M. Riehl Award, April 2013 The Graduate Dean's Citation for the Year 2008 and 2013 Who's Who Among Students, 2012 Grosscurth Doctoral Fellowship, August 2008-August 2010 The Alice Eaves Barns Award for Outstanding Achievement in a Master's Program, May 2008 CECS Masters of Science Award, April 2008