I, Olugbenga Awodokun, hereby submit this original work as part of the requirements for the degree of Master of Science in Electrical Engineering.

It is entitled:
Classification of Patterns in Streaming Data Using Clustering Signatures

Student's name: Olugbenga Awodokun

This work and its defense approved by:

Committee chair: Raj Bhatnagar, Ph.D.
Committee member: Gowtham Atluri
Committee member: Nan Niu, Ph.D.
Classification of Patterns in Streaming Data Using Clustering Signatures

A thesis submitted to the
Graduate School of the
University of Cincinnati
in partial fulfillment of the
requirements for the degree of
Master of Science

In the Department of Electrical Engineering and Computer Science
of the College of Engineering and Applied Science

by
Gbenga Awodokun

July 2017

Professor Raj Bhatnager, Thesis Advisor
Abstract

Streaming datasets often pose a myriad of challenges for machine learning algorithms, some of which include insufficient storage and changes in the underlying distributions of the data during different time intervals. This thesis proposes a hierarchical clustering based method (unsupervised learning) for determining signatures of data in a time window and thus building a classifier based on the match between the observed clusters and known patterns of clustering. When new clusters are observed, they are added to the collection of possible global list of clusters, used to generate a signature for data in a time window. Dendrograms are created from each time window, and their clusters were compared to a global list of clusters. The global clusters list is only updated if none of the existing global clusters that can model data points in any later time window. The global clusters were then used in the testing phase to classify novel data chunks according to their Tanimoto similarities. Although the training samples were only taken from 20% of the entire KDD Cup 99 dataset, we validated our approach by using test data from different regions of the datasets at multiple intervals and the classifier performance achieved was comparable to other methods that had used the entire datasets for training.
Acknowledgements

I would like to extend my deepest gratitude to Professor Raj Bhatnagar whose invaluable guidance, constant support and intellectual expertise contributed to the successful completion of this thesis. I would also like to thank Dr. Gowtham Atluri and Dr. Nan Niu for making time out of their demanding schedules to be on my thesis committee. In addition, I would like to thank all my other fellow students for providing useful suggestions and support during the course of this work. I am heavily indebted to my friends, Fatima, Abhro and Lalit for their constant moral support and valuable recommendations. I would like to take the opportunity to thank all the academic faculty who have inspired me and played an important part in helping me grow. Lastly, I will always be grateful to my loving family especially my wife Omowale for her motivation and encouragement.

I am grateful to the Almighty God for my christian faith which was a source of inspiration at difficult times during the course of the research work.
## Contents

1 Introduction .................................................. 1
   1.1 Background .............................................. 1
   1.2 Motivation .............................................. 2
   1.3 Achievements ........................................... 3
   1.4 Overview ................................................ 4

2 Related Work .................................................. 6
   2.1 Machine Learning for Intrusion Detection .............. 6
   2.2 Tracking Patterns in Streaming Background ........... 9
   2.3 Summary of Existing Algorithms ....................... 10

3 Approach ....................................................... 11
   3.1 Overview ................................................ 11
   3.2 Nature of Data Set ..................................... 14
       3.2.1 Data Preparation .................................. 16
   3.3 Experiment Design ...................................... 17
       3.3.1 Steps after Initialization ....................... 18
       3.3.2 Initialization ..................................... 20
       3.3.3 Pruning Negligible Occurrences ............... 20
       3.3.4 Linkages / Proximity Matrix ................... 21
List of Figures

1.1 Streaming mode data analytics at different time windows . . . . . . . 4
3.1 Percentage of data instances per group . . . . . . . . . . . . . . . . . 16
3.2 The algorithm explained with flowchart. - Global Cluster Update . . 19
3.3 Hierarchical Clustering Dendrogram . . . . . . . . . . . . . . . . . . . 22
3.4 Dendrogram (normal class) . . . . . . . . . . . . . . . . . . . . . . . . 23
3.5 Dendrogram (attack class) . . . . . . . . . . . . . . . . . . . . . . . . 24
3.6 Generation of Candidate Cluster - sample dendrogram x . . . . . . . 26
3.7 Hierarchical Clustering Dendrogram . . . . . . . . . . . . . . . . . . . 27
3.8 The algorithm explained with flowchart. - Classification tasks . . . 38
4.1 Execution Speed and Amount of Data required before Classification
   Comparison. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
4.2 Performance Comparison of relevant methods - normal class group. . 51
4.3 Performance Comparison of relevant methods - Dos class group. . . 52
4.4 Performance Comparison of relevant methods - Probe class group. . . 53
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Relevant Work Comparison</td>
<td>10</td>
</tr>
<tr>
<td>3.1</td>
<td>Count of the data class instance in 100% training data set</td>
<td>15</td>
</tr>
<tr>
<td>3.2</td>
<td>Count of the data class instance in 20% of the training data set</td>
<td>15</td>
</tr>
<tr>
<td>3.3</td>
<td>Count of the data class instance per group in 20% of the training data set</td>
<td>15</td>
</tr>
<tr>
<td>3.4</td>
<td>Existing global clusters $g_j$ &amp; their centroids at time window $t_w=3$</td>
<td>30</td>
</tr>
<tr>
<td>3.5</td>
<td>Candidate clusters $c_i$ &amp; their centroids at time window $t_w=4$</td>
<td>30</td>
</tr>
<tr>
<td>3.6</td>
<td>Pair-wise Euclidean distances between global &amp; candidate cluster centroids</td>
<td>31</td>
</tr>
<tr>
<td>3.7</td>
<td>Least pair-wise Euclidean distances between $g_j$ &amp; $c_i$ centroids at $t_w=4$</td>
<td>33</td>
</tr>
<tr>
<td>3.8</td>
<td>Signature at time window $t_w=4$</td>
<td>33</td>
</tr>
<tr>
<td>3.9</td>
<td>Candidate clusters $c_i$ &amp; their centroids at time window $t_w=4$</td>
<td>35</td>
</tr>
<tr>
<td>3.10</td>
<td>Least pair-wise Euclidean distances between $g_j$ &amp; data rows at $t_w=5$</td>
<td>36</td>
</tr>
<tr>
<td>3.11</td>
<td>Signature at time window $t_w=5$</td>
<td>36</td>
</tr>
<tr>
<td>4.1</td>
<td>Confusion Matrix Per Class Group - initial</td>
<td>46</td>
</tr>
<tr>
<td>4.2</td>
<td>Confusion Matrix Per Class - initial</td>
<td>47</td>
</tr>
<tr>
<td>4.3</td>
<td>Confusion Matrix Per Class Group - later</td>
<td>47</td>
</tr>
<tr>
<td>4.4</td>
<td>Confusion Matrix Per Class - later</td>
<td>48</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.5</td>
<td>Relevant Work Comparison - Execution Time</td>
<td>49</td>
</tr>
<tr>
<td>4.6</td>
<td>Relevant Work Comparison - Normal</td>
<td>50</td>
</tr>
<tr>
<td>4.7</td>
<td>Relevant Work Comparison - Dos</td>
<td>50</td>
</tr>
<tr>
<td>4.8</td>
<td>Relevant Work Comparison - Probe</td>
<td>50</td>
</tr>
</tbody>
</table>
List of Algorithms

1. Creation of Candidate Clusters ........................................... 28
2. Update the Global Cluster Set G ....................................... 32
3. Least Distance Cluster (LDC) Table for signature generation .... 34
4. Frequency table obtained from LDGC - novel time window signatures 36
5. Classification using Tanimoto Similarities Table. ................... 39
Chapter 1

Introduction

1.1 Background

Streaming data are often associated with certain challenges when it comes to applying data mining techniques. For example, since algorithm at every time interval only has some subset of the entire training samples, it may be difficult to begin classification without remaining part of the data stream. Furthermore, the data mining algorithm must consider the mutable underlying distributions of such data since they may be changing from one time interval to another. However, these challenges have not hindered time series data from remaining a commonplace in many applications and they account for a large variety of generated data in the industry. For example, data networks process billions of data logs every day that are crucial to most real-world businesses. Additionally, in order to monitor and secure these networks, these logs could be processed in real-time as part of a practical online intrusion detection system. While several data mining techniques that have been applied in the past to the problem of intrusion detection, most have operated in an offline mode and thereby unresponsive to the dynamic nature of a data network. In
some of the previous work on classification for intrusion detection and for streaming mode, fuzzy clustering have been used to generate training subsets from the data stream\cite{13} to be trained with an artificial neural network in an offline mode and another work generated an ensemble of decision trees in streaming mode\cite{12}. Our approach improved on the latter work \cite{12} with unsupervised learning (hierarchical clustering) to identify changes in the distribution of the streaming data over each data chunk i.e. sample of the data at a specific time interval. We compared dendrogram generated from each time window of the data set and created a global set of signatures that can be used for classification which enable us to train the classifier without entirety of the training dataset. Furthermore, we validate our algorithms using network monitoring data to identify nominal and attack types of traffic with the renown KDD Cup 99 dataset\cite{21}.

1.2 Motivation

Our goal is to provide a practical approach for classification in streaming dataset that can be applied to solve real-world problems like network intrusion detection. Previous work in table 2.1 have either used the entirety or some summarized form of the entire KDD Cup 99 dataset\cite{21} at once in their approach to solve the problem of network intrusion detection but this work differs in that we used only the sample of data points in the current time window and as the data grows, we evolve new models while keeping the historic models. Since we are training in streaming mode, we created contiguous windows of data with equal predetermined interval length from the KDD Cup 99 dataset\cite{21}. This implies we can start training the classifier immediately with a small chunk of data extracted from the entire dataset and improve the model as more data arrives compared to the other methods that required
training on entire dataset before classification can begin. In addition, since we are only considering data at each interval, we can analyze the data in the context of that time window which enables us to account for the mutable underlying distribution of data from one-time window to another. For example, the sample of data at time window $t_{w=m}$ may consist of only "normal" traffic data while at time window $t_{w=m+n}$ it may be denial of service (DOS) attacks. Our approach creates models based on data in each time window if they do not already exist, hence it can be put to use in operation (e.g. network intrusion detection) immediately without being trained on the entire dataset. Furthermore, it can be applied in online training mode\textsuperscript{[22]} where training on the entire datasets may be impractical due to large volumes of the data or parallelization with frameworks such as Hadoop may not be suitable. We updated the global sets with new models as more data arrived, hence the performance of our approach can be compared with other method that used the entire dataset at once for training.

1.3 Achievements

In this work, we proposed a practical online classification approach using hierarchical clustering (unsupervised learning) where we created models with available data and evolved new models as more data arrives without losing the historical models. In addition, a part of our algorithm (algorithm 1 & 2) can be used to compare two or more dendrograms created from hierarchical clustering. Classification of novel test samples into their predicted class was achieved using their tanimoto similarities\textsuperscript{[23]} from existing models or clusters. We performed classification using least amount of training data compared to previous approaches in Section 4.5 since we require only a subset of the entire dataset before we begin our classification tasks. In addition,
when we encountered a test sample at any time window where multiple class labels exist e.g. a test sample with observations having "normal" and "attack" class labels, our approach successfully distinguished data points within the sample and associate them with their corresponding global clusters. The solution can be suitable as a near or real-time network intrusion detection systems that classify incoming packets at the edge of the network into either "normal" or flag the traffic as "anomalous" for further investigation by the network security team. New global clusters and signatures will be generated at later time windows as new data arrive making the system "self-improving". Consider figure 1.1 a data space X with changing data patterns evolving over time $t_w$ where $t_w$ is the time window.

![ML Clustering across the Individual data space](image)

**Figure 1.1:** Streaming mode data analytics at different time windows

### 1.4 Overview

The remaining chapters were organized as follows - Chapter 2 describes previous work in tracking patterns in streaming datasets and other machine learning approach to network intrusion detection. In Chapter 3, we described our approach,
methodology and experimental setup to validate the performance of our work. Furthermore, in Chapter 4 we provide a comparison of our results with others previous work described earlier. Lastly, Chapter 5 provides the conclusion and future works based on or that could be adapted from our design.
Chapter 2

Related Work

2.1 Machine Learning for Intrusion Detection

Several machine learning algorithms have been applied for classification in streaming data mode. In addition, there have been numerous works on intrusion detection using network monitoring data (logs). Intrusion detection itself have been broadly categorized\textsuperscript{[18]} into host-based attacks [HAs] and network-based attacks [NAs]. In a host-based attack, an intruder tries to gain unauthorized access to services and resources on a computer host\textsuperscript{[18]}. While in network-based attacks, the intruder attempts to take control or deny access to network services and resources. An attack can also be hybrid, which is a combination of both HAs and NAs attacks. We will focus on the network based (HAs) attacks in our work.

Ekman and Holst\textsuperscript{[7]} incremental stream clustering tried to classify incoming data stream into different patterns by creating statistical models from the data stream but their approach requires a human user/agent to label these groups of models as "correct" clusters while pruning away anomalous ones. Our approach eliminates the need for a human agent in classification by using tanimoto similarities\textsuperscript{[23]} threshold
to existing models to classify new input models. In addition, we relied on an unsupervised method of clustering (hierarchical clustering) thereby allowing each data chunk to create its own clusters before comparing them to existing ones. Furthermore, Wang et al.\cite{13} approached the problem by using *fuzzy clustering and artificial neural networks (ANNs)*. A fuzzy generator was used to generate subsets from the entire training data to train the artificial neural networks (ANNs). The intermediate output from the ANN models was combined with fuzzy clustering (fuzzy aggregator) to produce the final result. Fuzzy clustering has an advantage of being able to create $n$ number of overlapping clusters with all the data points belonging to all clusters with some degree of "fuzziness" (overlaps) rather than $n$ distinct clusters (disjoint sets) in traditional clustering (hard clustering). Soft clustering may be more suitable for training the artificial neural networks in the intermediate phase of their algorithm. However, the main drawback of their work has to do with the prior task of determining the $n$ number of fuzzy clusters to be generated. In addition, another non-trivial task is to determine the fuzziness index $m$ which will be used for the fuzzy clustering. Our approach eliminates the need for these tasks by using an unsupervised method (hierarchical clustering). Also, we improved on Wang’s approach requiring only a subset and not the entire training datasets before we start our classification since we update our classification models as more data arrives. In summary, other notable work include Zhong et al.\cite{9} which used clustering-based intrusion detection on large wireless traffic dataset but their approach also require the entirety of the training dataset at least once before any classification can be done. In addition, some other works have also considered time-component of the network intrusion detection problem. Gao, J. et al.\cite{11} proposed using clustering huge volumes of continuously arriving data by using chunks of streaming data to increment a list of global clusters but their approach only considered dense points
(high number of partitioned data points) greater than a predetermined threshold for clusters and not all data points in each streams therefore some of the data points may never contribute to the classification models. In addition, Jaehak et al created signatures i.e. patterns from nominal traffic flow and pre-learned attacks using Support Vector Machines (SVM) which must be updated for new types of intrusion attack\cite{14}, however it takes a much higher amount of time to train an SVM and entire training dataset have to be considered or some summary of it making it unsuitable for a near or real-time intrusion detection system. Furthermore, Shi-Jinn et al\cite{25} proposed a combination of hierarchical clustering and support vector machine (SVM) for intrusion detection which summarized the training data set to reduce the amount of time to train the SVM. Hierarchical clustering using the BIRCH algorithm was used to pre-processed the entire KDD Cup 99 Dataset into smaller sized samples before support vector machine (SVM) was applied for classification. However, the BIRCH algorithm is quite sensitive to the order of arrivals of the data rows (i.e. connection records or chunks) which could lead to issues with the results for the second stage of the algorithm (SVM training). Our work differs from the former in that only a subset of training data set was considered in our approach before classification and not the summary of the entire training data set. Also, we used hierarchical clustering (agglomerative) as our unsupervised learning method. Lastly, Zhong et al\cite{9} and Moises et al have both used K-means clustering for finding anomalies in network monitoring data while Evangelos et al\cite{8} applied Co-clustering on the network intrusion monitoring data. K-means parameters could be difficult to determine and all these methods used the entire training data set. Our approach applied hierarchical clustering in near real-time mode and performs classification in shorter time compared with an SVM classifier method which required the entire training dataset.
2.2 Tracking Patterns in Streaming Background

Certain research work has focused on data mining techniques in streaming data. Street et al\textsuperscript{[15]} and Wang et al\textsuperscript{[16]} have applied data mining techniques (decision trees) on streaming data by building an ensemble of classifiers and carrying-out classifications by majority voting but historical patterns were not stored. Bhatnagar and Ramamurthy\textsuperscript{[12]} used an ensemble of decision trees in tracking recurrent concept drifts (changing patterns) in data streams. In tracking concepts drifts in streaming data\textsuperscript{[12]}, Bhatnagar and Ramamurthy created a global sets of classifiers (decision trees) from individual chunks of data and new classifiers are added provided none of the existing ones in the global sets meets the Permitted Error criterion. In classification, relevant classifiers are selected to form a committee using weighted Maximum Mean Square Error (Max. MSE) and a predetermined Acceptance factor.

Our approach will improve on the algorithm by using hierarchical clustering to overcome some of the limitations of the decision trees classification\textsuperscript{[17]}. A more recent approach to online classification is the Adaptive Random Forest (ARF) algorithm\textsuperscript{[24]} which generates an ensemble of Hoeffding trees from data streams but may require an optimally tuned parameters. Hoeffding trees are decision trees that can be used to train on data stream because small sized sample could be sufficient to choose an optimal splitting attribute. Gomes et al.\textsuperscript{[24]} created random forests of these trees to be used for online classification of data streams. Our approach makes use of a different learning method hierarchical clustering which does not require complex parameters has to be tuned correctly to achieve good classifications.
2.3 Summary of Existing Algorithms

While there has been a significant number of research work on network intrusion detection in offline mode, our approach differs from these works in a sense that we have designed the system to be able to start classification with available training data sets while improving the classification models as more data arrives (near real-time mode). Our approach also differs from or improved upon existing online streaming classifications methods that have been described in 2.2 above without being computational burdensome. In addition, this work also differs from other online classification works described earlier in using an unsupervised learning method which requires the least number of tunable parameters.

The table 2.1 is a summarized comparison of relevant work.

<table>
<thead>
<tr>
<th>Related Works</th>
<th>Similarities</th>
<th>Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracking Recurrent Concept Drift in Streaming Data using Ensemble Classifiers [^{12}]</td>
<td>Online Classification</td>
<td>Unsupervised Learning</td>
</tr>
<tr>
<td>Adaptive Random Forests for evolving Data Stream Classification [^{24}]</td>
<td>Online Classification</td>
<td>Unsupervised Learning</td>
</tr>
<tr>
<td>Intrusion Detection using Artificial Neural Networks and Fuzzy Clustering [^{8}]</td>
<td>Intrusion Detection</td>
<td>Unsupervised Learning &amp; Online Classification</td>
</tr>
<tr>
<td>A Novel Intrusion Detection System Based on Hierarchical Clustering &amp; Support Vector Machines [^{25}]</td>
<td>Intrusion Detection, Hybrid Learning (Unsupervised &amp; Supervised)</td>
<td>Online Classification</td>
</tr>
</tbody>
</table>

Table 2.1: Relevant Work Comparison
Chapter 3

Approach

3.1 Overview

We have created a system that can analyze sequential chunks of data on arrival and can perform classification without using an entire training data. If we consider $s$ as a sequence of data points $x$ that arrived on data space $\varphi$ within time window $t_w$, then we can represent $s$ as a collection of data points $x$ at $t_w$ such that $s$ is a data sample with $n$ data points at time window $t_w$.

If we consider each window of data $s$ as an independent sample and run hierarchical clustering on each window to generate its dendrogram. Using a recursive approach, we collect a list clusters, $c$ in the dendrogram with data points greater than the specified minimum threshold value (minimum point)\textsuperscript{[section 3.5]}, hence from each data sample $s$ we initially generate a list of overlapping clusters $L = \{c_1, c_2, c_3, \ldots c_n\}$ at $t_w$. However, due to the nature of hierarchical clustering, there maybe repetition of children nodes when parent nodes are selected to create candidate clusters. These repetitions will be handled in a later steps. We proceed to find the least distance cluster(s) $LDC$ from $L$ by finding the cluster(s) $c_i$ in $L$ that have least Euclidean
distances from the all the existing global clusters in the global set $G$ such that

$$\|d_i\| = d(g_j, c_i).$$

for all $g_j$ in the Global Set $G$

If the $d_i$ is greater than a predetermined threshold $\theta$, we update the global set $G$ with $c_i$ - Additional details will be covered later in Section 3.5.

Also, we create *signatures* from each time window $s$ using their candidate clusters. If we consider $D_{tw}$ as a set of distances of all the clusters $c_i$ in $L$ from the each global cluster $g_i$ in $G$ at time window $t_w$ i.e. $D_{tw} = d_1, d_2, d_3, ..., d_n$, then we can create a subset $LDC_{tw}$ from $D_{tw}$ containing Least distance candidate cluster(s) $c_i$ and their corresponding global cluster $g_i$ such that $LDC_{tw}^{LeastDistance} \subseteq D_{tw}$. Therefore a *signature* at time window $t_w$ is a set of Least Distance Candidate Cluster(s) $c_i$ and their global cluster pair $g_i$ i.e.

$$\text{signatures } (tbl_{ij}) = \text{Least}(D(g_j, c_i))_{tw}$$

In order to eliminate the repetition mentioned earlier from our signature table $tbl$, whenever any candidate cluster is selected from the list $L$, we remove the parent and children nodes of the parent node used to create the candidate cluster if they still exists on the dendrogram $H$, thereby ensuring there are no overlapping clusters.

We create *signature* for each window that will be used to perform classification as soon as possible without the need of entire training data set. Classification is performed once a sufficient set of signatures $tbl$ are generated typically from between 10% to 20% of the entire dataset can be used but the decision can be also heuristically determined as appropriate. On arrival of new data sample $s$, we find its signatures $tbl_{ij}$ and using the new signature, we find its Tanimoto similarities $T^{[23]}$ with existing ones in the global sets. Tanimoto similarity is akin to the Jaccard Distance metric,
however tanimoto is a similarity measure that can be computed as

\[
T(tbl_G, tbl_{ij}) = \frac{tbl_G \cdot tbl_{ij}}{|tbl_G|^2 + |tbl_{ij}|^2 - tbl_G \cdot tbl_{ij}}
\]

where \( tbl_G, tbl_{ij} \) are existing global cluster & new time window signatures.

Current data \( s \) in a time window are classified by their majority class \( C \) (i.e. majority voting) of most similar global cluster \( g_i \) in \( G \) using the Tanimoto similarities in \( T \).

As part of our future works, classification can also be done per data point in a time window using the class of the most similar global cluster \( g_i \) rather than the majority class as described earlier. In cases, where none of the global to candidate clusters Euclidean distances for a new time window is above a predetermined threshold value \( \theta \), the approach will be to add the novel signature and its class to that of the global set of signatures. The threshold distance value \( \theta \) could be determined based on experience or heuristics or both.

Traditionally offline learning method requires separation of an entire data sets into both training and testing samples for training and validation phases respectively. Using our approach, we initialize a set of global clusters \( G \) from the candidate clusters List \( L \) generated from each data sample \( s \) that arrived within time window \( t_{w1} \) and update the models as more data arrives in subsequent time windows \( t_{w2}, ..., t_{wn} \) hence we start our classification tasks sooner making it more suitable for a near real-time solution. However, we recommend using the initial 10% to 20% of the training data set to validate the threshold distance parameter \( \theta \).

One of the implementation of our hierarchical clustering system is for intrusion detection. As described in Section 2.1, we designed our system to be used to prevent network-based attacks [NAs]. We will validate its performance using the KDD Cup 99 Dataset. In order to prevent network-based intrusion attacks, the system must be able to distinguish data traffic (TCP connection records) that are nominal from anomalous ones. Additional detail of the implementation will be
3.2 Nature of Data Set

Time series data often have two main dimensions: time/period and value/attributes. Each row corresponds to an observation at period $t_i$ while the columns are the attributes of that observation. Since the underlying mode of the data distribution may be changing over time, it is essential to consider its transient mode in our design. A naive approach that clusters the entire training data all at once without consideration for the changing mode of operation in the data could lead to misleading results. However, a tedious approach of clustering each data point (row) could create meaningless clusters and become computational burdensome, hence the need for a heuristic balance in the length/period of each data window to create meaningful clusters. A combination of domain knowledge and experiment was used to select an appropriate period for the data chunks. In this work, we considered 250 sequential data rows to be a data window sufficient to generate signatures that can be used for classification. Once we had fair amount of signatures, we scaled this down to 50 sequential data rows as novel data at different time windows which can then be classified using the existing signatures. Our data set has each row vector of 41 attributes and each row represent a completed LAN TCP connections records at time $t$ which we later grouped into $n$ number of sequential connections of blocks of individual time windows. Column 2, 3 and 4 of the KDD Cup 99 data sets were categorical attributes while the rest are numerical (continuous) attributes. The entire data has over five million individual LAN TCP connection records or rows and there were five main class group label - Normal, Probe (unauthorized information gathering or snooping), denial of service (Dos), unauthorized access -user to root
(U2R) and unauthorized access from a remote machine (R2L). The data class group label can be broadly divided into normal and attack. The system encountered mainly normal and two groups of attack (dos and probe) in the initial 20% of the data set while generating the initial set of signatures (training).

Table 3.1 is a count of observations from various data class that exists in the entire training data

<table>
<thead>
<tr>
<th>Class</th>
<th>normal.</th>
<th>portsweep</th>
<th>satan.</th>
<th>ipsweep.</th>
<th>nmap.</th>
<th>smurf.</th>
<th>neptune.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance Count</td>
<td>972,781</td>
<td>10,413</td>
<td>15,892</td>
<td>12,481</td>
<td>2,316</td>
<td>2,807,886</td>
<td>1,072,017</td>
</tr>
</tbody>
</table>

Table 3.1: Count of the data class instance in 100% training data set

Table 3.2 is a count of observations from various data class that exists in the 20% of training data

<table>
<thead>
<tr>
<th>Class</th>
<th>normal.</th>
<th>portsweep</th>
<th>satan.</th>
<th>ipsweep.</th>
<th>nmap.</th>
<th>smurf.</th>
<th>neptune.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance Count</td>
<td>562,387</td>
<td>2,782</td>
<td>5,389</td>
<td>7,579</td>
<td>2,316</td>
<td>191,863</td>
<td>204,815</td>
</tr>
</tbody>
</table>

Table 3.2: Count of the data class instance in 20% of the training data set

Table 3.3 is a count of observations from various data class group that exists in the 20% of training data set

<table>
<thead>
<tr>
<th>Class Group</th>
<th>normal</th>
<th>probe</th>
<th>dos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance Count</td>
<td>562,387</td>
<td>15,750</td>
<td>396,678</td>
</tr>
</tbody>
</table>

Table 3.3: Count of the data class instance per group in 20% of the training data set
Figure 3.1 is the percentage distribution of data class instances by their group encountered by the system before classification began.

![Distribution of Class Instances in data stream used for Initial Signature Generation (20% of training data set)](image)

Figure 3.1: Percentage of data instances per group
The chart shows how percentage distribution of class instance per class group for the data used to generate the initial set of signatures before classification began.

### 3.2.1 Data Preparation

Certain pre-processing steps were undertaken on the training data set to convert the categorical attributes to numerical attributes to facilitate easier training. The step was necessary, since having a uniform columns of attribute has the advantage of simplifying the computation of across all the dimension of the data. First, we created a set of integer to correspond to all unique categorical attribute values in a column then we assign each categorical attribute an integer within the set. For example, if column $b$ in the data set has 6 unique categorical attributes value, then we will have set $b = \{1,2,3,4,5,6\}$, each row in the data will have a value from set $b$ for its column that correspond to its categorical attribute value. As a result, we were able to convert all the attribute to numerical (continuous) values which we
then normalized column wise across the rows or records to ensure every component contributes evenly towards the result.

3.3 Experiment Design

Once the data have been pre-processed, we can generate an hierarchy of clusters (dendrogram) from each sample at every time window. While the number of data points per window is subjective (we used a total of 300 data points per window - 250 for training and 50 for testing initially), the goal is to ensure there are sufficient data points to create "good" clusters with hierarchical clustering. A sufficient set of data points allowed per time window cannot be over-emphasized as too few data points could be "too grainy" while overly large number of data points per time window will be counter-productive for an online streaming analytics. A promixity matrix \(^{(3.3.1)}\) was obtained as an intermediate result from the hierarchical clustering \(^{(3.4)}\) which was used to generate an hierarchy of clusters called a dendrogram. A dendrogram was used to create a set of candidate clusters using parent nodes from the tree with each clusters having a total number of data points greater than the minimum point. Signatures were created from each window was compared with existing global set of signatures or model using Tanimoto similarities \(^{[23]}\). The result was used to classify the novel data window into the majority class of the global set of signatures or clusters i.e. the class of most frequent global clusters in the set \(G\). In cases, where none of the existing global clusters in \(G\) are "similar enough" (none greater than the predetermined threshold \(\theta\)) we can add the new signature as part of our global set. The MATLAB 2016 \(^{\text{TM}}\) computer program was executed on a 16GB Memory 1600MHz AMD Macintosh \(^{\text{TM}}\) Computer and it took approximately 600 seconds (10 minutes) to analyze data samples in 3265 time windows - See Section 4 for
performance comparison with other algorithms.

3.3.1 Steps after Initialization

Our approach can be described formally in the sequence of steps listed below -

Step 1: Create time window $s$ of data from the training data with every window having $p$ sequential data points

Step 2: From each window $s$, select $q$ sequential data points without replacement for testing signatures and use the remaining $p-q$ for training

Step 3: Run hierarchical clustering on each data sample $(p-q)$ to create a dendrogram

Step 4: Select the appropriate higher level nodes from the dendrogram to create clusters where the total number of data points in each cluster is greater than the minimum point

Step 5: *Initialize* a set of global clusters only at iteration $t_w = 1$ i.e. from the data in the first time window

Step 6: New clusters and their signatures are added to global set of clusters or signatures if & only if their Euclidean distances $d_i$ are greater than threshold distance $\theta$

Step 7: Obtain tanimoto similarities between existing signatures or global clusters and new signature from novel data in a new time window

Step 8: Using Tanimoto Similarities, classify novel data in a new time window using its most similar existing signatures to the corresponding class $C_j$ of the global cluster(s) $g_j$
Step 9: If all global to candidate cluster distances are greater than the predetermined threshold $\theta$ then update with new signatures if required (*Optional*).

The figure 3.2 describes each step of the algorithm for updating the global clusters list $L$ after initialization. In certain cases, where none of the candidate clusters created in data window is similar enough to the existing global cluster $g_i$ in $G$, all candidate clusters were added to the global cluster list i.e. all $c_i$ in $C$ are all added to $G$ ($k_{late}$ initialization).

*Initialization step described only occurs only during the first time window.*

![Figure 3.2: The algorithm explained with flowchart. - Global Cluster Update](image-url)
3.3.2 Initialization

Hierarchical clustering can be ran to create a set of $k$ predetermined number of clusters specified by the user. Our global clusters list $G$ was initialized by using all the $k$ clusters created from the first window $s_{tw}=1$ dendrogram at the beginning of the algorithm. The global sets $G$ will then be updated as described in Section 3.3.1 throughout the course of the operation of the system except in cases where all (or over 70%) of the candidate clusters in the current data window was not similar to the any of the existing global clusters in $G$. This scenario indicates that none of the existing models in $G$ describes the underlying distribution in the data of the current time window. In such cases, we regenerated the dendrogram with a predetermined number of $k$ clusters which will all be added to the global clusters list $G$. The initialization parameters $k$ may differ at the beginning of the algorithm $s_{i=1}$ ($k_{begin}$) and later in the course of the algorithm $k_{later}$.

3.3.3 Pruning Negligible Occurrences

A multitude of methods has been suggested to handle a negligible number of a data class instance in training data. These limited number of instances of some class labels are often insufficient to calculate performance measure such as a confusion matrix $^{[3]}$. The scenario is even more palpable for an online data streaming classification. Some of the data classes could have a limited number of observations in the time window being used to generate the signatures which may lead to insignificant results appearing on the confusion matrices. Our approach was to "prune" these negligible results from the table so as not to skew the performance results and we then recompute the confusion matrix based on the remaining data class results. Also, to further ensure the integrity of our result, we calculated metrics such as
3.3.4 Linkages / Proximity Matrix

The similarity measure or proximity distance method is an intermediate result required to generate a dendrogram. Let consider a data sample $X$, where $X = x_1, x_2, x_3, ..., x_N$, then $s(x_1, x_2)$ or $d(x_1, x_2)$, $s(x_1, x_3)$ or $d(x_1, x_3)$ and so on are similarities or distances between pair-wise observations. We create symmetric $N$ by $N$ matrix, where $N$ is the number of observations and diagonal are either maximum similarities or zero (minimum) distances. There are several distance metrics that could be used to create proximity matrices with each having an implication on the kind of clusters created\[20\]. For our work, we created a proximity matrix where the proximity of two clusters $s(c_1, c_2)$ is the average of pairwise distances between data points in the two clusters i.e. the distance between the centroids of the two clusters. The proximity matrix is hence less susceptible to noise and outliers but could be biased towards globular clusters\[20\].

3.4 Hierarchical Clustering with each chunk

Hierarchical clustering is an unsupervised method of learning where each observation is separated into individual nodes initially then we join these nodes agglomeratively ("bottom-up") using the distance between them. Qualitative measure to be considered is the type of similarity (distance) measure between each pair of observation within the data set in $n$ steps. While an entire discussion of hierarchical clustering is not subject of this thesis, it’s basic understanding should suffice for easier comprehension of this work. Given a data sample $X$, hierarchical clustering initially partitions the data into individual observations: $x^1$ through $x^N$ in step $t=0$ where
Figure 3.3 describes hierarchical clustering on a data at time window $t_w$.

Data Sample = $X$

$X = \{x_1, x_2, x_3, x_4 \ldots x_n\}$ at step $t=0$

Clusters = $\{x_1, \{x_2, x_3\}, x_4, x_5\}$ at step $t=1$

Clusters = $\{\{x_1\}, \{x_2, x_3\}, \{x_4, x_5\}\}$ at step $t=2$

Clusters = $\{\{x_1, x_2, x_3\}, \{x_4, x_5\}\}$ at step $t=3$

Clusters = $\{\{x_1, x_2, x_3, x_4, x_5\}\}$ at step $t=4$

Figure 3.3: Hierarchical Clustering Dendrogram

The dendrogram has an hierarchy of clusters. Each parent node has $n$ number of child nodes. Leaf nodes at the bottom of the dendrogram represent individual data points.

$N$ is the no. of observations. Subsequent steps, $t=i+1$, each pair-wise observations are further grouped based on their distance or similarity measure and repeated until $N-1$ sets are formed. Using data sample $X$ as described in figure 3.3, we created an hierarchy of clusters in five steps, starting from individual data points in the data sample at $step = 0$ to the top of the dendrogram binary tree at $step = 4$ by combining previous nodes based on the average pair-wise Euclidean distances between the group of data points represented by those nodes.

### 3.4.1 Threshold Parameters minimum points and $\theta$

A threshold we called *minimum point* was used to reduce the number of computations required after a dendrogram is generated from a data sample by "cutting" the dendrogram at a predetermined height i.e. the bottom child nodes of the dendrogram so as to focus on the "meaningful" parent clusters. The alternative will be
to use every single data point in the dendrogram which could be computationally burdensome or superfluous.

The global to candidate clusters Euclidean distance threshold $\theta$ may require some level of experience to determine. $\theta$ is used by the algorithm to decide if new cluster(s) should be added to the global set of clusters $G$ i.e. the available or existing are not sufficient to model data in a new time window. A "large value" of similarity threshold $\theta$ may cause an increase in misclassification errors because newer signatures are not getting updated on the global set, and conversely, if the value of $\theta$ is too small, the algorithm could be overfitting. We recommend running the algorithm initially with arbitrary values of $\theta$ then adjusting it based on the classification errors obtained till it is suitable. We created hierarchical clusters from data samples in each time window using average distance similarity measure.

Figure 3.4 is a dendrogram generated from data at a time window $t_{w1}$

![Dendrogram](image)

*Not drawn to scale
Figure 3.5 is a different dendrogram generated from data at time window $t_{w3}$

*Not drawn to scale

3.5 Candidate Clusters Generation from Dendrogram

The method explained earlier in 3.4 creates various groupings of data points which are nested to form a hierarchical binary tree called a Dendrogram. The level of the dendrogram describes the clusters formed at $n_{th}$ step of the agglomerative clustering while the height of the dendrogram is the distance or similarity of one node from another (or data points in the case of leaf nodes). Each dendrogram was cut wherever the number of points exceeds the *minimum point* to minimize computations. The minimum point value was initially arbitrary chosen but heuristically adjusted to obtain "good" clusters over the course of the experiment while reducing computation. Good judgment must be exercised to determine the appropriate *minimum point* to avoid using the entire dendrogram. Faster computation is achieved if the dendrogram is cut at the predetermined level where each individual clusters have
sufficient data points with distinct characteristics. For the experiment, we adjusted the *minimum point* to a sufficient value based on feedbacks from multiple runs and domain knowledge.

### 3.5.1 Generation of Candidate Clusters

The description of how dendrograms are generated was provided in 3.4. We can describe how we create candidate clusters from these dendrograms, once generated. Consider a dendrogram $H_x$ generated from an independent data sample at time window $t_{w1}$ with the linkages (3.3.4) as shown in figure 3.6(a,b). The *linkage tree* shows how a hierarchy of clusters (dendrogram) was formed starting from the initial individual data points (leaf nodes) at the bottom of the tree to the topmost parent nodes which are large clusters of data points from hierarchical clustering. A higher level node ($>size = 2$) must correspond to at least two original data points from the data sample. For example, let consider any 2nd level nodes = \{10, 12, 15, 13\} in figure 3.6, we can say that the height of the parent node (2nd level node) on the dendrogram corresponds to the Euclidean distance separation between their child nodes e.g. parent node 11 has child node $a = 2$ and $b = 10$ at a Euclidean distance (height) 0.148 from each other i.e. $\|h\| = d(a, b)$ (see figure 3.6). The definition can be extended to the higher level of nodes on the dendrogram hence we define the *level* of a node as the number of data points connected to it.

| The *Level* of a node on a dendrogram, are total the number of child nodes connected to it. |

Therefore, hierarchical clustering starts by subdividing the entire data points into individual 1st order nodes (leaf nodes) before combining the closest nodes together to form the next higher order $n$ nodes in an agglomerative process as shown in figure...
3.6 e.g, 2\textsuperscript{nd} order nodes are $c_{11} = \{2,10\}$, $c_{12} = \{5,8\}$, $c_{13} = \{6,7\}$, $c_{15} = \{1,4\}$, 3\textsuperscript{rd} order nodes are $c_{14} = \{9,5,8\}$, $c_{18} = \{3,6,7\}$ and so on. Therefore, we define a candidate cluster as a cluster that contains data points greater than or equal to the minimum point. We can generate a list of candidate clusters $L$ from a dendrogram $H_x$ using nodes with sufficient child nodes such that the total number of data points in the cluster is greater than or equal to the minimum point.

$\Rightarrow L = \{c_1, c_2, c_3, ... c_n\}$ at time window $t_w$ such that data points $\geq$ minimum point.

Repetition of child nodes occurred due to the agglomerative method of clustering which was handled later in the algorithm. One early step was to exclude the root node (top of the tree) from the candidate cluster list $L$ creation.

A candidate cluster is a cluster created whose data points greater than or equals to the minimum point

Consider the dendrogram shown in figure 3.7, using a minimum point $= 3$ i.e. if each cluster must have at least three or more data points to be considered as a candidate cluster. We can then generate a list of candidate clusters $L$ from figure 3.7 where $L = \{ c_1, c_2, c_3, c_4, c_5 \}$ where each $c_i$ in $L$ has its total number of data points greater than the minimum point.
Figure 3.7 describes how a dendrogram is split into clusters (candidate clusters)

Candidate clusters creation using dendrogram from figure 3.7

- 1st Level nodes: \{42, 50, 44, 45, 43, 48, 41, 46, 47, 49\}
- 2nd Level nodes: \{51, 54, 55\}
- 3rd Level nodes: \{52, 57\}
- 4th Level nodes: \{53\}
- 5th Level nodes: \{56\}
- 6th Level nodes: \{58\}

minimum point = 15, Candidate clusters list \( L = \{52, c_1, 53, c_2, 56, c_3, 57, c_4, 58, c_5\} \)

Algorithm 1 described generation of candidate clusters in pseudo code.
Algorithm 1: Creation of Candidate Clusters

Input: $H =$ Hierarchical Clustering Dendrogram from data sample
Output: $L =$ list of Candidate Clusters $c_i$ from $H$

Minimum Point $P$: $P =$ Minimum number of data points in cluster $c_i$

$a_i =$ child node attached to parent node $b_j$, where $j$ is position of parent node in $H$

1: procedure 1: (Starting at the bottom of the tree $H$)
2: for $i=1$ to length($H$) do ← get position $j$ of $b_j$
   where $i$ is the number of nodes in tree $H$
3: check ← Get total number of child nodes $a_{ij}$ for $b_j$
   where $a_{i....z,j}$ is the number of child nodes in tree $H$ directly below cluster $b_j$

4: top:
5: if $\sum_{n=1}^{z} a_{ij} > P$ then return Add the $b_j$ to Candidate Cluster List $L$*
6: else:
7: continue

8: procedure 2: (Obtain the centroids of each candidate cluster $b_j$ in $L$)
9: for each $c_i$ in $L$ obtain their child nodes $a_j$:
10: for $i=1$ to length($a_j$)
11: $a_i =$ Obtain the data row corresponding to $a_j$
12: $Centroid_j = \sum_{n=1}^{z} \text{datapoint}_j$

13: procedure 3: (Handling Repititions - Remove all Parent nodes of $n_j$ in $H$)
14: for each selected $n_j$ in $L$ remove their parent nodes $n_{\text{greater than } j}$:
15: for $i=1$ to length($H_j$)
16: deletednodes ← Obtain the nodes directly selected node $n_j$
17: deletednodes$_j =$ null

*The actual data points corresponding to the nodes were obtained using their node labels in the dendrogram
3.6 Global cluster update with selected candidate cluster(s)

In order for the system to be "self-improving" and remain relevant throughout the course of its operation, we need to be able to update the global cluster set $G$ with candidate cluster(s) $c_i$ from section 3.5, whenever none of existing global cluster $g_j$ in $G$ can describe the model from novel data at a new time window.

Consider table 3.4 as a set $G$ of global cluster $g_i$ with their respective centroids i.e. $g_i \in G$ which are all existing global clusters arrived before the current window. Each global cluster has its own centroid. A centroid is a row vector which whose values equals the mean of all data points within that cluster i.e. the centroid $j$ of a cluster with $m$ dimensional data with $n$ of data points.

$$\text{Centroid}_j = \frac{\sum_{n=1}^{n} \text{data row}_j}{z}$$

where data point $j$ and $n$ are data points with $m$ dimensions and number of data points respectively in the cluster.

A centroid is a row vector which whose values are the mean of the data points within the cluster.

For brevity, we assume that the dimension of our data to have five attributes (KDD Cup 99 training data set had 41 attributes), the mean values (centroids) are shown in table 3.4. For all the existing global clusters where each row vector is the average of all data points that are the member of each group $g_j$.

Assuming the dendrogram in figure 3.7 arrived at time window $t_w = 4$, we can find Euclidean distances between all its candidate clusters $c_i$ in $L$ and the existing global clusters $g_i$ in $G$. However, we first obtain each cluster centroid $c_i$ as shown in table 3.5. Each row vector in table 3.4 corresponds to the centroid of a global cluster $g_j$ while each row vector in table 3.5 corresponds to the centroid of a candidate
Table 3.4: Existing global clusters $g_j$ & their centroids at time window $t_w=3$

Table 3.5: Candidate clusters $c_i$ & their centroids at time window $t_w=4$

The Euclidean distances between each candidate cluster $c_i$ and global cluster $g_j$ can therefore be obtained as follows:

$$D : d_{ij}^2 = \text{Pairwise Distance}(g_j, c_i)$$

where all $g_j$ and $c_i$ are row vectors

The threshold distance $\theta$ will be used to determine whether any of candidate cluster should be added to the existing global cluster set. A candidate cluster(s) $c_i$ in $L$ will be added to global cluster set $G$ if and only if all its pairwise Euclidean distances to all existing global clusters centroid are greater than the threshold distance $\theta$ i.e. if $\parallel$
\[ d_i = d(g_j, c_i), \] all \( d_i \) obtained for \( c_i \) are greater than \( \theta \) for \( c_i \) to be a global cluster. Table 3.6 shows the pairwise Euclidean distances between the global cluster set \( G \) in table 3.4 and candidate clusters list \( L \) from figure 3.7 dendrogram in table 3.5.

A candidate cluster \( c_i \) is added to the global cluster set \( G \) if and only if all its pairwise Euclidean distance \( d_{ij} \) to existing global clusters is greater than the threshold distance \( \theta \). If we assume a threshold \( \theta = 15 \), using our definition, only candidate cluster \( c_4 \) is added to \( G \) i.e. none of the existing models before time window \( t_{w=4} \) fits the set of data points in cluster \( c_4 \). Algorithm 2 describe these steps in pseudo code.

<table>
<thead>
<tr>
<th>( c_1 )</th>
<th>( g_1 )</th>
<th>( g_2 )</th>
<th>( g_3 )</th>
<th>( g_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_1 )</td>
<td>7.6</td>
<td>9.9</td>
<td>0.4</td>
<td>10.8</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>7.7</td>
<td>9.5</td>
<td>5.5</td>
<td>12.7</td>
</tr>
<tr>
<td>( c_3 )</td>
<td>4.7</td>
<td>7.4</td>
<td>5.2</td>
<td>13.1</td>
</tr>
<tr>
<td>( c_4 )</td>
<td>21.5</td>
<td>26.7</td>
<td>17.9</td>
<td>18.9</td>
</tr>
<tr>
<td>( c_5 )</td>
<td>6.5</td>
<td>10.0</td>
<td>6.0</td>
<td>11.4</td>
</tr>
</tbody>
</table>

Table 3.6: pair-wise Euclidean distances between global & candidate cluster centroids

### 3.7 Signature table generation

#### 3.7.1 Global Cluster Signature

The pairwise Euclidean distances from 3.6 can also be used to identify a set of *least distance global candidate cluster(s)* pair. Consider the \( m \) by \( n \) matrix \( D \) where \( m \) and \( n \) corresponds to the number of candidate cluster(s) \( c_i \) in \( L \) and global cluster \( g_j \) in \( G \) respectively. If we find the least distance pairs \( d(g_j, c_i) \) in matrix \( D \) across each column i.e. for each global cluster \( g_j \), we obtain a \( m \) by \( n \) matrix which is subset from the set \( D \), referred to as *least distance global candidate cluster(s)* \( LDGC \) such that
**Algorithm 2**: Update the Global Cluster Set G

Procedure: continue from Algorithm 1

Input: \(L = \) Candidate Clusters with \(c_i\) having data points \(n_i > \) minimum point

Output: \(G = \) Global Cluster Set.

1: initialize Global clusters \(G\):
2: for \(i = 1\) do initialize Global clusters \(G \leftarrow\) with all clusters \(c_i\) in \(L\)
3: Update Global Clusters \(G\):
4: for \(i = 2\) to \(\text{length(iteration)}\) do centroid \(_i\) \(\leftarrow\) center of all data points in \(c_i\)
5: for \(j = 1\) to \(\text{length}(G)\) do clusters \(\leftarrow G_j\)
6: Calculate distance \(d_i\) as \(D(\text{centroid}_g_j, \text{centroid}_c_i)\) where \(g_i, c_i\) are clusters in \(G\) and \(L\) respectively
7: if (all \(d_i\)) > Threshold \(\theta\) then
8: \(g_i = c_i\) : Update Global clusters \(G\)
9: if \(i > \text{length(iterations)}\) : Repeat Update Global clusters \(G\) for all \(c_i\) in \(L\) then
10: break

*Euclidean distances are calculated from centroid to centroid of both clusters.

**Algorithm 3** describes these steps in pseudocode.

\(LDGC_{tw}^{LeastDistance} \subset D_{tw}\) at time window \(t_w\). Using least distance cluster LDGC matrix, we can create a frequency distribution (signature) with the number of occurrence of individual candidate cluster \(c_i\) and global cluster \(g_j\) pair in the LDGC i.e. \(f(g_j,c_i)\) as the entries. Therefore, we obtain a \(n \times 1\) matrix for each individual data that arrived at time window \(t_w\) which called signature where \(n\) is number of \(g_i\). Each signature was added to existing global cluster \(G\) signature to be used for classification. **Algorithm 3** describes these steps in pseudocode.

* Additional steps required before classification includes normalization and obtaining Tanimoto similarities between signatures.

A signature is the frequency of each \(g_i\) in Least Distance Clusters i.e. \(f(g_j,c_i)\)

For example, using the least distance clusters \(LDC\) in table 3.7 and the corresponding \(m \times n\) signature 3.8 for time window \(t_w = 4\) (table 3.6). We have \(LDGC\)
= \{(g_1,c_3), (g_3,c_2),(g_3,c_4),(g_3,c_5)\} and signature = [1,0,4,0]^T. The frequency table obtained describes which of the existing global clusters closely model the new data in a new time window or their dissimilarities

\[
\begin{array}{|c|cccc|}
\hline
 & g_1 & g_2 & g_3 & g_4 \\
\hline
 c_1 & 7.6 & 9.9 & 0.4 & 10.8 \\
 c_2 & 7.7 & 9.5 & 5.5 & 12.7 \\
 c_3 & 4.7 & 7.4 & 5.2 & 13.1 \\
 c_4 & 21.5 & 26.7 & 17.9 & 18.9 \\
 c_5 & 6.5 & 10.0 & 6.0 & 11.4 \\
\hline
\end{array}
\]

Table 3.7: Least pair-wise Euclidean distances between $g_j$ & $c_i$ centroids at $t_w=4$

\[
\begin{array}{|c|c|}
\hline
 & f \\
\hline
 g_1 & 1 \\
 g_2 & 0 \\
 g_3 & 4 \\
 g_4 & 0 \\
\hline
\end{array}
\]

Table 3.8: signature at time window $t_w=4$
Algorithm 3 : Least Distance Cluster (LDC) Table for signature generation

Procedure: continue from Algorithm 2

Input: \( G = \) Global Cluster Set.

Input: \( L = (c_i) \) where \( c_i \) is a candidate cluster created from a parent node with \( n_i \) child nodes in a dendrogram \( H \)

Output: \( tbl = \) Least Cluster Distance Table where \( tbl = (LD_i, g_i, c_i) \)

Procedure:
1. Update Least distance cluster table \( tbl \):
2. for \( i = 1 \) to \( \text{length}(G) \) do collect \( g_i \) ← where \( g_i \) is a cluster in \( G \)
3. for \( j = 1 \) to \( \text{length}(L) \) do collect \( c_i \) ← where \( c_i \) is a cluster in \( C \)
4. Calculate distance \( LD_i \) as \( \text{LeastD}(\text{centroid} \ g_j, \text{centroid} \ c_i) \) where \( g_i \) , \( c_i \) are clusters in \( G \) and \( C \) respectively

\[ \|LD_i\| = d(g_j, c_i). \]
5. Obtain Least distance \( LD_i \) with \( \text{LeastD}(g_j, c_i) \)
6. if \( (d_i) = \) Least distance then
7. \( tbl_{ij} = LD_i, g_i, c_i \) : Update Least distance cluster table \( tbl \) where \( LD_i \) is the Least distance between \( g_i \) and all the \( c_i \) in \( C \)
8. if \( j > \text{length}(L) \) : Repeat Update Least distance cluster table \( tbl \) for all \( G_j \) then
9. break
3.7.2 Novel Time Window Signature

We extended the same approach to data $s_i$ that arrived at time window $t_w$ by maintaining and updating a frequency table of most similar data point (row) $td_i k$ and their global cluster $g_j$ pair where $k$ is the number of data points in $s$ at the $i$th time window. In the initial phase of the operation of the system, we collected signatures from each data sample at different time windows within 20% of the KDD 99 Dataset. New signatures were then obtained from both 20% (different from global cluster signatures) and remaining 80% of the data set to be used for classification. For example, using the global clusters $g_i$ centroids in table 3.7 and new data sample (normalized with five attributes) at time window $t_w = 5$ in table 3.9. Algorithm 4 describes these steps in pseudo code.

| Global Clusters signatures used global to candidate clusters distances while novel (test) time window signatures used individual data rows to global clusters distances.

| $td$ | $attrb_1$ | $attrb_2$ | $attrb_3$ | $attrb_4$ | $attrb_5$
|-----|-----------|-----------|-----------|-----------|-----------
| $td_51$ | 7.1       | 5.8       | 9.5       | 9.2       | 3.1       
| $td_52$ | 11.1      | 14.8      | 12.5      | 8.6       | 9.8       
| $td_53$ | 7.9       | 3.0       | 2.6       | 9.6       | 8.9       
| $td_54$ | 2.9       | 1.8       | 7.6       | 1.2       | 8.9       
| $td_55$ | 6.8       | 9.5       | 4.7       | 8.5       | 8.6       

Table 3.9: Candidate clusters $c_i$ & their centroids at time window $t_w=4$
Table 3.10: Least pair-wise Euclidean distances between \( g_j \) & data rows at \( t_w=5 \)

<table>
<thead>
<tr>
<th>( td_j )</th>
<th>( g_1 )</th>
<th>( g_2 )</th>
<th>( g_3 )</th>
<th>( g_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( td_1 )</td>
<td>10.8</td>
<td>8.7</td>
<td>6.7</td>
<td>11.7</td>
</tr>
<tr>
<td>( td_2 )</td>
<td>16.4</td>
<td>19.2</td>
<td>12.0</td>
<td>10.3</td>
</tr>
<tr>
<td>( td_3 )</td>
<td>6.1</td>
<td>8.0</td>
<td>5.1</td>
<td>13.9</td>
</tr>
<tr>
<td>( td_4 )</td>
<td>7.9</td>
<td>11.1</td>
<td>9.8</td>
<td>12.8</td>
</tr>
<tr>
<td>( td_5 )</td>
<td>6.8</td>
<td>10.5</td>
<td>6.0</td>
<td>11.5</td>
</tr>
</tbody>
</table>

Table 3.11: signature at time window \( t_w=5 \)

<table>
<thead>
<tr>
<th>( g )</th>
<th>( f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g_1 )</td>
<td>1</td>
</tr>
<tr>
<td>( g_2 )</td>
<td>0</td>
</tr>
<tr>
<td>( g_3 )</td>
<td>3</td>
</tr>
<tr>
<td>( g_4 )</td>
<td>1</td>
</tr>
</tbody>
</table>

**Algorithm 4**: Frequency table obtained from LDGC - novel time window signatures

Procedure: continue from Algorithm 3

Input: \( tbl = \) Table of Least Distance Clusters \( tbl(g_i) \)

Input: \( TestData = \) Test Sample Data at a new time window i.e. \( s(td_{jk}) \) with \( k \) number of rows

Output: \( frq_{table} = \) Frequency Distribution of Least Distance Global Clusters \( (g_i) \) from Test Sample Data rows \( (td_{ikrows}) \)

Procedure:

1. Initialize \( frq_{table} \):
2. Obtain \( frq_{entry} \) at every \( i \)th iterations:
3. \( \text{for } i = 1 \) to \( \text{length}(tbl(g_j)) \) \( \text{do collect } g_{\text{centroids}} \leftarrow \text{of } g_j \) in \( tbl_i \)
4. \( \text{for } j = 1 \) to \( n(s) \) \( \text{do collect each row } \leftarrow s(td_{ikrow}) \)
5. Obtain all distances \( d_k \leftarrow D(g_j, td_{ikrows}) \) where \( d_k \) are \( k \) Least distances of each row in \( td_{jrow} \) from \( g_i \)
6. \( \text{for } k = 1 \) to \( \text{length}(d_k) \) \( \text{do} \)
7. \( frq_{entry} = \) Assign frequency \( f \) value based on number of times \( g_i \) has least distance value to Test Data \( td_{jrows} \)
8. Update \( frq_{table} \)
9. if \( i > 1 \): Repeat \( frq_{entry} \) at every \( i \)th iterations then
10. break
3.8 Classification using Tanimoto Similarity matrix

We classify a novel data at a new time window by finding its signature Tanimoto similarity from the existing signature table. Darko[^23] described Tanimoto similarity as a similarity measure whose coefficients are somewhat normalized to account for the common bits. As a result, the more similar two clusters are, the higher will be their Tanimoto coefficient value.

\[
T(A, B) = \frac{A \cdot B}{|A|^2 + |B|^2 - A \cdot B}
\]

\[
T(tbl_G, tbl_{ij}) = \frac{tbl_G \cdot tbl_{ij}}{|tbl_G|^2 + |tbl_{ij}|^2 - tbl_G \cdot tbl_{ij}}
\]

where \(tbl_G\) and \(tbl_{ij}\) are existing and new signatures from global cluster \(G\) and novel data \(s_i\) at a new time window respectively.

The most similar global clusters \(g_{closest}\) will be those with the largest value of Tanimoto coefficient \(T\) to the signature (candidate clusters) in the data.

Tanimoto similarity measure was used for finding most similar global clusters on arrival of a new data at a time window during the testing phase*. Tanimoto was selected as the similarity measure because of robustness of the now i.e. where the distance between global cluster to data row is null. Also, Tanimoto similarity technique was used to find best-matched groups for a data sample in a new time window in the classification tasks.

The Algorithm 4 described how we created a frequency table of the number of times a global cluster \(g_i\) in the global set \(G\) that was similar to data rows from new data \(s\) to create its signature. Tanimoto similarity is a similarity measure that provided additional robustness of the system without undue complexity by being able to handle nullipotent scenario e.g. for example where all the existing models or global clusters are dissimilar or unable to model a new data in a later time window. For
example, we see from signature in table 3.11 that the most similar global cluster will be $g_3$ and the new data will belongs to the same class with $g_3$ class. The flowchart described in Figure 3.8 provides visual representation of the flow of execution of the classification tasks, and Algorithm 5 describes these steps in pseudo code.

![Flowchart](image)

Figure 3.8: The algorithm explained with flowchart. - Classification tasks

### 3.8.1 Validation

Confusion matrices were used to validate and compare the performance of this work. We created two sets of confusion matrices with test samples from 20\% and 80\% of the training data. The predicted class is the class label of the most frequent and most similar (based on Tanimoto similarities and majority voting) global clusters to the data points in a test sample while the actual class is the majority class label of each test sample. Some negligible instances of data class label groups (R2L and U2R) had to be removed before computing these confusion matrices. Such remedial steps have been used by Wang et al\cite{13}. In addition to removing these insignificant instances, we calculated precision and recall, both of which are less susceptible to the limited number of certain instances in a training data set.
Algorithm 5: Classification using Tanimoto Similarities Table.

Procedure: continue from Algorithm 3 & 4

Input: \((frq_{table})_{norm}\) = normalized global cluster signatures; \(frq_j\) (column vector)

Input: \((frq_{table})_{norm}\) = normalized new signatures from novel data at a new time window: \(frq_i\) (column vector)

Output: \(T_{table}\) = Tanimoto similarities between all column \(frq_i\) and \(frq_j\) in both signatures

Output: \(PredictedClass_{table}\) : where \(C_i\) is the Class of Test Data Sample \(i\)

Procedure:

1. Initialize \(T_{table}\):
2. Obtain \(T_{ith}\) entry at every \(ith\) iterations:
3. for \(i=1\) to \(size((frq_{table})_i)\) do collect \(frq_i \leftarrow ((frq_{table})_i)_{column}\)
4. for \(j=1\) to \(size((frq_{table})_j)\) do collect \(frq_j \leftarrow ((frq_{table})_j)_{column}\)
5. Obtain all TanimotoSimilarities \(T_k \leftarrow T(frq_i, frq_j)\); where \(T_k\) is \(k\) similarities between frequency column vector \((frq_{table})_i\) & \((frq_{table})_{all_j}\) respectively
6. Classification:
7. for \(k = 1\) to \((length(T_{table})_{column})\) do
8. \(C_i = \text{Assign Class Label} C_i\) based on most similar or highest \(T_k\)
9. Find the number of \(g_i\) and its class \(g_{jclass}\) in \(G\) that corresponds to \(T_k\)
10. Assign \(C_i \leftarrow f(g_{jclass})\)
11. Update \(PredictedClass_{table}\)
3.9 Summary

In this work, we present a hierarchical clustering approach to the streaming data classification problem. We created contiguous time window from the training data and then we build a global set of clusters (classifiers) by performing hierarchical clustering on data in each time window. We designed the system to calculate the Euclidean distance $d_i$ from all the existing global clusters $g_i$ in $G$ to all the candidate clusters $c_i$ for data in each window. If none of the distances, $d_i$ is greater than the predetermined threshold $\theta$, we update the global set $G$ with the new cluster $c_i$ i.e. new clusters from a current data time window are only added to the global clusters set, if all their distances from the existing global clusters exceed the pairwise Euclidean distance threshold $\theta$. The Euclidean distances were calculated from the centroids of both the existing global clusters and new clusters ($signatures$) from the new window while generating signatures. In the testing phase, we measure the similarities of each global cluster signature generated to data points in a test sample and assign the new window to class labels of global clusters with frequent similarities measure (Tanimoto similarities). We evaluate our method on different novel test samples from the data set that were not used during the training and created some confusion matrices to summarize the performance of our approach.
Chapter 4

Results

4.1 KDD 99 Cup Datasets

The KDD Cup 99 dataset\cite{21} was used for the experiment. The dataset consists of nine (9) weeks of United States Air Force Local Area Network (LAN) Transmission Control Protocol (TCP) data dump infused with a variety of network intrusion attacks at various intervals. Each connection is a sequence of TCP packets at some well-defined times. Every data point (connection record) in the KDD 99 data set has 41 attributes and its class label. The classes are broadly divided into normal or attack traffic while the attack class group traffic can be further subdivided into four main attack types:

- Denial of Service (Dos): back, neptune, smurf, teardrop, pod, land
- Probe: satan, portsweep, ipsweep, nmap
- U2R: bufferOverflow, loadmodule, perl, rootkit
- R2L: ftpWrite, guessPasswd, imap, multihop, spy, warezclient, warezmaster
The data set contains 4,898,431 and 494,021 connection records for training and testing data respectively\textsuperscript{[21]}. A connection record has about 100 bytes of binary TCP dump and it is a well-defined sequence of TCP packets with specific duration \textsuperscript{[21]}. We can consider each TCP connection record as a data point in a stream of time series data with each row of the data as a value at some well-defined time $t$. Since part of the goal of this work is to perform classification sooner using the available data, we created chunks of sequential TCP connection records that formed contiguous blocks of data samples based on our predetermined time interval. Furthermore, we intend to achieve the goal of the original KDD 99 Cup contest which was to build an intrusion detector using a predictive model or classifier that can distinguish between normal and malicious connections \textsuperscript{[21]}, however in our context, individual chunk or window (sample) will be classified rather than each individual data point. Our approach can also be extended to classify individual data points (future works) rather than the windows of the data. The KDD Cup 99 Data set can be downloaded directly from the UCI KDD Archive website hosted by the University of California, Irvine\textsuperscript{[21]}.

4.1.1 Experiment Setup

Since the goal was to demonstrate the ability of our approach to evolve new models in streaming mode, training the classifier with the entire dataset at once will be superfluous. Therefore, we used 20\% of the entire KDD data set divided into contiguous chunks of 250 data points (connection records) which generated 3265-time window with each window having 250 data points (connection records). Each connection record has its duration, and it is a sequence of TCP packets at a well-defined time, therefore can be considered as a time series data for the purpose of this experiment. Each feature sets were normalized column wise to fit the data within a boundary $[0, 10]$. In validating the performance of the system, we used two
regions of the data set for selecting test samples. In the first approach, we selected an interval of 50 connection records at each time window leaving the remaining 250 data points to be used for training. This generated 3265 novel test data samples from 20% of the entire KDD Cup 99 dataset. In the second approach, we created each sample by randomly selecting intervals of 50 contiguous connection records from the remaining 80% of the KDD Cup 99 data set which generated another set of 3265 new test data samples. For this experiment, we used a threshold distance $\theta$ of 15 and minimum points of 15 data points ("cutting" height of each dendrogram). Other optional user-defined clustering initialization parameters (see Section 3.3.2) used were $k_{\text{begin}} = 12$ and $k_{\text{later}} = 8$. The lesser value of $k_{\text{later}}$ was carefully selected to minimize the number of candidate clusters $c_i$ to be added when used throughout the course of the algorithm. The algorithm had seven (7) independent run using test data samples from both 20% and 80% of the entire training data for each run. Insignificant values on the confusion matrices due to negligible observations were removed and the confusion matrices computed. We also calculated metrics such as precision and recall that are not easily biased by the size of the training samples.

4.2 Evaluation Metrics

We created confusion matrices to evaluate the performance of our classification model. The predicted class of data for a time window is the most frequent class of the global clusters that are most similar in Tanimoto similarities to the signature of data points within that window. For example, if there are 55 global clusters $g_i$ in the global set $G$ and 15 of these global clusters ($g_{\text{closest}}$) have the most similar signatures (Tanimoto similarities) to the current data window $s_{\text{current}}$. If the class label distribution of the most similar global cluster $g_{\text{closest}} = \{\text{normal: 12, smurf:3}\}$,
then the assigned class of the data \( s_{\text{current}} = \text{normal} \). Majority voting obtains the class label of each time window. Similarly, the actual class of data in a time window is the most common class label of the data points within a time window. Another approach would be to evaluate the predicted class label of data point using their most similar global cluster \( g_i \) from the global set \( G \). This is planned as a future work. We used the least possible number (\( n=50 \)) of data points per time window to ensure the interval remain suitable for an intrusion detection system.

Wang et al \cite{13} recommended evaluating the performance of a machine learning system using metrics that are not only dependent on the size of the training and testing samples with the KDD Cup 99 Dataset. We used the same approach in this work. True Positive (TP) indicates the system correctly identified most of the class labels (dominant class label) of the data points in a time window. Similarly, False Positive (FP), the system wrongly classified the dominant class of a data in the time window as an attack although they are not. False Negative (FN) means the system identified the dominant class as "normal" while most of the class labels was an attack. If a system consistently classifies nominal data in a time window as an attack (Type I error), the network team will become complacent and ignore "actual" network intrusion. On the other hand, if the system constantly classifies "attack" as nominal (Type II error), it will be an unreliable system that cannot be relied upon for practical network intrusion detection. We compared the performance of our system using Precision and Recall. Accuracy indicates the number of times the system classification was correct.

\[
\text{Accuracy} = \frac{TP+TN}{Total}
\]

\[
\text{Precision} = \frac{TP}{TP+FP}
\]

\[
\text{Recall} = \frac{TP}{TP+FN}
\]

44
4.3 Confusion Matrices

The training (initial phase) created an average of 55 global clusters from 20% of the training dataset which was used to classify novel test data samples into their predicted class labels at different time windows. The performance of the system was estimated using both the normal and individual attack class labels as well as the broader class groups (normal and four types of attacks). The class group performance estimation showed better results when compared with the individual attack class labels due to the uneven distribution of the individual class labels in the KDD Cup 99 dataset. For example, the following denial of service (Dos) attacks types: smurf, teardrop, and pod occurred 2,807,886 (57%), 979 (0.02%) and 264(0.005%) times respectively over the entire KDD Cup 99 training data set. Since our approach only considered the 20% of the entire dataset in streaming mode, it led to the negligible appearance of some individual class labels in the confusion matrices. While we could have trained the system with the entire training dataset, it is not in line with the goal of our approach. The highlight of the system prediction’s performance showed an overall accuracy of 99% and 75% for test data samples taken from 20% (initial) and 80% (later) of the entire training data respectively. Also, we had approximately 99% precision with denial of service (Dos) attacks in both cases (initial and later). Section 4.4 provides further details on the strategy used to select test data samples for the initial and later phase of the operation of the system. A review of the results showed the algorithm performed better in distinguishing normal data traffic from attack types of traffic but it struggles to identify the specific type of attack. Therefore, it will be suitable for a near real-time intrusion detection to immediately isolated data traffic for review by the data security team. The ability of the algorithm to separate out different attack types into their individual class
labels further deteriorated with novel data in later time windows. Furthermore, very few number of observations of probe attack types were encountered (1%) during training (signatures generation) which led to a lower classification performance when compared with others.

The table 4.1 is a Confusion Matrix Per Class Group - *initial*.

<table>
<thead>
<tr>
<th>Class Group</th>
<th>Normal</th>
<th>Dos</th>
<th>Probe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>13012</td>
<td>26</td>
<td>83</td>
</tr>
<tr>
<td>Dos</td>
<td>61</td>
<td>9246</td>
<td>4</td>
</tr>
<tr>
<td>Probe</td>
<td>92</td>
<td>6</td>
<td>315</td>
</tr>
</tbody>
</table>

*Negligible instances of Probe, U2R, and R2L were removed from the table except for probe attack (omitted for brevity).

Table 4.1: Confusion Matrix Per Class Group - *initial*

### 4.4 Performance

We compared the performance of our system with previous work on the KDD Cup 99 data set based on the percentage of the training data required before any classification task and the amount of time taken. While our approach required only 20% of the training data to begin its classification tasks, it was compared to other works which needed 100% of the training data. We used two methods described below in selecting these new data samples at different time windows.

1. *Initial*: We choose n=50 contiguous data points at each time window without replacement from a block of 300 data points leaving the remaining 250 data points to generate the signature (training) for that time window. In this case,
Negligible instances of Probe, U2R and R2L have been removed from the table.

Table 4.2: Confusion Matrix Per Class - *initial*

<table>
<thead>
<tr>
<th>Class</th>
<th>back.</th>
<th>ipsweep.</th>
<th>neptune.</th>
<th>nmap.</th>
<th>normal.</th>
<th>portsweep.</th>
<th>satan.</th>
<th>smurf.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipsweep.</td>
<td>0</td>
<td>115</td>
<td>0</td>
<td>19</td>
<td>38</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>neptune.</td>
<td>0</td>
<td>0</td>
<td>4776</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>nmap.</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>21</td>
<td>23</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>normal.</td>
<td>12</td>
<td>59</td>
<td>3</td>
<td>12</td>
<td>13012</td>
<td>10</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>portswep.</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>36</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>satan.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>114</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>smurf.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>4470</td>
</tr>
</tbody>
</table>

Table 4.3: Confusion Matrix Per Class Group - *later*

* Negligible instances of Probe, U2R, and R2L were removed from the table except for probe attack (omitted for brevity).

<table>
<thead>
<tr>
<th>Class Group</th>
<th>Normal</th>
<th>Dos</th>
<th>Probe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>1873</td>
<td>315</td>
<td>220</td>
</tr>
<tr>
<td>Dos</td>
<td>49</td>
<td>18789</td>
<td>1481</td>
</tr>
<tr>
<td>Probe</td>
<td>28</td>
<td>36</td>
<td>57</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class Group</th>
<th>TP</th>
<th>FN</th>
<th>FP</th>
<th>TN</th>
<th>TP/(TP+FP)</th>
<th>TP/(TP+FN)</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>1873</td>
<td>535</td>
<td>82</td>
<td>20365</td>
<td>0.96</td>
<td>0.78</td>
<td></td>
<td></td>
<td>0.91</td>
</tr>
<tr>
<td>Dos</td>
<td>18789</td>
<td>1530</td>
<td>353</td>
<td>2183</td>
<td>0.98</td>
<td>0.92</td>
<td></td>
<td></td>
<td>0.92</td>
</tr>
<tr>
<td>Probe</td>
<td>57</td>
<td>64</td>
<td>1701</td>
<td>21033</td>
<td>0.03</td>
<td>0.47</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

both the training data and test samples* are from the same region (20%) of the data set.

2. *later:* We choose test samples* from the remaining (80%) of the data i.e. test samples came from a region of the data set different from the region used
Negligible instances of Probe, U2R and R2L have been removed from the table.

Table 4.4: Confusion Matrix Per Class - later

to create signatures (training). Blocks of n=50 sequential data points were chosen to form these test data samples at various time windows from the other 80% of the data set.

The test sample selection method was designed to achieve the following:

- The signature generated from a data sample within any given time windows can model its chosen test sample.

- The system can learn and classify novel data in later windows successfully.

* The designation "test samples" may differ from its traditional meaning. In our definition, a test sample refers to a block of sequential data points from or to form a single time window data sample. This definition differs from the traditional meaning which refers to a partition of data set into training and testing data sets.

4.5 Comparative Analysis

To compare the performance of our approach to previous work on the KDD Cup 99 data set, we created comparison tables using similar methods and format from
Popular machine learning algorithms such as the Decision tree, Naïve Bayes, Back Propagation Neural Networks (BPNN) and Fuzzy Clustering with Artificial Neural Networks (FC-ANN)** were ran using the Weka Data Mining tool[26], although the training data was broken into time windowed chunks for suitable comparisons. The results obtained from their work was compared with our approach.

The table 4.5 is a summarized comparison of relevant work - Execution Time(Training)

<table>
<thead>
<tr>
<th>Method</th>
<th>Decision Tree</th>
<th>Naïve Bayes</th>
<th>BPNN</th>
<th>FC-ANN**</th>
<th>HC OnlineA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exec Seconds (s)</td>
<td>2.68</td>
<td>1.93</td>
<td>1538.17</td>
<td>2125.4</td>
<td>676.38*</td>
</tr>
<tr>
<td>Amt. of KDD Data (%)</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 4.5: Relevant Work Comparison - Execution Time

*Time taken to generate minimal signatures before classification of data in a new time window. The data was windowed, unlike the other algorithms

**Experiments conducted by Wang et al, [13] was on a single 1.83GHZ CPU machine while we ran the others on a single 2.5GHZ CPU Machine. Training data was not suitable to be broken into separate time windows because random selection used in selecting observations to reduce the size of the dataset [13] for the FC-ANN.

The table 4.6 is a summarized comparison of relevant work - (Normal)

The table 4.7 is a summarized comparison of relevant work - (Dos)

The table 4.8 is a summarized comparison of relevant work - (Probe)

The 20% region of the data set used in our work to generate the minimal signatures had negligible observations for R2L & U2L (unauthorized access) class labels, but
<table>
<thead>
<tr>
<th>Method</th>
<th>Decision Tree</th>
<th>Naïve Bayes</th>
<th>BPNN</th>
<th>FC-ANN</th>
<th>HC Online_A</th>
<th>HC Online_B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision (%)</td>
<td>91.22</td>
<td>89.22</td>
<td>89.75</td>
<td>91.32</td>
<td>98.84</td>
<td>95.81</td>
</tr>
<tr>
<td>Recall (%)</td>
<td>99.41</td>
<td>97.70</td>
<td>98.20</td>
<td>99.08</td>
<td>99.17</td>
<td>77.78</td>
</tr>
<tr>
<td>Amt. of KDD Data (%)</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>20</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.6: Relevant Work Comparison - Normal

<table>
<thead>
<tr>
<th>Method</th>
<th>Decision Tree</th>
<th>Naïve Bayes</th>
<th>BPNN</th>
<th>FC-ANN</th>
<th>HC Online_A</th>
<th>HC Online_B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision (%)</td>
<td>99.84</td>
<td>99.69</td>
<td>99.79</td>
<td>99.91</td>
<td>99.66</td>
<td>98.16</td>
</tr>
<tr>
<td>Recall (%)</td>
<td>97.24</td>
<td>96.65</td>
<td>97.20</td>
<td>96.70</td>
<td>99.30</td>
<td>92.47</td>
</tr>
<tr>
<td>Amt. of KDD Data (%)</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>20</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.7: Relevant Work Comparison - Dos

<table>
<thead>
<tr>
<th>Method</th>
<th>Decision Tree</th>
<th>Naïve Bayes</th>
<th>BPNN</th>
<th>FC-ANN</th>
<th>HC Online_A</th>
<th>HC Online_B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision (%)</td>
<td>50.00</td>
<td>52.61</td>
<td>60.94</td>
<td>48.21</td>
<td>78.36</td>
<td>00.00**</td>
</tr>
<tr>
<td>Amt. of KDD Data (%)</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>20</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.8: Relevant Work Comparison - Probe

there were removed to avoid insignificant values on the confusion matrices due to very few observations from these class instances.
The figure 4.1 below shows how long it took for classification for each algorithm.

![Graph showing execution speed and amount of data required before classification comparison.]

Figure 4.1: Execution Speed and Amount of Data required before Classification Comparison.
The signatures were already generated during the initial phase of the algorithm.

The figure 4.2 compares the **precision** and **recall** of related methods - normal

![Bar chart showing performance comparison of relevant methods - normal class group.]

Figure 4.2: Performance Comparison of relevant methods - normal class group. Initial and later phase of the algorithm had data from individual time window with each window having n=50 sequential data points (connection records).
The figure 4.3 compares the precision and recall of related methods - Dos class group.

Figure 4.3: Performance Comparison of relevant methods - Dos class group. Initial and later phase of the algorithm had data from individual time window with each window having n=50 sequential data points (connection records).
The figure 4.4 compares the precision of related methods - Probe

Figure 4.4: Performance Comparison of relevant methods - Probe class group. Initial and later phase of the algorithm had data from individual time window with each window having \( n=50 \) sequential data points (connection records).

The results showed our hierarchical clustering online system was able to start its classification with the least amount of data and therefore more suitable for a near real-time intrusion detection system since it did not require the entire training data set before to begin. Also, the algorithm can be said to be self-improving which means as more data arrives in a later time window, we can create more signatures and further improve its performance without disrupting ongoing classification tasks. Lastly, as shown from the results, although the system used the least amount of data to generate its signatures, its classification performance was comparable to well-known methods that had used the entire dataset. While an argument could be made against the system performance (the execution time), however, its execution speed can be improved by using additional processing power or the reduction of data points per time window. We demonstrated a later improvement effort in the second part of the system’s operation where we used only 50 data points per time window compared to the initial phase implementation which uses 250 data points per time window while creating the minimal set of signatures.
Chapter 5

Conclusion

5.1 Conclusion

One of the top predictions \[1\] from Gartner, a global information technology research company, anticipates that known vulnerabilities will continue to be exploited in computer networks. The situation becomes even more precarious as we become more dependent on computer network services which explain the geometric growth of intrusion attacks for ransomware and denial of service. As a result, the need for a more adaptive and online intrusion detection can no longer be overemphasized. The goal of this work is to provide a practical framework that can be used in a production environment for intrusion detection. Our system used a global set of "signatures" from data in each time window to identify normal or anomalous data traffic (TCP Connection records) in our testing phase. As more data arrives, we update the global sets of signatures, which makes the algorithm self-improving and adaptive to be used as an online classification system. The design will be suitable and can be adapted to handle the dynamic nature of attacks signatures in an intrusion detection system. Our approach can be used as a simple but yet effective means to
use machine learning algorithms to identify and distinguish anomalous data traffic from nominal as we prototyped using the popular KDD Cup 99 Dataset. Also, as an intermediate result, the algorithm developed as a part of this work can be used to compare two or more dendrograms obtained from different hierarchical clustering efficiently.

5.2 Future Works

The classification model in the current work distinguished dominant (majority) class labels at different time window from each other. Additional implementation could be adapted to classify individual data point at different time interval using the each data point similarities to the global cluster. Also, the system could be more dynamic by updating the global cluster to candidate cluster threshold distance parameter ($\theta$) as appropriate while continuously adding new signatures throughout course of the operation of the system.
Appendix A

Terms and Definitions

A.1 Definitions

1. *Size* of a node on a dendrogram is the number of children nodes connected to it.

2. A *candidate cluster* is any cluster from the dendrogram with generated from parent nodes such that the number of data points in the cluster is greater than the *minimum point*.

3. A *centroid* is a row vector which whose values equals the mean of the data points in the cluster.

4. A candidate cluster $c_i$ is added to the global clusters list $G$ if and only if all its pairwise Euclidean distances $d_{ij}$ to existing global clusters are all greater than the threshold distance $\theta$.

5. A *signature* is the frequency of each $g_i$ in *Least Distance Clusters* i.e. $f(g_j,c_i)$.

6. Global Clusters *signature* used global to candidate clusters distance while novel (test) data used individual data rows to global clusters distance.
Bibliography


