STREAM GENETIC PROGRAMMING FOR BOTNET DETECTION

by

Sara Khanchi

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

at

Dalhousie University
Halifax, Nova Scotia
November 2019

To my beloved Mom and Dad,

Farangis and Mahdi,

for their unconditional love, inspiration and gracious support

Table of Contents

List of Tables . . . . **vii**

List of Figures . . . . **xi**

Abstract . . . . **xv**

List of Abbreviations . . . . **xvi**

Acknowledgements . . . . **xx**

Chapter 1 Introduction **1**

1.1 Genetic Programming 3

1.2 Botnet 4

1.3 Objective 5
1.3.1 Streaming Classification 5
1.3.2 Network Behaviour Detection 6

1.4 Contributions 8

1.5 Thesis Outline 10

Chapter 2 Related Work **11**

2.1 Non-evolutionary Methods 11

2.2 Evolutionary Methods 14

2.3 Network Methods 16

2.4 Summary 24

Chapter 3 Framework for Streaming GP Teams **27**

3.1 Streaming Data Environment Under a Label Budget 27

3.2 Overall Framework 28

3.3 Sampling Policy 30

3.4 Archiving Policy 31

3.5 Champion Classifier Identification 34

3.6 Symbiotic Bid-based GP 35

3.7 Summary 37

Chapter 4 Evaluation methodology 39

4.1 Datasets 39
4.1.1 CTU-13 39
4.1.2 CTU13-mixed 42
4.1.3 ISOT 43
4.1.4 NSL-KDD 44

4.2 Comparator Algorithms 44
4.2.1 Naive Bayes 45
4.2.2 Decision Trees 46
CART 46
Hoeffding Tree 48
Random Forest 50

4.3 Comparator Frameworks 51
4.3.1 Massive Online Analysis (MOA) 52
4.3.2 Apache Spark Streaming 53

4.4 Performance Metrics 54

4.5 Experimental Design and Parameterization 56

4.6 Statistical Significance Testing 59
4.6.1 Wilcoxon-Mann-Whitney Test 59
4.6.2 Friedman Test 59
4.6.3 Bonfferoni-Dunn Post-hoc Test 60
4.6.4 Nemenyi Post-hoc Test 60

4.7 Summary 61

Chapter 5 Results 62

5.1 Botnet Detection in Real-world Network Traffic 62
5.1.1 Overall Performance Evaluation 64
5.1.2 Detection Rate Dynamics: Comparing the Best Streaming Classifiers 67
5.1.3 Detection Rate Dynamics: Stream-GP Sampling and Archiving Policies 73
5.1.4 Capacity for Detecting Botnet Signals 78
5.1.5 Capacity for Detecting Botnet C&C Signals 79

5.1.6 Distribution of Minor Classes 80
5.1.7 Real-time Operation 82
5.1.8 Summary 84

5.2 Botnet Detection in Multi-bot Network Traffic 85
5.2.1 Overall Performance Evaluation 86
5.2.2 Detection Rate Dynamics: Comparing the Best Streaming Classifiers 88
5.2.3 Capacity for Detecting Botnet Signals 93
5.2.4 Capacity for Detecting Botnet C&C Signals 96
5.2.5 Feature Selection by Stream-GP Champion 98
5.2.6 Real-time Operation 100
5.2.7 Summary 102

5.3 Network Security in Real-world Network Traffic 103
5.3.1 Overall Performance Evaluation 103
5.3.2 Detection Rate Dynamics: Comparing the Best Streaming Classifiers 104
5.3.3 Summary 108

5.4 A Comparison to the Apache Spark Streaming Network Tool 109
5.4.1 Overall Performance Evaluation 110
Streaming Simulation Scenario 111
Classical Scenario 113
Comparison of the Top Performing Algorithms: 114
5.4.2 Dynamic Properties of the Detection Rate 115
Streaming Simulation Scenario 115
Classical Scenario 122
5.4.3 Capacity for Detecting Botnet Signals 126
5.4.4 Capacity for Detecting Botnet C&C Signals 127
5.4.5 Feature Selection by Decision Tree 128
5.4.6 Computational Cost of Streaming Classifier Operation 130
5.4.7 Complexity of Algorithms 133
5.4.8 Summary 135

5.5 Network Analytic Applications 136
5.5.1 Traffic Analysis 136
5.5.2 Summary 139

5.6 Summary 141

Chapter 6 Conclusion and Future Work 143

6.1 Conclusion 143

6.2 Future Works 145

Appendix A Flow Features **146**

A.1 ISOT Flow Features 146

A.2 NSL-KDD Packet Features 146

Bibliography **147**

List of Tables

2.1 Existing network related works (architecture and learning process), Arch.: Architecture, Data P.: Data Processing, MC: Multi-Class, CD: Change Detection, DI: Data Imbalance, LB: Label Budget 22

2.2 Existing network related works (application and metric), Sec.:Security, Mon.:Monitoring, BD,ID,ITD:Botnet,Intrusion,Insider Threat Detection 23

4.1 Argus flow features for the CTU-13 dataset 40

4.2 Generic properties of the CTU-13 streaming datasets. N cardinality, and k is the total number of classes over the entire duration of the stream. Each dataset has $D = 8$ flow attributes. Classes are represented in the order: Background, Normal, Botnet and Botnet C&C. A combined Botnet/C&C label was assumed in the case of datasets in which the C&C class represents less than 0.01% of the original dataset (Capture 3, 4, 10, 11, 12). 41

4.3 Generic properties of the CTU13-mixed dataset. 7 types of Botnet behaviour are available. Classes are represented in the order: Background, Normal, Botnet and Botnet C&C. N is the Cardinality, and the class distribution is expressed in percentages. . 43

4.4 Generic properties of the ISOT streaming datasets. N cardinality, and k is the total number of classes over the entire duration of the stream. Each dataset has $D = 79$ flow attributes. 43

4.5 Generic properties of the NSL-KDD streaming datasets. N cardinality, and k is the total number of classes over the entire duration of the stream. Each dataset has $D = 22$ packet attributes. 44

4.6 Decision Tree Algorithms 47

4.7 Comparator algorithms List. 51

4.8 Best case configurations of MOA comparator algorithms for operation under drifting streaming data with label budgets [106]. 52

4.9 Stream-GP algorithms with their corresponding configurations. Uniform implies the identification of either sampling or archiving data using uniform sampling (Sections 3.3 and 3.4, respectively). Biased refers to either sampling or archiving data that happens under the corresponding biased algorithms (Algorithms 1 and 2 respectively). 57

4.10 Stream-GP Parameters. Mutation specifies the rate of adding/deleting programs from a learner or changing the action. *Gap*($Tgap$) denotes the number of records in the Data subset (teams) to be deleted at each non-overlapping window location. For each Data subset update, τ generations are performed. 57

4.11 Stream Dataset Parameters. Label Budget (β) is defined as a function of the window size $W(i)$ where for each non-overlapping window location there can only be *Gap* size (20) samples. . . . 58

5.1 Algorithm ranks w.r.t. streaming AvDR metric under a 5% label budget. Bracketed entries represent median AvDR values to 1 decimal place. Naive Bayes (NB) and Hoeffding tree classifiers (from MOA) appear with either 'split' or 'variable' sampling policies. Table 4.9 declares the 4 sampling/replacement policies for stream-GP. R_j denotes the average rank across all datasets. 64

5.2 Algorithm ranks w.r.t. streaming AvDR metric under a 1% label budget. Bracketed entries represent median AvDR values to 1 decimal place. Naive Bayes (NB) and Hoeffding tree classifiers (from MOA) appear with either 'split' or 'variable' sampling protocols. Table 4.9 declares the 4 sampling/replacement policies for stream SBB. R_j denotes the average rank across all datasets. 65

5.3 Algorithm ranks w.r.t. streaming AvDR metric under a 0.5% label budget. Naive Bayes (NB) and Hoeffding tree classifiers (from MOA) appear with either 'split' or 'variable' sampling policies. Table 4.9 declares the 4 sampling/replacement policies for stream SBB. R_j denotes the average rank across all datasets. 65

5.4 Result of Friedman test χ_F^2 and corresponding value for F-distribution F_F. The critical value of $F(7, 84)$ for $\alpha = 0.01$ is 2.86, so the null-hypothesis is rejected in each case. 66

5.5 Ranks for the Botnet class streaming AvDR alone. 78

5.6 χ_F^2 and F_F values for different label budgets. Assuming $\alpha = 0.01$ returns a critical value of $F(7, 84) < 2.86$, the null-hypothesis (of random ranking) is rejected. 78

5.7 Ranks for Botnet C&C class streaming AvDR alone. 79

5.8 χ^2_F and F_F values for different label budgets. Assuming an $\alpha = 0.01$ returns a critical value of $F(7, 84) < 2.86$, the null-hypothesis (of random ranking) is rejected. 79

5.9 Median values w.r.t. streaming AvDR metric, under 0.5% and 5% label budgets. Three Stream-GP configurations are declared in Table 4.9. Hoeffding Tree and Naive Bayes (from MOA) appear with the 'variable' sampling policy, the *CTU13-mixed* dataset 87

5.10 p-value from Mann-Whitney U test for Stream-GP and MOA algorithms, for the 5.0% label budget 88

5.11 p-value from the Mann-Whitney U test for the Stream-GP and MOA algorithms for the 0.5% label budget 88

5.12 median values w.r.t. streaming Botnet AvDR metric under 0.5% and 5% label budgets. Three Stream-GP configurations are given in Table 4.9. Hoeffding Tree and Naive Bayes (from MOA) appear with the 'variable' sampling policy, *CTU13-mixed* dataset 93

5.13 p-value from the the Mann-Whitney U test for the Stream-GP and MOA algorithms based on Botnet median AvDR for the 5.0% label budget 95

5.14 p-value from the Mann-Whitney U test for the Stream-GP and MOA algorithms based on Botnet median AvDR, for the 0.5% label budget 95

5.15 median values w.r.t. streaming Botnet C&C AvDR metric under 0.5% and 5% label budgets. Three Stream-GP configurations are declared in Table 4.9. Hoeffding Tree and Naive Bayes (from MOA) appear with the 'variable' sampling policy, *CTU13-mixed* dataset 96

5.16 p-value from the Mann-Whitney U test for the Stream-GP and MOA algorithms based on Botnet C&C median AvDR for the 5.0% label budget 98

5.17 p-value from Mann-Whitney U test for Stream-GP and MOA algorithms based on Botnet C&C median AvDR, for the 0.5% label budget 98

5.18 Algorithm ranks w.r.t. the streaming AvDR metric under a 5.0% label budget. Bracketed entries represent median AvDR values to 1 decimal place. 104

5.19 Result of Friedman test χ_F^2 and corresponding value for F-distribution F_F. The critical value of $F(2, 28$ for $\alpha = 0.01$ is 2.503, so the null-hypothesis is rejected in each case. 105

5.20 Median values, under a 5% label budget. The best Stream-GP configuration (GP–Hybrid) and the best Apache Spark Algorithm (Decision Tree), *CTU13-mixed* dataset 115

5.21 p-value from the Mann-Whitney U test for the Stream-GP and Spark best algorithms for the 5.0% label budget 115

5.22 Median values for Botnet AvDR metric, under the 5% label budget. The best Stream-GP configuration (GP–Hybrid) and the best Apache Spark Algorithm (Decision Tree), *CTU13-mixed* dataset 126

5.23 p-value from the Mann-Whitney U test for the Stream-GP and Spark best algorithms based on Botnet median AvDR, for the 5.0% label budget, 95% confidence level 126

5.24 Median values for the Botnet C&C AvDR metric under the 5% label budget. The best Stream-GP configuration (GP–Hybrid) and the best Apache Spark Algorithm (Decision Tree), *CTU13-mixed* dataset 128

5.25 p-value from the Mann-Whitney U test for the Stream-GP (GP–Hybrid) algorithm vs. the comparator Apache Spark and MOA algorithms based on Botnet C&C median AvDR, for the 5.0% label budget, 95% confidence level 128

List of Figures

1.1 Botnet architectures [115] 5

2.1 Network streaming applications 17

2.2 Data mining categories 17

3.1 Overall Stream-GP framework. Sampling policy, (S) determines which exemplars will be quarried for their true labels (under label budget constraint β). Archiving policy, (A) keeps a finite size Data Subset, (DS), where 'Gap' records are subject to replacement. On the update of DS with $Gap(i)$ labelled records, τ generations of GP are performed. At anytime, a single champion individual is available to predict the labels, $y(t)$, which may influence the Sampling policy operation. 29

3.2 Symbiotic Bid-Based GP. Each team indexes a different combination of programs, but the same program may appear in multiple teams. The action (class label) of a program is expressed through colour. 36

4.1 Apache Spark architecture. 53

5.1 Overall structure of GP-Stream evaluations and comparisons . 63

5.2 Capture 5 class-wise Detection rate through the stream. 5% versus 0.5% label budget. 68

5.3 Capture 6 class-wise Detection rate through the stream. 5% versus 0.5% label budget. 69

5.4 Capture 8 class-wise Detection rate through the stream. 5% versus 0.5% label budget. 70

5.5 Capture 9 class-wise Detection rate through the stream. 5% versus 0.5% label budget. 71

5.6 Class-wise Detection rate for Stream-GP sampling and archiving policies on the Capture 1 dataset at a 5.0% label budget. . 74

5.7 Class-wise False positive rate for Stream-GP sampling and archiving policies on the Capture 1 dataset at a 5.0% label budget. . 75

5.8 Typical Distribution of classes present in the Data Subset for Stream-GP sampling and archiving policies on the Capture 1 dataset at a 5.0% label budget. 76

5.9 Distribution of minor classes over the course of the stream for the five capture datasets appearing in Sections 5.1.2 and 5.1.3. Note the use of a log scale and the colour coding, which corresponds to that adopted for the original Stream DR figures. Background class is omitted for clarity (always 90 to 99%). Normal class represents 'normal' traffic corresponding to the CTU filters, represents Botnet and represents Botnet C&C. The log scale also implies that 10^{-2} is synonymous with zero content (e.g. the earliest that Botnet C&C appears at the 40% point in Captures 5 and 9). 81

5.10 Wall clock time for (a) champion individual to make predictions and (b) fitness evaluation to update the content of the population on a new non-overlapping window location for the Capture 3 dataset on a 2.67 GHz CPU. 83

5.11 Violin plots for a 5% label budget based on streaming AvDR metric (Average of 20 runs) 87

5.12 Class-wise Detection rate through the stream at a 5% label budget based on streaming AvDR metric (Average of 20 runs) 89

5.13 False positive rate through the stream at a 5% label budget (Average of 20 runs) 90

5.14 Typical Distribution of classes present in the Data Subset for the GP–Archive and GP–Both algorithms on the CTU13-mixed dataset at a 5.0% label budget. 92

5.15 Botnet distributions at each 10% of the CTU13-mixed dataset 93

5.16 Violin plots for comparison of algorithms in Botnet detection, 0.5% and 5% label budgets. 94

5.17 Violin plots for the comparison of algorithms in Botnet C&C detection, 0.5% and 5% label budgets. 97

5.18 Overview of the features used by the GP–Archive champion throughout the stream, 5% label budget 99

5.19 Overview of the features used by the GP–Archive champion throughout the stream based on action, 5% label budget . . . 99

5.20 The average number of execution code lines based on action (class label), 5% label budget 100

5.21 Wall clock time for (a) the champion individual to make predictions and (b) the fitness evaluation to update the content of the population on a new non-overlapping window location for the CTU13-mixed dataset on a 2.67GHz CPU. 101

5.22 NSL-KDD class-wise Detection rate through the stream at a 5% label budget based on the streaming AvDR metric (Average of 20 runs) 106

5.23 ISOT class-wise Detection rate through the stream at a 5% label budget based on the streaming AvDR metric (Average of 20 runs) 107

5.24 GP–Archive class-wise false positive rate through the stream at a 5% label budget based on the streaming AvDR metric (Average of 20 runs) 108

5.25 Apache Spark Experiments flowchart. 110

5.26 CTU13-mixed class-wise Detection rate through the stream. Stream simulation scenario. 5.0% label budget. The Y-axis shows the Split where training data are sampled from. 112

5.27 CTU13-mixed class-wise Detection rate in the training phase. 5.0% label budget. The Y-axis shows the Split where training data are sampled from. 113

5.28 CTU13-mixed class-wise Detection rate through the stream. Classical scenario. 5.0% label budget. The Y-axis shows the algorithms. 114

5.29 CTU13-mixed class-wise Detection rate through the stream. Stream simulation scenario. Split 9. 5.0% label budget (Average of 10 runs) 118

5.30 CTU13_mixed class-wise False positive rate through the stream. Stream simulation scenario. Split 9. 5.0% label budget (Average of 10 runs) 119

5.31 CTU13_mixed class-wise Detection rate through the stream. Stream simulation scenario. Split 7. 5.0% label budget (Average of 10 runs) 120

5.32 CTU13-mixed class-wise False positive rate through the stream. Stream simulation scenario. Split 7. 5.0% label budget (Average of 10 runs) 121

5.33 CTU13-mixed Class-wise Detection rate through the stream at a 5% label budget based on the streaming AvDR metric (Average of 10 runs) 123

5.34 CTU13_mixed class-wise Detection rate through the stream. Classic Scenario. 5.0% label budget (Average of 10 runs) . . . 124

5.35 CTU13_mixed class-wise False positive rate through the stream. Classic Scenario. 5.0% label budget (Average of 10 runs) . . . 125

5.36 CTU13_mixed class-wise Detection rate through the stream. 5.0% label budget. 127

5.37 Overview of the Apache Spark Decision Tree in a stream simulation scenario for a run on split 7, 5% label budget 129

5.38 Wall clock time, GP–Archive (a) champion individual to make predictions and (b) fitness evaluation to update the content of the population on a new non-overlapping window location for the CTU13-mixed dataset on a 2.67GHz CPU. 131

5.39 Wall clock time (a) model to make predictions for each exemplar and (b) Off-line model that is trained on the training set for the CTU13-mixed dataset on a 2.67GHz CPU. 132

5.40 GP–Archive average number of execution codes, whole number and effective number, for the champion throughout the stream. 5% label budget. 134

5.41 GP–Archive average champion team size throughout the stream. 5% label budget. 134

5.42 Distribution of Ground Truth traffic, CTU13-mixed dataset, Labels: Normal (1), Botnet C&C (2) and Botnet (3) 137

5.43 Botnet activities in ground truth traffic, CTU13-mixed dataset, Labels: Normal (1), Botnet C&C (2) and Botnet (3) 138

5.44 Stream-GP analytics on malicious botnet activities, CTU13-mixed dataset, Labels: Normal (1), Botnet C&C (2) and Botnet (3) . . . 140

Abstract

Algorithms for constructing classification models in streaming data scenarios are attracting more attention in the era of artificial intelligence and machine learning for data analysis. The huge volumes of streaming data necessitate a learning framework with timely and accurate processing. For a streaming classifier to be deployed in the real world, multiple challenges exist such as 1) Concept drift, 2) Imbalanced data; and 3) Costly labelling processes. These challenges become more crucial when they occur in sensitive fields of operation such as network security. The objective of this thesis is to provide a team-based genetic programming (GP) framework to explore and address these challenges with regard to network-based services. The GP classifier incrementally introduces changes to the model throughout the course of the stream to adapt to the content of the stream. The framework is based on an active learning approach where the learning process happens in interaction with a data subset to build a model. Thus, the design of the system is founded on the introduction of sampling and archiving policies to decouple the stream distribution from the training data subset. These policies work with no prior information on the distribution of classes and true labels. Benchmarking is conducted with real-world network security datasets with label budgets in the order of 5 to 0.5 percent and significant class imbalance. Evaluations for the detection of minor classes have been performed that represent the classifier behaviour in case of attacks. Comparisons to the current streaming algorithms and specifically network state-of-the-art frameworks for streaming processing under label budgets demonstrate the effectiveness of the proposed GP framework to address the challenges related to streaming data. Furthermore, the applicability of the proposed framework in network and security analytics is demonstrated.

List of abbreviations

ANOVA	Analysis of variance
API	Application Programming Interface
AUC	Area Under the Curve
AvDR	Average Detection Rate
CART	Classification and Regression Tree
C&C	Command and Control
CC	Corner Classification
CD	Critical Difference
CPU	Central Processing Unit
DoS	Denial of Service
DDoS	Distributed Denial of Service
DNS	Domain Name System
DR	Detection Rate
DS	Data Subset
FPR	False Positive Rate
GB	Giga Byte
GHz	Giga Hertz
GP	Genetic Programming
GT	Ground Truth
HTTP	Hypertext Transfer Protocol
HTTPS	Secure Hypertext Transfer Protocol

ICMP	Internet Control Message Protocol
IP	Internet Protocol
IRC	Internet Relay Chat
IT	Information Technology
k-NN	k Nearest Neighbour
LCS	Learning Classifier System
MAP	Maximum A Priori
ML	Machine Learning
MLP	Multi Layer Perceptron
MoA	Massive Online Analysis
NB	Naive Bayes
NTMA	Network Traffic Monitoring and Analysis
P2P	Peer to Peer
Prec	Precision
R2L	Remote to Local
RAM	Random Access Memory
Rnd	Random
SBB	Symbiotic Bid Based
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
SSH	Secure Shell
TPG	Tangled Program Graph
U2R	User to Remote

UDP	User Datagram Protocol
WEKA	Waikato Environment for Knowledge Analysis

Symbols

β	Label budget
τ	Generations per data subset
$\mathbf{A}$	Archiving policy
$\mathbf{d}$	Dimension of stream
$\mathbf{k}$	Number of algorithms
$\mathbf{n}$	Number of datasets
$\mathbf{S}$	Sampling policy
$\mathbf{y}$	Model prediction

Acknowledgements

First and foremost, I would like to express my greatest gratitude to my supervisor, Dr. Nur Zincir-Heywood, for her dedicated support throughout my PhD, for her patience, encouragement and positive energy. She has been a tremendous mentor and provides an excellent example of a successful and powerful Woman in Tech.

Also, I would like to thank my co-supervisor, Dr. Malcolm Heywood, for his guidance, caring and thorough vision. He has aroused my interest in an amazing Artificial Intelligence (AI) field, Genetic Programming. Moreover, he has taught me new technical and scientific methods as well as how to see life's philosophy from a brand-new perspective. I have been blessed to have them both Nur and Malcolm as my supervisors. They have provided a welcoming atmosphere for me away from home.

A very special gratitude goes out to the rest of my thesis committee: Dr. Uyen Trang Nguyen, Dr. Srinivas Sampalli, and Dr. Andrew McIntyre, for generously offering their time, interest and helpful comments. I would also like to recognize the technical contributions of my fellow office mates, Ali Vahdat, Stephan Kelly, and Duc Le.

Hereby, I would like to thank my friends for their emotional support, which has always helped me stand up stronger after times of difficulty.

Finally, I would like to thank my family. I am so grateful for all the sacrifices they have made, in raising me and supporting me in all my pursuits. Words cannot express how much I appreciate their efforts.

Chapter 1

Introduction

Streaming data refers to the continuous, unbounded flow of data that is mostly subject to non-stationary properties such as concept drift [101, 37, 59, 76]. A concept drift occurs when the statistical properties of a label change over time in unpredictable ways. In other words, consider records ($\overrightarrow{x}$) that arrive sequentially at discrete points in time, t, where $p(\overrightarrow{x}, d)$ is the joint probability and d refers to the record's unknown true label. If there exist times t and $t+1$ where $p_t(\overrightarrow{x}, d) \neq p_{t+1}(\overrightarrow{x}, d)$, then concept drift has occurred. The changes could happen gradually or suddenly, and are subject to repetition which may affect various subsets of classes at different points in the stream. Concept drift is encountered in many real-world scenarios such as surveillance systems, financial fraud detection, weather prediction and analyzing customer preferences. Currently, classifiers are available that work perfectly under a static data environment where the whole dataset is available for learning. Static environments refer to environments where the distribution of data do not change throughout time, whereas dynamic environments are subject to change. The classifiers that work on static data are able to work on large-scale datasets with high accuracy, but they are unable to perform in non-stationary environments. Recently, many research studies have explored learning methods in dynamic environments in which the primary asset is the ability to detect *concept drift*.

The goal of a classifier operating on these streams is therefore multifaceted. The classifier should be able to suggest labels for multiple classes in real time throughout the course of the stream. There should also be a mechanism to recommend what data to learn from as in a non-stationary environment, as the training data are interactively selected from the stream. The selection of data is also subject to a limitation, as a human expert is needed to suggest the ground truth label for the training data. Because providing the ground truth is costly, a small fraction of data would be offered for labelling according to a predefined *label budget*. This constraint is available in many applications, such as satellite data labelling and financial services [119].

In this thesis, the focus is concentrated on the discovery of multiple behaviours such as different botnets in network traffic. Network traffic precisely reveals the characteristics of a stream under a non-stationary environment. This task is difficult due to the following reasons:

1. Network load and application mix are time-varying parameters.
2. Some applications suddenly switch between modes of operation; e.g. services like Tor and Skype are trying to hide their transactions.
3. Malicious behaviours are combined with legitimate (normal) behaviours.
4. Normal behaviour encompasses a broad category in which users also have different normal behaviours.
5. Different versions of the same application coexist, which may create different 'fingerprints' of the same application.
6. The malicious behaviours ratio is very low in comparison with the non-malicious behaviours ratio.

Another focus point of current data stream processing studies is performance under *class imbalance* issues. The nature of the data stream implies that classes may appear in different distributions during the course of the stream. This is apparent when streaming data encounters the 'burst' of one class, in which a simple one-bit state machine classifier outperforms many other sophisticated models. Therefore, along with the label budget and class imbalance constraints, the sensitivity of the classifier performance under *multi-class* scenarios is examined in this thesis.

The network behaviour detection is framed as follows. For network security applications, the time when malicious behaviours appear cannot be known a priori when normal and malicious traffic are mixed together. The normal behaviour is non-stationary and changes due to variants such as users, applications, etc. Therefore, it is not feasible to pre-train a model and then apply it to the rest of the stream. During this time, the trained model would become outdated and useless when changes occur in the stream. Human experts provide labels for small subsets of stream data, based on the label budget, during the course of the stream. Keeping the human in the loop of machine learning is necessary, as attacks to the machine learning leads the attack behaviours to be considered normal by manipulating the stream data [7, 6]. The proposed Stream-GP framework is responsible for introducing data points to human experts for providing ground truth labels. The human experts are assumed to be trustworthy; otherwise, they will mislead the Stream-GP framework (similar to an attack to the ML). A GP champion should be available at all times to predict labels before any queries for the true labels. This way, the Stream-GP framework is continuously interacting with the stream, labelling the stream content and then querying a fraction of the stream based on the label budget. Then, the data with their true labels are used by the learning process to update the model throughout the stream. The proposed framework is able to be applied on the incoming/outgoing traffic of a wide range of network devices such as servers, routers and client devices. It could be utilized by financial, medical or any other type of institution with the support of human security to act as a reliable source of label information. IT security companies could use the proposed framework to provide anytime classifiers to service subscribers and retain the other components of the architecture.

1.1 Genetic Programming

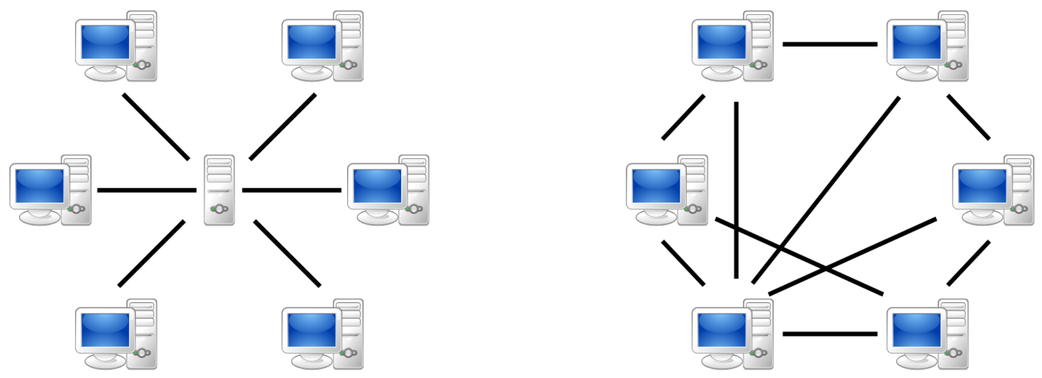
Genetic Programming (GP) is a special case of a genetic algorithm where the population individuals are computer programs [75]. It is a biologically-inspired approach to Machine Learning, which generalizes the basic biological evolutionary process to identify sophisticated prediction models. It begins with a population of simple programs that are randomly generated in basic GP. Then, the programs evolve to more complex versions after generations, which fit to the task definition. The evolution is directed toward the task goal by computing fitness score at each generation. The worst performing individuals are removed, and some individuals are sampled from the remaining population based on a pre-determined probability, which is then modified to make the next generation's population. This process is done using three main operators: 1) Selection, 2) Crossover, and 3) Mutation. In selection phase, some individuals are selected probabilistically based on fitness. That means the better performing individuals are more likely to be selected to have children. Then variation operators, crossover and mutation, are applied on the selected population to generate new individuals for the next generation. The crossover is mostly done on two selected individuals and merge them to create new individuals. However, mutation is a random change to one individual to create an offspring with modified behaviours.

Generally speaking, there are various advantages in adopting GP. Specifically, programs are capable of representing complex and non-linear functions. As well, the teaming solution makes it possible to decompose the sophisticated problems into smaller parts (divide & conquer) and assign several programs to solve each specific part. Moreover, GP has an implicit embedded approach for variable selection. This internal capability helps GP to select different variable sets based on the changes in the environment during the learning process.

1.2 Botnet

A botnet is a number of Internet-connected devices (bots) that form a network that is used for malicious activities such as (Distributed) Denial of Service attacks, sending spam, stealing data and accessing a victim's device. The word "botnet" is derived from the words "robot" and "network". Bots communicate with each other through Command and Control (C&C) communication protocols to send/receive commands. The most commonly used protocols are IRC, HTTP and DNS. In terms of architecture, botnets are either client/server models or P2P. Recent botnets rely on the latter architecture. Figure 1.1 depicts the types of botnet architectures.

The use of botnets is rapidly increasing, and their behaviour is becoming more complex. Nowadays, they are trying to hide their identity by concealing inside the traffic of the most commonly used protocols like HTTP and DNS (fast fluxing [122]). Encrypted communication is another way of keeping botnet behaviour from being detected [30].



(a) Client/Server model (b) Peer-to-Peer

Figure 1.1: Botnet architectures [115]

1.3 Objective

The primary objective of this research is to provide a team-based GP streaming classifier that works under class imbalance and label budgets simultaneously. The application of the proposed framework is investigated specifically in relation to network behaviour identification. Therefore, the objective is reviewed in two separate aspects: 1) Streaming classification in general;, and 2) Network behaviour detection specifically.

1.3.1 Streaming Classification

The generic challenges a streaming classifier encounters are listed; the proposed streaming classifier provides a solution that covers them all.

Non-stationary process: Streaming data are usually generated by a non-stationary process. This leads to gradual or sudden changes in points throughout the stream, which is called "concept drift".

Partial observability: Access to the content of the stream is limited to a window location, that is sliding or non-overlapping. Therefore, the overall characteristics of the stream may not be met at any time and only some properties of the stream will be revealed. Consequently, the model should deal with this partial observability during the learning process.

Anytime operation: A model, '*GP champion*', should be available at any time throughout the course of the stream to predict the label.

Single-pass operation: As the stream passes, the previous data in the stream are not revisited. As a result, the ML model only has one opportunity to predict any exemplar's label.

Label budget: The volume of data in streaming applications is enormous, which makes labelling all the data impossible or extremely costly. To reduce the cost of the learning process, only a small fraction of the stream is asked for their true label. The queried exemplars for the ground truth information act as the 'train' dataset.

Class imbalance: The probability of the imbalanced distribution of the classes in streaming data is extremely high. The imbalanced data situation could happen locally at each window as well, where only a limited number of classes are available at that window.

Classes (re)appearance: Some classes may continuously drop in and out of the stream, which is in contrast with the regular presence of other classes.

Generally speaking, the proposed framework should tackle the mentioned challenges as a verified streaming classifier.

1.3.2 Network Behaviour Detection

The applicability of the proposed framework under real-world scenarios is evaluated in network applications. The goal is to benchmark the performance of the system in the detection of network behaviours on real-world network streaming traffic. Network traffic corresponds to the definition of streaming data and its special constraints.

One of the most important fields in detecting network behaviours relates to network security investigations, in which finding malicious behaviours is essential. Detecting the malicious behaviours in network traffic is like finding a needle in haystack. The reasons why detecting behaviours in network (security) traffic is a perfect match for consideration as a difficult real-world streaming processing application are as follows.

Non-stationary environment: The network traffic undergoes many changes throughout the stream. These changes are due to a combination of user and application influences on the content of the stream. Every time a new user is added to or leaves the network, it will change the behaviour of the network traffic. Not only does adding a user cause a change, but also any user's behaviour also changes throughout the time, and their preferences change as well. The application's effect on the traffic can be based on introducing new software, updating currently available applications and the co-existence of different versions of applications. The interference of these changes ultimately causes the nature of network traffic to be continuously dynamic. In the special case of network security, the same formula exists with regard to attack types. For example, in Botnet detection, not only are there different botnet architectures, but stronger bots are also introduced to the network time. This evolving botnet trend introduces complex behaviours that put at risk the reliability or, in an even broader perspective, the social behaviour of the network.

Other streaming constraints: The tremendous volume of network traffic makes its processing a challenge. The network hardware becomes more powerful, which makes it necessary to process and route traffic at a high speed. Providing labels for all the content of the traffic is practically impossible. Moreover, the distribution of classes is not likely to be balanced. In network security in particular, most of the traffic is Normal behaviour, which, at times, becomes mixed with malicious behaviour. The malicious behaviour is not only rare in the context of the class distribution, but it also happens in varying periods of time, so it drops in and out of the stream repeatedly or just once.

Encryption: The other factor in network traffic that makes the detection of behaviours difficult is use of encryption techniques to hide the content of the packet. The packet payload is not useful in such a case, and only the header information is a reliable source.

Considering the described circumstances, the goal of this thesis is to provide a framework that can detect multiple network behaviours under the a highly imbalanced stream of network traffic. The primary target is to identify very rare classes whose miss-classification is costly. In network security, the minor classes are related to high-risk malicious activities. Failing to detect these classes leads to destructive effects that may not be possible to recover from completely. The network flow information is used to address the encrypted payload without causing privacy-related concerns for the traffic under analysis.

1.4 Contributions

The main contribution of this thesis is the provision of a team-based GP streaming classifier to handle the streaming constraints mentioned earlier in the objective section (Section 1.3). To the best of the author's knowledge, this is the *first* time that both class imbalance and label budget issues have been considered in a streaming classifier for network security behaviour analysis. The thesis contributions are summarized as follows.

1. **Designing sampling/archiving policies:** The proposed streaming classifier, Stream-GP, uses an active learning approach to decouple the stream distribution from the training data distribution. To make a more balanced data subset for training the classifier, intelligent policies are needed to sample/archive the exemplars to/from the data subset. In this research, sampling/archiving policies are introduced to make the training subset more balanced under the limitation of a label budget [68, 69]. The combination of these policies leads to algorithms with interesting behaviours that could be used in different applications.

2. **Application of the framework under real-world network traffic:** Network traffic has been explained to demonstrate the streaming data challenges as well. Despite this fact, few works have applied streaming learning in the network domain (reviewed in Chapter 2). Even these research studies have also

lacked the advantage of working in a non-stationary imbalanced environment with a label budget. Thus, this thesis provides the opportunity to introduce a framework to deal with the dynamics of network behaviour classification. To this end, the framework was first benchmarked for its effectiveness in working on specific streaming datasets that aim to reveal the strength of the proposed streaming classifier under general streaming classification challenges [68, 69]. The success on the general streaming classification problem led to the main focus of this thesis, in which the classifier is applied to network streaming applications [71, 70]. Several empirical evaluations are done on different network security datasets and compared to the streaming algorithms. A specific metric is used for the classifier's evaluation which works under class imbalance whereas other research studies used accuracy as their evaluation metric.

3. **Investigating the learner behaviour on re-ocurring patterns:** A challenging mixed botnet dataset is introduced that consists of several variations of botnet behaviours [71]. It is designed to create gaps within the same botnet traffic. The ability of the proposed classifier to remember previously learned behaviours is investigated.

4. **Comparison against a state-of-the-art network streaming tool:** A thorough comparison of Stream-GP with Apache Spark Streaming as a current state-of-the-art streaming toolset in network application is done. The evaluation is performed on the challenging mixed multi-bot dataset mentioned earlier. This comparison demonstrates the gaps a network streaming tool is facing and suggests the Stream-GP classifier as a solution.

5. **Demonstarte the ability to work on other types of data:** The effectiveness of the Stream-GP classifier is investigated under other network applications. Firstly, an empirical study was conducted in which the applicability of Stream-GP as an streaming classifier to identify insider threat detection is demonstrated [78]. The dataset features in this case are not traffic features but the user behaviours logged throughout the time. The comparison to the classical off-line GP approaches investigates the possibility of using Stream-GP in a non-stationary environment. In another study, Stream-GP is used to reveal

the unknown behaviours in the context of the Background traffic in the mixed multi-bot dataset [72].

1.5 Thesis Outline

The thesis is organized as follows.

Chapter 2 reviews studies on streaming processing. Existing works are investigated from three aspects: Non-evolutionary, Evolutionary and Network methods.

Chapter 3 describes the proposed GP framework (Stream-GP) for handling the streaming classification under label budget and class imbalance constraints. An active learning method is designed for this purpose, and its corresponding sampling and archiving policies are explained. The process of choosing a GP champion in each window location to label the rest of the stream is described. Ultimately, the specific GP framework (SBB) in general is explained.

Chapter 4 contains the necessary definition and parameters settings for the evaluation of the proposed framework (Stream-GP). The datasets and comparator algorithms that have been utilized are explained. In addition, the parameter setting of Stream-GP and its comparator algorithms are also described in this section. Moreover, the multi-class metric and statistical methods used to calculate the performance of the algorithms and their significance are also described.

Chapter 5 documents the empirical evaluations of the Stream-GP framework and its comparison to the other available streaming systems. The evaluations are done under challenging scenarios targeting network security.

Finally, Chapter 6 concludes the thesis, summarizes the research contributions, and discusses future work.

Chapter 2

Related Work

Several recent surveys have overviewed model building for the classification of non-stationary streaming data [37, 59, 76]. In this section, highlights of the streaming classification issues for streaming classification under imbalanced data streams are reviewed. Section 2.1 discusses the methods under imbalanced data, change detection and (online) active learning issues, whereas Section 2.2 overviews the same in evolutionary methodologies. Section 2.3 investigates the current solutions used in a variety of network applications. Finally, Section 2.4 summarizes related works in streaming processing.

2.1 Non-evolutionary Methods

Change detection is a mechanism used to trigger the necessity of model reconstruction in streaming applications. In this category, after a change is detected, the model is subject to be updated or replaced. In the latter case, an outdated model is replaced with a new model reflecting the observed change. Also, the model building is decoupled from the need to provide label information. For instance, Lindstrom *et al.* define a process in which a reference distribution is constructed and used to calibrate the model [81]. As the stream passes, a divergence measure (expressing the model confidence independent from the label information) is calculated to trigger the model reconstruction. When the trigger is activated, model is rebuilt based on the latest window content. The model is evaluated on two spam datasets collected from [31] as a natural concept drift source. Three text datasets are also utilized both in imbalanced and balanced modes, *Reuters*[1], *20 Newsgroups*[2] and their custom dataset called *News Sources*. In this study, labels are only requested in the case of change occurrence.

[1]`http://www.daviddlewis.com/resources/testcollections/reuters21578`
[2]`http://people.csail.mit.edu/jrennie/20Newsgroups`

However, they assume that change happens with regard to the unconditional distribution of data $p(\vec{x})$. Therefore, any variation to the posterior distributions of the data $p(y|\vec{x})$ remains undetected [106]. Wozniak handles the changes in the stream by changing the line-up of the classifier ensemble and assigning weights to the base classifiers [117]. There is no explicit change detector in his method. For each chunk of data, a new classifier is trained, and then the system selects the most valuable ensemble to continue with. The update of the ensemble happens in a mixture of two ways: 1) Dynamic combiners: individual classifiers are trained in advance and will not change. The combination of rule parameters changes. 2) Changing the ensemble line-up: Outdated classifiers are replaced with new ones trained on the most recent data. They used MOA platform [14] to implement their own software. Three benchmarking data streams are used to evaluate the system under three main drift types in the stream: HYP (incremental), LED (gradual) and SEA (sudden). Hammer *et al.* use the Anti-Baysian technique to classify the dynamic data streams [55]. In this method, quantile tracking is done instead of the mean value. This is shown to be more robust in the case of outliers in comparison to the Baysian models. Synthetic data are used to showcase their method in different outlier happenings. Gautam *et al.* propose a kernel-based one-class extreme learning machine (ELM) classifier for the detection of outliers [51]. They compared their system on both stationary and non-stationary data with batch learning-based and online one-class classifiers respectively. Two synthetic and six real-time datasets are used for the stationary environment and 16 synthetic and four real-time datasets are used for the non-stationary environment.

Active learning implies that only a fraction of data is requested for their true labels. Therefore, a change mechanism or uncertainty threshold is employed to initiate label requests. Several authors have proposed bias/variance minimization schemes for this purpose [73, 86, 101, 116, 124]. Empirical benchmarking has demonstrated that uniform sampling (under label budget) could effectively work under the data stream in case the data is well mixed [106, 124]. Žliobaitė *et al.* introduced a combined sampling policy where model uncertainty and uniform sampling are used to cover both $p(\vec{x})$ and $p(y|\vec{x})$ changes under a label budget. This generic mechanism is suitable to be applied to the modified streaming version of the Hoeffding Tree and Naive Bayes classification models. The performance of the models is demonstrated in three real data sets: *Electricity* [56], *Cover Type* [47] and *Airlines*, which they created based on the inspiration of [63]. Moreover, they extended the evaluations based on three text data sets: IMDB-E and IMDB-D, which are constructed based on IMDB, which originates from the MEKA repository[3] and *Reuters*. The proposed streaming algorithms are gathered in a platform called MOA for the researchers to work with [14]. Ksieniewicz *et al.* introduced active learning to the neural networks [77]. They used catastrophic forgetting in a neural network to smoothly adapt to the changes in the stream. Their system is compared to both semi-supervised and fully-supervised methods. The MOA platform is used for the implementation of their system, in which they utilized an MLP, Multi-layer Perceptron classifier in a scikit-learn library, where they optimized the log-loss function. Three separate label budgets were used: 30%, 50% and 70%.

Several studies focus on the challenge of imbalanced data in streaming applications. Some consider a bagging solution with under or oversampling of the data to create the 'Data Subset', and the model is built upon that [36, 112]. The data subset decouples the distribution of data from the direct stream distribution. In this category, Ditzler *et al.* emphasize incremental (i.e. batch) updating while providing anytime labelling, whereas the work by Wang *et al* emphasizes online (i.e. exemplar-wise) updating. Although Ditzler *et al.* assumed SMOTE algorithm working on the stationary data with class imbalance [24], this is not the most effective solution under non-stationary data [36]. They use their own prepared datasets to showcase their methods under specific conditions. Wang *et al.* produce 12 two-class data sets with different distributions and class imbalance rates. Ditzler *et al* use four synthetic data sets (*drifting Gaussians*, *rotating checker board*, *shifting hyperplane*, and *rotating spirals*) and two real-world data sets: *Electricity* and new *weather prediction*. The second category of solution for data imbalance constraints is to dynamically reweigh the class cost[89, 52]. This solution has been utilized in active learning algorithms as well [15]. More recently, some research studies have concentrated on scenarios in which classes drop in and out of the stream against a background pattern of classes on a continuous bias [102]. However, in these scenarios, all the class labels are available; also, the overall number of classes is also known (i.e. attack or normal) but the

[3]Available at `http://meka.sourceforge.net/`.

information on when they appear in the stream is unknown.

Several semi-supervised methods are proposed for the streaming data context; and provide a framework for working with labelled and unlabelled data [42, 61, 104]; i.e. the algorithm starts with an initial labelled data stream to build a primary model and then changes to an 'online' mode for labelling the data in an unsupervised methodology. Specific points of interest include operation under class imbalance and non-stationary data. However, the utilization of the unsupervised methodology makes this solution susceptible to *adverserial attacks* [7, 6]. Moreover, a claim was recently made that the Naive Bayes algorithm has privacy leakages [120]. It was stated that an attacker can infer information, so there is leakage of sensitive information in this case. Adding a little noise has been introduced as a simple solution for this problem.

2.2 Evolutionary Methods

Model building for streaming tasks has not been extensively investigated using evolutionary algorithms as a non-evolutionary counterpart. Some GP works have been done on tasks such as financial forecasting [66], which is considered a kind of streaming classification. The goal of this task is to predict the direction of movement in the next temporal time. However, these tasks are not targeted to be limited by the label budget, and the label information is available after a time delay. Folino and Papuzzo propose a distributed GP ensemble to use to cope with changes in non-stationary data stream classification [45]. A change detection mechanism is used based on the statistical differences between two consecutive windows. The proposed model could only detect changes in $p(\vec{x})$ but not in $p(y|\vec{x})$. Furthermore, another parallel framework was necessary to rebuild the GP ensemble. For the verification of their system, they modified the dataset generator[4] and simulated data sets that consisted of the circle function, the sine function, the SEA moving hyperplane [100] and the STAGGER boolean concept [98]. Evaluations were performed on the datasets they generated. Dempsey *et al.* investigate the role of genotype-to-phenotype mapping under dynamic environments (stock market data in particular) [32]. Specifically, the emphasis is more on the significance of evolvability/plasticity in facilitating adaptation under non-stationary data.

[4]`www.cs.bham.ac.uk/~flm/opensource/DriftGenerator.zip`

Earlier works using the framework drawn on in this thesis assumed that GP teams are evolved from: 1) a Data Subset that is available to decouple the model training from the cardinality of the stream, 2) a Sampling policy that specifies which exemplars are to be selected from the stream and placed in the data subset, and 3) an Archiving policy to determine which exemplars from the data subset are to be replaced. The initial attempt to utilize Pareto archiving as the archiving policy demonstrates that label error (a form of noise disrupting the ability to accurately model $p(y|\vec{x})$) has a significant negative impact on building robust models [4]. Adding a uniform sampling as the sampling policy to the framework under a label budget introduced a more robust starting point [108], though full label information was necessary to keep the data subset balanced. Moreover, it has been shown that a GP teaming solution is more effective in detecting changes than using the monolithic GP individual as a solution [107]. Ultimately, the issue of class imbalance in the data subset was noted to have an impact on the quality of the GP model in the preliminary steps of this thesis [68, 69]. The earlier works were evaluated on the artificial datasets, which were explicitly designed to benchmark the efficiency of the models under various non-stationary streaming data. In this thesis, the GP classification is focused on a specific cybersecurity problem, namely botnet detection. This is a highly challenging problem given the streaming network traffic data. Such data are very imbalanced data because botnet traffic constitutes a low percentage of the network traffic, in fact botnet command and control (C&C) traffic is usually less than half a percent of the traffic seen. Moreover, different types of applications and attacks (classes) appear and disappear. This results in a significant cost for miss-classification, and low label budgets. The GP framework is considered for this purpose as it is able to solve the complex problem by decomposing it to smaller easier problems and solve them. The programs in GP are capable of representing complicated, non-linear functions. Moreover, GP has the ability for implicit feature selection based on the data. Thus, it does not require other approaches for feature selection throughout the course of the stream.

Several earlier works were based on prototype-style representations, such as learning classifier systems (LCSs). Dam *et al.* focus their research on the reaction time of the LCS on the ‘multiplexer’ problem with modifications on $p(\vec{x})$. They proposed that population reinitialization is the best strategy when a change is detected [29]. Also, The real-6-multiplexer problem with/without noise is utilized to showcase their system. Behdad and French investigate an approach on an LCS in which the exploit and explore phases are reversed in comparison to the traditional off-line batch learning [8]. A random subset of a *corpus* dataset [87] is used that contains email messages of senior managers of the company. Also, there is an approach that utilizes k-NN in an evolutionary mode using swarm optimizatioich potentially has been applied to the data stream classification tasks [23]. Two artificial data sets(*SEA* and *Checkerboard* [43]) and two real data sets (*Electricity* and *weather prediction* [43]) are employed for evaluation. All the methods in this category fulfill just a number of the constraints that have been covered in this thesis.

2.3 Network Methods

Streaming applications in the network area are gaining more interest as streaming data constraints are applied more precisely to network traffic. However, this is a new topic for network managers, and few works are available in this area. Network streaming applications are widely categorized in four main groups, as shown in Figure 2.1. All categories may utilize machine learning algorithms to some extent, but the red-coloured area is where it is mostly used. The classical network style, the blue-coloured categories, is mostly done by performing aggregations on traffic or applying SQL-based queries on passing traffic where no learning is demanded.

Data mining in general is divided into batch or stream processing and can be done on a centralized or a distributed architecture. Figure 2.2 illustrates the hierarchy and available tools in each category. This is a general overview, and the tools might be more or less utilized in network applications.

Several research studies are available on network security that are based on centralized processing architecture. Masud *et al.* propose a multi-partition multi-chunk ensemble method to detect botnets in the network traffic stream [85]. They demonstrate that their system is capable of detecting drift changes, $p(\vec{x})$, where the hypothesis is that labels are available and the class distribution is consistent throughout the course of the stream. They create a dataset, based on a hyperplane simulator with the same parameters as [111], to illustrate the effectiveness of their model in change detection. Also, they generate a botnet dataset based on Nugache bot to showcase their system under the botnet detection area.

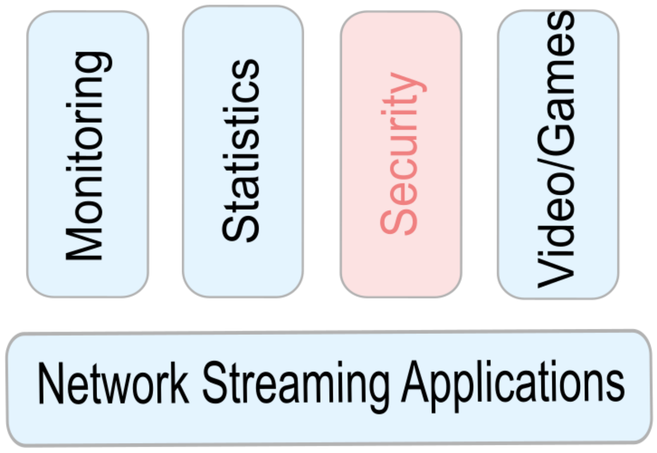


Figure 2.1: Network streaming applications

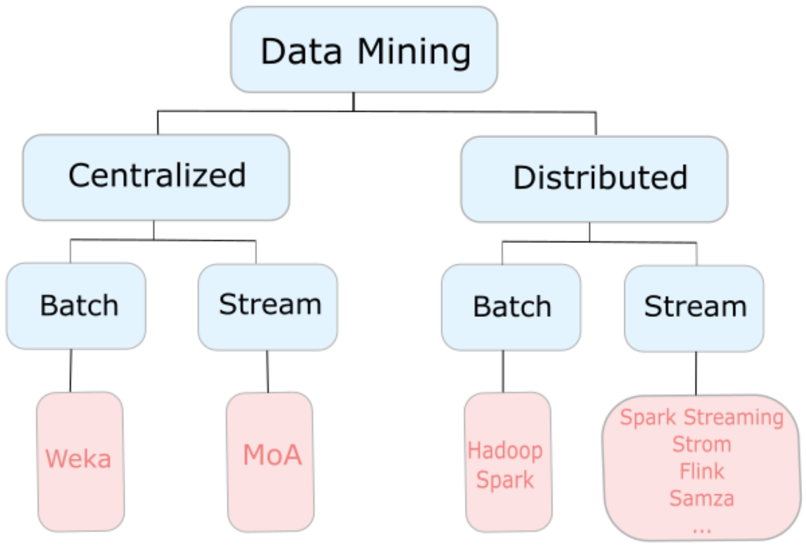


Figure 2.2: Data mining categories

Garg *et al.* propose an on-line peer-to-peer Botnet detection system that relies on failure and communication traffic [50]. In their proposed system, network traffic is divided into small time-based chunks where decisions are made based upon each chunk separately. The benchmarking is done on a variety of botnets and popular P2P applications. They conclude that only the first phase of the botnet traffic could be detected based on the predefined patterns. Moreover, their methodology does not adapt to the changes in the network traffic.

Baer *et al.* introduce a streaming warehouse, DBStream, that is tailored for Network Traffic Monitoring and Analysis (NTMA) applications [5]. The data are processed in the order of sliding windows where SQL queries are defined to be applied to the content of each window. These query jobs are controlled by a scheduler. In addition, a classification plugin is available that connects the DBStream to the WEKA toolkit[5]. Using this module, they classify Machine-to-Machine (M2M) traffic in a cellular network. They compare DBStream to Apache Spark in this specific scenario. Their system is a data stream management system, but the provided ML is not streaming itself.
Le *et al.* demonstrate the application of evolutionary solutions to *Insider threat detection* under stationary and non-stationary data assumptions [78]. In the non-stationary data assumption, the framework of this thesis is utilized.
Some works focus on benchmarking the streaming classification ML algorithms provided by the MOA open-source toolset. Desale *et al.* benchmark four streaming classification algorithms in intrusion detection on the NSL-KDD dataset [34]. They end up with Naive Bayes and Hoeffding Tree as the best-performing algorithms. Morgan *et al.* [90], likewise, benchmarked MOA classification algorithms' performances on publicly available network botnet datasets. The algorithms are evaluated against four sets of data: *KDD Cup'99*, two sets of *NIMS* collections[6] and *ISOT*[7].

Several works benchmark machine learning classification algorithms in a distributed processing architecture for Intrusion Detection purposes. Gupta *et al.* use the Apache Spark toolset in a batch mode to benchmark several classification algorithms in the MLlib library of Apache Spark[8] on detecting intrusions in network

[5]`https://www.cs.waikato.ac.nz/ml/weka/`
[6]`https://projects.cs.dal.ca/projectx/data/NIMS.arff.zip`
[7]`http://www.isot.ece.uvic.ca/dataset/isot_botnet.php`
[8]`http://spark.apache.org/`

traffic [54]. The *KDD Cup'99*[9] and *NSL-KDD* datasets are used for their evaluations. Belouch *et al.* apply the similar benchmarking of five algorithms in Apache Spark on the *UNSW-NB15* dataset [10]. Vimalkumar *et al.* use Apache Spark for detecting intrusions in smart grids [109]. Manzoor *et al.* followed the same approach as [10] but with a different tool, Apache Storm, in a batch learning mode [84]. The above studies are based on a batch learning process in which the model is then applied to a huge volume of data for classification.

Mylavarapu *et al.* use Apache Storm as the streaming platform to detect attacks in network traffic [91]. Corner Classification (CC4) and Multi Layer Perceptron (MLP) algorithms are used to detect unknown and known attacks, respectively. The models are trained prior to the utilization by the Storm streaming toolset. Wang *et al.* use Apache Storm to implement their own methodologies on the streaming architecture [113]. They target two types of intrusions: 1) a small number of large network flows, and 2) a large number of small network flows. Three methods are proposed, and their results are aggregated to get a final decision. Both these methods apply an ML model on streaming data for classification. The former uses a trained model for this purpose whereas the latter approach utilizes streaming learning methodologies as well, but it is not open-source.

There are several tools that provide opportunities to process data in a streaming mode in which network managers base their utilization and implementation on them. A review of the tools in distributed streaming processing based on Map-Reduce (introduced by Google [62]) is found in [9]. Based on available platforms that provide the opportunity to do stream processing, the work of Žliobaitė *et al.* [106], available in the MOA toolset, is extended to be embedded to the distributed streaming architectures that are commonly used as network tools to provide streaming ML algorithms on network traffic. StormMOA is a project that combines MOA with Apache Storm to provide a scalable implementation of streaming ML frameworks[10]. Apache SAMOA is an open-source framework that provides mining of big data streams [12, 74]. It can be run on top of Apache Flink/Storm/Samza tools. There is also a demonstration of StreamDM which provides streaming algorithms on top of Apache Spark Streaming

[9]`https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`

[10]StormMOA: `https://github.com/vpa1977/stormmoa`

[11]. All these implementations were originally based on MOA ML algorithms, which the proposed algorithms in this thesis are evaluated against and compared to.

Moreover, Casas *et al.* introduce a big data analytics framework named BigDAMA for network monitoring applications [21]. Their system uses Apache Spark Streaming for stream-based analysis, Apache Spark for batch analysis and Apache Cassandra for query and storage. They test their system to utilize five models to detect five different attack types on MAWILAB network datasets [46]. They build a binary-based detection on each of the attacks in a 10-fold cross validation. The ten top features from the feature correlation graph are extracted and used out of all 245 features available. Cermak *et al.* proposed a pattern-matching framework based on Apache Spark Streaming to detect SSH authentication attacks [22]. Their framework keeps known patterns for the specific SSH attacks and processes flows of data to find these malicious patterns. They broaden their matching scheme by employing a similarity-based approach to detect new attacks as well. No learning process is involved in this approach, and fixed patterns are defined prior to the stream processing.

Some semi-supervised solutions are also provided for network streaming data. Raja *et al.* use an ensemble of similarity-based and online GA classifiers using a sliding window over the network packet stream [95]. They apply their proposed method to detect intrusions in KDD Cup'99. Chen *et al.* utilize the Flume and Spark streaming framework to perform network monitoring and intrusion detection [25]. They apply a combination of streaming K-means and fuzzy C-means to detect abnormal activities. As mentioned earlier, these solutions are in danger of being misled by adversarial attacks.

Specific use-cases of streaming processing are available that specifically operate on graph algorithms [65, 26]. Example applications of this area include social networks, sensor networks and traffic flow networks where the relationships in the data generated are represented as graphs. GraphX [118] and Giraph [27] are Graph APIs that provide the opportunity to work on graph stream applications in Spark Streaming/Flink and Storm, respectively. Video streaming classification topic, is another available topic in the context of streaming classification. This relates to algorithms that stream the content of a video online [39, 40, 114, 92]. The focus of this thesis is far from these two specific areas.

Tables 2.1 and 2.2 represent the overall review of the existing works on streaming processing from a network perspective. This table demonstrates that there is still a gap in the network area to actually apply streaming ML classification algorithms that could encounter the constraints associated with streaming data. Either they use the streaming tools with the idea of stationary data in which an off-line model can be applied throughout the whole stream, or there is no learning included, and SQL-based queries or statistical operations are performed on the stream of data for network monitoring and analytical purposes.

Table 2.1: Existing network related works (architecture and learning process), Arch.: Architecture, Data P.: Data Processing, MC: Multi-Class, CD: Change Detection, DI: Data Imbalance, LB: Label Budget

Ref.	ML Type	Arch.	Data P.	Learning	Learning properties			
					MC	CD	DI	LB
[85]	Classification	Centralized	Stream	Stream	×	✓	×	×
[50]	Classification	Distributed	Stream	N/A	×	×	×	×
[5]	Classification	Centralized	Stream	Batch	×	×	×	×
[78]	Classification	Centralized	Stream	Stream	✓	✓	✓	✓
[34]	Classification	Centralized	Stream	Stream	×	✓	×	✓
[90]	Classification	Centralized	Stream	Stream	✓	✓	×	✓
[54]	Classification	Distributed	Stream	Batch	×	×	×	×
[10]	Classification	Distributed	Stream	Batch	×	×	×	×
[109]	Classification	Distributed	Stream	Batch	✓	×	×	×
[84]	Classification	Distributed	Stream	Batch	×	×	×	×
[91]	Classification	Distributed	Stream	Stream	×	✓	×	×
[113]	Classification	Distributed	Stream	Stream		✓	×	×
[21]	Classification	Distributed	Stream	Batch	×	×	×	×
[22]	Classification	Distributed	Stream	N/A	×	×	×	×
[95]	Clustering	Centralized	Stream	Stream	✓	✓	×	×
[25]	Clustering	Centralized	Stream	Stream	×	✓	×	×

Table 2.2: Existing network related works (application and metric), Sec.:Security, Mon.:Monitoring, BD,ID,ITD:Botnet,Intrusion,Insider Threat Detection

Ref.	Application	Open Source/Tool	Dataset	Metric
[85]	Sec. (BD)	×/(N/A)	P2P Botnet (private)	
[50]	Sec. (BD)	×/Hadoop	P2P Botnet (private)	
[5]	Mon.	✓/DBStream	METAWIN	
[78]	Sec. (ITD)	×	CERT	
[34]	Sec. (ID)	✓/MOA	NSL-KDD	Acc,Kappa,Time
[90]	Sec. (BD)	✓/MOA	KDD Cup'99/NIMS/ISOT	Acc,Dr
[54]	Sec. (ID)	✓ (Spark)	KDD Cup'99/NSL-KDD	Acc,Dr,Specificity,Time
[10]	Sec. (ID)	✓ (Spark)	UNSW-NB15	Acc,Dr,Specificity,Time
[109]	Sec. (ID)	✓ (Spark)	synchrophasor (private)	Acc,FR,Dr,Specificity,Time
[84]	Sec. (ID)	✓ (Storm)	KDD Cup'99	Confusion matrix
[91]	Sec. (ID)	× (Storm)	ISCX 2012	FP,FN
[113]	Sec. (ID)	× (Storm)	CERNET2+anomaly	mean,std
[21]	Sec. (ID)	✓ (Spark Streaming)	MAWILAB	ROC,Time
[22]	Sec. (ID)	✓ (Spark Streaming)	Stream4Flow (public)	Acc,Prec,Recall
[95]	Sec. (ID)	×	KDD Cup'99	TPR,FPR,Prec,Recall,F-measure
[25]	Sec. (ID)	×	KDD Cup'99	ROC,Time

2.4 Summary

The related works on building models for streaming data classification and the role of streaming processing in the network field are reviewed in this chapter. The review is classified into three main types of solutions: 1) Non-evolutionary algorithms, 2) Evolutionary solutions, and 3) Network applications. The summary of the investigations comes next.

Non-evolutionary streaming classification solutions are reviewed based on the main characteristics of streaming processing challenges: 1) Change detection, 2) Label budget, 3) Class imbalance, and 4) Others. The first category investigates the solutions provided to detect the changes throughout the stream to be able to update or replace the model. A variety of changes in data or their associated labels could occur that some research studies have deficiencies in detecting. This is a crucial part of streaming processing in non-stationary areas, specially in the network field.
Secondly, active learning methods are introduced in which implicitly only a fraction of the data is queried for their true labels. These algorithms decouple the stream complexity from the model evaluations by keeping a subset of data from the stream. They repeatedly update this subset with the content from the stream. The data are considered uniformly balanced in these solutions.
The third category in this group of algorithms focuses on dealing with imbalanced data in the stream. Under or oversampling solutions are part of these solutions. The research studies in this category consider that data labels are available for all data throughout the stream. In reality, this is not feasible, as the labelling process is costly.
Last, another challenging characteristic of the streaming classification is dropping in and out of the stream. The access for data labels are for-granted in these solutions.
Semi-supervised solutions are also provided that target the class imbalance and non-stationary data, but they are shown to be susceptible to adversarial attacks.
All solutions in the non-evolutionary algorithms provide the requirement of streaming processing to some extent, but no work is available that gathers them all in one place.

Evolutionary methods are not as extensive as non-evolutionary methods in this area. Some works are utilized in areas such as financial and forecasting fields where the label information is available after a period of time and the labels are fully available after a delay. The earlier GP works that this thesis is based on, started with change detection in streaming scenarios and reached a point of applying a label budget on top. In these previous solutions, class imbalance is not considered a factor. Moreover, there are approaches such as LCS and swarm intelligent systems that consider some stream challenges.

Network-oriented solutions regarding streaming data are then introduced extensively. Research studies in this category are mostly focused on the currently used and applied systems on the network traffic in a streaming mode. The network streaming applications are divided into four main categories: 1) Monitoring, 2) Statistics, 3) Security, and 4) Video/Games. The streaming tools are also categorized in two main centralized and distributed groups, where each could also be classified in a Batch and Stream mode. The investigation on network streaming tools reveals that they mainly provide a framework for streaming processing of network traffic, but the models are fixed and pre-trained before being applied on the stream. These solutions are only applicable in stationary environments (usch as Monitoring, Statistics and Video/Games), but in a non-stationary stream (Security), they are not applicable because they cannot update its model based on the new changes. Few works have applied stream learning models on streaming data, but they are not available as open-source solutions. Therefore, the network solutions are either not using streaming learning models or not available in open source solutions.

In summary, general streaming solutions focus intensely on the streaming processing of data and building streaming models. However, they do not consider all streaming classification constraints in one place. In addition, evolutionary solutions are not extensively researched in this regard. On the other hand, network streaming platforms provided for streaming processing are mostly focused on speeding up the process of predicting in real-time without considering the explicit challenges associated with streaming classification, e.g. change detection, class imbalance, label budget, etc. These shortcomings necessitates a framework as an integrated solution to provide the main streaming classification constraints, which are pre-requisite of network streaming processing.

In the next chapter, the proposed GP active learning framework is explained to provide a desirable system. This solution specifically considers non-stationary data, class imbalance, and label budgets in comparison with other requirements for stream classification such as any-time operation, one-time pass, partial observability and classes (re)appearance. It also prevents the adversarial attacks by keeping the human expert in the loop for the learning process.

Chapter 3

Framework for Streaming GP Teams

3.1 Streaming Data Environment Under a Label Budget

Streaming algorithms are either categorized as online [88] or incremental [93]. Online algorithms perform on the the content of the *sliding* window of sequential instances in an exemplar-wise manner.[1] On the other hand, incremental learning will perform on the 'chunks' of data from the *non-overlapping* window where the latest content of the stream is available. All the queries and processes take place in that specific window location (sliding or non-overlapping). These criteria will ensure that the implicit memory constraint of streaming processing is fulfilled. In this thesis, the incremental non-overlapping approach is utilized (Figure 3.1). The incremental approach is used to require low label budgets. The label budget specifies the window size. The lower the label budget is, the larger the size of the window would be. For an exemplar-wise algorithm, the window size needs to be one, so it is useful when the ground truth is available for all data, which is not the case in practice most of the time.

The stream is defined as a d-dimensional continual sequence of records $..., x(t), x(t+1), ...$ where t represents the temporal index. Streaming data implies that $t \rightarrow \infty$. Each exemplar in the stream has a (true) label, $d(t)$, which is *not* available until explicitly requested. The label is requested under two conditions: 1) Only from the current window, $W(i)$ and 2) Based on the label budget criteria, β. The $\beta = 0.5$ label budget declares that *half* of exemplars (50%) in the stream could be asked for their true labels.

The operation under label budget criteria requires a **sampling policy**, S, to be available that identifies which exemplars are to be selected for true label inquiries. This policy only works on the current window location, $W(i)$, and an exemplar could not be revisited once a decision has been made. However, **anytime** operation ensures

[1]As in FIFO data structure, 'First-in-First-out' queuing, where one exemplar enters the queue from one side pushes the oldest exemplar out from the other side.

that for each exemplar, $x(t)$, in the current window, a label prediction, $y(t)$, is made in (real-time) by the streaming classifier. A population-based paradigm is used (GP in this research), which implies that a *champion classifier* (i.e. an individual from the population) has to be available to predict the labels *before* they appear in the window. Figure 3.1 illustrates the relation of these concepts in this work.

The above formulation of the streaming classification task implies that:

- All of the data in the stream have their labels predicted firstly by the champion classifier. Any changes to the GP population takes place only after the champion classifier suggests its labels, *y(t)*;
- The training phase is an interactive process in which the stream-GP framework decides which exemplar's true label is requested under the label budget, β.
- Records arrive based on the properties of the underlying process, so class balance is not likely met within the local region of the stream.
- The champion is changed by the stream-GP framework but once the change takes place, the new champion does not revisit the previous labelling decision(s).

3.2 Overall Framework

The overall framework is assumed to be what is depicted in Figure 3.1. The **Sampling policy** specifies a *Gap* number of records to be asked for their true labels, $d(t)$, under label budget constraint (β). After their true labels are identified, they are replaced with some records in the Data subset ($DS(i)$). This is the **Archiving policy**'s duty to mark a fixed number of records in the Data Subset to be replaced by the new labelled records. The Data Subset provides the opportunity to introduce biases to the representation of classes.

An *'incremental'* learning process is assumed in which the Data Subset decouples the fitness evaluation from the cardinality of the stream. This implies that only after the 'Gap' replacement in the Data Subset can the fitness evaluation be performed. Thus, $DS(i)$ denotes the point where the replacement happens and an updated Data Subset is available. A τ number of GP generations are performed and a new champion individual is selected for future predictions. Streaming operations begin with a cold start that is necessary to identify the first champion individual. Note that the rate of champion identification does not exceed the DS updates, but it is not necessarily the same.

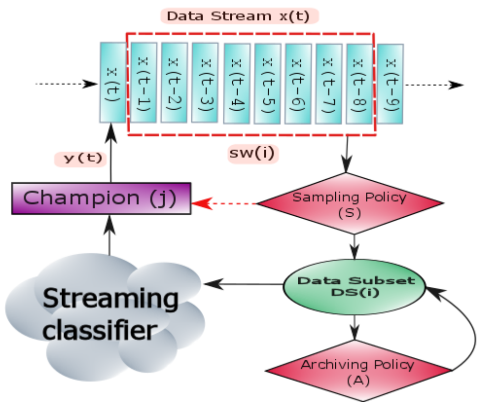


Figure 3.1: Overall Stream-GP framework. Sampling policy, (S) determines which exemplars will be quarried for their true labels (under label budget constraint β). Archiving policy, (A) keeps a finite size Data Subset, (DS), where 'Gap' records are subject to replacement. On the update of DS with $Gap(i)$ labelled records, τ generations of GP are performed. At anytime, a single champion individual is available to predict the labels, $y(t)$, which may influence the Sampling policy operation.

A Simbiotic Bid-Based (SBB) formulation is assumed for defining the solution in the form of GP teams [79]. Such a framework co-evolves GP individuals through a bidding process that identifies the context for an action, class label in this study. Each program is associated with an action that is assigned randomly at the initialization step. Teams and programs are separate independent populations. There are only two constraints available in building a team: 1) At least two programs should be members of a team, and 2) At least two different actions should be available among the same team members (programs). The SBB framework can be used for multi-class classification without any modification.

Previous works have demonstrated that the SBB framework is more effective than monolithic (canonical) GP solutions on off-line classification tasks [80] and streaming classification [107]. The focus of this thesis is on defining Sampling and Archiving policies to make the framework suitable to be applied in network applications, specifically the network security field. The proposed sampling and archiving policies are independent of the underlying GP framework. A detailed description of SBB architecture and how it works is found in Section 3.6.

Sections 3.3 and 3.4 explore the definitions of the proposed Sampling and Archiving policies. One main characteristic of network security traffic, as our target application, is its very imbalanced nature. The attack percentage is highly low in comparison to the normal transactions but costly to dismiss as well. Therefore, the goal is to build classifiers that are robust to the imbalanced stream of data. This robustness could be introduced to the classifier by introducing a bias to the content of the data subset in an active learning methodology. Moreover, it is notable that there is always a trade-off between keeping the data subset content balanced and losing the sensitivity towards the most frequent records. The theme of the research is to maintain this balance as much as possible and represent various policies with regard to the evaluation of this balance. Finally, Section 3.5 explains the mechanism of champion individual identification. The champion individual is then utilized to predict label $y(t)$ for a given $x(t)$.

3.3 Sampling Policy

The sampling policy determines which exemplars from the window $(W(i))$ are to be selected under the label budget constraint to request their true labels. In this section, the proposed sampling policies used in this thesis are explained.

A **uniform sampling policy** is assumed as the baseline or control approach. This policy samples data from the (non-overlapping) window, $W(i)$, based on uniform probability before any label information is available. In other words, the record $x(t)$ is sampled uniformly with β probability from the window $W(i)$. Previous works have shown that this policy as a start point is not necessarily better when more complex algorithms are encountered [124, 106].

On the other hand, another sampling policy is introduced that takes advantage of GP champion prediction in favour of weighting the records for sampling. In this case, the GP champion individual at the time, gp^*, suggests its label for a record in the (non-overlapping) window. Once the champion individual *predicts* a record as one of the under-represented classes (based on the content of the data subset), it is prioritized by the sampling policy to be selected (subject to label budget β). This helps to promote records that *could* possibly lead the process to lean toward re-balancing the content of the data subset, $DS(i)$. As the performance of the Stream-GP classifier relies on the distribution of classes in the data subset, this sampling policy, which actively addresses the issue, is proposed without relying on the true label information. The proposed sampling policy is referred to as **biased sampling policy** hereafter.

Algorithm 1 introduces a summary of the process in a pseudo-code format. The content of the previous Data Subset, $DS(i-1)$, is required to be characterized beforehand. Let C denote the number of classes currently present in $DS(i-1)$ and $\overline{c}$ be the set of classes appearing with frequency $\geq \frac{\|DS\|}{C}$ in $DS(i-1)$, where $\|DS\|$ is the size of the Data Subset (identifies the over represented class(es)).

The content of the non-overlapping window, $W(i)$, is marked based on the predicted label, $y(t) \notin \overline{c}$ (Step 1). If less marked instances are available than the capacity of *Gap*, the true labels of the marked ones are requested and placed in the Data Subset (Step 2). The rest of the Gap capacity is filled with random sampling of non-marked records from the window, $W(i)$, if there is any (Step 2b). In case more instances are marked than the capacity of *Gap*, then the selection happens in a roulette wheel of the marked ones from $W(i)$ (Step 3). The roulette wheel samples from $W(i)$ with th frequency inversely proportional to (marked) $DS(i-1)$ class content.

The only exception is in the case of a cold start where no GP champion individual is available to suggest its predictions. In this case, the uniform sampling policy is considered for $W(i == 0)$.

3.4 Archiving Policy

In active-learning architecture (Figure 3.1), a Data Subset, (DS), is utilized with a limited capacity. The data subset keeps the training instances, the Stream-GP classifier is updated based on them. Once it gets full, a mechanism to free up *Gap* records is necessary to let the content be updated with the new records from the stream from $W(i)$. This is the Archiving Policy that determines which exemplars from the data subset are to be replaced by the newer records. Again, two policies are considered for this purpose. The **uniform archiving policy** is assumed as the base/control case where *Gap* records are identified for replacement from $DS(i)$ with uniform probability.

Algorithm 1 Biased Sampling Policy. Let $rnd(A)$ return a randomly sampled instance from set A without a replacement. $roulette(A, b)$ returns a randomly sampled instance (without a replacement) from the subset $A \wedge b$ with frequency inversely proportional to $DS(i-1)$ class content. $Gap(i)$ is the set of records transferred to $DS(i)$ (Figure 3.1) at a non-overlapping window location i.

Input: The current content of the (non-overlapping) window $W(i)$ and predicted labels, $y(t)$. The set $\overline{c}$ of over represented classes from the Data subset $DS(i-1)$;
Initial state: $Gap(i) = \emptyset$; $cnt1 = cnt2 = 0$

```
1. For all $t \in W(i)$
    (a) IF $y(t) \notin \overline{c}$ THEN $M_t = 1$ AND $cnt1 = cnt1 + 1$
        ELSE $M_t = 0$
2. IF $cnt1 \leq Gap$ THEN
    (a) For all $t$ in which $M_t == 1$
        i. Request $d(t)$
        ii. $Gap(i) = Gap(i) \cup (\vec{x}(t), d(t))$
    (b) WHILE $cnt1 < Gap$
        i. $t = rnd(W(i))$ subject to $M_t == 0$
        ii. Request $d(t)$
        iii. $Gap(i) = Gap(i) \cup (\vec{x}(t), d(t))$
        iv. $cnt1 = cnt1 + 1$
3. ELSE
    (a) For any $t$ in which $M_t == 1$
        i. $(\vec{x}(t), d(t)) \leftarrow roulette(W(i), M_t)$
        ii. Request $d(t)$
        iii. $Gap(i) = Gap(i) \cup (\vec{x}(t), d(t))$
        iv. $M_t = 0, cnt2 = cnt2 + 1$
    (b) Repeat Step 3a WHILE $cnt2 < Gap$
```

The **biased archiving policy** is defined to incrementally (re)balance the content of the data subset. Once a record is placed in the data subset, its true label is revealed. The label information is utilized to proportionally remove the over-represented records in the data subset. The detailed process is summarized in 2. Firstly, all the records in the data subset are grouped into class categories and ranked in a descending order based on their 'age' (Step 1). The 'age' of a point indicates the number of generations since it is added to the point population. The number of records, c_k, for each class in counted (Step 2a). The number of extra records that exceeds the count of a class in an ideal balanced data subset, i.e. $\frac{DS-Gap}{C}$, for each class is calculated (Step 3a). Under-represented classes are not targeted for record deletion (Step 3b), hence additional records can be accumulated. For each class, over represented records (relative to the ideal distribution) are removed from the data subset where older classes are prioritized (Step 4a). Finally, the content of $DS(i-1)$ after removal of the Gap records is added to the $DS(i)$.

3.5 Champion Classifier Identification

A champion classifier, $gp*$, is identified by applying a robust performance metric to evaluate the operation of the population based on the content of the current Data Subset, $DS(i)$. The data subset content is the only source with true labels. The performance metric in this thesis is assumed to be multi-class Detection rate (DR), or

$$DR = \frac{1}{C} \sum DR_{j=1}^{C} \text{ and } DR_j = \frac{tp_j}{tp_j + fn_j} \tag{3.1}$$

where C is the count of classes present in $DS(i)$; tp_j and fn_j are the counts of true positive and false negative for class j, again with respect to the class distribution present in $DS(i)$.

The champion classifier could be changed based on the content of the data subset (i.e. relative to the content of DS by index i). This rate never exceeds the data subset update rate, which is another reflection of incremental and online algorithms. Once the first champion classifier is identified, the anytime operation guarantees that a champion classifier is available at all t times. Therefore, the delay in introducing the new champion will not interrupt the labelling process in the stream.

Algorithm 2 Biased Archiving Policy. Let a_j be the 'age' of record j in $DS(i-1)$, where this is a scalar count for how long record j has appeared in the Data Subset. c_k is a count of the number of class k instances in $DS(i-1)$. C is the number of different classes currently represented in $DS(i-1)$. T is the total number of instances removed from the over-represented classes.

Input: Set of labelled instances $Gap(i)$ and the last available Data Subset, $DS(i-1)$
Initial state: $T = 0$

```
1. For all $j \in DS(i-1)$ identify class and rank w.r.t. record 'age' $a_j$;
2. For each class $k$ present in $DS(i-1)$
   (a) Count the number of records with class $k$ in $DS(i-1)$. Let $c_k$ be the count
       for class $k$.
3. For $k = 1$ to $C$
   (a) $remove_k = c_k - \frac{DS-Gap}{C}$
   (b) IF $remove_k > 0$ THEN $T = T + remove_k$
       ELSE $remove_k = 0$
4. For $k = 1$ to $C$
   (a) Delete the oldest $Gap \times \frac{remove_k}{T}$ records of class $k$ from $DS(i-1)$
5. $DS(i) \leftarrow Add(DS(i-1), Gap(i))$
```

3.6 Symbiotic Bid-based GP

The Symbiotic Bid-Based (SBB) GP provides a solution in a team formulation. In this formulation, the task could be broken down into smaller ones and separately solved by team members. The SBB GP consists of two separate populations, teams and programs, with a symbiotic relation between them [60]. Linear GP formulation is assumed in the program population for the sake of ease where the 'intron' codes could be identified and skipped during the fitness evaluation [16]. Each program produces a scalar value named 'bid' in this context. The bid is representative of the program performance based on the fitness evaluation. Moreover, each program is associated with an action, a, which in this case $a = 1, 2, ..., C$ is the class label and C is the number of classes. The action is initially assigned to the program randomly.

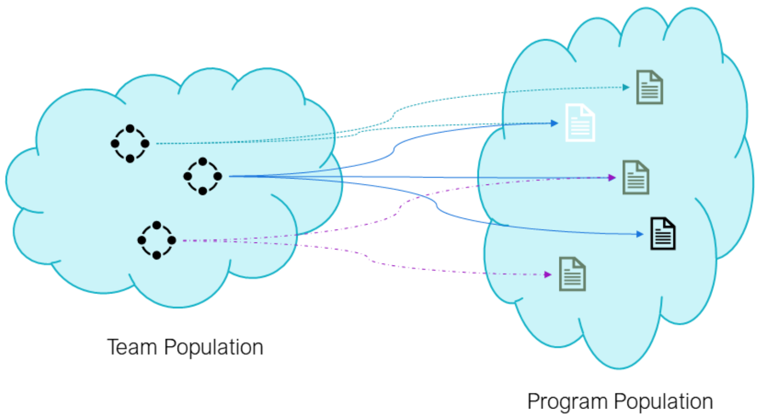


Figure 3.2: Symbiotic Bid-Based GP. Each team indexes a different combination of programs, but the same program may appear in multiple teams. The action (class label) of a program is expressed through colour.

The team population represents similar identification as of the variable length GA. Each team tm_i contains a list of indexes pointing to the programs in the program population. For a given record $\vec{x}(t)$ and a team tm_i, all the programs in the team are evaluated and their 'bid' is announced. The program with the highest bid indicates the 'winning' program in team tm_i for the record $\vec{x}(t)$. The winner program has the right to suggest its action, class label here. As the team is represented like a variable length chromosome, the team size and also the team complement evolve. This flexibility is of high importance, as no prior decision on the decomposition of the task is necessary. In a three-class classification problem, even the requirement to have at least three distinct programs introduces a poor learning bias. Instead, in SBB, each team evolves so it firstly identifies the 'easiest' classes and then adds the complex ones to its solution complements. Thus, it is common to share the same program in multiple teams. The only two constraints are: 1) each team has at least two programs, and 2) two different actions should be present in each individual team.

The fitness evaluation is calculated at the team population level. All programs across the team population are evaluated, then sorted based on the fitness value and T_{gap} programs from the bottom of the list are deleted. The same number of programs are then added to the team population from the offsprings of the surviving individuals (i.e., a breeder model of selection/replacement). Before team reproduction, the individuals from the program population are inspected to find any program(s) that do not receive any index from a team. These individuals are removed from the program population. Variation operations operate hierarchically and probabilistically delete or clone (add) programs (Table 4.10). Only cloned programs see further modifications and only these programs are added to the new teams.[2] This way, there is no disruption to the operation of the surviving programs in generations. Also, it implicates that only the team population is limited in size, explicitly limited to P_{size}, and there is no limitation on the number of individuals in the program population.

3.7 Summary

An active learning framework based on streaming GP teams is proposed. This solution is considered an incremental learner working on a non-overlapping window. limiting the access of the streaming classifier to the window location guarantees the *memory limitation* of streaming data processing. The active learning method keeps a limited capacity of stream content as a data subset. Two policies are needed to change the content of a data subset: a sampling policy and an archiving policy. The Sampling policy is based on a *label budget* constraint. These two policies could be utilized to overcome the imbalanced stream problem by introducing a more balanced data subset decoupled from the stream content. The focus of this thesis is on tuning these two policies and trying to benchmark the different combinations of them to determine how they affect dealing with the imbalanced stream of data in network applications and impact the overall performance of the Stream-GP classifier. The definition of the policies and the process of selecting a champion classifier to predict the upcoming data are discussed in this chapter. In addition, a description of the Simbiotic Bid-Based (SBB) GP is also provided to give a rough idea of the teaming GP used by the

[2]For example, having the program action changed, or instructions randomly modified.

framework in this area. The Stream-GP framework alongside the mentioned policies define a whole system that could deal with the streaming challenges in an aggregated approach.

The definition of comparator algorithms, datasets, metrics and other settings are explained in detail in the next chapter.

Chapter 4

Evaluation methodology

In this chapter, the definitions of all the parameters, datasets and algorithms used to evaluate the performance of the proposed system are given.

4.1 Datasets

In this section, the network datasets used to evaluate the streaming systems are introduced.

4.1.1 CTU-13

The CTU-13 dataset is a real-world public dataset that has been collected in the University of Czech Republic [49]. The dataset consists of four main classes: Background, Normal, Botnet and Botnet C&C. The dominant class is the Background (Table 4.2), which demonstrates that the network traffic has been captured in a real-world scenario. In the past, Normal traffic was characterized as the most difficult to express accurately, which results in using benchmark datasets where Normal detection is much easier than in real-world situations, see for example [83]. It is then difficult to label Normal and Attack behaviours because the Background traffic may actually consist of attack behaviours as well.

The solution provided by the network community in this regard is to label all the network traffic as 'Background'. Then apply specific filters that clearly identify Normal behaviour [97, 99]. Any portion of Background traffic that corresponds to the Normal filters is labelled as Normal. Then, the attacks using (Botnet) tools are injected to the (virtual) network from specific IP addresses, which explicitly identifies the attack behaviour. In the specific case of CTU-13, the Botnet attacks are labelled separately as Botnet and Botnet C&C, with the latter specifically identifying the Botnet Command and Control signals. The attack is first run using Botnet C&C signals ruled by Botnet masters to command the slaves to perform specific types

Table 4.1: Argus flow features for the CTU-13 dataset

Feature	Description
stime	Start time
ltime	End time
dur	Duration
saddr	Source IP address
sport	Source port number
proto	Protocol
dir	Direction of transaction
daddr	Destination IP address
dport	Destination port number
State	Transaction state
SToS	Source ToS byte value
DToS	Destination ToS byte value
pkts	Total transaction packet count
bytes	Total transaction bytes
srcBytes	Total source transaction bytes

of attacks on the victims' systems. Therefore, it is undeniable that the sooner the Botnet masters are detected, the lower the damage costs will be.

All Thirteen datasets in the CTU-13 collection [49] will be used in this research; and are hereafter referred as Capture 1 to 13.[1] Botnet and Botnet C&C classes are combined in the case of Captures 3, 4, 10, 11, 12.[2] Each dataset consists of network flow information with 15 features extracted per flow by the Argus network flow analyzer.[3] The flow features with their corresponding description are listed in Table 4.1.

However, out of 15 features, the start/end times, State, IP addresses and Port numbers are excluded, as they are not reliable sources of information. IP addresses can be spoofed by attackers to operate malicious behaviours or by proxies in legitimate utilization to protect user identities on the Internet. Port addresses also could not be assured as belonging to a certain service, as many applications (Voice over IP, social media and network-based games) dynamically change their corresponding port

[1] `https://mcfp.felk.cvut.cz/publicDatasets/CTU-13-Dataset/CTU-13-Dataset.tar.bz2`

[2] The combined label is used in the case of data sets in which the Botnet C&C class represents less than 0.01% of the original data set.

[3] `http://qosient.com/argus/`

Table 4.2: Generic properties of the CTU-13 streaming datasets. N cardinality, and k is the total number of classes over the entire duration of the stream. Each dataset has $D = 8$ flow attributes. Classes are represented in the order: Background, Normal, Botnet and Botnet C&C. A combined Botnet/C&C label was assumed in the case of datasets in which the C&C class represents less than 0.01% of the original dataset (Capture 3, 4, 10, 11, 12).

Dataset	Botnet	Botnet Architecture	N	$\approx$ Class Distribution (%)
Capture 1	Neris	Centralized	2,824,637	[97.47, 1.08, 1.438, 0.01]
Capture 2	Neris	Centralized	1,808,123	[98.34, 0.5, 1.12, 0.04]
Capture 3	Rbot	Centralized	4,710,638	[96.95, 2.48, 0.57]
Capture 4	Rbot	Centralized	1,121,077	[97.52, 2.25, 0.23]
Capture 5	Virut	Centralized	129,833	[95.70, 3.6, 0.67, 0.02]
Capture 6	Menti	Centralized	558,920	[97.83, 1.34, 0.79, 0.04]
Capture 7	Sogou	Centralized	114,078	[98.47, 1.47, 0.03, 0.02]
Capture 8	Murlo	Centralized	2,954,230	[97.33, 2.47, 0.17, 0.04]
Capture 9	Neris	Centralized	2,087,508	[89.7, 1.44, 8.72, 0.14]
Capture 10	Rbot	Centralized	1,309,792	[90.67, 1.21, 8.12]
Capture 11	Rbot	Centralized	107,251	[89.85, 2.54, 7.61]
Capture 12	NSIS.ay	P2P	325,472	[96.99, 2.34, 0.657]
Capture 13	Virut	Centralized	1,925,150	[96.26, 1.66, 2.05, 0.03]

numbers when they are blocked or unavailable. Therefore, classifiers could not rely on this information due to retaining their ability to generalize in real world.

From a network security perspective the following malicious behaviours are present in each group of datasets based on the Botnet type activated on there:

Captures 1, 2, 9: consist of instances of the Neris Botnet, hence traffic content pertaining to Internet Relay Chat (IRC), spam, click fraud and scanning activities are explicitly present.

Captures 5, 13: consist of instances of the Virut Botnet, hence traffic content pertaining to Distributed Denial of Service (DDoS), spam, fraud and data theft attacks are explicitly present.

Capture 6: consists of instances of the Menti Botnet, hence traffic content pertaining to identity theft and login credentials are explicitly present.

Capture 7: consists of instances of the Sogou Botnet, hence traffic content pertaining to spam and popup adware to collect personal information are present.

Capture 8: consists of instances of the Murlo Botnet, hence traffic content pertaining to the use of scanning activities and proprietary mechanisms for establishing C&C.

Captures 3, 4, 10, 11: consist of instances of the Rbot Botnet, hence traffic content pertaining to IRC and Internet Control Message Protocol (ICMP) based DDoS attacks are explicitly present.

Captures 12: consists of instances of the NSIS.ay Botnet, hence traffic content pertaining to identity theft and login credentials by using extra payloads are explicitly present.

Table 4.2 depicts how highly imbalanced the datasets are, as all but the major class appear and disappear throughout the course of the stream (Section 5.1.6).

4.1.2 CTU13-mixed

The CTU13-mixed dataset is generated by concatenating CTU-13 traffic captures to produce a more challenging dataset for the study of multi-bot scenarios. The goal of generating this dataset is to study how algorithms perform when there is more than a Botnet behaviour available throughout the course of the stream. To gain this insight, the order of the Capture files from CTU-13 are arranged in a way to put 'distance' between Captures with the same type of Botnet. Thus, as a result, the 13 independent CTU-13 datasets are concatenated into a single stream, sequenced as follows: Capture5 (Virut), Capture6 (Menti), Capture3 (Rbot), Capture1 (Neris), Capture4 (Rbot), Capture7 (Sogou), Capture2 (Neris), Capture10 (Rbot), Capture8 (Murlo), Capture9 (Neris), Capture11 (Rbot), Capture12 (NSIS) and Capture13 (Virut). The outcome dataset consists of over 19 million network flows and 7 different types of Botnet. Moreover, more than 95% of the dataset are 'Background' traffic in which, conversely, other classes are considered minor classes (Table 4.3).

Table 4.3: Generic properties of the CTU13-mixed dataset. 7 types of Botnet behaviour are available. Classes are represented in the order: Background, Normal, Botnet and Botnet C&C. N is the Cardinality, and the class distribution is expressed in percentages.

Dataset	Botnet	N	$\approx$ Class Distribution (%)
Mixed CTU-13	Neris, Rbot, Virut, Menti, Sogou, Murlo, NSIS.ay	19,175,568	[95.99 , 1.78, 2.2, 0.03]

4.1.3 ISOT

The public ISOT dataset is created based on a combination of several malicious and non-malicious datasets to represent a semi-real-world Botnet dataset [123]. It contains traces of Storm and Waledac botnets where it is mainly categorized under ‘UDP flooding’ and ‘SMTP Spam’. Waledac is one of the most prevalent P2P botnets and is considered the successor of the Storm botnet with a more decentralized communication protocol. It utilizes HTTP communication and a fast-flux-based DNS network. The legitimated part is from two sources: the Traffic Lab at Ericsson Research in Hungary [103] and the Lawrence Berkeley National Laboratory (LBNL) in the U.S.[4] The dataset is originally provided in Pcap format with relative information on how to label the dataset accordingly. To provide the network flow-based version of the dataset for the research, the Tranalyzer network flow exporter [20] is utilized. After conversion to the flow version, the labels are assigned based on the provided information. Each dataset record consists of 79 features of network flow with the class label. The first 15% of the dataset is truncated as it contains no attack types.

Table 4.4: Generic properties of the ISOT streaming datasets. N cardinality, and k is the total number of classes over the entire duration of the stream. Each dataset has $D = 79$ flow attributes.

N	D	k	Class Distribution (%) [Normal, SMTP spam, UDP flooding]
254,291	79	3	[79%, 13%, 8%]

[4]LBNL Enterprise Trace Repository: `http://www.icir.org/enterprise-tracing`

Table 4.4 represents the statistics for the ISOT dataset.The list of flow features extracted for this dataset is available at Section A.1.

4.1.4 NSL-KDD

The NSL-KDD dataset is an improvement of the KDD'99 dataset [105], and it contains the packet-based information of network traffic. The original KDD'99 dataset contains almost five million connections. Each connection has 41 attributes and a class type, and some attributes are gathered throughout the time for some specific lengthy attacks. 22 different attack types are available that can be categorized in four main categories: 1) DoS (Denial of Service), 2) R2L (Remote to Local), 3) U2R (User to Root) and 4) Port Scanning (Probe). In NSL-KDD, duplicates are removed from KDD'99 to minimize the classification biases [105]. Several arrangements of the dataset are prepared for various learning purposes. For the streaming classification, the concatenation of the training partition (KDDTrain+) and test partition (KDDTest+) is used as a single new dataset in this thesis.

Table 4.5: Generic properties of the NSL-KDD streaming datasets. N cardinality, and k is the total number of classes over the entire duration of the stream. Each dataset has $D = 22$ packet attributes.

N	D	k	Class Distribution (%) [Normal,DoS,Probe,R2L,U2R]
148,515	41	5	[52%, 35%, 9%, 3%, $\sim$1%]

Table 4.5 represents the NSL-KDD classes' distribution. The two slow, lengthy types of attacks (R2L and U2R) with the Probe attack types are the minor classes in this dataset. The list of the features of this dataset is available in Section A.2.

4.2 Comparator Algorithms

Stream-GP is compared to many state-of-the-art classification algorithms, which vary from streaming learning models to the classical off-line learned models. This section describes the comparator algorithms solely, irrespective of details such as the evaluation methodology, which will be discussed in Section 4.3. Stream-GP is compared to two types of algorithms. One group is related to the streaming algorithms, in which the model is updated throughout the stream based on the content and changes that have appeared throughout the stream. These algorithms are rarely applied on network streaming traffic. On the other hand, some algorithms are used in network applications, in which the models are learned off-line and then applied to the streaming network traffic.

4.2.1 Naive Bayes

In both cases of Off-line and Streaming learning models, Naive Bayes is used as one of the comparator algorithms. The Naive Bayes classifier belongs to the family of probabilistic classifiers and is based on the Bayes Theorem (Eq. 4.1) with the "naive" assumption of conditional independence between each pair of features given the value of the class variable [121].

Let $(x_1, ..., x_n)$ be the list of attribute values where x_i is the value of attribute X_i. Moreover, C represents the class label where c is its value. The probability of $(x_1, ..., x_n)$ belonging to class c is:

$$P(c|x_1, ..., x_n) = \frac{P(c)P(x_1, ..., x_n|c)}{P(x_1, ..., x_n)} \tag{4.1}$$

for which the naive conditional assumption is

$$P(x_i|c, x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = P(x_i|c) \tag{4.2}$$

for all i, the relationship is simplified to

$$P(c|x_1, ..., x_n) = \frac{P(c)\prod_{i=1}^{n} P(x_i|c)}{P(x_1, ..., x_n)} \tag{4.3}$$

Since $P(x_1, ..., x_n)$ is a constant value, the following classification rule can be used:

$$P(c|x_1, ..., x_n) \propto P(c)\prod_{i=1}^{n} P(x_i|c) \rightarrow \hat{c} = \underset{c}{\operatorname{argmax}}\, P(c)\prod_{i=1}^{n} P(x_i|c) \tag{4.4}$$

Now, the Maximum A Posteriori (MAP) estimation can be used to estimate $P(c)$ and $P(x_i|c)$ where the former is the relative frequency of class c in the training set. The Naive Bayes classifier selects the class label with the highest probability that results in minimum error.

For the classical utilization of Naive Bayes, all training information is known a priori, so all the calculations and estimations can be done before the label predication.

However, in streaming learning, there is no insight to the upcoming data in the stream, which leads to applying Naive Bayes incrementally in streaming applications [96]. The streaming learning process starts with an initial training set and computes two parameters: prior probability ($P(c)$) and conditional probability ($P(x|c)$). As the stream goes on, these two parameters are updated based on the new training data. So, the class label prediction is done at any time based on the available information (a.k.a. the corresponding parameters) to calculate the post probability ($P(c|x)$). Moreover, a generalization to the case of operation under label budgets is readily available [106].

4.2.2 Decision Trees

Decision Trees create a hierarchical partitioning of data where at the leaf level, the partitions are assigned to a class label. The hierarchical partitioning at each level happens based on *"split criteria"*. These criteria may be based on a single attribute or a function of several attributes referred to as *univariant* and *multivarient* respectively. This approach involves maximizing the discrimination between different classes over different nodes while splitting. This maximization is going to happen if the level of skew among different classes in a given node be maximized. A measure such as gini-index or entropy is used to maximize this skew.

In construction of the training model, the splitting happens in a way to minimize the weighted sum of the gini-index or entropy of the two nodes. The process is repeated until a termination criterion is met. The obvious termination criterion would be that all instances of a node belong to one class. More generally, the termination criterion is either a minimum level of skew or purity or a minimum number of records exist in a node to prevent overfitting problems. One problem with Decision Trees is that the time to stop the splitting can not be predicted. Therefore, there are ways of pruning the tree to avoid overfitting.

Table 4.6 represents the algorithms used in this thesis; their specific description is next.

CART

The CART, Classification and Regression Trees, algorithm [17] provides a general framework through which different decision trees can be gained.

Table 4.6: Decision Tree Algorithms

Algorithm	Type	Single/Ensemble
Hoeffding Tree		Single
CART	univariant	Single
Random Forest		Ensemble

The CART decision tree is a binary recursive partitioning tree capable of working with both continuous and nominal attributes as targets. The trees could be stopped from splitting by a stop criterion or could be expanded completely and pruned afterwards. The CART mechanism includes (optional) automatic class balancing and automatic missing value handling. Some main characteristics of the CART algorithm are explained briefly in the following [41].

Branching factor: The tree is branched in a binary format ($B = 2$). The authors believe that the binary split is preferred, as 1) the data are fragmented more slowly than the multi-splits, 2) the same split on a value is allowed and can be repeated.

Splitting Rule: It is always in the form of

An instance goes left if ***CONDITION****, and goes right otherwise*

where the CONDITION is “attribute $X_i <= C$” for continuous attributes. For nominal attributes, it is as membership in an explicit list of values.

The Gini impurity algorithm (Eq. 4.5) is used as the metric for splitting criteria in the training set. It computes how often a randomly selected attribute would be incorrectly labelled if it was randomly labelled based on the distribution of labels in the subset.

$$i(t) = 1 - \sum_{t=0}^{1} P_t^2 \tag{4.5}$$

The smaller the Gini impurity index is, the better the candidate is to split. Later, the information gain metric is also added to the algorithm as an optional metric.

In the case of using stop criteria, node splitting continues until it can not create purer children or a stop criterion is reached. The stopping rules are:

- The depth of the tree reaches the maximum depth.

- There is no split candidate, which makes the information gain greater than the minimum information gain.

- There is no split candidate that produces child nodes that have at least a minimum number of instances per node.

The pruning strategy could also be used instead of stop rules. This allows the tree to completely expand, and then the tree can be pruned afterwards. The pruning strategy is more costly than the stopping criteria.

Class balancing CART always computes the ratio of class frequency in each node to the root node. This is equivalent to automatically reweighting the data to balance the classes. It allows CART to work with imbalanced data.

Greedy Algorithm The decision to split the node is made in a greedy mode in the direction of minimizing the information gain at a tree node. The optimal solution at each node is local so there is no assurance that it will lead to a global optimal solution as well.

Hoeffding Tree

The Hoeffding Tree [38] is designed to work with extremely large (potentially infinite) datasets. This is unlike classic decision trees like ID3, C4.5 and CART, which assume that the whole dataset can be stored in the main memory. Loading dataset partially is possible by considering only a small subset of training data to be kept at each node. The decision on how many exemplars are needed at each node is based on the statistical Hoeffding bound.

Consider variable r whose range is R. Suppose n observations of the variable have been made, and their mean is $\bar{r}$. The Hoeffding bound states that with $1 - \sigma$ probability the true mean of the variable is at least $\bar{r} - \epsilon$, where

$$\epsilon = \sqrt{\frac{R^2 ln(1/\sigma)}{2n}} \tag{4.6}$$

Algorithm 3 Hoeffding Tree algorithm.

Input: S is a sequence of examples, X is a set of discrete attributes, $G(.)$ is a split evaluation function, δ is one minus the desired probability of choosing the correct attribute at any given node.
Output: HT is a decision tree.

```
1: procedure HoeffdingTree($S$,$X$,$G$,$\delta$)
2:     Let $HT$ be a tree with a single leaf $l_1$ (the root).
3:     Let $X_1 = X \cup \{X_\phi\}$.
4:     Let $\bar{G}_1(X_\phi)$ be the $\bar{G}$ obtained by predicting the most frequent class is $S$.
5:     for each class $y_k$ do
6:         for each value $x_{ij}$ of each attribute $X_i \in X$ do
7:             Let $n_{ijk}(l_1) = 0$.
8:         end for
9:     end for
10:    for each sample $(x, y_k)$ in $S$ do
11:        Sort $(x, y)$ in to a leaf $l$ using $HT$.
12:        for  each $x_{ij}$ in $x$ such that $X_i \in X_l$ do
13:            Increment $n_{ijk}(l)$.
14:        end for
15:        Label $l$ with the majority class among the examples seen so far at $l$.
16:        if the examples seen so far at $l$ are not all of the same class,  then
17:            Compute $\bar{G}_l(X_i)$ for each attribute $X_i \in X_l - X_\phi$ using the counts $n_{ijk}(l)$.
18:            Let $X_a$ be the attribute with highest $\bar{G}_l$.
19:            Let $X_b$ be the attribute with second-highest $\bar{G}_l$.
20:            Compute $\epsilon$ using 4.6
21:            if $\bar{G}_l(X_a) - \bar{G}_l(X_b) > \epsilon$ and $X_a \neq X_\phi$ then,
22:                Replace $l$ by an internal node that splits on $X_a$.
23:                for each branch of the split do
24:                    Add a new leaf $l_m$, and let $X_m = X - \{X_a\}$. Let $\bar{G_m}(X_\phi)$ be the $\bar{G}$ obtained
    by predicting the most frequent class at $l_m$.
25:                    for each class $y_k$ do and each value $x_{ij}$ of each attribute $X_i \in X_m - \{X_\phi\}$
26:                        Let $n_{ijk}(l_m) = 0$.
27:                    end for
28:                end for
29:            end if
30:        end if
31:    end for
32:    Return $HT$.
33: end procedure
```

The attractive property of the Hoeffding bound is that it is not dependent on the probability distribution of observations. If $G(X_i)$ is assumed to be the heuristic to choose test attributes, the goal is to make sure the attribute chosen using n observations will be chosen when infinite observations are available as well. The Hoeffding bound guarantees that this will happen.

The algorithm progressively splits the leaves and creates decision nodes, as there is enough information available at the leaf. It traverses the tree from the node to the leaf in an exemplar-wise manner and updates the statistics along the way. The Hoeffding Tree pseudo code taken from [38] is given in Pseudocode 3.

The Hoeffding trees are successfully applied on the streaming applications [48] and have been utilized under label budget criteria as well [106].

Random Forest

Random Forest is an ensemble classifier algorithm developed by Breiman [18]. In this algorithm, a number of binary trees are constructed where voting takes place between these trees on each test exemplar to decide the final decision. Each tree in the Random Forest is learned on different subsets of the training set, the goal of which is to decrease the variance. This is actually a modified instance of bagging, which is a technique used to reduce the variance of an estimated prediction function.

Given $X = x_1, ..., x_n$ with $C = c_1, ..., c_n$ as labels, bagging will repeatedly (B times) sample randomly with replacement from the training set and fit trees with these sample; so each tree is constructed on a subset of the dataset that is selected randomly. The first step of the Random Forest algorithm is exactly the same as bagging, but the next step, splitting the nodes, differs. In each tree, node branching takes place based on the best feature in a randomly selected subset of features (bagging uses all features). At each internal node of the tree, a subset of attributes are selected randomly, and the decision on how to split is determined based on an entropy metric. After training, the predicted label is given based on the voting of trees' outcome. Random Forest offers improvements on the overfitting problem in decision trees [57]. The pseudo code of Random Forest taken from [57] is illustrated in Algorithm 4.

Algorithm 4 Random Forest for Classification.

```
1. For $b = 1$ to B:
    (a) Draw a bootstrap sample Z* of size N from the training data.
    (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by re-cursively
        repeating the following steps for each terminal node of the tree, until the
        minimum node size $n_{min}$ is reached.
        i. Select $m$ variables at random from the $p$ variables.
        ii. Pick the best variable/split-point among the $m$.
        iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.
```

To make a prediction at a new point x:
Classification:Let $\hat{C}_b(x)$ be the class prediction of the *bth* random-forest tree. Then $\hat{C}_{rf}^B(x) = majority\ vote\{\hat{C}_b(x)\}_1^B$

4.3 Comparator Frameworks

In this section, the comparator algorithm's parameterization is explained. A detailed description of algorithms were previously given in Section 4.2. Table 4.7 shows the list of algorithms, the model learning methodology and in which platform they are used.

Table 4.7: Comparator algorithms List.

Algorithm	Model (Off-line or Streaming)	Toolset (MOA or Apache Spark)
Naive Bayes	Both	Both
Hoeffding Tree	Streaming	MOA
Random Forest	Off-line	Apache Spark
CART	Off-line	Apache Spark

4.3.1 Massive Online Analysis (MOA)

The work of Žliobaitė *et al.* [106] addresses streaming classification under label budget criteria where changes to the data are expected as well. In their benchmarking study only one dataset consists of more than two classes. To the best of the author's knowledge, no (multi-class) imbalanced streaming classification under a label budget has been proposed before this thesis (Section 2).

The MOA software suite[5] provides the implementation of these algorithms, where modifications were made to provide reporting using the performance metric from Section 4.4.

Table 4.8: Best case configurations of MOA comparator algorithms for operation under drifting streaming data with label budgets [106].

Classifier	Policy
Naive Bayes	Split
Naive Bayes	Variable uncertainty
Hoeffding tree	Split
Hoeffding tree	Variable uncertainty

Table 4.8 represents the combinations of the classifier and the corresponding sampling policy from the MOA toolset that performed the strongest in the original study [106]. The algorithms are changed to adapt to the streaming situation where the models are constructed incrementally throughout the course of the stream. Naive Bayes and Hoeffding Tree algorithms are explained in Section 4.2.1 and Section 4.2.2, respectively. Split and Variable Uncertainty are the most effective sampling policies [106] used to sample a finite set of data based on a label budget to query true label information. The definition of these sampling policies comes next.

Variable Uncertainty

Variable uncertainty sampling policy relies on the classifier's confidence. If the confidence is below the threshold, it is asking for true labels. When a change occurs and there is a need for new labels, the most uncertain ones are asked for true labels according to the label budget.

[5] `http://moa.cms.waikato.ac.nz`

Split

In the case of Split, two models are maintained concurrently. One model uses variable uncertainty as the sampling policy and the other is using uniform sampling. Both policies are used to request the true label information. Uniform sampling is most likely to be used if a change is detected in the stream.

4.3.2 Apache Spark Streaming

Apache Spark[6] is a clustering framework suitable to be used in big data analysis and computation. This tool is a state-of-the-art solution for analyzing network behaviour from the perspective of network managers. The Spark Streaming component in Apache Spark provides the streaming processing of data. The process could be a simple statistical operation or more complicated machine learning processes. In the case of ML operations, the MLlib library[7] provides a distributed ML library on top of an Apache Spark core. The overall architecture of the Apache Spark is depicted in Figure 4.1.

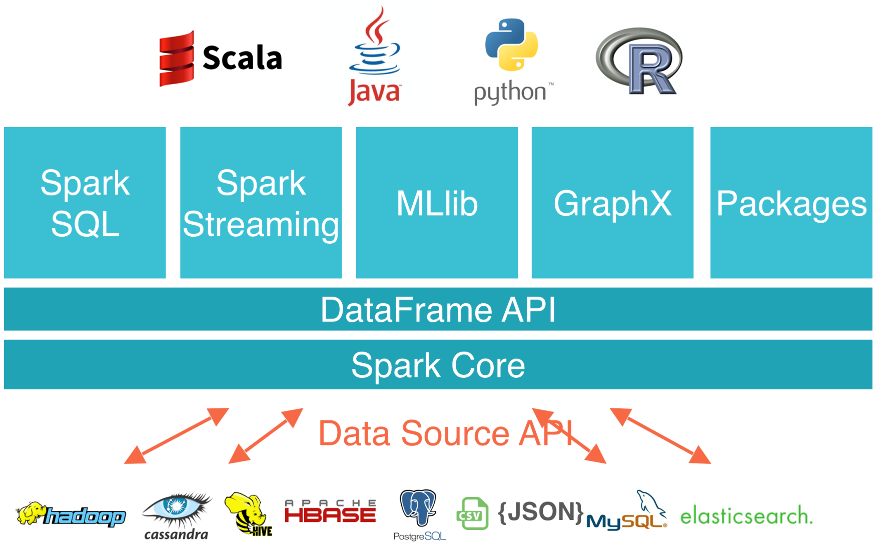


Figure 4.1: Apache Spark architecture.

The publicly available MLlib library supports many machine learning algorithms

[6]Apache Spark: https://spark.apache.org/.
[7]Apache Spark MLlib, `http://spark.apache.org/docs/latest/mllib-guide.html`

for the classification, regression or clustering of data. In this research, among the provided algorithms, Decision Tree, Random Forest and Naive Bayes algorithms are selected and benchmarked as the most well-known ML classification algorithms used in the literature [2, 3, 94, 90]. The CART algorithm (Section 4.2.2) is used for building trees in Decision Tree where Random Forest (Section 4.2.2) uses the ensemble of CART trees as well. In Decision Tree, the maximum depth of the tree is set to 7 ($max\_depth = 7$). Similarly, in Random Forest, the maximum depth of the trees is set to 4 ($max\_depth = 4$) and 50 trees ($num\_trees = 50$) are constructed. The rest of the parameters are set to their default values in both algorithms.

4.4 Performance Metrics

Performance metrics for streaming data classification take one of the three classes of metrics [82].

The Prequential error metric characterizes the 'accuracy' measurement, which there is a decay relative to the older instances, hence forgetting the older instances.[8] The problem with this kind of metric arises when imbalanced data are encountered. In this case, a simple model that labels all instances as the most frequent class seems to be more 'accurate'. Under the network security application, this frequent class is either 'Background' or 'Normal' traffic, where a degenerate classifier could get 80%-90% accuracy (depending on the distribution of the most frequent class, e.g. Tables 4.2, 4.4 and 4.5). In contrast, a better metric is required to quantify the classifier's ability to operate under multi-class scenarios.

Measure of (label) autocorrelation characterizes the performance of a classifier based on its ability to defeat a single bit classifier operating on the label space alone [13]. In other words, when the distribution of the data is not well mixed, there are some sequences that all have the same label. A one/two bit state machine could predict these sequences with a low rate of miss classification.[9] This is a better metric than the previous error minimization, but it still does not quantify the ability of the classifier to work under a multi-class scenario. (i.e., the distribution of labels is mostly determined by major classes in the imbalanced data scenario).

[8]The use of the recal rate on the single smallest class also has been proposed [112].

[9]See for example, the widespread use of one/two-bit finite state machines in branch taken/not taken sequences for conditional statements associated with loop constructs [58].

Rate-based metrics incrementally construct the confusion matrix throughout the stream [59, 107]. Then, the resulting confusion matrix can be used to obtain any number of scalar (rate-based) performance metrics, e.g. F-measure, Detection rate, Precision. Furthermore, this metric explicitly quantifies the performance of classifiers under a multi-class scenario [64].

As an explicit demonstration of the classifier's performance could be quantified using a rate based metric, such a metric is used in this thesis. The AUC metric (Area Under the Curve) could be used as one solution that illustrates the curve characterizing the interaction between Detection rate and False positive rate. However, this metric could only be applied to a two class scenario which also needs a complete reconstruction for each new window location [19].[10]

Alternatively, Detection rate (DR) is used in this research and is explicitly computed for each class. Such a metric can be calculated incrementally throughout the stream and could be visualized with time on the dependent (x) axis and class-wise performance metric on the independent (y) axis. The champion classifier always has to predict the labels before any update to the model. Thus, the overall DR could be obtained by averaging the detection rate of all classes dynamically estimated throughout the stream [82]. On the other hand, the False positive rate (FPR) is calculated the same way as the Detection rate.

The streaming estimation of the Detection rate for each class is calculated by:

$$DR_c(t) = \frac{tp_c(t)}{tp_c(t) + fn_c(t)} \tag{4.7}$$

where t is record index and $tp_c(t), fn_c(t)$ are the online true positive and false positive rates until that point of the stream.

Also, the streaming estimation of the False positive rate for each class is computed as:

$$FPR_c(t) = \frac{fp_c(t)}{fp_c(t) + tn_c(t)} \tag{4.8}$$

where t is the record index and $fp_c(t), tn_c(t)$ are the online false positive and true

[10]The alternative would be to re-estimate the entire AUC for each time step, limiting its application to short streams [36].

negative rates until that point of the stream.

The multi-class Detection rate is calculated as:

$$DR(t) = \frac{1}{C^*} \sum_{c=[1,\ldots,C^*]} DR_c(t) \tag{4.9}$$

where C^* is the true total number of classes at any point in the stream. If this number is not known from the start of the stream, every time an unseen label is encountered, a stepwise effect happens in the metric.

Hence, the multi-class Detection rate is the sum of each class detection rate. In addition, although the stream is considered endless, for benchmarking purposes, a finite sequence of streaming data is assumed. Therefore, the overall detection rate is computed as the surface under the multi-class Detection rate plot, estimated as the sum of all the multi-class detection rates, over the course of the stream. This calculation returns a scalar value to be used as a metric called AvDR.

$$AvDR = \frac{1}{s_{max}} \sum_{t=0}^{s_{max}-1} DR(t) \tag{4.10}$$

where s_{max} is the cardinality of the stream.

The combination of Detection rate and False positive rate give a precise evaluation of the multi-class classifiers. However, another combination is also applicable where Recall and Precision metrics could be used together. Recall is another name for Detection rate, whereas Precision is formulated as:

$$Prec_c(t) = \frac{tp_c(t)}{fp_c(t) + tp_c(t)} \tag{4.11}$$

which specifies how precisely a class is detected.

The two metric sets, Detection rate, False positive rate and Recall, Precision, indicate how precisely the classifiers are able to detect each class in the imbalanced stream of data. In this thesis, the first metric set is used for this purpose.

4.5 Experimental Design and Parameterization

Sections 3.3 and 3.4 introduced biased approaches for sampling and archiving policies, respectively. Five combinations of these policies are proposed that are addressed in

Table 4.9. All these algorithms are benchmarked in this thesis to check the relative effectiveness of sampling and archiving policies in the field of streaming multi-class classification. Also, the parametrization of the Stream-GP is indicated in Table 4.10.

Table 4.9: Stream-GP algorithms with their corresponding configurations. Uniform implies the identification of either sampling or archiving data using uniform sampling (Sections 3.3 and 3.4, respectively). Biased refers to either sampling or archiving data that happens under the corresponding biased algorithms (Algorithms 1 and 2 respectively).

Model	Sampling Policy	Archiving Policy
Random	Uniform	Uniform
Sample	Biased	Uniform
Archive	Uniform	Biased
Both	Biased	Biased
Hybrid	(Biased, Uniform)*	Biased

* In the Hybrid algorithm, the Sampling is Biased at first, and whenever the minor classes reach a pre-determined Detection rate, it changes to the Uniform configuration.

Table 4.10: Stream-GP Parameters. Mutation specifies the rate of adding/deleting programs from a learner or changing the action. $Gap(Tgap)$ denotes the number of records in the Data subset (teams) to be deleted at each non-overlapping window location. For each Data subset update, τ generations are performed.

Parameter	Value
Data Subset size (DS)	120
DS gap size (Gap)	20
GP gap size ($Tgap$)	20
Team pop. size (P_{size})	120
Max. programs per team (ω)	20
Prob. Program deletion (pd)	0.3
Prob. Program addition (pa)	0.3
Prob. Action mutation (μ)	0.1
Generations per DS update (τ)	5

The Data Subset size (DS) specifies the size of the archive from the stream content. At each generation of GP the following takes place:

- Gap size (Gap) from data subset is replaced by new exemplars.
- Team population size (P_{size}) is replaced by new teams.

- Programs are deleted by pd probability.
- Programs are added by pa probability.
- Actions are mutated by μ probability.

The maximum number of programs in a team is identified by ω and τ is the number of generations per data subset update. The value of each parameter is set to its best performing value given in the previous work [108].

The label budget, β, specifies the portion of data in each window location that are asked for true labels. The higher the parameter value, the more 'costly' the learning process would be, as a user/expert is required to provide the labels for each query. The lower the parameter value, the higher the risk of losing minor classes at all or missing the change that is happening through the stream.

For instance, if the label budget is 5%, and a minor class is appearing at a 1% frequency rate throughout the stream, the chance of a uniform sampling policy catching an exemplar of this class is 0.05%. On the other hand, if a sampling or archiving algorithm is more 'intelligent', then the performance of the classifier would less be affected by the dominant classes. In this thesis, different sampling and archiving policy combinations are benchmarked to investigate how decreasing the label budget would affect the performance of each classifier and at what rate. Thus, three different label budgets, $\beta = \{0.005, 0.01, 0.05\}$ are considered to be examined in different sections. Due to the low distribution of the minor class, e.g. Table 4.3, it provides a challenging streaming classification task.

Table 4.11 refers to the label budgets and their impact on the size of the non-overlapping window, $W(i)$. Keep in mind that the content of window $W(i)$ is predicted by the champion team from the previous window $W(i-1)$ before any updates to the champion (i.e., incremental operation). Also, lower label budgets are required to delay to get the true label information.

Table 4.11: Stream Dataset Parameters. Label Budget (β) is defined as a function of the window size $W(i)$ where for each non-overlapping window location there can only be *Gap* size (20) samples.

Label Budget (β)	$W(i)$ cardinality
0.5%	4,000
1.0%	2,000
5.0%	400

4.6 Statistical Significance Testing

The statistical significance tests are utilized to quantify the performance of the algorithms in the results section. They are used to measure if the differences in the algorithms' performances are significant. In this thesis, the tests are done on two domains: 1) the comparison of the algorithms on a single dataset, and 2) the comparison of the algorithms on multiple datasets. In this section, the statistical tests that have been used for this purpose are explained in both domains. Section 4.6.1 explains the Mann-Whitney pair-wise test for multiple algorithms on a single dataset (domain 1) with Section 4.6.3 defining the post-hoc test for it. Section 4.6.2 describes the Friedman test for the ranking of multiple algorithms on multiple datasets (domain 2) where Section 4.6.4 is a post-hoc algorithm for this test to group similar algorithms based on their performances.

4.6.1 Wilcoxon-Mann-Whitney Test

A nonparametric Mann-Whitney U test is performed to show the statistical significance of the results obtained. This test is used for comparison of the algorithms' performance on a single dataset in a pair-wise manner. The preliminary step to use in this statistical test is to determine if the data is not uniformly distributed; one way to identify is through violin plots. The shape of the violin plot demonstrates if the data distribution belongs to a normal distribution or not. The following assumptions are considered:

- The null hypothesis (H_0) is that the distribution of classifiers' results are equal.
- The alternative hypothesis (H_1) is that the distributions are not equal.

If the p-value between two algorithms is less than the critical difference (CD) then the null hypothesis is rejected and they are considered significantly different.

4.6.2 Friedman Test

The Friedman test is a non-parametric repeated measures statistic that is equivalent to the parametric ANOVA test [33] but more robust. The test is used to compare different algorithms over various datasets. The analysis is done based on the ranking of algorithms over datasets, not the performance measures. In doing so, the following steps are taken:

First, each classifier is ranked over each single dataset in an ascending order, i.e. the best performing algorithm gets the lowest rank. For dataset D_i, the classifier C_i is ranked first if its performance is greater than every one else's, $pm_{ij} > pm_{ij\prime}, \forall j\prime, j, j\prime \in$

$\{1, 2, ..., k\}, j \neq j\prime$. If there is a d-way tie between algorithms, the rank is computed as, $[(r+1)+(r+2)+...+(r+d)]/d$ for each tied algorithm.
Let R_{ij} be the rank of classifier C_j on dataset D_i, quantities are computed as:

- The mean rank of the classifier C_j on all datasets:

$$\overline{R}_j = \frac{1}{n}\sum_{i=1}^{n} R_{ij} \tag{4.12}$$

- The Friedman statistic is calculated based on Eq. 4.12 as:

$$\chi_F^2 = \frac{12n}{k(k+1)}\left[\sum_{j=1}^{n} \overline{R_j}^2 - \frac{k(k+1)^2}{4}\right] \tag{4.13}$$

  where n is the number of datasets and k is the number of algorithms.

The null hypothesis assumes that all classifiers are equivalent in their performances and also in their average ranks, R_j. A comparison of the χ_F^2 with the F distribution with $k-1$ degree of freedom determines if the null hypothesis is rejected. The null hypothesis is rejected if the p value of the corresponding χ_F and $F_F(n, k-1)$ in the table is smaller than the critical difference (CD).

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2} \tag{4.14}$$

4.6.3 Bonfferoni-Dunn Post-hoc Test

In the case of the comparison of mean values to a control classifier, the Bonfferoni-Dunn test could be applied. For this purpose, the significance level α is divided by the number of comparisons made. Then, the new significance level is used for comparison of the designated control algorithm with others. This test is used to compare the proposed algorithm against the comparator algorithms on a single domain scenario after the Mann-Whitney U test.

4.6.4 Nemenyi Post-hoc Test

A post-hoc test is used after the rejection of null hypothesis to demonstrate which algorithms are performing equally or similarly. Through this test, the significance of the algorithms can be distinguished more specifically. The Nemenyi post-hoc test computes a q statistic over the difference in the average mean ranks of the classifiers. It is calculated after the Friedman test rejects the null hypothesis. So, considering the prerequisite steps of the Friedman test, for any two classifiers, C_{j1} and C_{j2}, the q statistic is calculated as:

$$q = \frac{\overline{R_{j1}} - \overline{R_{j2}}}{\sqrt{\frac{k(k+1)}{6n}}} \tag{4.15}$$

The null hypothesis is rejected if the q value exceeds the critical value with the desired significance level, hence q_α where α is the significance level. Therefore, if the q value of two classifiers is more than or equal to the critical difference, CD, with the appropriate amount of uncertainty, the two algorithms are considered significantly different.

4.7 Summary

In this chapter, all the requirements for evaluating Stream-GP are described. These requirements include the network datasets used, the evaluation metrics and the statistical tests. Moreover, the definitions of the comparator algorithms were explained in detail, and the parameter settings for the Stream-GP and comparator algorithms were explained.

All the above are then used in various scenarios in the results section (Section 5) to evaluate the Stream-GP algorithms and compare them to the available state-of-the-art frameworks.

Chapter 5

Results

The benchmarking of the proposed GP streaming system, Stream-GP, with various sampling and archiving policies is presented for real-world network security problems, specifically botnet detection. Moreover, the system's utilization in a network analytic application is demonstrated to showcase the generality of the solution for other network streaming problems. The overall structure of the evaluations in this section is illustrated in Figure 5.1. Section 5.1 evaluates the performance of the system in network streaming traffic with *Botnet* malicious activities. The evaluations are done on a *CTU-13* dataset collection, where each dataset is representative of a single bot behaviour. Section 5.2 investigates the system's performance, where several botnet tracks exist in the stream of network traffic. This challenging scenario investigates how the system reacts to previous learned botnet behaviours. Section 5.3 explores the best algorithms' performances on more network security scenarios. In previous sections, Stream-GP variants are compared with the best MOA toolkit algorithms, which are explicitly designed to perform under a streaming mode. To address the necessity of comparing the system with current real-world network solutions, Apache Spark (Streaming) is chosen. It is widely employed by IT / Network Management teams in commercial, government and academic environments. The discussion on the comparison of Stream-GP and Apache Spark (Streaming) is addressed in Section 5.4. Furthermore, Section 5.5 introduces the Stream-GP application in network analytic tasks such as unknown network traffic exploitation. Finally, Section 5.6 summarizes this chapter.

5.1 Botnet Detection in Real-world Network Traffic

The utilization of Stream-GP and its comparator algorithms from the MOA toolkit is analyzed under various botnet scenarios. In Section 5.1.1, the overall performance for different Stream-GP algorithms (configurations) and their comparator MOA algorithms is evaluated for the thirteen Botnet datasets (Table 4.2). Thereafter, some of the datasets are selected based on the primary results for a broader investigation to provide more insight into the algorithm's operation and its distinguishing factors. Specifically, Section 5.1.2 demonstrates the advantage of detection rate visualization through the course of the stream to provide insight into the dynamic behaviour of the algorithms. Section 5.1.3 examines the various archive / sampling policies of Stream-GP and how they affect the distribution of the records in the Data subset. Section 5.1.4 specifically determines the capabilities of the algorithms in the detection of botnet behaviour. This class is of interest from a network security perspective, as it determines how well the algorithm can identify malicious behaviour in streaming network traffic. Section 5.1.5 focuses solely on the detection of the least frequent class, which are the command and control signals. Success in this task amounts to providing an early warning of botnet activity. Section 5.1.6 specifies the distribution of each class record in the data subset throughout the course of the stream. Section 5.1.7 quantifies the computation time for fitness evaluation and champion individual execution to label the data. Finally, Section 5.1.8 summarizes the findings on botnet detection.

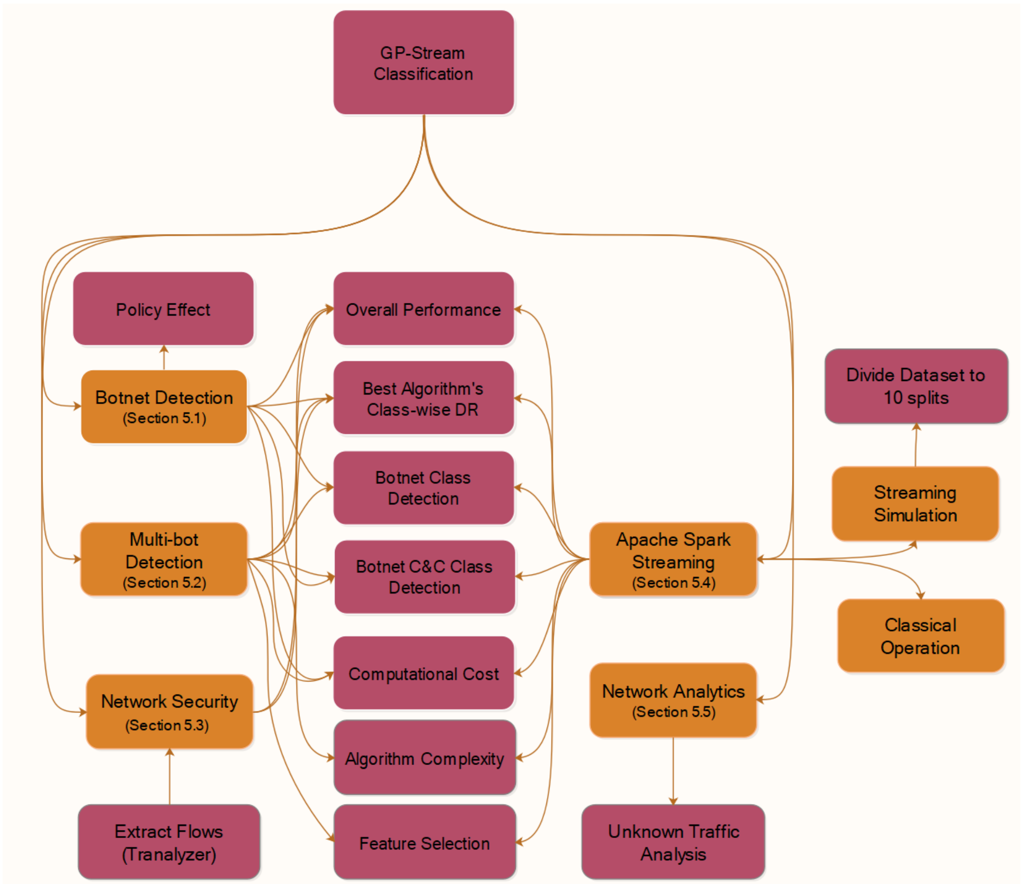

GP-Stream Classification
Policy Effect
Botnet Detection (Section 5.1)
Multi-bot Detection (Section 5.2)
Network Security (Section 5.3)
Extract Flows (Tranalyzer)
Overall Performance
Best Algorithm's Class-wise DR
Botnet Class Detection
Botnet C&C Class Detection
Computational Cost
Algorithm Complexity
Feature Selection
Apache Spark Streaming (Section 5.4)
Network Analytics (Section 5.5)
Unknown Traffic Analysis
Divide Dataset to 10 splits
Streaming Simulation
Classical Operation

Figure 5.1: Overall structure of GP-Stream evaluations and comparisons

Table 5.1: Algorithm ranks w.r.t. streaming AvDR metric under a 5% label budget. Bracketed entries represent median AvDR values to 1 decimal place. Naive Bayes (NB) and Hoeffding tree classifiers (from MOA) appear with either 'split' or 'variable' sampling policies. Table 4.9 declares the 4 sampling/replacement policies for stream-GP. R_j denotes the average rank across all datasets.

Dataset	Stream-GP				Hoeffding		NB	
	Random	Sample	Archive	Both	split	variable	split	variable
Capture 1	5 (32.4)	4 (36.3)	**1 (56.8)**	2 (51.5)	7.5 (25.0)	6 (26.7)	7.5 (25.0)	3 (43.7)
Capture 2	5 (36.8)	4 (41.5)	**1 (68.1)**	2 (67.7)	7.5 (25.0)	6 (36.5)	7.5 (25.0)	3 (54.8)
Capture 3	4 (65.9)	3 (76.0)	2 (81.5)	**1 (83.5)**	7.5 (33.3)	6 (55.3)	7.5 (33.3)	5 (59.5)
Capture 4	5 (45.5)	4 (51.4)	**1 (62.7)**	2 (60.8)	7.5 (33.3)	6 (42.2)	7.5 (33.3)	3 (55.2)
Capture 5	4 (29.3)	5 (28.8)	**1 (36.1)**	2 (34.5)	7.5 (25.0)	6 (26.3)	7.5 (25.0)	3 (30.7)
Capture 6	5 (35.5)	4 (42.5)	**1 (65.8)**	2 (64.6)	7.5 (25.0)	6 (25.5)	7.5 (25.0)	3 (48.6)
Capture 7	4 (29.8)	5 (28.4)	2 (29.8)	3 (29.8)	7.5 (25.0)	6 (25.9)	7.5 (25.0)	**1 (32.5)**
Capture 8	5 (39.9)	4 (46.2)	**1 (78.0)**	2 (76.2)	7.5 (25.0)	6 (28.2)	7.5 (25.0)	3 (57.9)
Capture 9	5 (34.8)	4 (35.8)	**1 (54.1)**	2 (48.5)	7.5 (25.0)	6 (26.5)	7.5 (25.0)	3 (45.4)
Capture 10	5 (57.6)	3 (61.6)	**1 (70.3)**	2 (68.7)	7.5 (33.3)	6 (54.6)	7.5 (33.3)	4 (58.7)
Capture 11	3 (48.7)	4 (47.9)	**1 (54.8)**	2 (52.4)	7.5 (33.3)	6 (42.2)	7.5 (33.3)	5 (47.6)
Capture 12	4 (41.6)	5 (40.2)	**1 (52.5)**	3 (47.2)	7.5 (33.3)	6 (36.0)	7.5 (33.3)	2 (48.3)
Capture 13	5 (41.8)	3 (53.9)	**1 (70.2)**	2 (67.4)	7.5 (25.0)	6 (27.3)	7.5 (25.0)	4 (42.9)
R_j	4.54	4.00	**1.15**	2.08	7.50	6.00	7.50	3.23

5.1.1 Overall Performance Evaluation

The (first) four Stream-GP algorithms (configurations) (Table 4.9) and four MOA comparator algorithms (Section 4.3.1) are evaluated. The MOA algorithms are selected as the strongest algorithms/sampling strategies by Žliobaitė *et al.* for operation under label budgets [106]. All eight algorithms are run 20 times per dataset. The evaluation of their performance is based on the multi-class streaming AvDR metric (Eq. 4.10 of Section 4.4) on the mean value of the runs. Thereafter, ranking is performed on the overall performances using the Friedman test. Together with a Nemenyi post-hoc test, the general trend of the best algorithms/most challenging datasets are identified. This evaluation is considered a robust solution in comparison with the ANOVA equivalent for the comparison of algorithms' performance on several datasets [33, 64]. The evaluation is conducted separately for the three label budgets, Table 4.11.

Tables 5.1, 5.2 and 5.3 demonstrate the results of the ranking of eight algorithms under the multi-class AvDR metric for different label budgets. The Friedman non-parametric repeated measures statistic (and Nemenyi post-hoc test) are then used to identify the general trends of the best algorithms and the most challenging datasets. Such an approach does not make assumptions regarding the underlying distributions of the performance data, and represents the preferred approach for conducting comparisons between multiple algorithms

Table 5.2: Algorithm ranks w.r.t. streaming AvDR metric under a 1% label budget. Bracketed entries represent median AvDR values to 1 decimal place. Naive Bayes (NB) and Hoeffding tree classifiers (from MOA) appear with either 'split' or 'variable' sampling protocols. Table 4.9 declares the 4 sampling/replacement policies for stream SBB. R_j denotes the average rank across all datasets.

Dataset	Stream-GP				Hoeffding		NB	
	Random	Sample	Archive	Both	split	variable	split	variable
Capture 1	5 (32.9)	4 (33.3)	**1 (48.3)**	2 (45.3)	7.5 (25.0)	6 (25.3)	7.5 (25.0)	3 (37.5)
Capture 2	5 (37.2)	4 (39.2)	**1 (56.6)**	2 (54.6)	7.5 (25.0)	6 (35.3)	7.5 (25.0)	3 (48.0)
Capture 3	4 (64.2)	3 (70.8)	**1 (78.4)**	2 (75.8)	7.5 (33.3)	6 (47.4)	7.5 (33.3)	5 (61.8)
Capture 4	5 (43.5)	4 (44.3)	**1 (55.8)**	2 (52.4)	7.5 (33.3)	6 (33.4)	7.5 (33.3)	3 (51.0)
Capture 5	4 (27.1)	5 (26.7)	**1 (29.2)**	3 (27.5)	6.5 (25.0)	8 (24.9)	6.5 (25.0)	2 (29.1)
Capture 6	5 (33.3)	3 (37.9)	**1 (50.4)**	2 (43.8)	7.5 (25.0)	6 (25.0)	7.5 (25.0)	4 (36.5)
Capture 7	3 (25.5)	8 (24.7)	**4 (25.4)**	5 (25.3)	6.5 (25.0)	1 (25.9)	6.5 (25.0)	2 (25.6)
Capture 8	4 (34.4)	5 (33.1)	2 (60.9)	**1 (64.3)**	7.5 (25.0)	6 (26.0)	7.5 (25.0)	3 (52.3)
Capture 9	4 (33.9)	5 (32.9)	**1 (46.0)**	2 (42.9)	7.5 (25.0)	6 (25.1)	7.5 (25.0)	3 (42.7)
Capture 10	4 (57.1)	3 (57.9)	**1 (64.6)**	2 (63.2)	7.5 (33.3)	6 (54.0)	7.5 (33.3)	5 (56.3)
Capture 11	2 (45.7)	5 (41.9)	**1 (46.6)**	4 (43.1)	7.5 (33.3)	6 (41.3)	7.5 (33.3)	3 (43.8)
Capture 12	4 (39.6)	5 (36.4)	**1 (43.6)**	3 (40.4)	7.5 (33.3)	6 (34.1)	7.5 (33.3)	2 (43.1)
Capture 13	4 (39.5)	3 (43.8)	**1 (57.1)**	2 (55.1)	7.5 (25.0)	6 (25.1)	7.5 (25.0)	5 (33.2)
R_j	4.08	4.38	**1.31**	2.42	7.35	5.77	7.35	3.31

Table 5.3: Algorithm ranks w.r.t. streaming AvDR metric under a 0.5% label budget. Naive Bayes (NB) and Hoeffding tree classifiers (from MOA) appear with either 'split' or 'variable' sampling policies. Table 4.9 declares the 4 sampling/replacement policies for stream SBB. R_j denotes the average rank across all datasets.

Dataset	Stream-GP				Hoeffding		NB	
	Random	Sample	Archive	Both	split	variable	split	variable
Capture 1	5 (32.4)	4 (33.0)	**1 (45.2)**	2 (41.0)	7.5 (25.0)	6 (25.5)	7.5 (25.0)	3 (36.9)
Capture 2	4 (38.0)	5 (37.4)	**1 (48.8)**	2 (46.0)	7.5 (25.0)	6 (34.5)	7.5 (25.0)	3 (40.5)
Capture 3	4 (62.3)	3 (67.9)	**1 (76.2)**	2 (73.3)	7.5 (33.3)	6 (48.1)	7.5 (33.3)	5 (59.2)
Capture 4	5 (45.9)	4 (46.7)	**1 (59.6)**	2 (58.7)	7.5 (33.3)	6 (33.5)	7.5 (33.3)	3 (49.5)
Capture 5	2 (27.2)	8 (25.0)	3 (27.1)	4 (26.2)	6.5 (25.0)	5 (25.3)	6.5 (25.0)	**1 (28.4)**
Capture 6	5 (33.4)	4 (34.7)	**1 (44.5)**	2 (41.6)	7.5 (25.0)	6 (26.0)	7.5 (25.0)	3 (34.7)
Capture 7	3 (26.0)	8 (24.8)	4 (25.2)	7 (25.2)	5.5 (25.0)	2 (26.5)	5.5 (25.0)	**1 (27.0)**
Capture 8	4 (32.4)	5 (30.3)	**1 (55.7)**	2 (54.1)	7.5 (25.0)	6 (26.0)	7.5 (25.0)	3 (43.9)
Capture 9	4 (33.4)	5 (32.2)	**1 (42.5)**	2 (40.6)	7.5 (25.0)	6 (25.0)	7.5 (25.0)	3 (39.6)
Capture 10	3 (56.0)	4 (56.0)	**1 (61.8)**	2 (59.7)	7.5 (33.3)	6 (53.7)	7.5 (33.3)	5 (55.9)
Capture 11	4 (42.7)	6 (37.7)	3 (44.9)	5 (37.9)	7.5 (33.3)	2 (45.4)	7.5 (33.3)	**1 (45.5)**
Capture 12	3 (37.2)	5 (36.9)	2 (38.5)	4 (37.1)	7.5 (33.3)	6 (34.2)	7.5 (33.3)	**1 (40.2)**
Capture 13	4 (38.3)	3 (41.5)	**1 (52.7)**	2 (49.5)	7.5 (25.0)	6 (25.0)	7.5 (25.0)	5 (34.8)
R_j	3.85	4.92	**1.62**	2.92	7.27	5.31	7.27	2.85

Table 5.4: Result of Friedman test χ^2_F and corresponding value for F-distribution F_F. The critical value of $F(7, 84)$ for $\alpha = 0.01$ is 2.86, so the null-hypothesis is rejected in each case.

Label budget	5%	1%	0.5%
χ^2_F	84.9	73.2	65.3
F_F	166.7	49.2	30.49

and datasets [33, 64]. The last row reports the average rank (R_j) calculated based on Eq. 4.12, where it forms the basis for the Friedman test (Eq. 4.13). The Friedman's parameters in the current test case are as follows: $n = 13$ is the number of datasets, and $k = 8$ is the number of algorithms. Thereafter, the null hypothesis is tested by mapping χ^2_F into the F-distribution with $k - 1$ and $(k - 1)(N - 1)$ degrees of freedom using Eq. 4.14.

The null hypothesis (i.e. that the ranks are random) is strongly rejected in each case (Table 5.4). To get a deeper sense of the differences between algorithms, a Nemenyi post-hoc test may be applied to identify which groups of algorithms are performing equivalently. Specifically, if the average algorithm ranks are within the critical difference of $CD = q_\alpha \times \sqrt{\frac{k(k+1)}{6N}}$, then they are deemed equivalent. For $q_\alpha = 0.1$, the critical difference is 2.671. Based on that, the top performing algorithms in all three label budgets are: GP–Archive, GP–Both and NB–variable. More specifically, GP–Archive is by far the most consistently performing model irrespective of dataset or label budget with GP–Both always appearing as the runner up.

A general analysis of the overall performance with the Botnet detection perspective is now considered. Captures 1, 2, 8, and 9 are dominated by port-scanning activities, for which access to the IP addresses and port numbers are assumed typically (limiting the generality of the classifier). Conversely, the proposed Stream-GP classifier is able to detect these activities without the IP address and port number information in an overall streaming AvDR between 54% and 78% (dropping to no less than 42% under the 0.5% label budget). Captures 3, 10 and 11 represent protocol-based attacks so it is assumed that the attack could be identified more easily with flow features. Captures 3 and 10 validate this assumption (high overall streaming AvDR disregards the label budget), whereas Capture 11 returns the overall AvDR in the range of 54% to 44%. Last, Captures 5, 6 and 7 relate to payload attacks (such as operating system vulnerabilities), where the detection mostly relies on the content of the payload, and the flow features alone are not much help in this case. Captures 5 and 7 show a performance level that is slightly better than that of a degenerate classifier[1].

[1]Equivalent to labelling all the data as a single class or $AvDR = \frac{DR(t)}{C*} = 0.25$ where $DR_c(t) = 1$

Conversely, the best detector could identify Capture 6 data with an overall streaming AvDR between 65.5% and 44.5% depending on the label budget.

5.1.2 Detection Rate Dynamics: Comparing the Best Streaming Classifiers

At the next level of performance evaluation, the dynamic properties of the classifiers are reviewed by tracing the *detection rate* of classes *through* the course of the stream. This valuable evaluation provides more insight into the classifiers' ability to interact with the stream and the learning process. As the stream of data is in an ever-changing status, unlike off-line formulation of learning, the models should be able to adapt to the changes in the stream content over the course of the stream. The results provide some insight into whether results with similar overall AvDR also exhibit similar preferences in class detection.

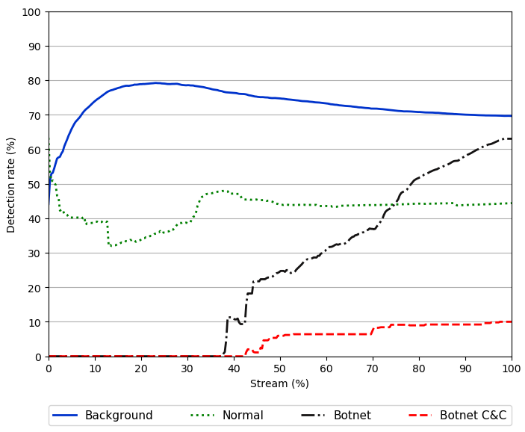
The concentration of this section is on: (i) Captures 5 and 6 (payload attacks) and (ii) Captures 8 and 9 (port scanning) with the two top-performing configurations of Stream-GP and Naive Bayes (Section 5.1.1): GP–Archive and NB–Variable. Captures 5 and 6 contain payload attacks, in which attack-related information is embedded in the payload of the packet and is not available in the flow features used in this research. In the network security field, payload information is usually omitted due to privacy issues. Therefore, these two datasets are good examples of 'hard-to-detect' attacks. Captures 8 and 9 are also related to port-scanning attacks, where in this research the information of IP addresses and Port numbers are omitted from the flow features to provide a more reliable set. It is possibly more convenient to detect these kinds of attacks using the mentioned information, but this also reduces the generality of the classifier. So, these two datasets also raise the question of how well algorithms would perform on them without direct attack information.

Figure 5.2 summarizes the results of the class-wise detection rate, averaged over 20 runs per algorithm, over the course of the stream (Eq. (4.7)) in the specific case of the Capture 5 dataset. Subplots 5.2(a) and 5.2(b) demonstrate the detection rate under a 5% label budget. The NB–Variable model could not detect the minor Botnet C&C class (class 4) at all. Moreover, the detection of important minor classes results in the reduction of the major class.[2] On the other hand, the GP–Archive could detect Botnet C&C after half of the stream has passed with no loss in detection of the major class.

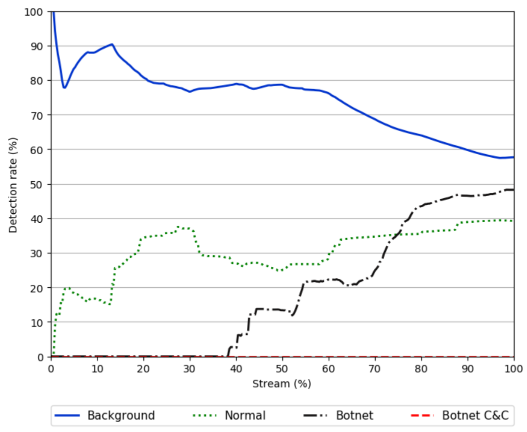
Subplots 5.2(c) and 5.2(d) illustrate the performance of the algorithms for the same dataset but under a 0.5% label budget. NB–Variable continues to detect the major class at

occurs for one class alone and $DR_{i \neq c}(t) = 0$.

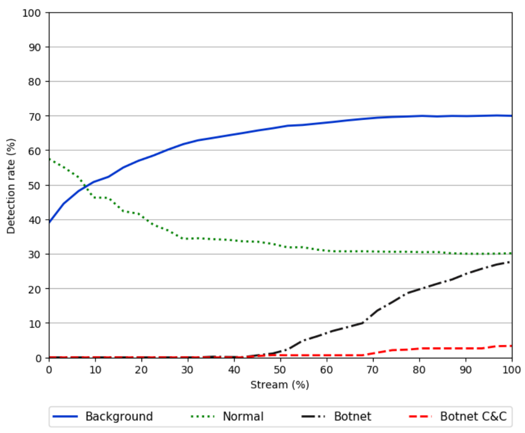
[2]Given the degree of class imbalance in evidence (Table 4.2), the major class is always class 1. The minor classes are always the remaining classes.



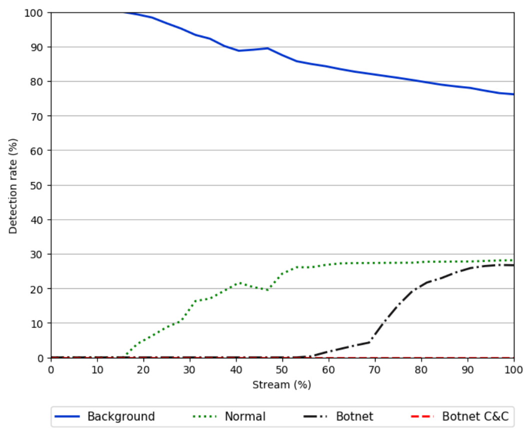
(a) SBB–Archive (5%)



(b) NB–Variable (5%)

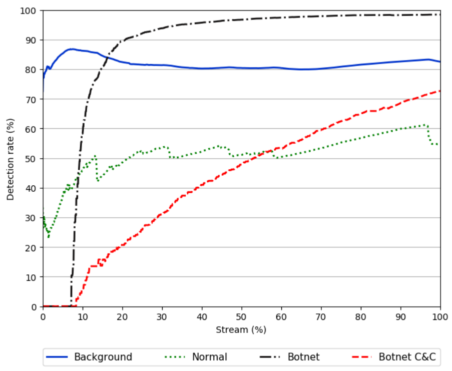


(c) SBB–Archive (0.5%)

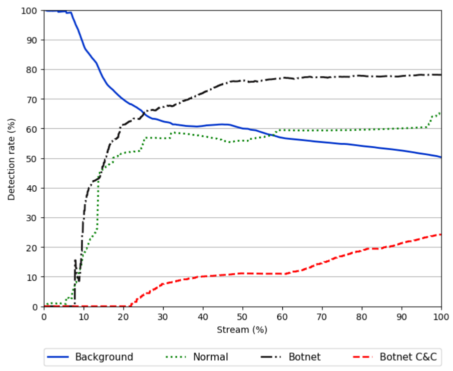


(d) NB–Variable (0.5%)

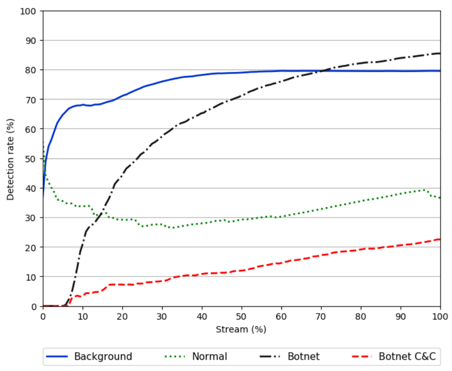
Figure 5.2: Capture 5 class-wise Detection rate through the stream. 5% versus 0.5% label budget.



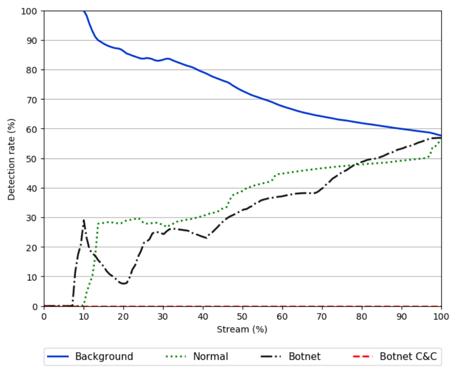
(a) GP–Archive (5%)



(b) NB–Variable (5%)

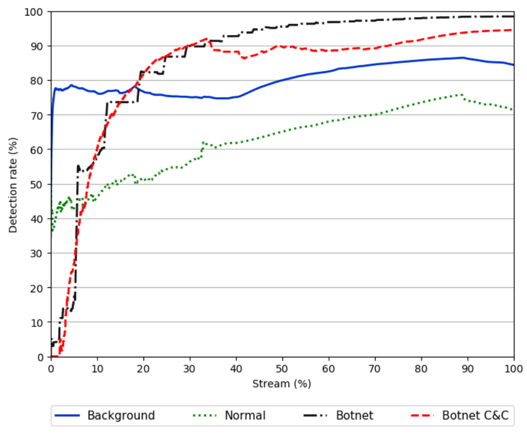


(c) GP–Archive (0.5%)

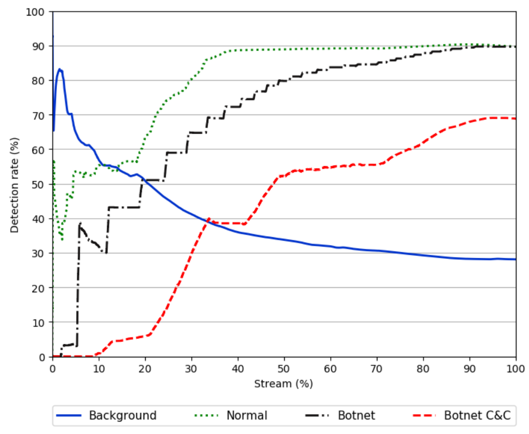


(d) NB–Variable (0.5%)

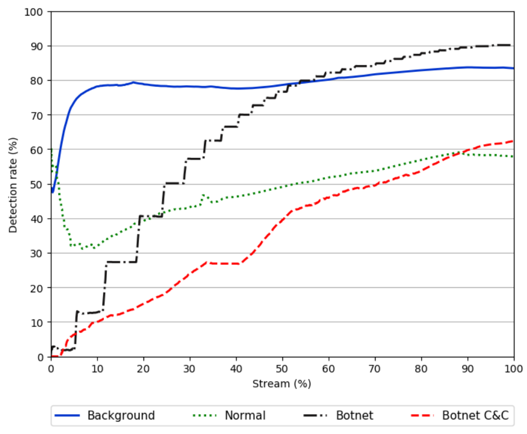
Figure 5.3: Capture 6 class-wise Detection rate through the stream. 5% versus 0.5% label budget.



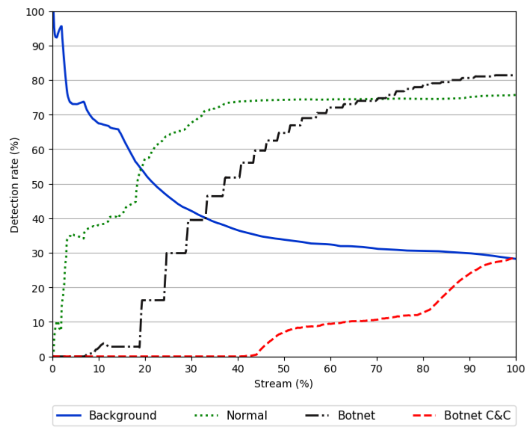
(a) GP–Archive (5%)



(b) NB–Variable (5%)

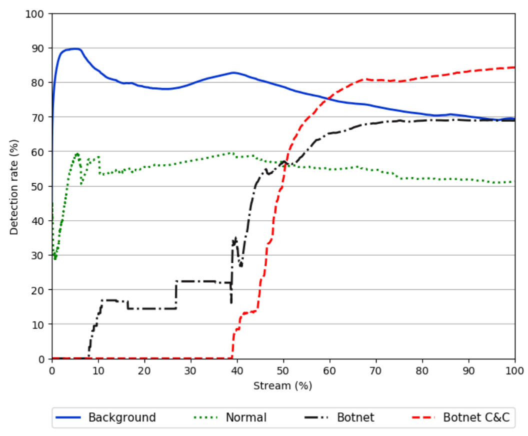


(c) GP–Archive (0.5%)

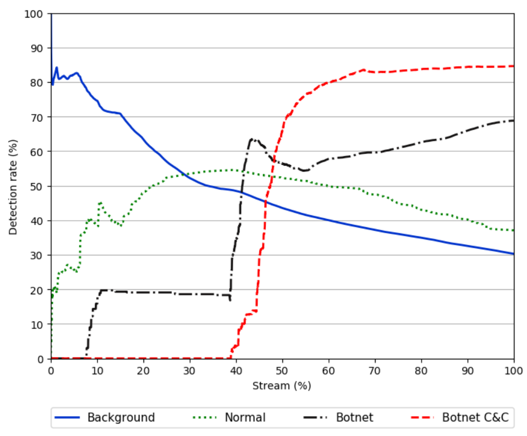


(d) NB–Variable (0.5%)

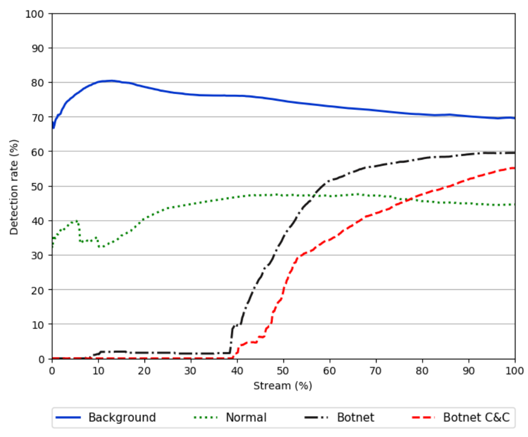
Figure 5.4: Capture 8 class-wise Detection rate through the stream. 5% versus 0.5% label budget.



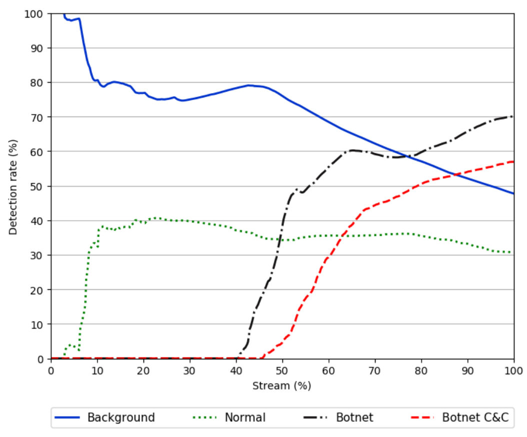
(a) GP–Archive (5%)



(b) NB–Variable (5%)



(c) GP–Archive (0.5%)



(d) NB–Variable (0.5%)

Figure 5.5: Capture 9 class-wise Detection rate through the stream. 5% versus 0.5% label budget.

the expense of losing the detection rate of the important minor classes. Conversely, GP–Archive could detect the minor classes as they appear in the stream, and it was able to detect Botnet C&C but at a lower rate than the 5% label budget. Thus, although NB–Variable is shown to have the higher overall AvDR for a 0.5% label budget under Capture 5 (Table 5.3), it happens through a higher detection rate on the major class. Similar observations hold for the Capture 7 and 11 datasets.

Figure 5.3 demonstrates the per class detection rate for the specific Capture 6 dataset. Like Capture 5, the GP–Archive could detect all classes throughout the course of the stream at varying degrees irrespective of the label budget; NB–Variable was able to do so only under a 5% label budget. In addition, it seems that NB–variable first detects every class as the major class, and improvement in the detection of minor classes results in the reduction of major class detection. This pattern is consistent in the NB–Variable framework regardless of the dataset.

This general property of the NB-Variable is not evident in the GP–Archive framework. Each class in the GP–Archive once detected would not directly affect the detection rate of other classes. The independence of classes in detection is a result of the following:

- *Population-based solution:* Each solution consists of several learners, each of which is responsible for the detection of a single class. Therefore, they are trying to maximize their performance based only on their own class regardless of what is going on with the other learners. This decentralized structure makes the classes' detection rates independent of each other, and an increase/reduction in one class detection rate would not affect the others.

- *Robust measurement metric:* A proper metric is needed to evaluate the performances of the nominees for the champion position. This metric should be able to consider the challenging properties of the streaming data and evaluate solutions based on how well they perform under these challenges (Section 3.5).

- *Ability to balance the DS:* The streaming data are highly imbalanced. If the same distribution is applied to the training data, the learning leads to better detection of the major classes and, in contrast, discards the detection of minor classes. The policies designed and applied in Section 5.1.3 help to force the DS to be more balanced.

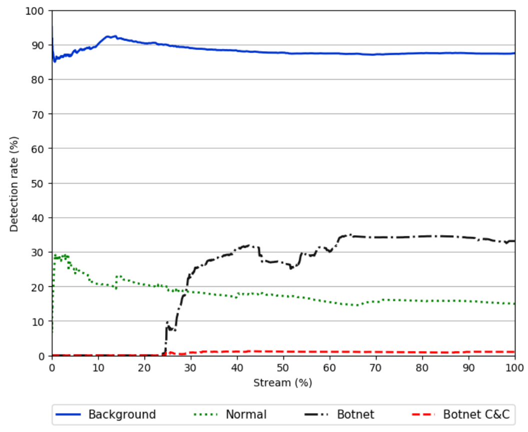
Similar observations carry over to the case of Capture 8 (Figure 5.4) and Capture 9 (Figure 5.5). Comparison of the detection rate plots to the distribution of the classes throughout the course of the stream, (Figure 5.9, Section 5.1.6), demonstrates that Stream-GP is capable of detecting minor classes as soon as they appear in the stream. As for Capture 8, Botnet C&C starts with a burst at the beginning of the stream and then changes to an intermittent basis later. Figure 5.4 illustrates that Stream-GP was successful to detect it from the first occurrence, even in the case of the 0.5% label budget. Another interesting behaviour evident in AvDR and the distribution plots for Capture 8 is related to the Botnet class. As is seen in Figure 5.9, the Botnet traffic is happening at an interval mode, which begins in a burst but changes to a lower frequency in a longer duration. This botnet appearance in the stream results in a step-wise detection rate (shown in Figure 5.4) that improves over time as both algorithms, Stream-GP and NB, become better at sampling/detection.

From a network security perspective, although the AvDR metric shows the order of 54% to 78% in the case of Capture 6, 8 and 9 (Section 5.1.1), the Botnet C&C (master-to-slave) communication ended up with a Detection rate of 70 to 95% by the 'end' of the stream in the GP–Archive case. The malicious behaviours are definitely scarce (Table 4.2) and transitory (5.1.6), but they are also more likely related to the operating system and/or application vulnerabilities attacks, which are reliant on payload data. In this research, the only available information is reliant on the flow features, and the payload is completely unavailable. Therefore, the detection is done solely based on the 'fingerprints' left on the flow features to trace the botnet behaviour.

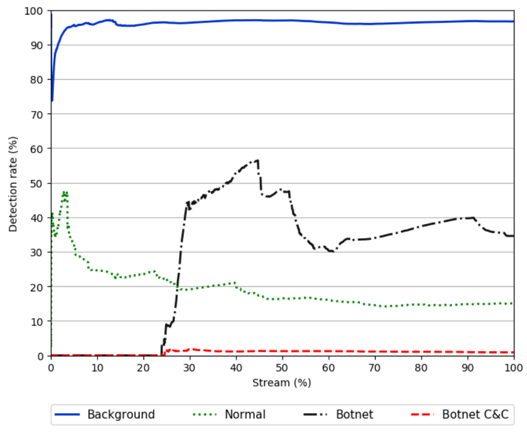
5.1.3 Detection Rate Dynamics: Stream-GP Sampling and Archiving Policies

The four variations of sampling/archiving policies (Table 4.9) in Stream-GP and their contributions to the algorithms' performance is the subject of this section. In Section 5.1.1, the overall ranking for the four variants were identified as: Archive > Both > Random > Sample where 'Random' represents the control/baseline parameterization. The interest is how each of the sampling and archiving policies contributes to this result. For this purpose, the Capture 1 dataset is selected to be investigated more in depth. Capture 1 introduces classes at different phases of the stream (Figure 5.9), thus there is a need for an 'Archiving' policy to create 'new' categories in the data subset (Figure 3.1), whereas the sampling policy needs to detect the changes at first. Capture 1 consists of several port-scanning activities that are not straightforward to detect without the port number information (the current case).

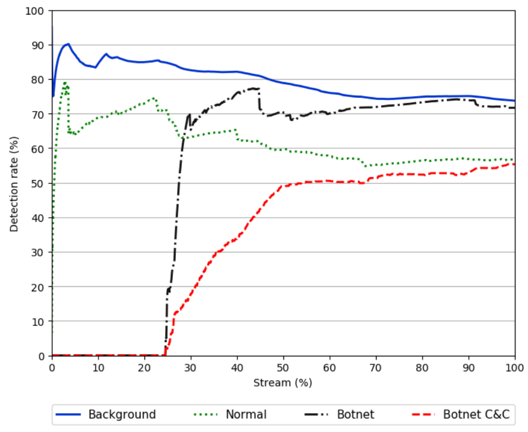
Figure 5.6 summarizes the per-class detection rate for Capture 1 under a 5% label



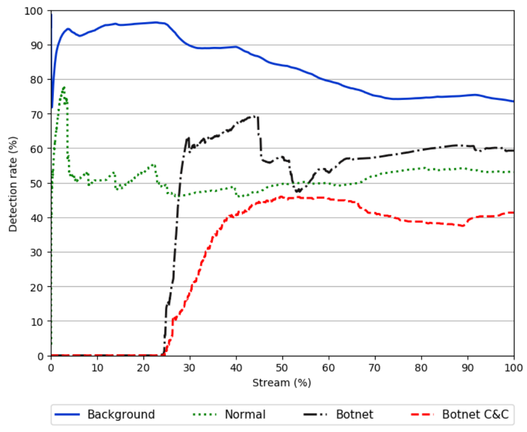
(a) Random



(b) Sample



(c) Archive



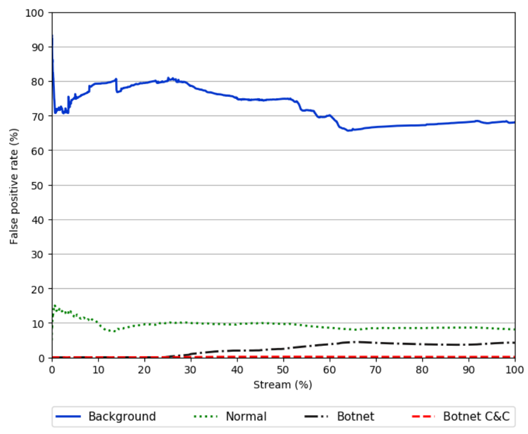
(d) Both

Figure 5.6: Class-wise Detection rate for Stream-GP sampling and archiving policies on the Capture 1 dataset at a 5.0% label budget.

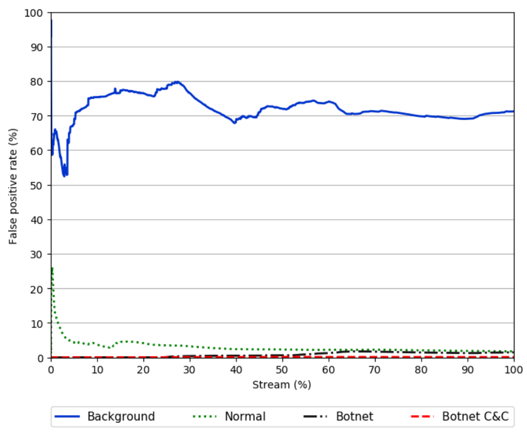
budget. The control parameterization of 'Random' samples from the stream window, $W(i)$, with uniform probability and the archiving also replaces the records in the Data Subset (DS) uniformly (Table 4.9). Comparing Subplot 5.6(a) to the distribution of minor classes (Figure 5.9) results in the following observations:

1. The underlying details of each class: Background class (class 1) – the major class – is detected strongly, and Botnet C&C class (class 4) – the least frequent – very rarely.

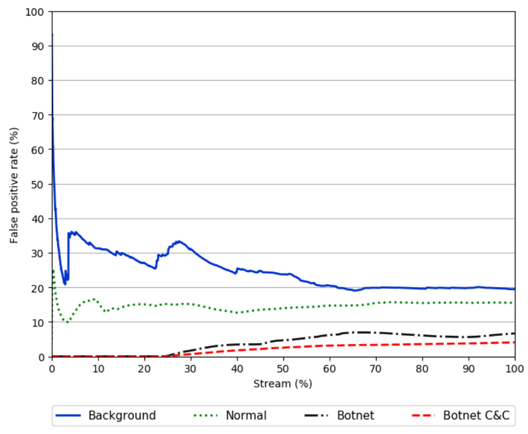
2. When classes appear for the first time in the stream: Botnet class (class 3) appears later in the stream than Normal class (class 2).



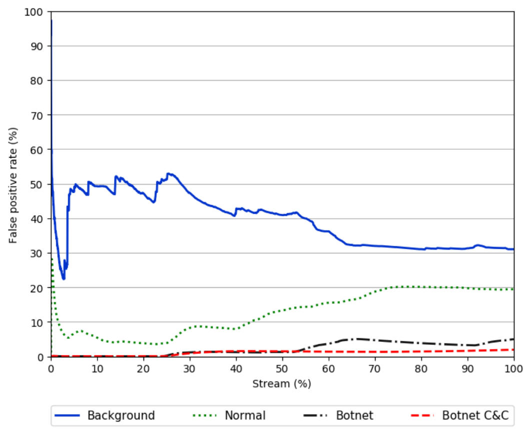
(a) Random



(b) Sample

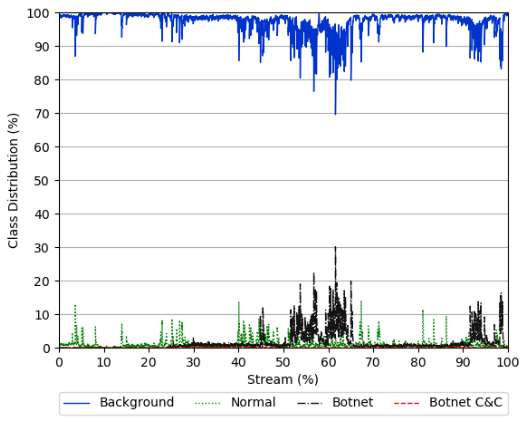


(c) Archive

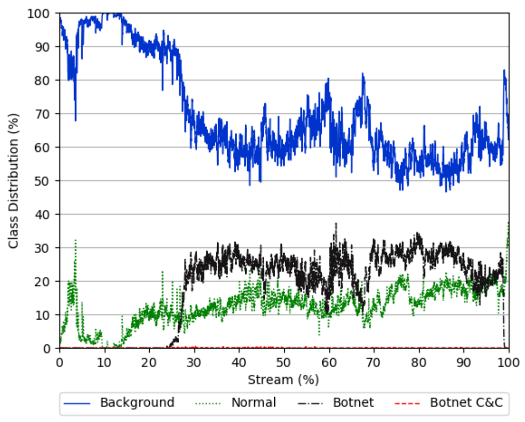


(d) Both

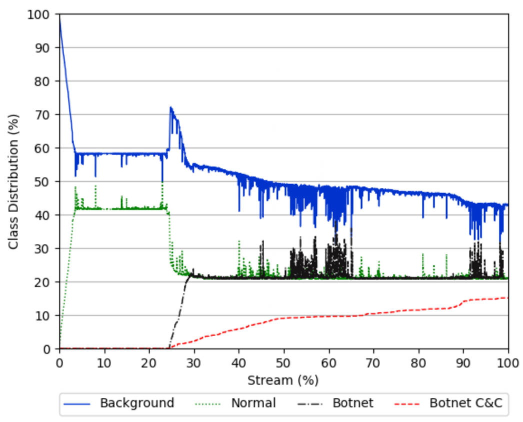
Figure 5.7: Class-wise False positive rate for Stream-GP sampling and archiving policies on the Capture 1 dataset at a 5.0% label budget.



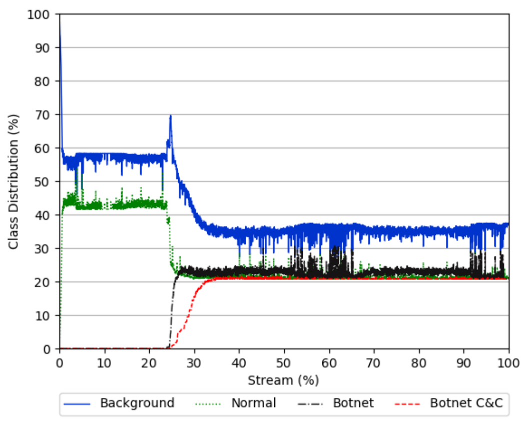
(a) Random



(b) Sample



(c) Archive



(d) Both

Figure 5.8: Typical Distribution of classes present in the Data Subset for Stream-GP sampling and archiving policies on the Capture 1 dataset at a 5.0% label budget.

The impact of introducing the *biased sampling* while the archiving policy remained random, is reported in subplot 5.6(b). The observation shows that the Background class is detected more strongly, and the Botnet class is now detected more effectively despite its late appearance in the stream. The Botnet class has seen a small reduction in the detection rate compared to the 'Random' policy.

Examining the *biased archiving policy*, with a random sampling policy (Table 4.9), illustrates its effectiveness in improving the minor classes' detection rates, as demonstrated in subplot 5.6(c). This indicates that the Archive policy is effectively working in the network security area, as it could detect minor classes at an acceptable detection rate. Some decay in Background detection is apparent, which may be due to sharing the space in the data subset with minor classes as they appear later in the stream. By the 'end' of the stream, the Botnet and Botnet C&C are identified with detection rates of $\approx$70% and $\approx$55%, respectively.

Ultimately, the conjugated biased sampling and archiving policy is demonstrated in subplot 5.6(d). In comparison to the best policy combination, the 'Archive', Normal and Botnet classes declined. The reason for this behaviour could be investigated by checking the status of the classes distribution retained in the data subset throughout the stream.

Figure 5.8 refers to the distribution of classes in the data subset during the stream. The content of the data subset is a key point in the learning infrastructure in the active learning scheme, here Stream-GP, where the fitness evaluation and champion selection is happening relative to it. Therefore, the distribution of classes in the data subset plays an important role in defining the overall Stream-GP performance. All policies start with a dominant Background class (major class); however, GP–Rnd and the GP–Sample policies fail to guarantee the presence of minor classes in sufficient quantity (subplots 5.8(a) and 5.8(b)); whereas, both the GP–Archive and GP–Both policies achieve this (subplots 5.8(c) and 5.8(d)).

Figure 5.7 demonstrates the False positive rate for each sampling / archiving policy set. The high detection rate in GP–Random and GP–Sample algorithms is gained with a corresponding high false positive rate as well. This leads to the conclusion that these two policies intend to label flows to the major class, Background, based on their lack of information on minor classes based on the data subset content (subplots 5.8(a) and 5.8(b)). GP–Archive and GP–Both algorithms have low false positive rates in the Botnet and Botnet C&C classes, where GP–Archive has a relatively low false positive rate for the Background class as well. However, discussion on the Background False Positive rate is not completely reliable, as it may actually represent Normal or Botnet behaviours due to its unknown

Table 5.5: Ranks for the Botnet class streaming AvDR alone.

Label Budget	Stream-GP				Hoeffding		NB	
	Random	Sample	Archive	Both	split	variable	split	variable
5%	3.62	4.38	1.08	2.69	7.23	6.15	7.23	3.62
1%	3.77	4.38	1.00	2.62	7.31	6.27	7.31	3.35
0.5%	3.62	4.38	1.08	2.69	7.23	6.15	7.23	3.62

Table 5.6: χ^2_F and F_F values for different label budgets. Assuming $\alpha = 0.01$ returns a critical value of $F(7, 84) < 2.86$, the null-hypothesis (of random ranking) is rejected.

Label Budget	Statistic	
	χ^2_F	F_F
5%	79.60	83.86
1%	79.25	80.93
0.5%	74.12	52.72

identity.

It is notable that the GP–Both parameterization is by far the most consistent. However, GP–Archive gradually introduces a change to the distribution of class representation. As the GP–Archive policy is ranked as the best classifier in the test cases, it seems that gradual updating of the records in the data subset is the key to the performance differences.

5.1.4 Capacity for Detecting Botnet Signals

The performance of the Stream-GP and MOA comparator algorithms in Botnet detection under different label budgets is analyzed in this section. The Botnet class is an important class, and it is costly to miss-classify from a network security perspective. Miss-classification of this class leads to probable destructive consequences that are hard to recover from. To identify the performance of the algorithms on this specific class, a similar significance test (Section 5.1.1) is applied only to the Botnet class, and a ranking table is prepared.

The final ranking of the algorithms on the specific Botnet class based on streaming AvDR for all class labels is summed up in Table 5.5. Remember that Botnet and Botnet C&C classes are aggregated in to one class in Captures 3, 4, 10, 11 and 12. GP–Archive and GP–Both are once again the two top algorithms in the list and could detect the Botnet class better than all the other algorithms.

Table 5.6 summarizes the Friedman's Test statistics indicated as (χ^2_F) and the corresponding F-distribution (F_F) value, which demonstrates that the null hypothesis is comfortably rejected. The degree of freedom is still unchanged (from the earlier analysis), so

Table 5.7: Ranks for Botnet C&C class streaming AvDR alone.

Label Budget	Stream-GP				Hoeffding		NB	
	Random	Sample	Archive	Both	split	variable	split	variable
5%	4.81	4.27	1.46	2.15	6.92	6.35	6.92	3.12
1%	4.00	4.38	2.00	2.38	6.69	6.23	6.69	3.62
0.5%	3.62	3.73	1.38	2.27	6.96	6.54	6.96	4.54

Table 5.8: χ_F^2 and F_F values for different label budgets. Assuming an $\alpha = 0.01$ returns a critical value of $F(7, 84) < 2.86$, the null-hypothesis (of random ranking) is rejected.

Label Budget	Statistic	
	χ_F^2	F_F
5%	69.23	38.16
1%	52.82	16.60
0.5%	70.05	40.13

the critical difference is also unchanged (2.671). The Nemenyi post-hoc test applied on the highest ranked model implies that GP–Archive, GP–Both and NB–Variable are statistically independent (of the remaining six models) at a confidence of $q = 0.1$ for a 1% label budget. The results on other label budgets are still consistent for these three algorithms. Moreover, the low ranking of Hoeffding and NB–Split is a general reflection of their inability to detect the minor class in a significant number of Capture datasets.

5.1.5 Capacity for Detecting Botnet C&C Signals

The analysis of the Stream-GP and MOA comparator algorithms on the least frequently occurring class, Botnet C&C, under different label budgets is given in this section. The Botnet C&C class, i.e. Botnet Command and Control, refers to the indication of the start of a malicious botnet behaviour where commands are transferred from Bot master(s) to slave(s). The ability to perform well in detection of the Botnet C&C class leads to the early prevention of such attacks. An investigation of similar to that in Section 5.1.1 is taking place here with specific focus on the botnet C&C class to check if the same ranking applies.

The final ranking of all algorithms based on the minor class's streaming AvDR for each label budget is summarized in Table 5.7. Both GP–Archive and GP–Both formulations had the highest ranks as well. This success is reliant on the fact that these two algorithms are able to effectively balance the content of the data subset, i.e. the observation in Section 5.1.3.

Table 5.8 indicates the Friedman test statistic (χ_F^2) and the corresponding F-distribution

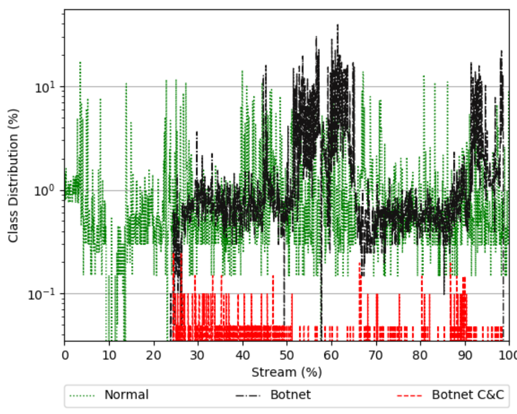
(F_F) value which demonstrates that null hypothesis is comfortably rejected. The degree of freedom is still unchanged (from the earlier analysis), so the critical difference is also unchanged (2.671). The Nemenyi post-hoc test applied to the highest-ranked model implies that GP–Archive, GP–Both and NB–Variable are statistically independent (of the remaining six models) at a confidence of $q = 0.1$ for a label budget of 5%. As the label budget decreases to 1% and 0.5%, these three models continue to be consistently identified. the low ranking of Hoeffding and NB–Split is still evident in the detection of the minor class. This continues to emphasize the incapability of these algorithms to detect the minor classes.

5.1.6 Distribution of Minor Classes

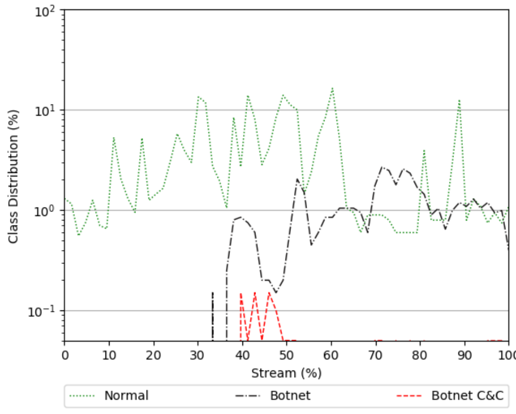
The distribution of minor classes (all classes except the Background class) that illustrates the behavioural properties of the streaming botnet detection task for the CTU-13 datasets, e.g. Captures 1, 5, 6, 8 and 9, is summarized in Figure 5.9. At any point, the value of each class represents the sum of that class presentation in the unique non-overlapping window location for a 1% label budget.

The figure explicitly demonstrates that there is no common ‘behavioural’ property for the Botnet C&C class with the lowest frequency overall, and it mostly has a very burst mode nature.

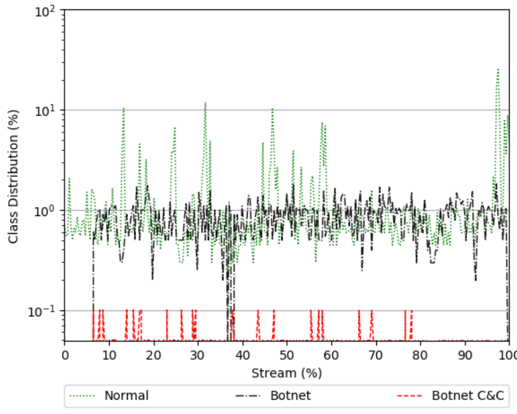
Normal Class (representing data corresponding to the CTU ‘normal filters’) appears in all datasets throughout, but in most cases, and Capture 5 and Capture 6 specifically, it disappears and appears at particular points. The Botnet class is interesting in Capture 8 as it goes on and off throughout the stream in non-periodic intervals. Although each class represents a definitive class (Background/Normal/Botnet/C&C), but they actually represent multiple different behaviours. For instance, Capture 1, 2 and 9 represent Neris Botnet, thus the Botnet class may contain any combination of Spam, Click Fraud or Scanning activities. See Section 4.1.1 for a summary of the types of malicious behaviours present in each stream.



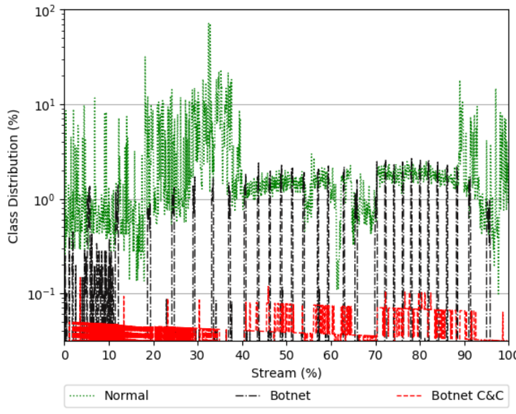
(a) Capture 1



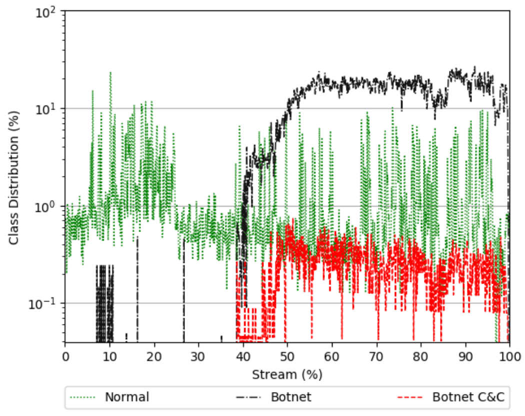
(b) Capture 5



(c) Capture 6



(d) Capture 8



(e) Capture 9

Figure 5.9: Distribution of minor classes over the course of the stream for the five capture datasets appearing in Sections 5.1.2 and 5.1.3. Note the use of a log scale and the colour coding, which corresponds to that adopted for the original Stream DR figures. Background class is omitted for clarity (always 90 to 99%). Normal class represents 'normal' traffic corresponding to the CTU filters, represents Botnet and represents Botnet C&C. The log scale also implies that 10^{-2} is synonymous with zero content (e.g. the earliest that Botnet C&C appears at the 40% point in Captures 5 and 9).

5.1.7 Real-time Operation

The common perception of GP is that it is not good to be used in real-time operations due to its computational overhead. In this application domain, the network traffic packets are first pre-processed into traffic flows using network flow exporters such as Argus [49]. Each flow is aggregated information of a couple of network packets that are common in the 5-tuple information: source/destination IP, protocol, and source/destination port. The number of packets in each flow depends on the functionality of the service/application (10's to 100's packets per flow). CISCO[3] defines an upper bound of 600ms as the time interval for the completion of any flow; however, this represents a worst-case figure, and the inter-arrival time of packets is a function of network topology and load.

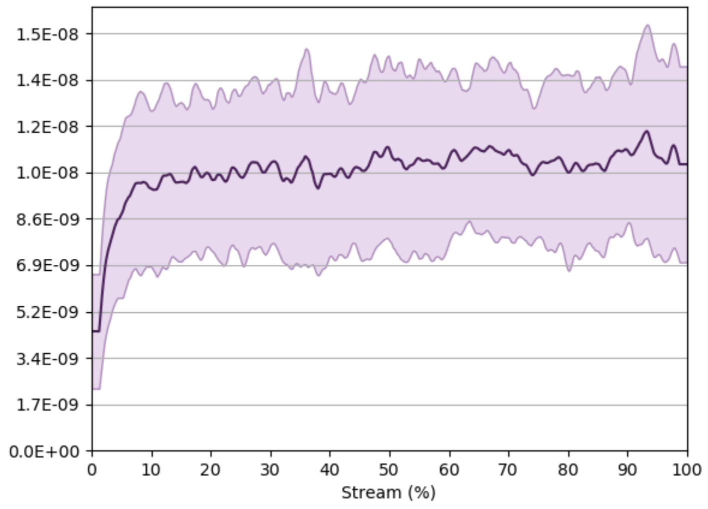
The capacity of Stream-GP in a real-time operation will be investigated from the following two perspectives:

- The time a Stream-GP champion takes to suggest its predicted class label for each flow record (anytime operation).
- The time that fitness evaluation takes to be completed after each data subset update.

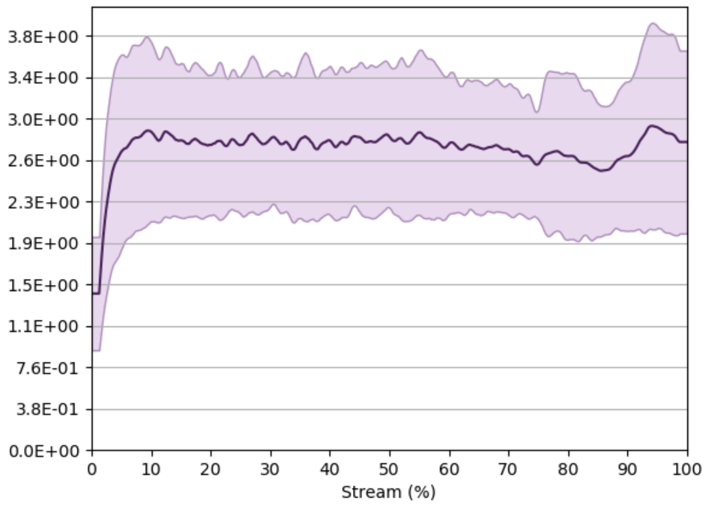
In the latter case, based on the parameterization assumed in this research (Table 4.10), the fitness evaluation would evaluate $P_{size} = 120$ teams on $DS = 120$ flow records for $\tau = 5$ generations and identify the champion individual. However, only twenty training records are added to the data subset in each individual window location, and just twenty teams are replaced at each generation ($Gap = Tgap = 20$); therefore, the computational cost *per window location* is in the order of $20 \times 20 \times 5$ evaluations.

Figure 5.10 summarizes the execution time under each of these conditions for a common Intel i5 CPU (2.67GHz, 48GB RAM). The plot demonstrates the mean and variance of 20 runs over Capture 3, which is the dataset with the largest cardinality, i.e. it should be clear whether the computational costs stabilize or not. In all cases, this reflects a code base that executes as a *single thread*.

[3]`https://www.cisco.com/c/en_ca/index.html`



(a) Champion



(b) Fitness evaluation

Figure 5.10: Wall clock time for (a) champion individual to make predictions and (b) fitness evaluation to update the content of the population on a new non-overlapping window location for the Capture 3 dataset on a 2.67 GHz CPU.

The average execution time for the champion is ≈10 ns (Figure 5.10(a)), implying an average (single threaded) throughput of 6×10^7 flows per second. On the other hand, the time taken to update the champion predictor is between 2.7 to 3.8 seconds.[4] Furthermore,

[4]Adopting a multi-threaded operation, say, eight threads, could potentially reduce this to between

the biggest impact on the time to identify a champion is the time it takes the human expert to provide labels for the *Gap* records associated with each (non-overlapping) window location $W(i)$. However, this does not have any effect on the ability of the current champion classifier to provide labels, and it would be synonymous with the current practice for deploying updates to 'signature-based' detectors.

5.1.8 Summary

Active learning under the streaming data context decouples the ML learning process from the raw throughput of the stream and provides the opportunity to handle the distribution of classes for model building. This property of active learning alongside GP would provide a great combination to be applied on a stream of data, especially when low label budgets are available. The key point of this thesis is to introduce the appropriate interaction between sampling and archiving policies.

- GP–Random: represents the control base for the policy combinations. The distribution of the data subset is indicative of the actual distribution of the stream, which specifically impacts the anytime classifier's accuracy.

- GP–Sample: is unable to retain 'useful' queries from the minor classes in the data subset. This could happen because the selected champion (anytime) classifier always performs based on the two or three most frequently occurring classes in the data subset. Therefore, it loses its sensibility to the smallest class(es) where the penalty of miss-classifying the class(es) becomes typically low.

- GP–Archive: emphasizes balancing the data subset based on biased record replacement from the data subset. This appears to provide the best balance of keeping the major class exemplars up to date while identifying instances of the minor class(es).

- GP–Both: combines the targeted queries on the stream for labelling with biased record replacement in the data subset. This introduces an aggressive algorithm in promoting the minor class(es) in the data subset at the expense of a reduction in the major class(es) performance.

The comparison of four alternative formulations explicitly designed to operate under label budgets, the Stream-GP policies of GP–Archive and GP–Both are generally ranked 1st and

0.34 to 0.5 seconds.

2$^{\text{nd}}$, respectively. Previous results on the entirely artificial stream data resulted in the same ranking [69].

In pursuing a Botnet detection task, the potential for addressing a network analysis is illustrated under particularly challenging conditions, e.g. class imbalance, the high cost of labelling, anytime operation. Classifying the minor class in Botnet detection, Botnet C&C, is costly and results in constructing a streaming classifier under a low label budget and class imbalance becomes significantly challenging. For the *first* time, this thesis addresses the mentioned two limiting constraints at the same time under the streaming classification. The GP–Archive policy is shown to be an effective algorithm for working in GP streaming with regard to the mentioned challenges. In addition, the detection of Botnet behaviours improves over the course of the stream, resulting in the better detection of Botnet and Botnet C&C classes, even though they might appear at a rate of less than 1% of the total stream. To minimize the effect of adversarial attacks against learning algorithms, a human expert is in the loop to suggest the true labels. However, the streaming algorithm itself identifies which exemplars are to be labelled. Also, the anytime classifier can make predictions regarding the class labels, which prioritizes the records for the expert to label first.

GP–Archive and GP–Both algorithms from Stream-GP are selected as the best performing algorithms to be evaluated on other applications later in this thesis. Similarly, the Naive Bayes and Hoeffding Tree algorithms with Variable Uncertainty sampling policy from MOA comaparator tool are employed for further evaluations.

5.2 Botnet Detection in Multi-bot Network Traffic

In this section, the Stream-GP and the MOA comparator algorithms are evaluated in a more realistic network traffic scenario, *multi-bot* traffic. In the multi-bot scenario, different bots that are distinct in operational modes appear throughout the stream. Several varieties arise when various bots appear in the stream, such as: 1) The structure of the botnets, centralized vs. peer-to-peer. 2) The command and control (C&C) protocol between the bot and the botmaster, HTTP, DNS, etc. 3) The type of attacks, DoS, Click-Fraud, Spam, etc. 4) The procedure used to perform the attack.

Sometimes a single bot switches between the different attack types it could provide. Altogether, this scenario provides a great opportunity to benchmark the system under a more challenging network traffic stream and determine if the system can remember the seen bot behaviour when it appears later in the stream. In Section 5.2.1, the overall performance of the highest ranked Stream-GP and MOA comparator algorithms is quantified over the huge mixed Botnet dataset, CTU13-mixed, which created based on mixing the thirteen Botnet datasets (Table 4.3), in regard to 5.0% and 0.5% label budgets. Section 5.2.2 overviews the underlying behaviour of algorithms by tracking their detection rates throughout the course of the stream. Section 5.2.3 and Section 5.2.4 identify the performance of the algorithms on the specific Botnet and Botnet C&C classes respectively. In Section 5.2.5, the overview of the flow features selected by the Stream-GP champion throughout the stream is investigated. Section 5.2.6 demonstrates the average time taken to evaluate the fitness function and predict the label for an exemplar from the stream by the Stream-GP champion. Ultimately, Section 5.2.7 concludes the investigations on algorithms' behaviours and reactions on the multi-bot scenario.

5.2.1 Overall Performance Evaluation

The four top-ranked Stream-GP and MOA comparator algorithms are selected for evaluation purposes based on the observations in Section 5.1.1. A novel Stream-GP configuration, GP–Hybrid, is also introduced and added to the list for evaluation. All algorithms are deployed for 20 independent runs with two label budgets, 5% and 0.5% ($\beta = \{0.05, 0.005\}$). The streaming AvDR (Section 4.4) is used as the metric for evaluation of the algorithms. Table 5.9 details the median of the AvDR results for each algorithm's runs under both label budgets, 0.5% and 5%. Then, a nonparametric Mann-Whitney U test is performed on the mean values to verify the statistical significance between each pair of algorithms. Violin plots (Figure 5.11) for the 5% label budget confirmed that the distribution of $AvDR$ does not conform to a normal distribution. The same observation is valid for the 0.5% label budget as well. The violin plot also visually demonstrates the distinctive differences on algorithms' performances. To provide numerical values to reflect the visual performance distinction, p-values for each pair of algorithms are computed in Tables 5.10 and 5.11 separately for the 5% and 0.5% label budgets.

The overall performance (Table 5.9) explicitly shows that the Stream-GP algorithms perform better in comparison to the MOA comparator algorithms under both label budgets. The observation also demonstrates that there is a $\approx$15% increase in the performance of the Stream-GP algorithms when the label budget increases. On the other hand, the same conclusion could be made on the MOA algorithm, but this time with a $\approx$10% increase.

Table 5.9: Median values w.r.t. streaming AvDR metric, under 0.5% and 5% label budgets. Three Stream-GP configurations are declared in Table 4.9. Hoeffding Tree and Naive Bayes (from MOA) appear with the 'variable' sampling policy, the *CTU13-mixed* dataset

	Algorithms	AvDR Median (0.5%)	AvDR Median (5%)
Stream-GP	Archive	62.3%	78.1%
	Hybrid	63.4%	78.4%
	Both	62%	76.4%
MOA	Hoeffding	33%	40.4%
	NB	47.2%	56.9%

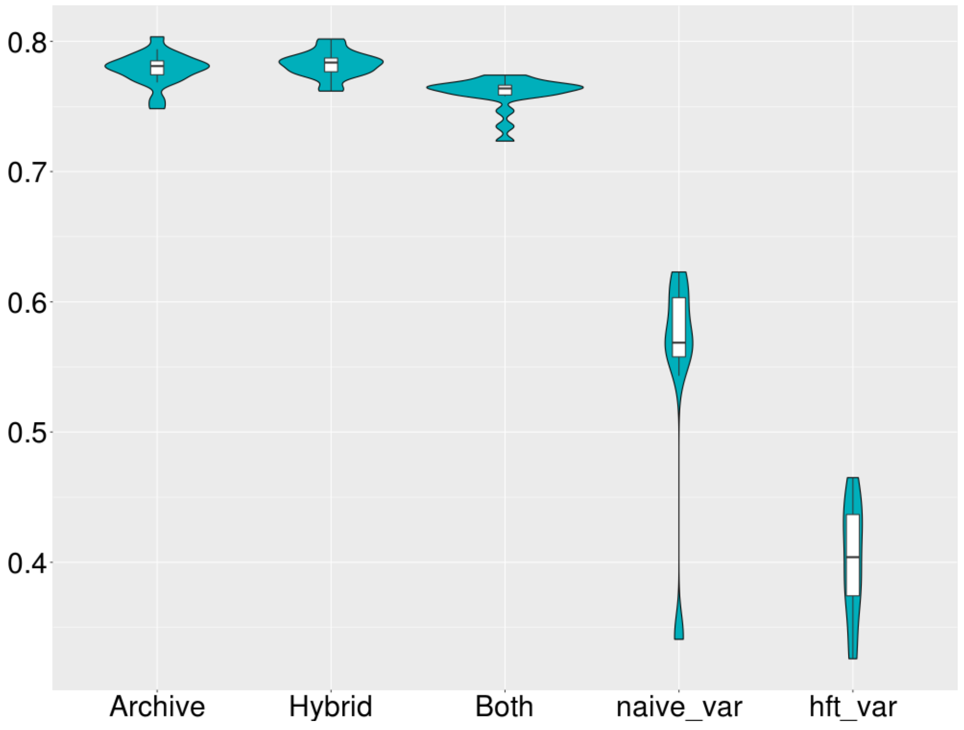


Figure 5.11: Violin plots for a 5% label budget based on streaming AvDR metric (Average of 20 runs)

The null hypothesis (i.e. that the distribution of results is equal) is strongly rejected between the Stream-GP and MOA comparator algorithms (Tables 5.10 and 5.11). This implies that the Stream-GP algorithms perform significantly better than the MOA algorithms. This result illustrates the effectiveness of Stream-GP variants in detecting multiple botnets throughout the course of the stream using the knowledge of previously seen behaviour repeated in the stream. GP–Archive versus GP–Hybrid could not be rejected under both label budgets and versus GP–Both could not be rejected under the 0.5% label budget (Table

Table 5.10: p-value from Mann-Whitney U test for Stream-GP and MOA algorithms, for the 5.0% label budget

Alogrithms					
Stream-GP	Archive	Archive			
	Hybrid	0.37	Hybrid		
	Both	3.4×10^{-6}	8.3×10^{-8}	Both	
MOA	Hoeffding	0.0	0.0	0.0	Hoeffding
	NB	0.0	0.0	0.0	1.8×10^{-5}

Table 5.11: p-value from the Mann-Whitney U test for the Stream-GP and MOA algorithms for the 0.5% label budget

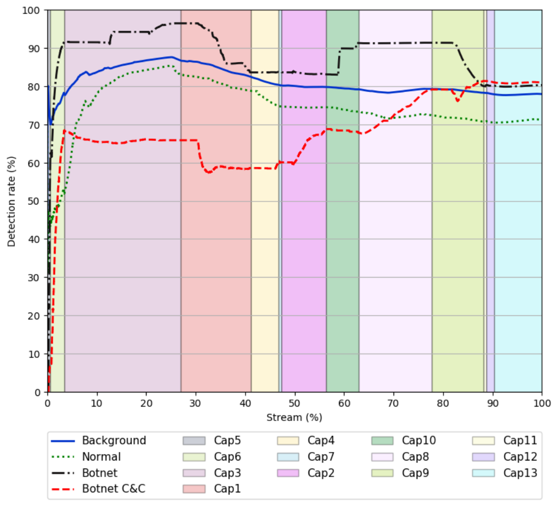
Algorithms					
Stream-GP	Archive	Archive			
	Hybrid	0.68	Hybrid		
	Both	0.11	0.04	Both	
MOA	Hoeffding	0.0	0.0	0.0	Hoeffding
	NB	0.0	0.0	0.0	0.0

5.11), although the p-value is clearly decreasing. There is at least a 20% difference between the performance of the Stream-GP and MOA algorithms under the 5% label budget and 15% under the 0.5% label budget (Table 5.9).

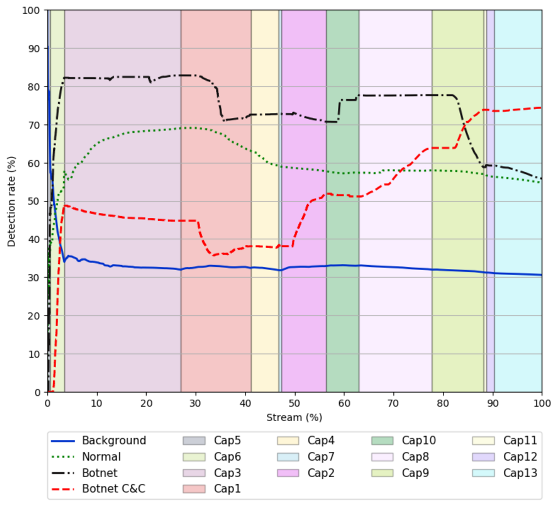
The comparison of overall performances, in the case of the multi-bot dataset (Tables 5.10 and 5.11) with that of separate botnet datasets (Tables 5.1 and 5.3) indicates that Stream-GP algorithms could keep up the streaming AvDR to the highest detection rates in the case of single-bot datasets. This observation suggests that the complexity of the dataset does not affect the Stream-GP algorithm to change, which implicitly relies on the ability of the algorithms to remember the learned behaviours.

5.2.2 Detection Rate Dynamics: Comparing the Best Streaming Classifiers

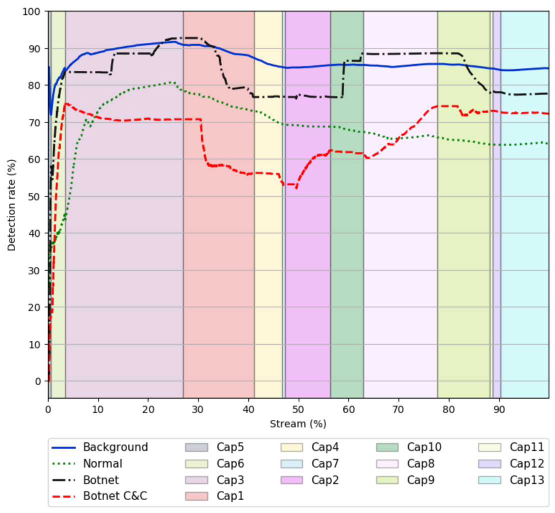
The behaviour of the classifiers throughout the course of the stream in the presence of multi-bot activities, is of interest in this section. For this purpose, the algorithms from Stream-GP and Naive Bayes from MOA are chosen for the comparison of behaviours. Thus, the streaming class-wise detection rate, streaming AvDR (Eq. 4.7), is plotted as a function of time (Figure 5.12). The plots are superimposed over segments of coloured backgrounds. Each segment is representative of one CTU-13 Capture, i.e. one single bot behaviour. Moreover, the transition from one type of botnet to another is identified by a colour change.



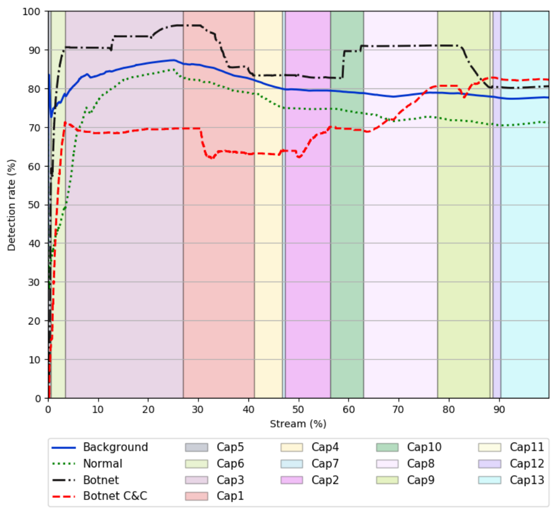
(a) GP–Archive



(b) NB–Variable

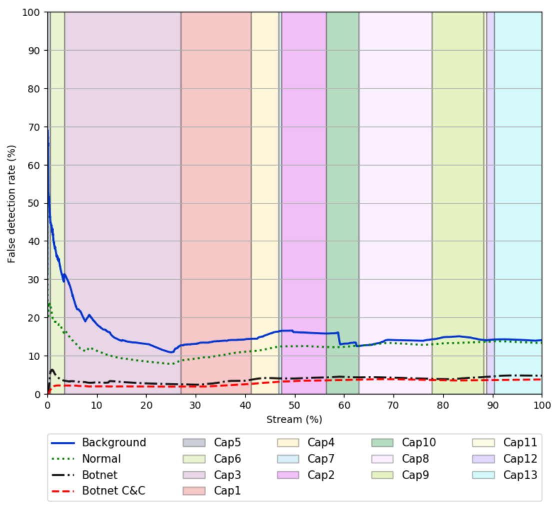


(c) GP–Both

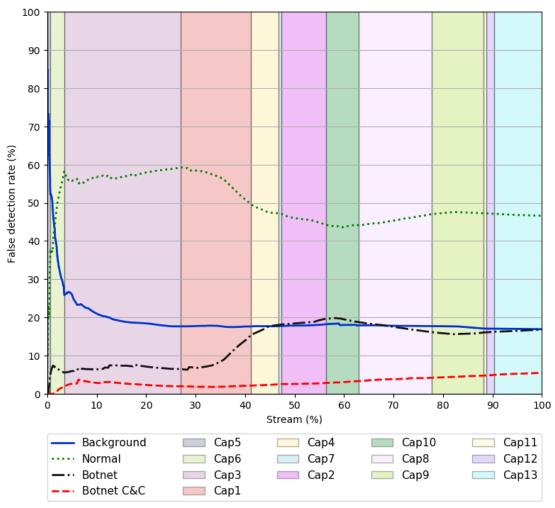


(d) GP–Hybrid

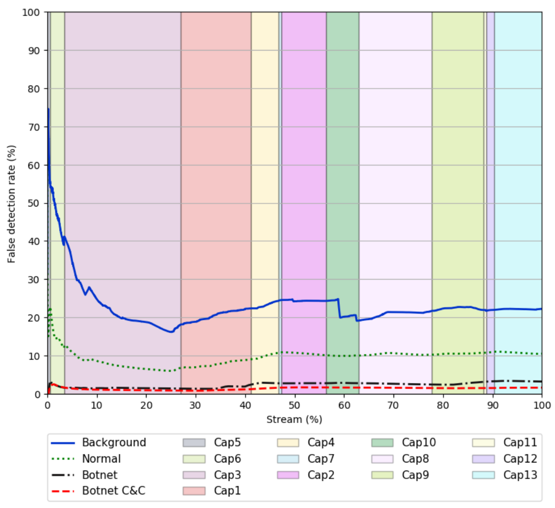
Figure 5.12: Class-wise Detection rate through the stream at a 5% label budget based on streaming AvDR metric (Average of 20 runs)



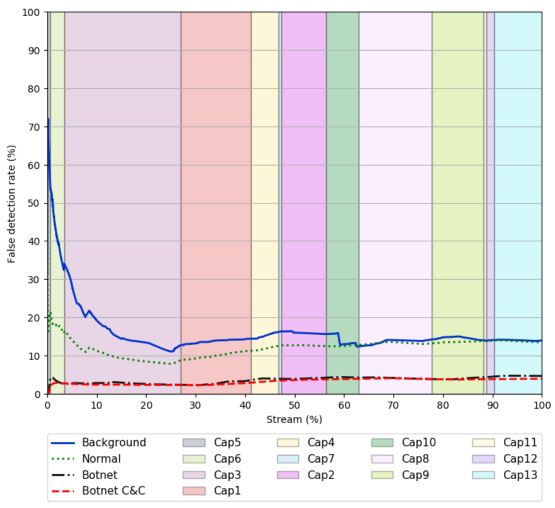
(a) GP–Archive



(b) NB–Variable



(c) GP–Both



(d) GP–Hybrid

Figure 5.13: False positive rate through the stream at a 5% label budget (Average of 20 runs)

All classifiers have a cold start during the initial Capture datasets (Captures 5 and 6). The three Stream-GP algorithms are closely related in terms of reaction to the changes throughout the stream (Figure 5.12). The least detected class is always the Botnet C&C class, which is also the least frequent class, and its general shape of profile is similar under all algorithms. Likewise, The Normal and Botnet classes have the common trend in all cases, whereas the Background class performs poorly in the specific case of Naive Bayes with the 'variable' sampling policy. Firstly, NB–Variable starts with a 100% Background detection rate, which immediately reduces to almost 30% when the model becomes able to identify the minor classes. This observation matches the Naive Bayes properties under several single-bot scenarios previously identified in Section 5.1.2. Moreover, the Naive Bayes algorithm appears to be slower at building classifiers / reacting to change over the course of the stream. All classes are detected above the $\approx 60\%$ AvDR through the course of the stream for Stream-GP algorithms, whilst this only applies to the Botnet class in the case of Naive Bayes with the Background detection rate that is even lower than the Botnet C&C DR.

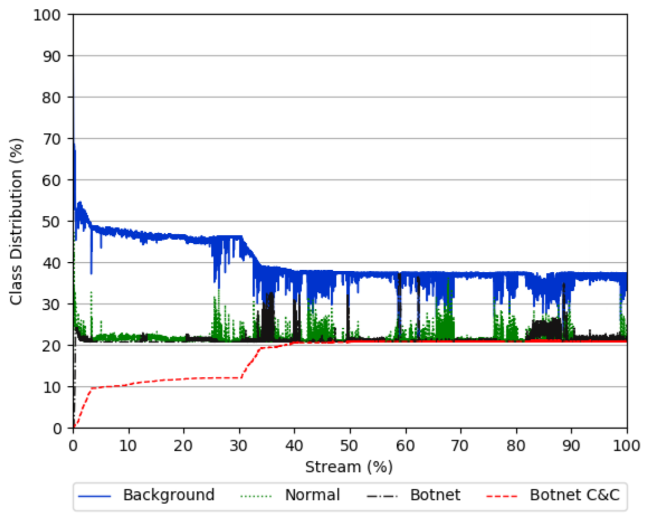
Figure 5.13 summarizes the *False Positive* rate over the course of the stream for each algorithm. GP–Archive and GP–Hybrid perform similarly in false detection, where both algorithms tend to stay at $\approx 10\%$ for Background and Normal traffic and lower than 5% for Botnet and Botnet C&C classes. GP–Both is different from the previous two algorithms in that it has a higher false detection rate of the Background and a little lower rate for the Botnet and Botnet C&C classes. This indicates the insistence of GP–Both to strictly keep the data subset balanced, which leads to the lower number of Background (major class) in this case, while it detects Botnet and Botnet C&C more precisely. On the other hand, NB–Variable detects Normal traffic 60% on average but at the expense of getting a $\approx 50\%$ false positive rate. The trend in the false positive plot for the Background class in the Naive Bayes algorithm confirms the hypothesis that it starts with labelling all classes as Background class (100% false positive), then goes to a lower false positive rate by starting to detect other classes.

Figure 5.15 visualizes the botnet existence throughout the stream in chunks of 10% of the stream. The appearance of different Botnet behaviours throughout the course of the stream and the Stream-GP algorithms' reactions to it leads to the following observations:

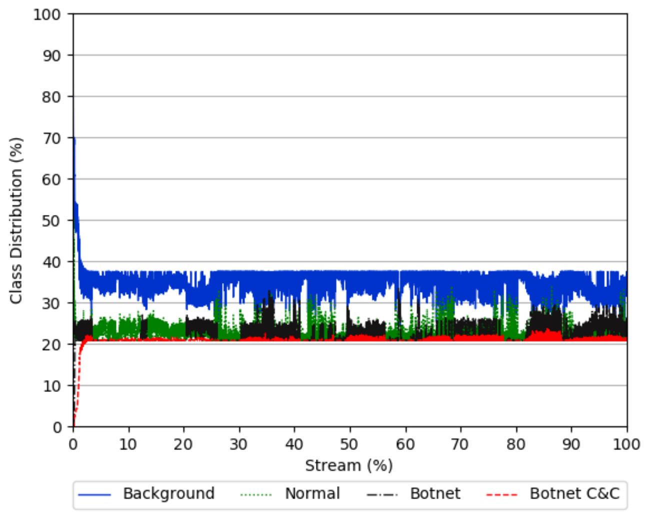
- **RBot** activities (red colour in Figure 5.15) first appear in Capture3 (3rd segment) and the classifier incrementally improves its detection rate. The next occurrences of this attack happen in Capture 4, Capture 10 and Capture 11 (5th, 8th and 11th segments respectively) where the Botnet detection remains the same or increases.

- **Neris** (blue colour in Figure 5.15) begins its malicious activities in Capture 1 (4th segment), which causes a $\approx 10\%$ reduction in the Botnet detection rate. In the next appearance of this Botnet in Capture 2 (7th segments), the detection rate stays stable; whereas a $\approx 10\%$ reduction in Botnet detection again occurs in Capture 9 (10th segment) due to the change in the Bot behaviour.

- **Virut** bot (purple colour in Figure 5.15) experiences the longest absence in the stream, where it starts at the very first and the last segments of the stream. The observation does not show any decrease in the detection of the Botnet class when it again appears lastly.



(a) GP–Archive



(b) GP–Both

Figure 5.14: Typical Distribution of classes present in the Data Subset for the GP–Archive and GP–Both algorithms on the CTU13-mixed dataset at a 5.0% label budget.

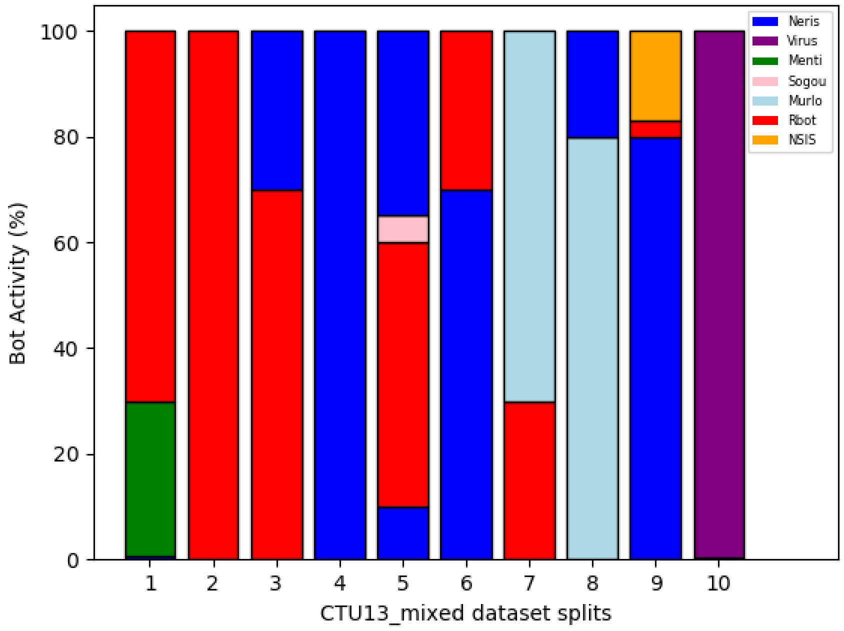


Figure 5.15: Botnet distributions at each 10% of the CTU13-mixed dataset

The summary of observations indicates that a reduction in Botnet class only happened in two cases: 1) the introduction of the Neris Botnet for the first time in Capture 1 and 2) when the same Neris Botnet changes its behaviour in Capture 9. In other cases, either the Botnet detection improves or stays stable.

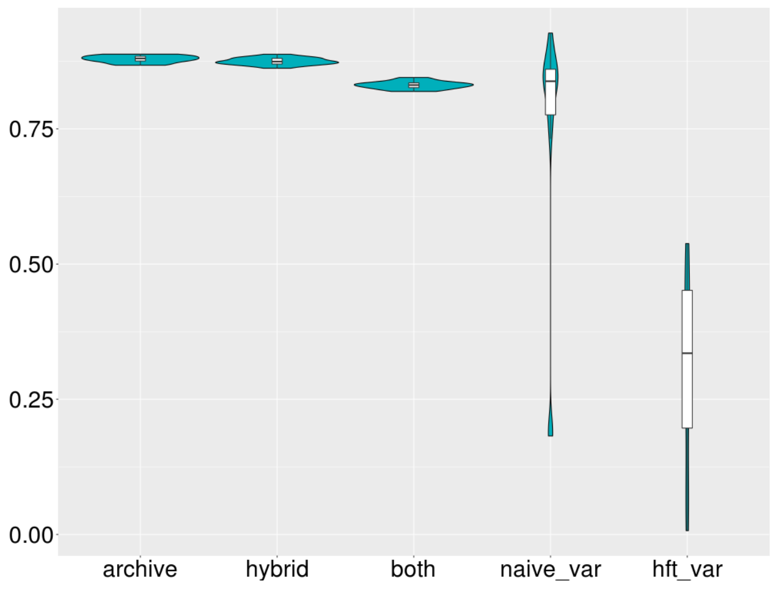
5.2.3 Capacity for Detecting Botnet Signals

The performance of the algorithms in detecting different botnet behaviours available in the CTU13-mixed dataset is investigated in this section. Table 5.12 indicates the median AvDR

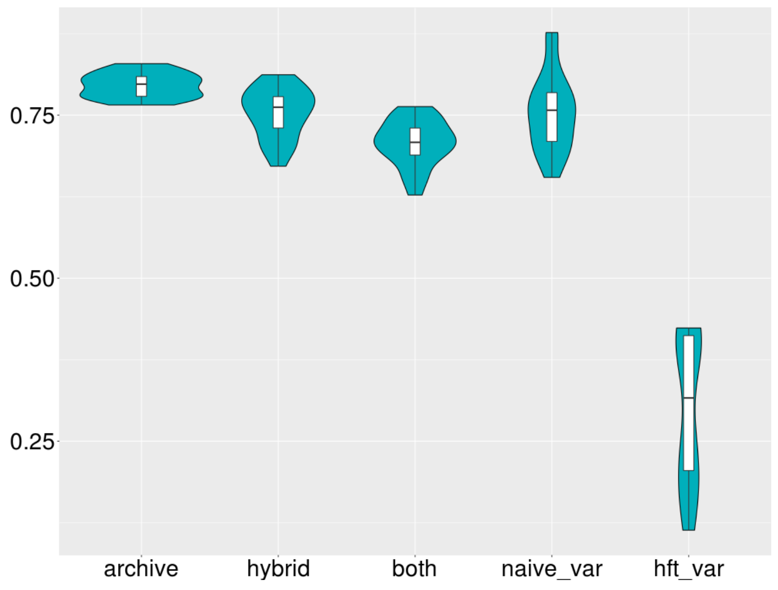
Table 5.12: median values w.r.t. streaming Botnet AvDR metric under 0.5% and 5% label budgets. Three Stream-GP configurations are given in Table 4.9. Hoeffding Tree and Naive Bayes (from MOA) appear with the 'variable' sampling policy, *CTU13-mixed* dataset

	Algorithms	AvDR Median (0.5%)	AvDR Median (5%)
Stream-GP	Archive	79.7%	88.0%
	Hybrid	76.2%	87.4%
	Both	70.8%	83.1%
MOA	Hoeffding	31.6%	33.5%
	NB	75.8%	83.8%

of the runs for each algorithm. The values implies that GP–Archive is the best algorithm at detecting the botnet signals (79.7% and 88.0% for 0.5% and 5% label budgets). All other algorithms, except MOA–Hoeffding with variable uncertainty sampling method, perform well in Botnet detection. In addition, almost 10% growth in the AvDR median is observed by label budget increment.



(a) 5% label budget



(b) 0.5% label budget

Figure 5.16: Violin plots for comparison of algorithms in Botnet detection, 0.5% and 5% label budgets.

The overall performance of the algorithms on the Botnet class, the least frequent class, is illustrated via violin plots in Figure 5.16 for 0.5% and 5% label budgets. The pair-wise comparison of the algorithms based on the Mann-Whitney U test on the Botnet class detection is performed for 5% and 0.5% label budgets (Tables 5.13 and 5.14, respectively). Based on the results, GP–Archive and GP–Hybrid are the top-ranked algorithms in Botnet detection, and the null-hypothesis could not be rejected for them under the 5% label budget. Consequently, the null hypothesis is comfortably rejected for the top-ranked Stream-GP algorithms, GP–Archive and GP–Hybrid, versus MOA comparator algorithms. However, under the 0.5% label budget, only GP–Archive maintains the significant difference against the MOA comparator algorithm.

Further, a Bonferroni-Dunn post-hoc test is applied to showcase the significance of GP–Archive over the rest of the algorithms. The newly computed critical value based on the post-hoc test is computed as $\alpha = \frac{0.05}{4} = 0.0125$, in which 0.05% is the Mann-Whitney U test critical value, and 4 is the number of comparisons made. Based on the computed critical value, GP–Archive performs significantly better than other algorithms in Botnet detection under both 0.5% and 5% label budgets, except in the specific case of GP–Hybrid, in which the null hypothesis could not be rejected for the 5% label budget. In general, GP–Archive and GP–Hybrid are the best Stream-GP variants for detecting the botnet class, and they significantly outperform the MOA comparator algorithms.

Table 5.13: p-value from the the Mann-Whitney U test for the Stream-GP and MOA algorithms based on Botnet median AvDR for the 5.0% label budget

Algorithms					
Stream-GP	Archive	Archive			
Stream-GP	Hybrid	0.05	Hybrid		
Stream-GP	Both	0.0	0.0	Both	
MOA	Hoeffding	0.0	0.0	0.0	Hoeffding
MOA	NB	3.9×10^{-5}	6.9×10^{-5}	0.56	0.0

Table 5.14: p-value from the Mann-Whitney U test for the Stream-GP and MOA algorithms based on Botnet median AvDR, for the 0.5% label budget

Algorithms					
Stream-GP	Archive	Archive			
Stream-GP	Hybrid	1.0×10^{-4}	Hybrid		
Stream-GP	Both	0.0	1.0×10^{-3}	Both	
MOA	Hoeffding	0.0	0.0	0.0	Hoeffding
MOA	NB	1.0×10^{-3}	0.82	9.4×10^{-3}	0.0

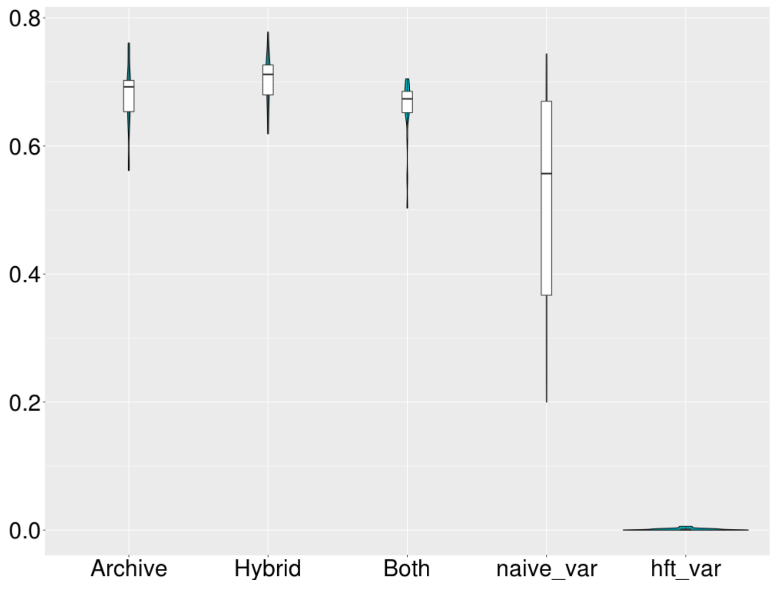
5.2.4 Capacity for Detecting Botnet C&C Signals

The performance of algorithms in detecting the minor class, Botnet C&C, the distribution of which is distinctively lower than the rest of the classes, is investigated in this section. Table 5.15 demonstrates the median values for the AvDR of the runs for each algorithm. It shows that the Stream-GP algorithms are performing distinguishably better than the MOA comparator algorithms. Hoeffding with variable uncertainty is shown to be completely incapable of detecting the Botnet C&C. There is a huge difference between the median values in 0.5% and 5% label budgets ($\approx 40\%$ for all algorithms). This increase in median values determines the power of the label budget when a minor class with a huge gap in its distribution with other classes exists.

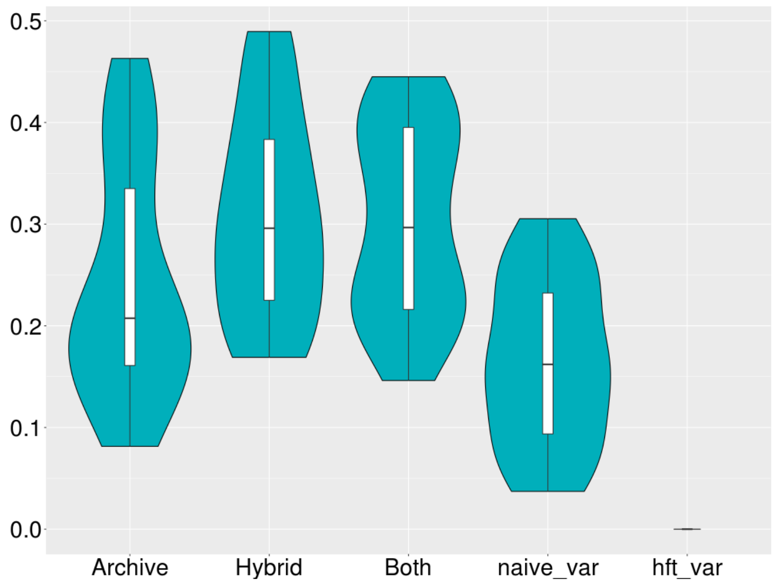
Table 5.15: median values w.r.t. streaming Botnet C&C AvDR metric under 0.5% and 5% label budgets. Three Stream-GP configurations are declared in Table 4.9. Hoeffding Tree and Naive Bayes (from MOA) appear with the 'variable' sampling policy, *CTU13-mixed* dataset

	Algorithms	AvDR Median (0.5%)	AvDR Median (5%)
Stream-GP	Archive	20.7%	69.2%
	Hybrid	29.6%	71.2%
	Both	29.7%	67.3%
MOA	Hoeffding	0.0%	0.0%
	NB	16.2%	55.7%

The overall performance of the algorithms on the Botnet C&C class, the least frequent class, is illustrated via violin plots in Figure 5.17 for 0.5% and 5% label budgets. A more detailed Mann-Whitney U test is applied pair-wise on every two algorithms of the set to investigate if the null hypothesis, is rejected or not (Figures 5.16 and **??** for 5% and 0.5% label budgets, respectively). The null hypothesis is rejected for Stream-GP variants versus MOA comparator algorithms under a 5% label budget, which shows that Stream-GP variants are performing significantly better than the comparator MOA algorithms. However, GP–Archive versus the other Stream-GP variants test did not reject the null hypothesis. GP–Hybrid and GP–Both are the best algorithms for Botnet C&C detection. The leading Stream-GP algorithms, GP–Hybrid and GP–Both, still perform significantly better than the MOA comparator algorithms in Botnet C& C detection. The comparison of GP–Hybrid and GP–Both demonstrates that GP–Hybrid is successful in increasing the Botnet C&C detection rate when the label budget is low, which illustrates that it could successfully play its role as it was designed. Therefore, GP–Hybrid and GP–Both perform best on Botnet C&C detection, whereas GP–Archive performs better on Botnet Detection.



(a) 5% label budget



(b) 0.5% label budget

Figure 5.17: Violin plots for the comparison of algorithms in Botnet C&C detection, 0.5% and 5% label budgets.

Table 5.16: p-value from the Mann-Whitney U test for the Stream-GP and MOA algorithms based on Botnet C&C median AvDR for the 5.0% label budget

	Algorithms				
MOA Stream-GP	Archive	Archive			
	Hybrid	0.06	Hybrid		
	Both	0.20	5.9×10^{-4}	Both	
	Hoeffding	0.0	0.0	0.0	Hoeffding
	NB	2.9×10^{-3}	1.5×10^{-4}	0.01	0.0

Table 5.17: p-value from Mann-Whitney U test for Stream-GP and MOA algorithms based on Botnet C&C median AvDR, for the 0.5% label budget

	Alogrithms				
MOA Stream-GP	Archive	Archive			
	Hybrid	0.03	Hybrid		
	Both	0.04	0.94	Both	
	Hoeffding	0.0	0.0	0.0	Hoeffding
	NB	0.05	2.9×10^{-5}	6.9×10^{-5}	0.0

5.2.5 Feature Selection by Stream-GP Champion

The features used to detect the classes throughout the stream may vary based on the content of the stream. In this section, the frequency of the flow features which used by the stream-GP champion is reviewed. For this purpose, GP–Archive is considered as the representative of Stream-GP algorithms.

Figure 5.18 illustrates an overall view of the flow features used in the GP–Archive champion team to label the stream. This stacked column chart is plotted based on the average number of flow features utilized by the learners of the champion team. In other words, the number of features in each learner is computed, and then the average of these numbers between the learners of a team is calculated. The number is conjugated in several numbers of consecutive generations on average to produce a readable and clear image of the feature usage throughout the course of the stream. This represents the active behaviour of the Stream-GP classifier throughout the course of the stream where the frequency of features used in the champion team differs from time to time throughout the course of the stream. *Direction*, *protocol* and *source bytes* features are the three most frequently used features in general.

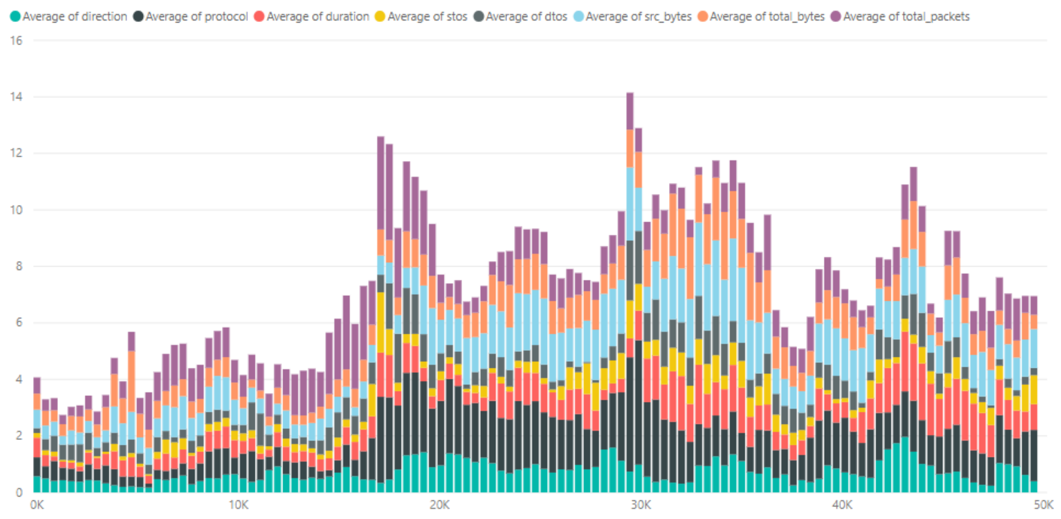


Figure 5.18: Overview of the features used by the GP–Archive champion throughout the stream, 5% label budget

Figure 5.19 demonstrates the average feature usage in a champion team based on the action (class label). This specifically shows which features are most important in the detection of each class. *Protocol* is important in the detection of all classes. *Duration* plays an important role in Botnet detection, as attack behaviour is different from the Normal behaviour in terms of time. It may be affected by the burst of the Botnet attack in a short period of time as is illustrated in Figure 5.9. Again, *Source Bytes* are important in distinguishing between different classes and also between Botnet and Botnet C&C.

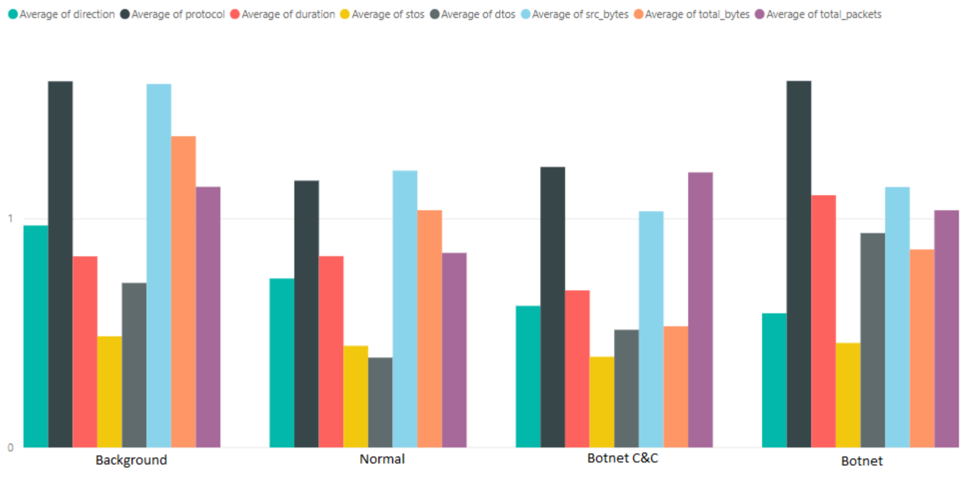


Figure 5.19: Overview of the features used by the GP–Archive champion throughout the stream based on action, 5% label budget

Figure 5.20 illustrates the distribution of the average effective (non-introns) code lines based on class labels. It implies that there is a balanced distribution in the overall distribution of the effective executable code lines throughout the stream, which indicates that the attempt to balance the learning data subset was successful.

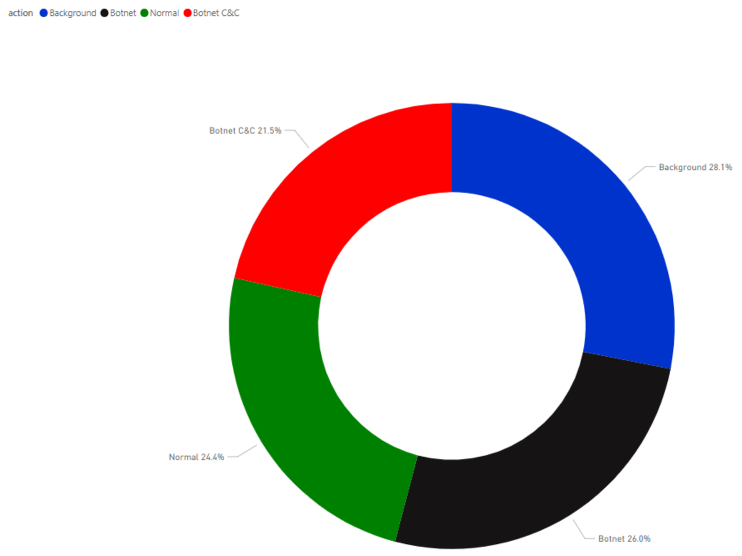


Figure 5.20: The average number of execution code lines based on action (class label), 5% label budget

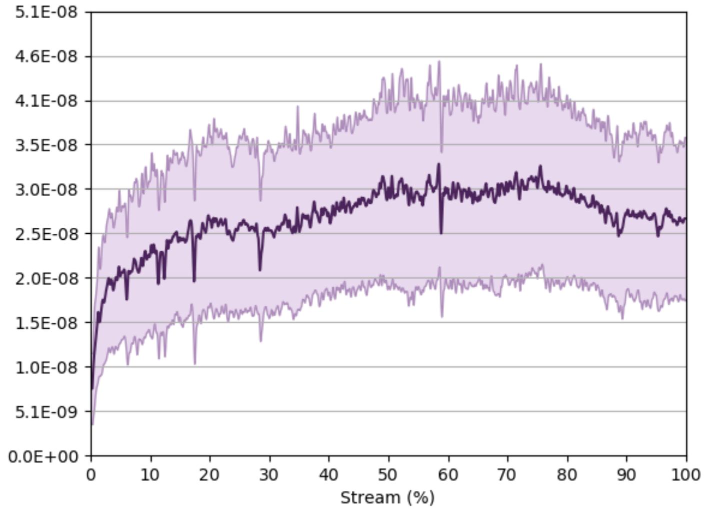
5.2.6 Real-time Operation

The computational cost of the streaming classifiers on the challenging CTU13-mixed dataset is reviewed in this section. The preprocessing of the network traffic and the general idea to evaluate the computational time is similar to Section 5.1.7. Therefore, the time is computed in two modes:

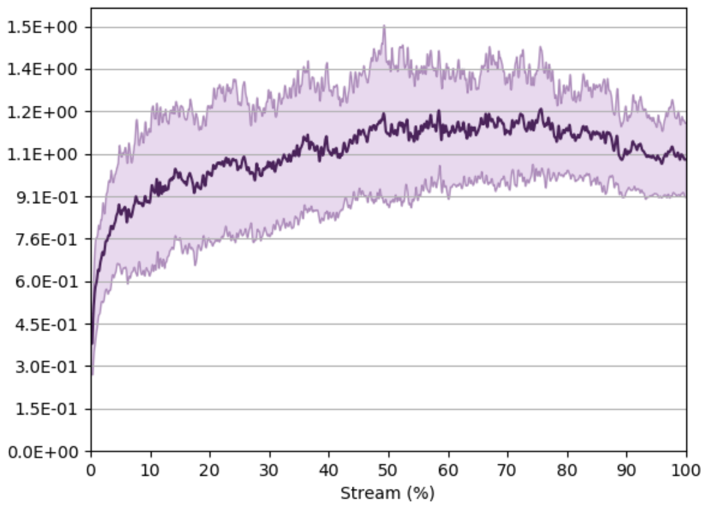
- **Champion prediction**, which is the amount of time the Stream-GP champion at the time, anytime operation, takes to suggest a label for each flow record.

- **Training time** refers to the cost of performing $\tau = 5$ training epochs for every data subset change, i.e. each window location.

These two time calculations are independent as the corresponding processes can happen in parallel.



(a) Stream-GP Champion



(b) Fitness evaluation

Figure 5.21: Wall clock time for (a) the champion individual to make predictions and (b) the fitness evaluation to update the content of the population on a new non-overlapping window location for the CTU13-mixed dataset on a 2.67GHz CPU.

Figure 5.21 summarizes the average and mean time taken for the 20 runs of the Stream-GP algorithm, which is deployed *single-threaded* on an Intel i5 CPU 16@2.67GHz and 48GB memory. It takes $\approx$ 30ns (Figure 5.21(a)) for the Stream-GP champion to predict an exemplar under a 5% label budget. This time could be reduced further if multiple threads are utilized by the champion.

The training time is computed based on the fitness evaluations on a per window location. This time is a function of the data subset size ($DS(i) = 120$ flows), the number of teams ($P_{size} = 120$) and the number of Stream-GP generations on each data subset, $DS(i)$, update ($\tau = 5$ generations) plus any overhead in champion selection (the parameters can be seen in Table 4.10). However, just a part is changing at each window location; for example, only 20 new flow records ($Gap = 20$) are introduced and replaced at each DS update, and for the team updates, only 20 teams are replaced by new instances ($Tgap = 20$). Thus, the overall computational cost for training on *a single window* takes $20 \times 20 \times 5$ number of evaluations. In the case of training on CTU13-mixed, the average time for training on a window is $1.1s$ (Figure 5.21(b)).

Still, the bottleneck for the training time is the time the human expert takes to suggest true labels for the classifier, where it is common to stream learning algorithms and also the best practice for developing updates for signature detection models. Whilst the expert takes their time to suggest labels, there is always a champion classifier to label the stream of network flows based on the anytime operation. In addition, the framework applied on the CTU13-mixed works on the network flows, so it is considered to have a 'near' real-time performance.

5.2.7 Summary

The best Stream-GP algorithms and their best MOA comparator (based on the performance ranking in Section 5.1.1) is selected to benchmark their performance on a multi-bot dataset, CTU13-mixed. This dataset includes thirteen different Botnet scenarios (re)occurring at different points in the stream. The evaluations are done on 0.5% and 5% label budgets. The GP–Hybrid algorithm is introduced, which initially starts with a greedy selection of flows predicted to be under-represented in the data subset to develop classifiers. As the performance of the initially under-represented classes improves, uniform sampling is assumed. This switch impacts the sampling of traffic to be more of the typical behaviours after a cold start, in which the samples reflect the underlying distribution of the traffic. The positive point is that the infrequent classes, which are representative of the Botnet behaviour, are available in the data subset and are not lost.

The evaluations demonstrate that GP–Hybrid and GP–Archive are the two best algorithms in multi-botnet detection. There might be a slight difference between these two algorithms; but in the case that the minor class(es) is tremendously infrequent with a poor / slow introduction, GP–Hybrid would suggest better samples than GP–Archive for rapid minor class(es) identification. GP–Hybrid has been shown to perform better than GP–Archive in Botnet C&C detection. This implies that the start of the Botnet activities could be identified more rapidly by the GP–Hybrid algorithm. On the other hand, GP–Archive performs slightly better than GP–Hybrid in Botnet detection. In addition, the analysis illustrates that Stream-GP is able to retain Botnet signatures learned and re-use them when they occur again in the stream. Last, the ability of Stream-GP to perform in 'near' real-time is also demonstrated.

The Hoeffding Tree algorithm with Variable Uncertainty from MOA comparator tool was incapable of performing under the stream of data with multiple botnet behaviours. Therefore, this algorithm is removed from the list of algorithms for further evaluations. On the other hand, GP–Hybrid as the best performing algorithm under this complex stream is added to the list of best performing algorithms.

5.3 Network Security in Real-world Network Traffic

The Stream-GP and MOA comparator algorithms are examined under malicious network traffic content, e.g. Network Bots and Intrusions, for more evaluations of network security topics. In Section 5.1, the performances of Stream-GP variants and MOA algorithms were identified under various botnet datasets, and the relative ranking of the algorithms were concluded in general. In this section, a detailed analysis of the top-ranked algorithms under the 5% label budget are exposed over two more network security scenarios. Section 5.3.1 quantifies the overall performance of the best-ranked algorithms identified in Section 5.1 over two network security datasets (Table 5.18) for the 5.0% label budget. Section 5.3.2 demonstrates the dynamics of algorithms throughout the course of the stream. Finally, Section 5.3.3 concludes the results for analysis of the aforementioned network security datasets.

5.3.1 Overall Performance Evaluation

The overall performance of the Stream-GP algorithm with the best sampling-archiving policy combination (*GP–Archive*) and the MOA comparator algorithms with the best sampling policy (*Variable Uncertainty*) on two network security datasets, namely ISOT and NSL-KDD, are explored. The streaming AvDR (Section 4.4) is considered the metric for the evaluations of algorithms. All results are the average of 20 independent runs under the 5.0% label budget ($\beta = \{0.05\}$). In Table 5.18, the ranking of all network datasets are given, where the CTU-13 datasets' results are taken from Table 5.1. The Friedman test is

Table 5.18: Algorithm ranks w.r.t. the streaming AvDR metric under a 5.0% label budget. Bracketed entries represent median AvDR values to 1 decimal place.

Dataset		GP–Archive	Hoeffding	NB
NSL-KDD		**1 (65.5)**	2.5 (20.0)	2.5 (20.0)
ISOT		2.5 (85.6)	2.5 (85.6)	**1 (88.4)**
	Capture 1	**1 (56.8)**	3 (26.7)	2 (43.5)
	Capture 2	**1 (68.1)**	3 (36.5)	2 (54.8)
	Capture 3	**1 (81.5)**	3 (55.5)	2 (59.5)
	Capture 4	**1 (62.7)**	3 (42.2)	2 (55.2)
	Capture 5	**1 (36.1)**	3 (26.3)	2 (30.7)
	Capture 6	**1 (65.8)**	3 (25.5)	2 (48.6)
CTU-13	Capture 7	2 (29.8)	3 (25.9)	**1 (32.5)**
	Capture 8	**1 (78.0)**	3 (28.1)	2 (57.9)
	Capture 9	**1 (54.1)**	3 (26.5)	2 (45.4)
	Capture 10	**1 (70.3)**	3 (54.6)	2 (58.7)
	Capture 11	**1 (54.8)**	3 (42.2)	2 (47.6)
	Capture 12	**1 (52.5)**	3 (36.0)	2 (48.3)
	Capture 13	**1 (70.2)**	3 (27.3)	2 (42.9)
R_j		**1 (1.16)**	3 (2.93)	2 (1.9)

done on the median AvDR for the 20 runs of each algorithm to specify if a pattern in the algorithms' ranking exists. Thereafter, a Nemenyi post-hoc test determines the significance of the algorithms based on the critical difference, Eq. 4.15.

The last row of Table 5.18 represents the average ranking of algorithms over all the network datasets. This ranking is then used by the Friedman test to check the null hypothesis, i.e. the ranks are random.

The null hypothesis is tested by comparing χ_F^2 to F_F, F-distribution with $k-1$ and $(k-1)(n-1)$ degrees of freedom [33]. Where $n = 15$ is the number of datasets, and $k = 3$ is the number of algorithms in this case. Table 5.19 demonstrates that the null-hypothesis is rejected as the critical value of $F(2, 28)$ for $\alpha = 0.01$ is less than 2.503.

Thereafter, a Nemenyi post-hoc test is done to specify the significance level of algorithms and group them into similarly performing algorithms. By setting $q_\alpha = 0.1$, the critical difference becomes 0.036. This value shows that each algorithm is significantly different from their comparator algorithms where the *'stream-GP'* is ranked **first**.

5.3.2 Detection Rate Dynamics: Comparing the Best Streaming Classifiers

The underlying dynamics of the specific two new datasets, NSL-KDD and ISOT, are investigated to gain more insights in to the behaviour of algorithms over the course of the

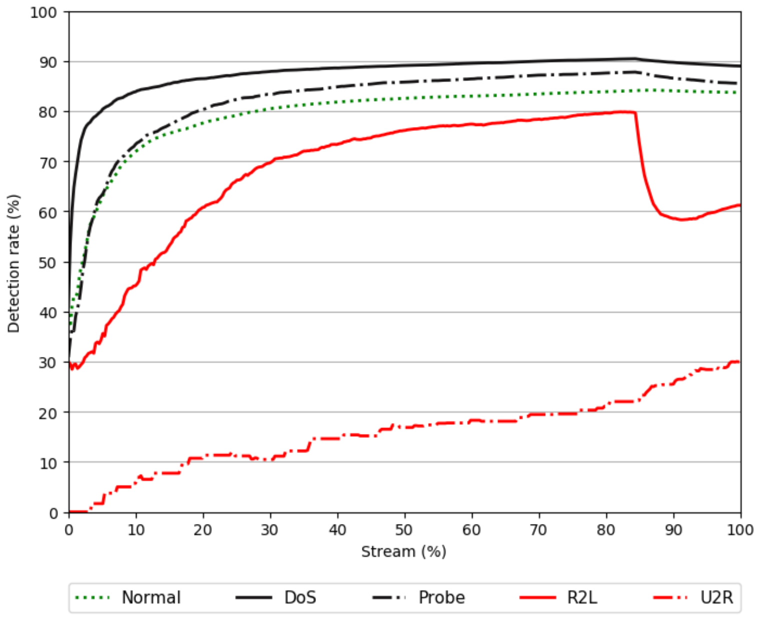
Table 5.19: Result of Friedman test χ^2_F and corresponding value for F-distribution F_F. The critical value of $F(2, 28$ for $\alpha = 0.01$ is 2.503, so the null-hypothesis is rejected in each case.

Label budget	5.0%
χ^2_F	23.11
F_F	46.96

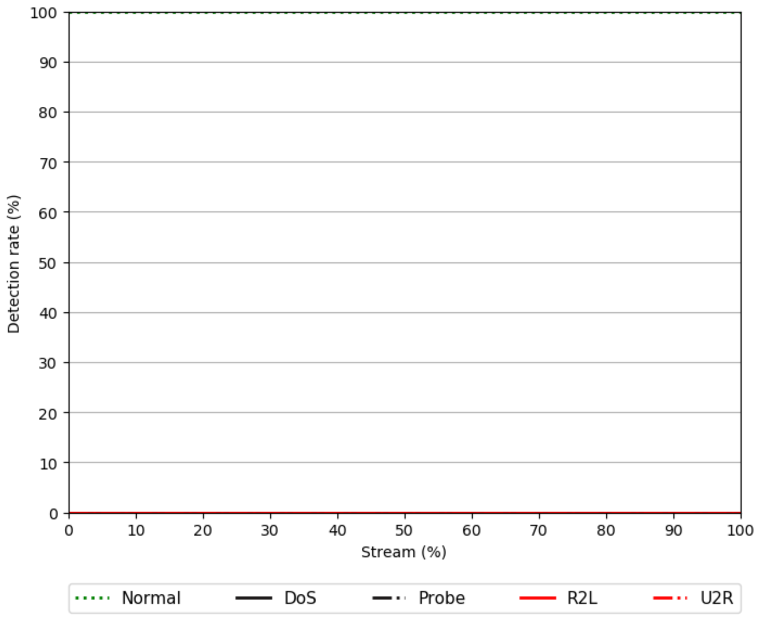
stream. The comparison is done between GP–Archive as the champion over the network datasets (rank 1st) and the best MOA comparator candidate, NB–Variable (ranked 2nd). Figures 5.22 and 5.23 are the class-wise AvDR plots throughout the stream for NSL-KDD and ISOT, respectively.

In the case of NSL-KDD, Figure 5.22, it is obvious that Naive Bayes labels all classes as Normal traffic which is the major class. The poor performance of Naive Bayes over NSL-KDD may be relative to the type of this dataset. NSL-KDD is a packet-based dataset, as each record is a packet. In previous datasets, network flows were utilized that aggregate relative packets and provide more meaningful content. Naive Bayes seems to not capable of performing well in this situation. On the other hand, Figure 5.24 demonstrates low false positive rates of GP–Archive in both datasets.

Both algorithms perform well on the ISOT dataset, Figure 5.23. GP–Archive has with a cold start but soon gets to the desirable detection rate level. As the figures depict, once the class is detected, its detection rate stays at a consistent level. This behaviour suggests that the streaming data are stationary, and do not change throughout the course of the stream, or on the other hand, that the attack appears at one point, and there is no indication of it until the end of the stream. In this scenario, the Naive Bayes algorithm labels everything as Normal class, and then if it applies to the specific thresholds, which are not going to change throughout the course of the stream, it assigns their class labels. On the other hand, the GP–Archive recognizes each class separately, which leads to a slightly lower detection of the Normal and Smtp-Spam classes. The Naive Bayes performs well when there are some specific ranges with a different behaviour so it is not difficult to track the changes. The ISOT dataset is a case of a non-complex dataset, as no concurrent track of the boundaries are required, and boundaries are well separated.

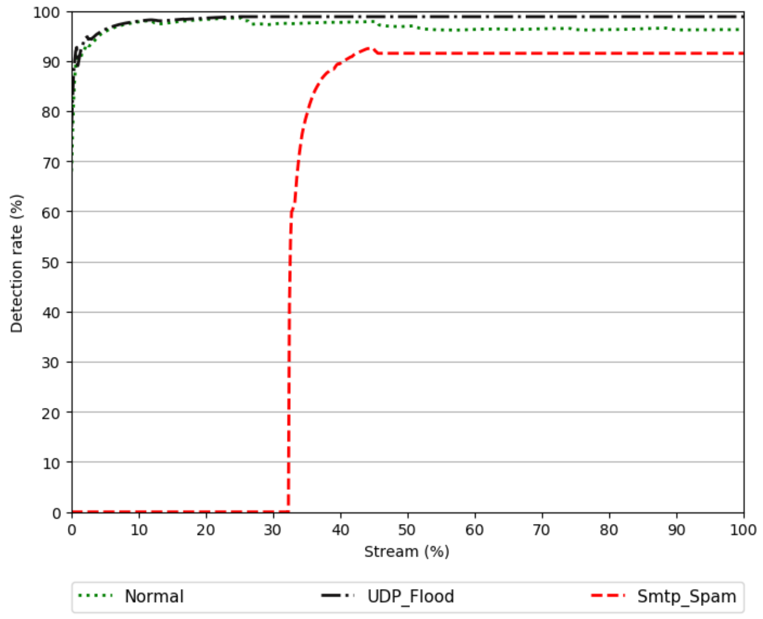


(a) GP–Archive

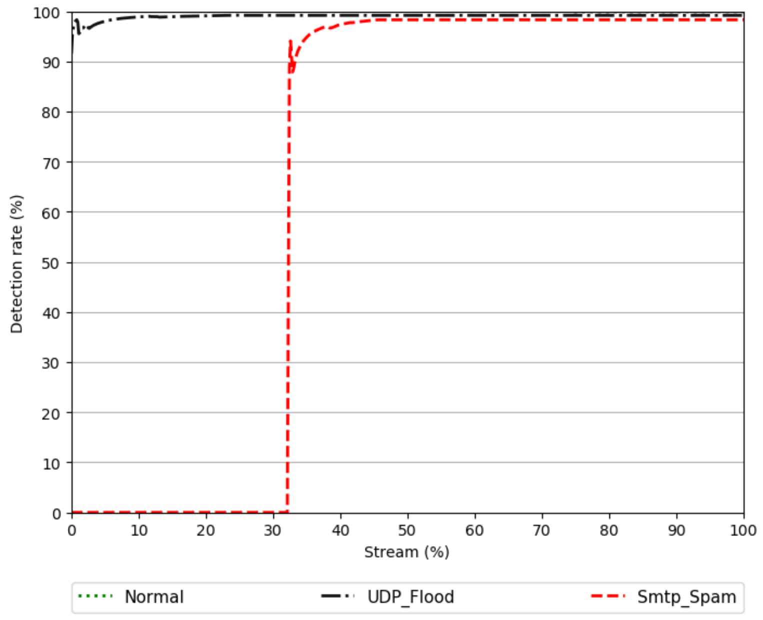


(b) NB–Variable

Figure 5.22: NSL-KDD class-wise Detection rate through the stream at a 5% label budget based on the streaming AvDR metric (Average of 20 runs)

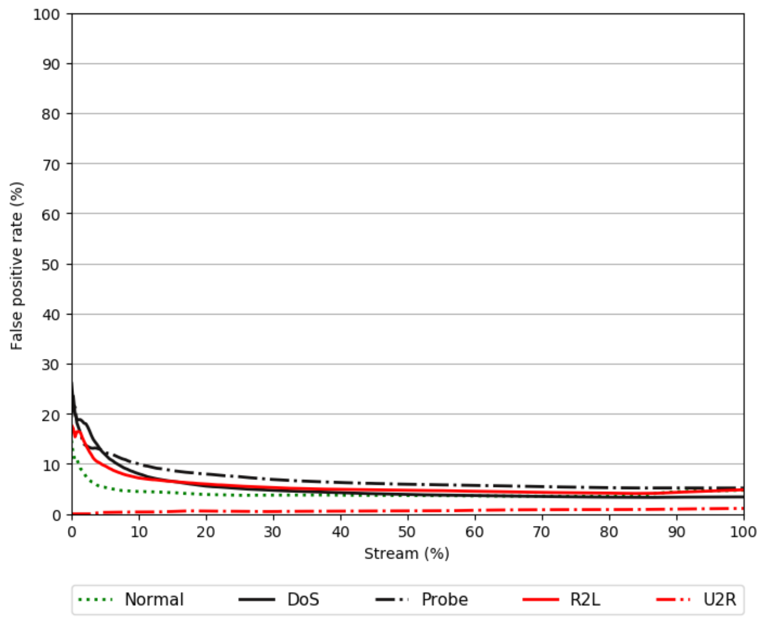


(a) GP–Archive

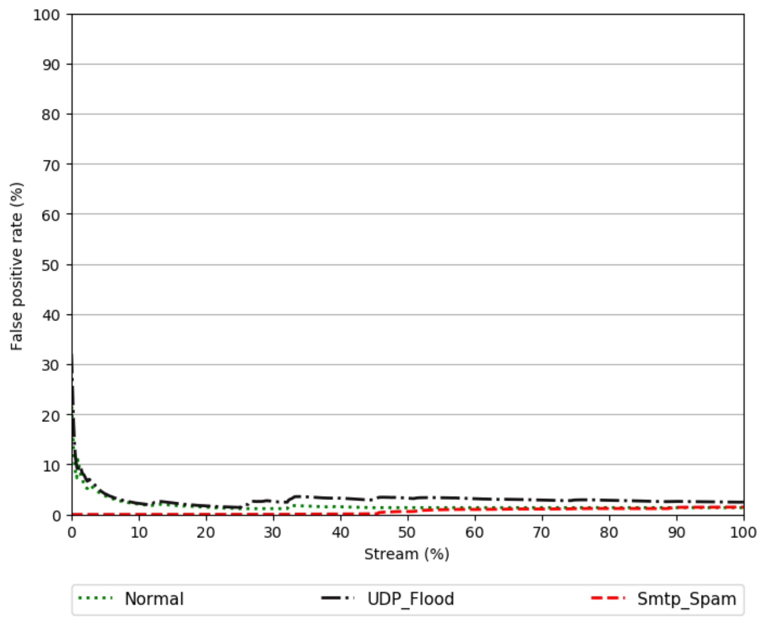


(b) NB–Variable

Figure 5.23: ISOT class-wise Detection rate through the stream at a 5% label budget based on the streaming AvDR metric (Average of 20 runs)



(a) NSL-KDD dataset



(b) ISOT dataset

Figure 5.24: GP–Archive class-wise false positive rate through the stream at a 5% label budget based on the streaming AvDR metric (Average of 20 runs)

5.3.3 Summary

Three top-ranked streaming classifiers: GP–Archive, Hoeffding–Variable and NB–Variable, have been applied on two more public network security datasets, ISOT and NSL-KDD. The overall evaluations on all network datasets were performed, which strongly indicates that GP–Archive is the best algorithm to be applied on the network stream. The GP–Archive algorithm, by selectively making the right decisions to remove highly populated exemplars from the data subset, not only performs well based on the streaming AvDR metric but also has been shown to do so under a low false positive rate and low label budget (5.0%). This latter case is an important factor in the network security field, where malicious behaviours are mostly scarce.

In summary, Stream-GP algorithms (GP–Archive, GP–Hybrid and GP–Both) and MOA algorithm (NB–Var) are identified as the best performing algorithms on different network scenarios. These algorithms are kept for further evaluations.

5.4 A Comparison to the Apache Spark Streaming Network Tool

The Stream-GP classifier is shown to effectively work in real-world network scenarios in Section 5.1 to Section 5.3. The evaluations are performed on real-world network scenarios, and the comparisons are always done with the MOA algorithms, which MOA is the streaming version of the Weka toolkit[5] that provides ML algorithms for streaming data. To get a sense of how the Stream-GP performance stands in comparison with current network tools, a comparison to the Apache Spark (Streaming) tool is performed in this section. Apache Spark is a state-of-the-art network tool that is widely used in network security and operations. The overall architecture of Apache Spark is illustrated in Figure 4.1. This tool has a streaming component, which provides the ability to work on streaming data in real-world scenarios. Section 5.4.1, presents an overall comparison of the best Stream-GP algorithms with three popular Apache Spark ML classification algorithms. In Section 5.4.2, the visualization of the algorithms' underlying behaviour throughout the course of the stream is prepared. Sections 5.4.3 and 5.4.4 dig into the investigation of Stream-GP behaviour in comparison to its comparator algorithms in specific cases of Botnet and Botnet C&C detection. In Section 5.4.6, the computational cost of training the Apache Spark classifier model and the response time to label the stream exemplars are illustrated. Finally, Section 5.4.8 summarizes the comparative analysis of the Stream-GP and Apache Spark network tool.

A multi-class streaming AvDR metric (Eq. (4.10) of Section 4.4) is used to calculate the overall performance. For evaluation of each algorithm's performance, 10 runs are performed.

[5]`https://www.cs.waikato.ac.nz/ml/weka/`

5.4.1 Overall Performance Evaluation

The top three Stream-GP algorithms (Table 5.10) and three most commonly used Apache Spark ML classification algorithms (Decision Tree, Random Forest and Naive Bayes) are selected for evaluations on the CTU13-mixed dataset. The *Stream-GP* algorithms are utilized under a 5.0% label budget in a completely streaming mode where both learning and prediction processes are streaming. In contrast, *Apache Spark Streaming* does not provide streaming learning but supports the utilization of an off-line learned ML model on the stream of data[6]. For evaluations, two modes of operation are considered: 1) Streaming simulation, and 2) Classical operation. The overall architecture of the scenarios is illustrated in Figure 5.25. For both approaches, the 5.0% label budget is considered. A description of each scenario follows next.

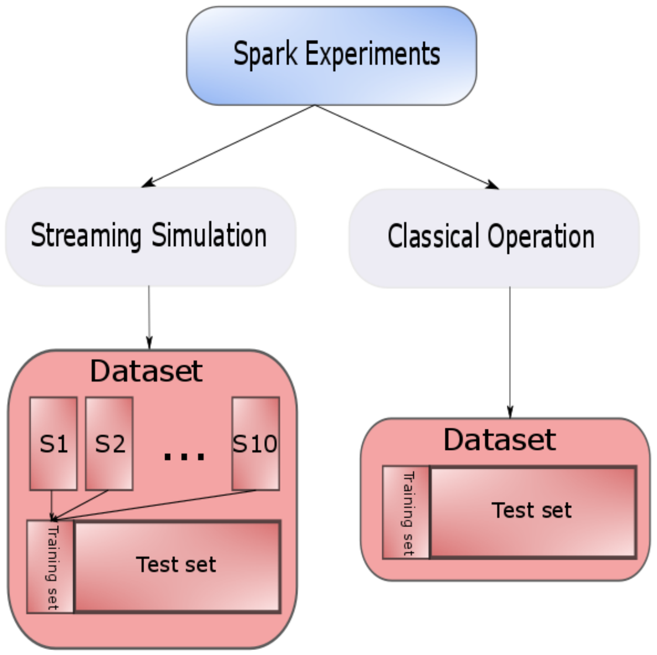


Figure 5.25: Apache Spark Experiments flowchart.

The **Streaming simulation** is more like the real-world application of off-line models being applied to the stream of data, as only access to the *first* portion of the stream is guaranteed and the future of the stream is not known. Therefore, for a rich comparison, the CTU13_mixed divided into *ten* non-overlapping splits in which each split indicates a

[6]Later, DStream is introduced based on Spark Streaming to provide streaming learning. However, the ML library provided in this tool is based on MOA which is benchmarked previously

separate experiment where the stream starts from there. Dividing up the dataset into 10 splits provides the opportunity to test what happens if the stream starts at each of the splits and how it affects the performance of an off-line model on predicting the label for the rest of the stream. The training data are sampled from the first split in each case based on the label budget. For each split, 10 different runs are performed to evaluate the algorithms' performance with the assumption that the stream is started from that specific split. Overall, it makes 100 runs on the dataset (training set number (10) * number of runs (10)).

The **Classical operation** ignores the constraint of access to the streaming data and works like the usual off-line learning (batch) model but on a huge dataset, which is also considered a big-data problem. For this case as well, 10 runs are performed with random sampling from all over the dataset this time.

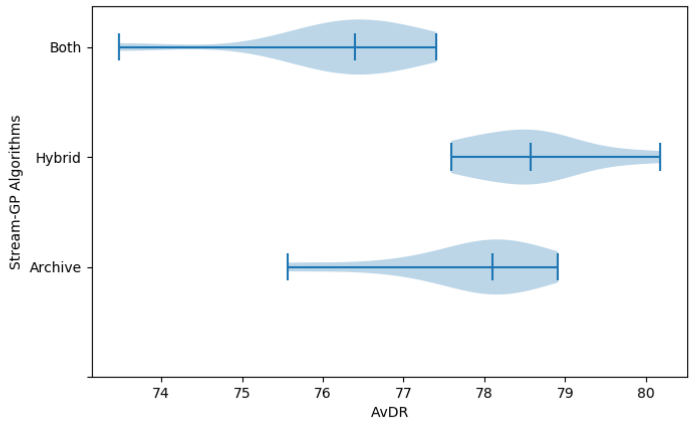
In both modes of operations, the sampling process happens in a random mode so the training set classes' distributions correspond to the original stream distribution, which is highly imbalanced in the CTU13-mixed dataset. This imbalanced training set would lead the model to learn the major class the most, and it would be unable to detect minor classes. To overcome this shortcoming, the number of major class samples is kept the same as the sum of the remaining minor classes. This may lead to using the training data less than the 5% label budget.

Streaming Simulation Scenario

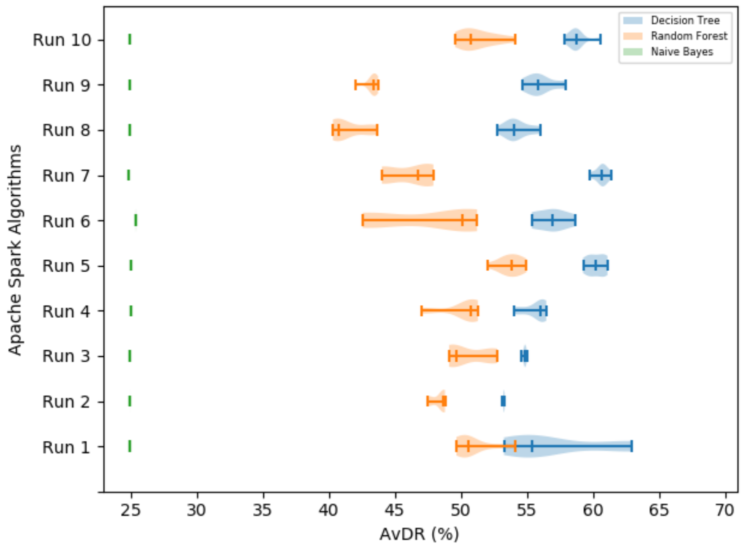
For the sake of clarity, the results are demonstrated as violin plots in Figure 5.26 for the Stream-GP and Apache Spark algorithms. The training violin plots are also shown in Figure 5.27.

The Apache Spark violin plots (Figure 5.4.1) indicate that *Naive Bayes* is not performing well. The figure shows that it is only capable of detecting the major class. Moreover, the plot demonstrates that *Decision Tree* is performing better than Random Forest ($>\approx 5\% - 10\%$). Inspecting the training violin plots (Figure 5.27(a)) reveals that the same relationship is evident between the Decision Tree and Random Forest as well. This evidence diminishes the possibility of Random Forest's over-fitting. This also shows that the random forest parameterization could be changed for better performance. One key point of Stream-GP is that it does not need to be tuned at the beginning, and it will change throughout the stream based on the observations and the need. The plots also illustrate that Detection rates become higher when trained on some specific splits of the dataset, e.g. Splits 5 and

10.



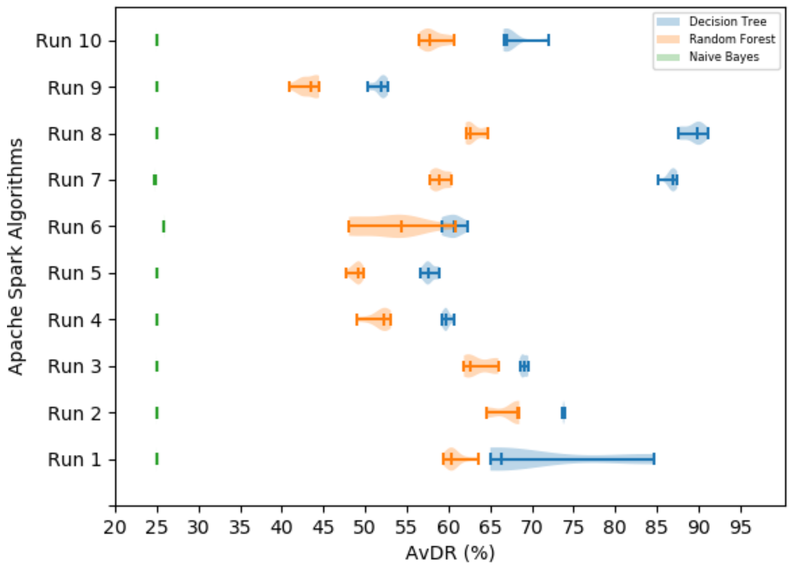
(a) Stream-GP (5.0%)



(b) Apache Spark (5.0%)

Figure 5.26: CTU13-mixed class-wise Detection rate through the stream. Stream simulation scenario. 5.0% label budget. The Y-axis shows the Split where training data are sampled from.

The comparison of the performance of Stream-GP and Apache Spark algorithms shown in the violin plots (Figure 5.26) indicates that the Stream-GP algorithms ($\approx 75\% - 80\%$) are performing significantly better than the Apache Spark ML algorithms ($< 63\%$) in terms of the Average Detection Rate (AvDR). This corresponds to the adaptation of Stream-GP to the changes throughout the stream and the ability to deal with the imbalanced data in learning. This experiment clearly demonstrates that the off-line models can not effectively work in a stream with non-stationary data properties.



(a) Apache Spark (5.0%)

Figure 5.27: CTU13-mixed class-wise Detection rate in the training phase. 5.0% label budget. The Y-axis shows the Split where training data are sampled from.

Classical Scenario

The Random Forest and Decision Tree algorithms from Apache Spark are selected for this experiment. The results are demonstrated as violin plots in Figure 5.28 for the Stream-GP and Apache Spark algorithms.

The comparison of violin plots (Figure 5.28) illustrates that the Stream-GP algorithms are performing significantly better than the classical application of the Apache Spark algorithms ($\approx 78\% > \approx 55\%$). This result emphasizes that the dynamic properties of streaming data require the classifier to be dynamic as well. Therefore, any static classification model is unable to perform well over all the stream, and the performance will decay due to the shift/drift changes throughout the stream. Although the limited access to the stream content is not considered in this scenario, this does not help Decision Tree to have a better performance in comparison to the Stream simulation scenario. This may represent that limited sampling of the data from a huge pool of input entries in an imbalanced dataset can not provide more informative selections compared with choosing from a limited window position to be embedded in a monolithic solution. Similar to the stream-based scenario, Decision Tree performs better than the Random Forest algorithm (≈ 7%).

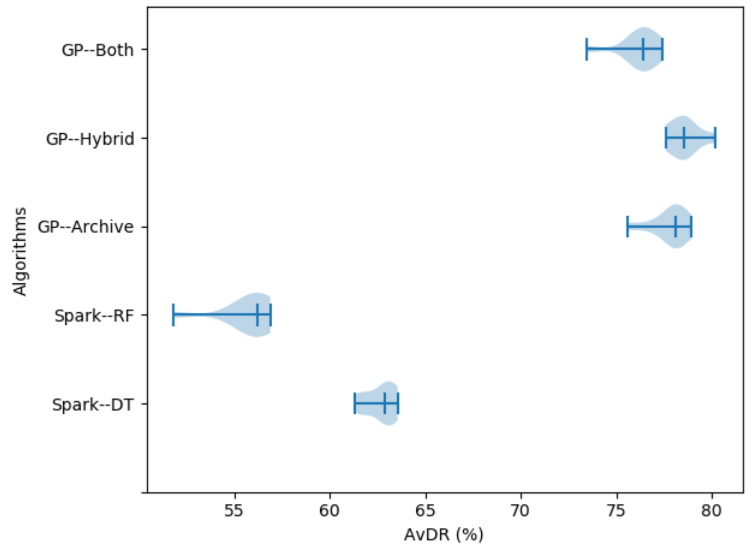


Figure 5.28: CTU13-mixed class-wise Detection rate through the stream. Classical scenario. 5.0% label budget. The Y-axis shows the algorithms.

Comparison of the Top Performing Algorithms:

In this section, the performance of best algorithms in Apache Spark (Decision Tree, both scenarios), MOA (Naive Bayes-Variable Uncertainty) and Stream-GP (GP–Hybrid) are compared and statistical analysis is given. For this purpose, all 10 different runs on 10 splits of the dataset in the case of the Spark stream-like scenario are averaged to form a general perspective of the algorithm on the overall dataset.

The overall performance (Table 5.20) illustrates the extreme difference of the Stream-GP algorithm with the Apache Spark algorithm in two modes of operation. The decision tree in the classical scenario is slightly better than the stream-based scenario. This is due to the unlimited access of the classical scenario to the content of the stream from the beginning, which in practice is not possible in stream-processing tasks.

The pair-wise Mann-Whitney U test on the Stream-GP versus comparator algorithms is applied for a 5% label budget (Table 5.25). Thereafter, a Bonferrori-Dunn post-hoc test is applied to find out if the Stream-GP algorithm is significantly different than all the other algorithms overall. For this purpose, the α value ($\alpha = 0.05$) is divided by the number of comparisons ($k - 1 = 3$) in which the new α equals 0.017. With regard to the new alpha, the null hypothesis is comfortably rejected, and the Stream-GP has been shown to be significantly better than the rest of the comparator algorithms. There is at least a 15% difference between the performance of the Stream-GP and Apache Spark and MOA comparator algorithms under a 5% label budget (Table 5.20). Overall, the necessity of having streaming learning throughout the course of the stream is clear now. Even the classical scenario of the Apache Spark, which has no limitation to access the data, is demonstrated to not be applicable, as it is not adaptable to the changes throughout the course of the stream.

Table 5.20: Median values, under a 5% label budget. The best Stream-GP configuration (GP–Hybrid) and the best Apache Spark Algorithm (Decision Tree), *CTU13-mixed* dataset

Algorithms	AvDR Median (5%)
GP–Hybrid	78.6%
MOA–NB	57.7%
DT–Stream	56.4%
DT–Classic	62.9%

Table 5.21: p-value from the Mann-Whitney U test for the Stream-GP and Spark best algorithms for the 5.0% label budget

Alogrithms	GP–Hybrid
MOA–NB	0.001
DT–Stream	0.0001
DT–Classic	6.4×10^{-5}

5.4.2 Dynamic Properties of the Detection Rate

A deeper look in to the behaviour of algorithms throughout the course of the stream reveals the properties of the Detection rates for each algorithm. For this purpose, the class-wise tracking of the model's reaction to the content of the stream at any time is performed in the mentioned two base Apache Spark applications: 1) Streaming simulation, and 2) Classical operation.

Streaming Simulation Scenario

In the streaming simulation scenario, each of the three Apache Spark algorithms are run 10 times for each of the training sets. To prepare the training set, the dataset is divided into 10 splits, and to simulate the streaming mode, each time, one split is considered to be the start of the stream. Every time, the training set is prepared from the start of the stream based on the label budget the model is trained on. Then, the model is applied all over the stream for the label predictions. For each split, 10 runs of each algorithm are performed, and the average of the detection rates are computed.

The plots for the average detection rate of each algorithm's runs throughout the course of the stream on different splits are then reviewed to check the performance of each algorithm. In the following, the summary of the observations for each algorithm's performance on 10 splits is given. Later, certain plots are selected for a visualized comparison of the algorithms on some of the interesting splits.

Decision Tree detects Background (the major class) $> 80\%$ for all the splits, and the Normal detection rate is in the range $\approx 45\% - 85\%$. Botnet detection is the highest rate, $\approx 85\%$, when Split 9 is considered as the start of the stream and the lowest, $\approx 40\% - 50\%$, when Split 7 is the start of the stream. Split 9 contains Neris, Rbot and NSIS botnet traces, which includes the most frequently occurring botnets in the overall stream. Split 7 only contains Rbot and Murlo botnet traces. Botnet C&C is not detected in almost all splits, except for *Split 6* and *Split 7*, which have a low rate.

Random Forest detects Background the best of all other classes. Botnet is best detected when *Split 9* is the start of the stream but on the other hand, the Normal DR is almost 0% in this case; this implies that gain in the Botnet detection rate results in the lowest Normal detection rate.

Naive Bayes is completely unable to detect minor classes, i.e. Normal and Attack classes, and could detect the major class (Background) up to 100%. There is only a slight detection of the Botnet class at *Split 6*, which happens at the expense of the lower detection rate for Background (95%).

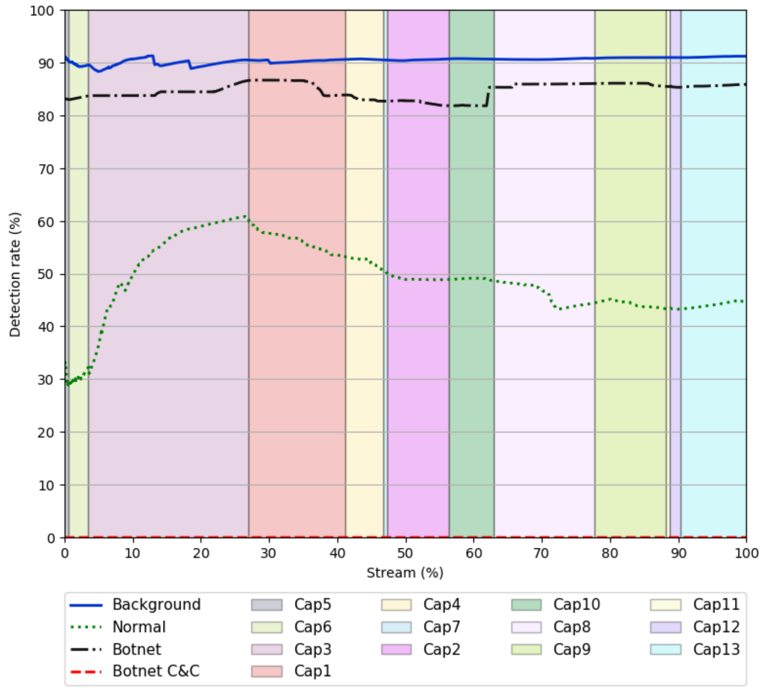
Based on the observations, Naive Bayes is shown to be completely incapable of performing well in this regard. For better comparison of Decision Tree and Random Forest, their performances are plotted under *Split 9* and *Split 6* scenarios as the start of the stream for the AvDR measurement (Figures 5.29 and 5.31, respectively) and for false positive rates (Figures 5.29 and 5.31 respectively).

Figure 5.29 demonstrates the detection rates throughout the course of the stream for Decision Tree and Random Forest from Apache Spark ML algorithms for the Split 9 (start of the stream) scenario. They seem to react the same in all classes except the Normal class, which Random Forest was not capable of detecting. The Botnet C&C is not detected at all in two algorithms, which is an indication of the inability of the models to learn this minor class. Comparison to the Stream-GP algorithms in Figure 5.33 for GP–Archive and GP–Both illustrates that only Background traffic is higher in the detection rate in Apache Spark. The lower level of the detection rate for Normal (20% less) as well as there being no detection of C&C in Apache Spark algorithms (70% difference), ensures us that the inability to handle changes throughout the stream and imbalanced data lead to ignoring minor classes and strengthening the major class detection rate. Split 9 has the properties of Rbot, Neris and NSIS botnets (Figure 5.15) and contains almost all the most important botnets, which is also occurring frequently (70%) in this dataset. So the Botnet detection is high and is over 80% throughout the stream.

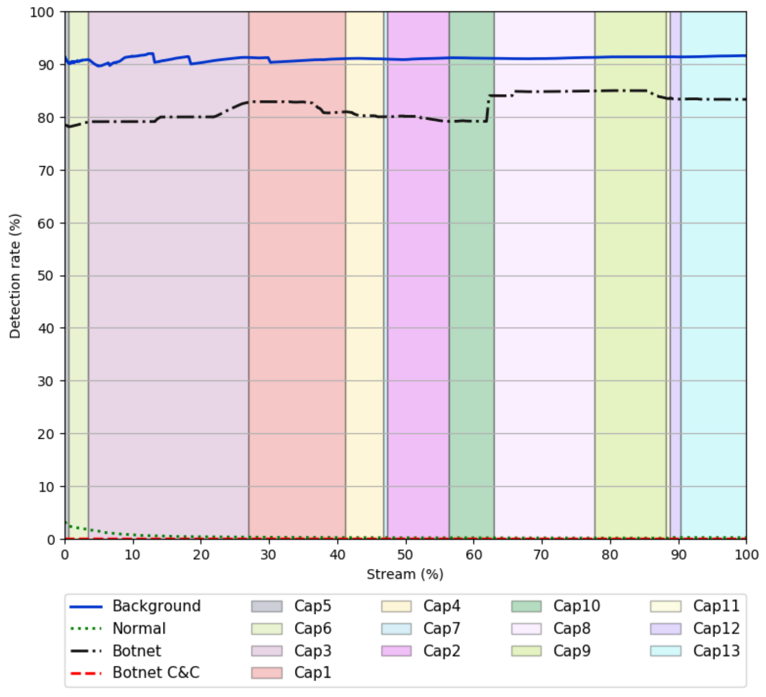
Figure 5.30 provides the false positive rate of the Apache Spark algorithm throughout when Split 9 is considered as the start of the stream. The Botnet class has a false positive rate of 10% for both algorithms, and the other major classes are close to 0%. Only the Background class (major class) has a high false positive rate (30% for Decision Tree and 60% for Random Forest). Previously, it was shown that Random Forest is incapable of detecting C&C and has a very poor Normal detection rate in this case (Figure 5.29), so this high Background false positive rate indicates that Random Forest is incapable of distinguishing between Normal and Background behaviour and mostly label it as Background.

Now, the performance of the algorithms are compared for the Split 7 (start of the stream), which demonstrates the best overall AvDR for the Decision Tree as seen in Figure 5.4.1. Figure 5.31 illustrates that the algorithm gets a high overall detection rate by having the high Background and Normal detection rates. The Botnet detection rate is low. The Botnet C&C is also detected at a pretty low rate, $\approx 15\% - 40\%$, but it is not still comparable to the streaming solution that Stream-GP provides. Random Forest behaviour is similar to the Decision Tree's behaviour, but Normal DR is lower, and Botnet C&C is not detected at all. Split 7 has the properties of Rbot and Murlo botnets (Figure 5.15), which explains why Botnet and Botnet C&C are best detected at segments 1-3 of the stream (contains Rbot activities) and segment 8 (contains Murlo activities).

Figure 5.32 also demonstrates the detection rates throughout the course of the stream for Decision Tree and Random Forest from Apache Spark ML algorithms for the Split 7 (start of the stream) scenario. In this case, not only is the false positive rate for the Botnet class low but also the detection rate. The normal class false positive rate ($\approx 10\%$) is the same for both algorithms, but Decision Tree could detect the Normal class at an 80% rate, while Random Forest could only detect it at a 50% rate.

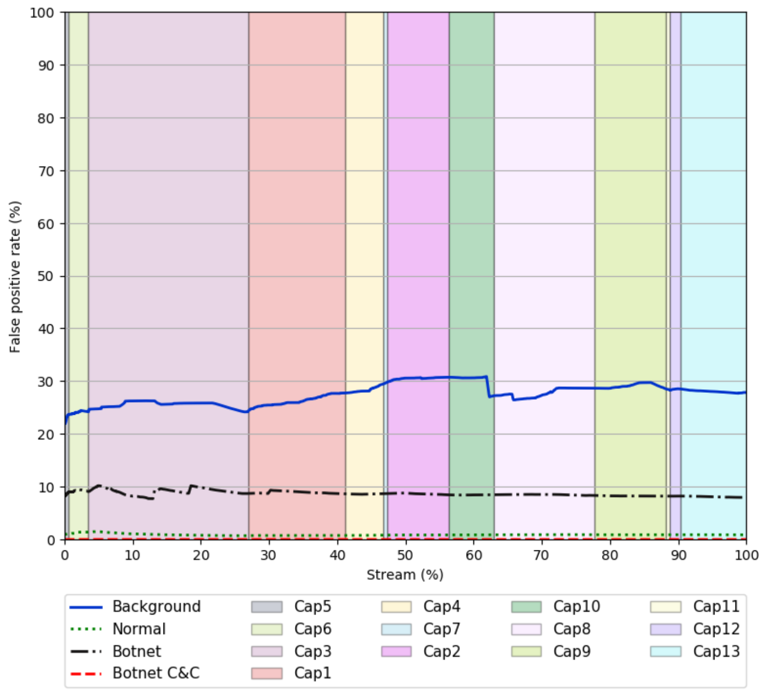


(a) Decision Tree (5.0%)

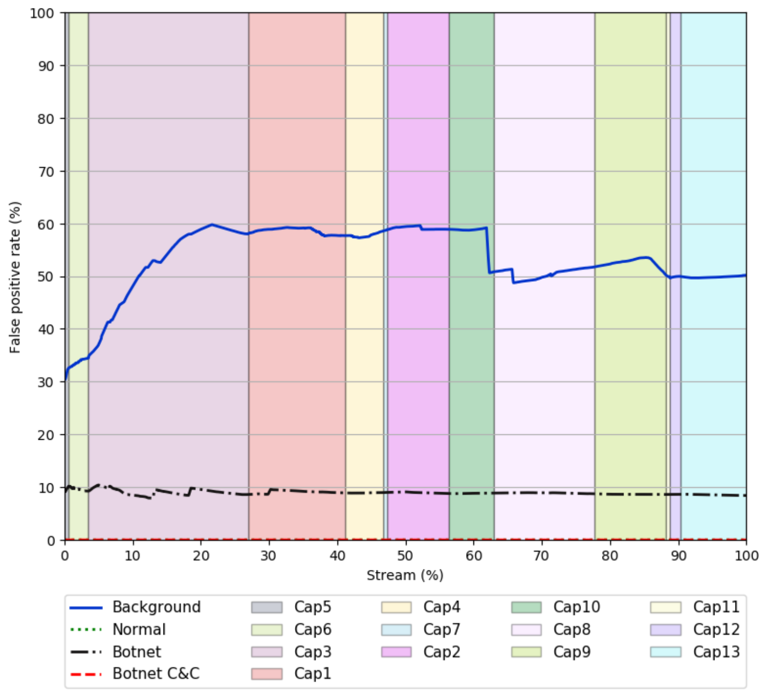


(b) Random Forest (5.0%)

Figure 5.29: CTU13-mixed class-wise Detection rate through the stream. Stream simulation scenario. Split 9. 5.0% label budget (Average of 10 runs)

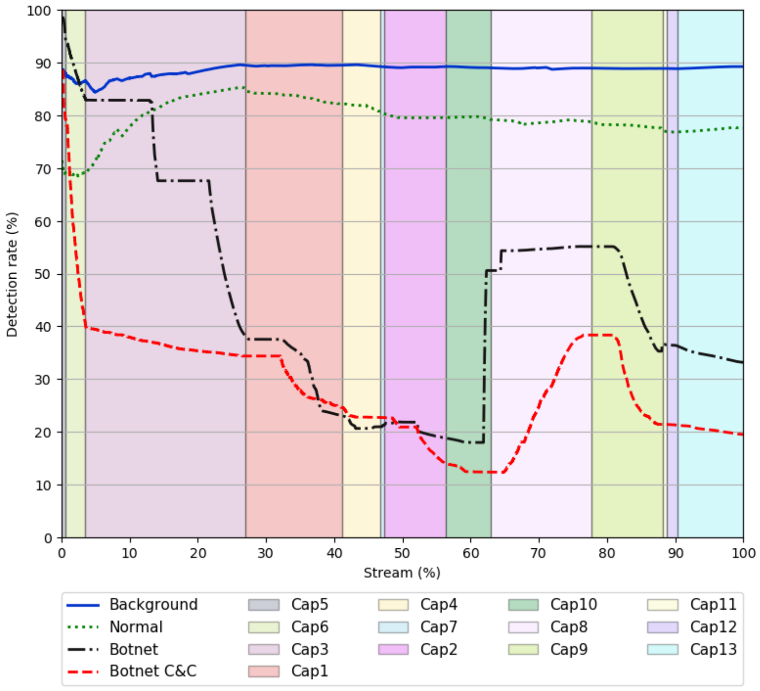


(a) Decision Tree (5.0%)

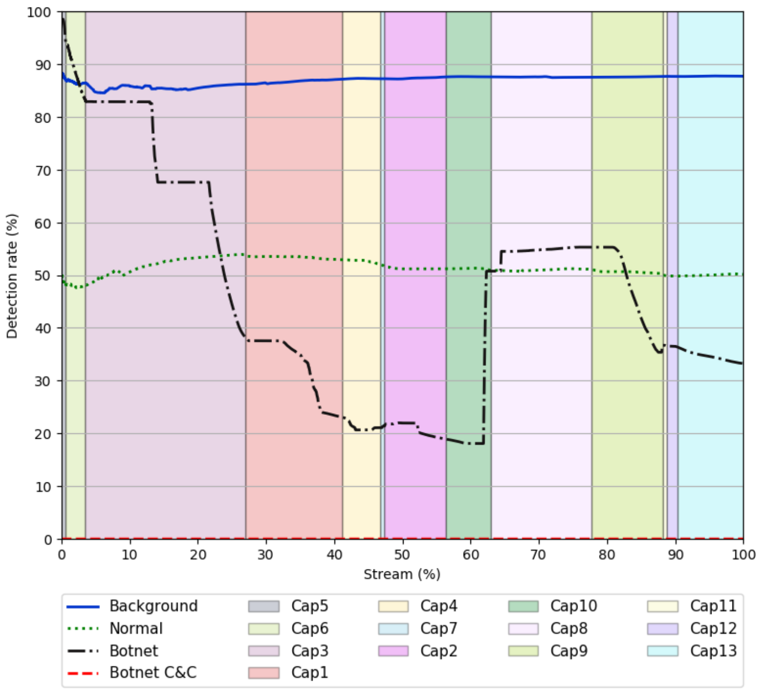


(b) Random Forest (5.0%)

Figure 5.30: CTU13_mixed class-wise False positive rate through the stream. Stream simulation scenario. Split 9. 5.0% label budget (Average of 10 runs)

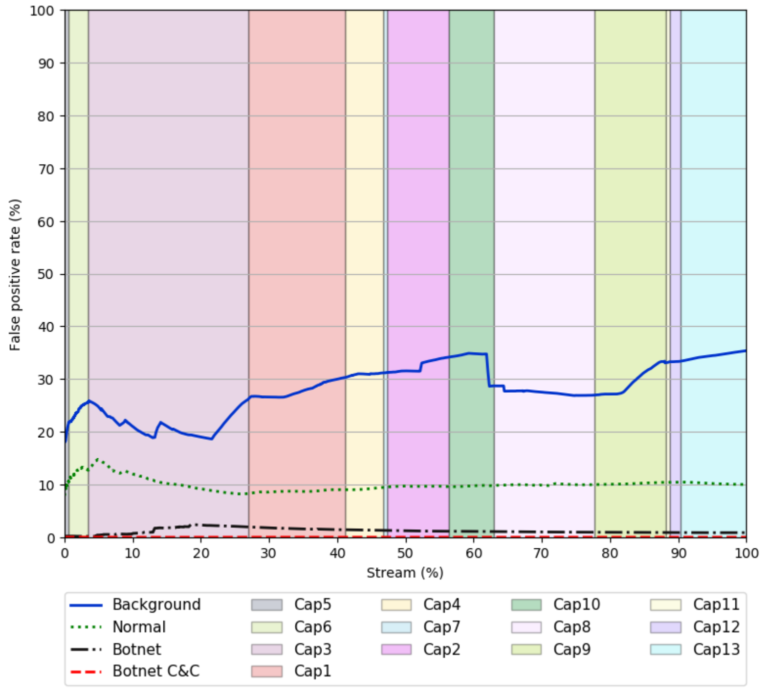


(a) Decision Tree (5.0%)

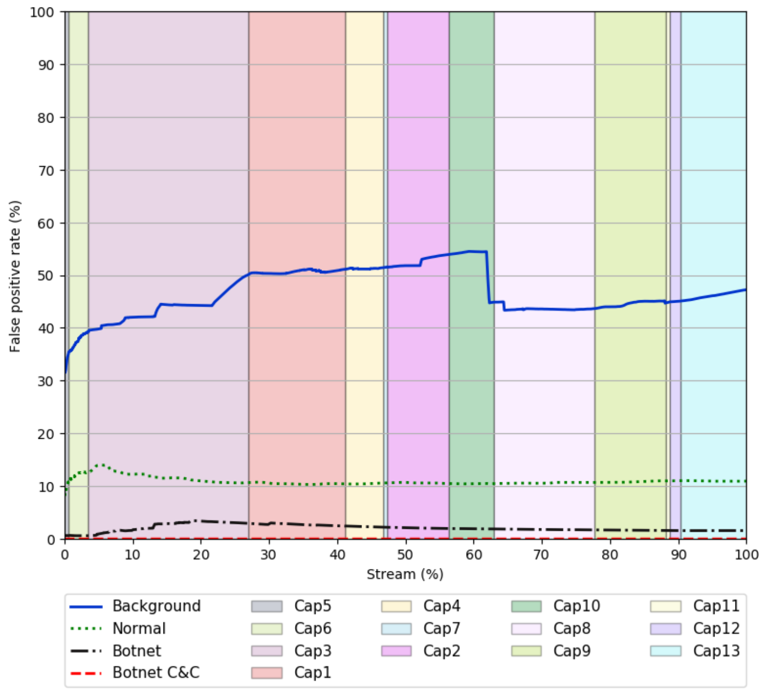


(b) Random Forest (5.0%)

Figure 5.31: CTU13_mixed class-wise Detection rate through the stream. Stream simulation scenario. Split 7. 5.0% label budget (Average of 10 runs)



(a) Decision Tree (5.0%)



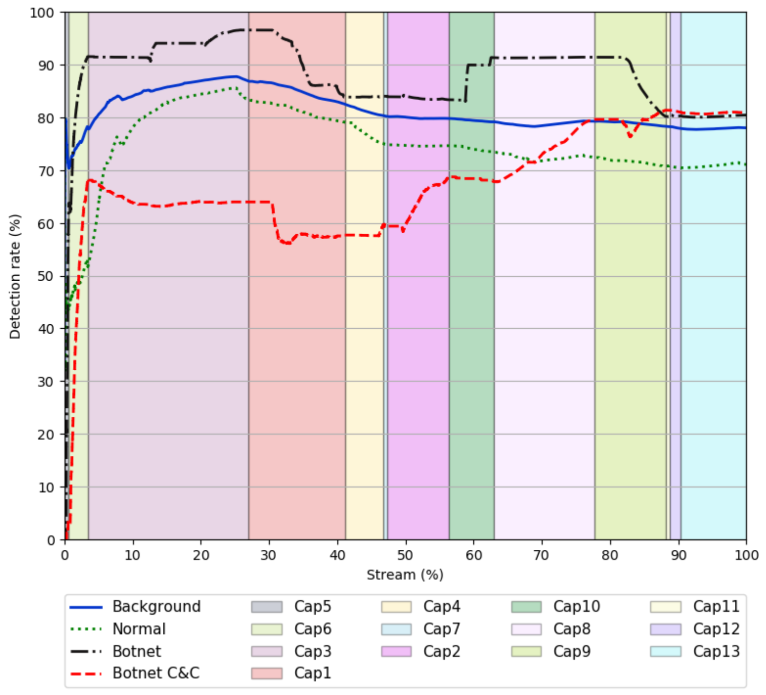
(b) Random Forest (5.0%)

Figure 5.32: CTU13-mixed class-wise False positive rate through the stream. Stream simulation scenario. Split 7. 5.0% label budget (Average of 10 runs)

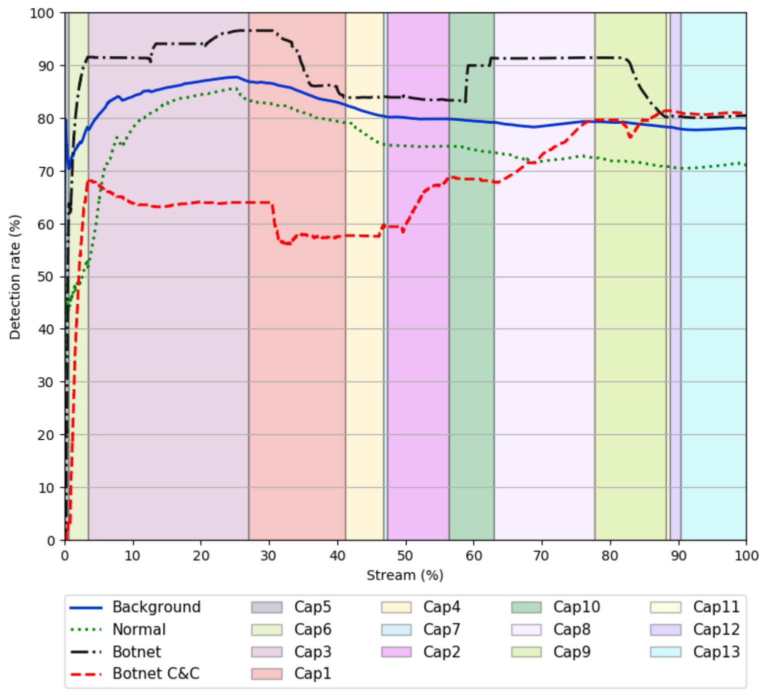
Classical Scenario

In the classical scenario, access to the content of the stream is assumed to be limitless so the training data are sampled randomly from all over the dataset based on a 5% label budget. The number of the Background class (major class) is kept the same as the sum of all minor classes. Decision Tree and Random Forest models are then trained on the training set in an off-line mode. Then, 10 runs of each algorithm are performed to label the rest of the data.

Figure 5.34 demonstrates the performance of the Apache Spark algorithms, Decision Tree and Random Forest, in the classical scenario, where only the label budget from the streaming scenario is kept. The comparison of this Figure to the performance of the Stream-GP algorithms (Figure 5.33) illustrates that Normal and Botnet behaviours are similar to the Stream-GP classifier with a lower Normal detection rate (this difference is higher in Random Forest, $\approx$ 20%), whereas Botnet C&C is not detected at all. The Background is detected about 10% more than in Stream-GP. This indicates that accessing the data from all over the stream helps to improve the minor class detection rates, except for the very rare Botnet C&C case, which is mostly missed in sampling universally from all over the stream. Figure 5.35 summarizes the false positive rate for the Apache Spark algorithms in the classical mode (Figure 5.34). The figure depicts that the false positive rates are less than 10% for all minor classes in both algorithms, and there is a 20% and 35% false positive rate for the Background class in the Decision Tree and Random Forest algorithms, respectively.

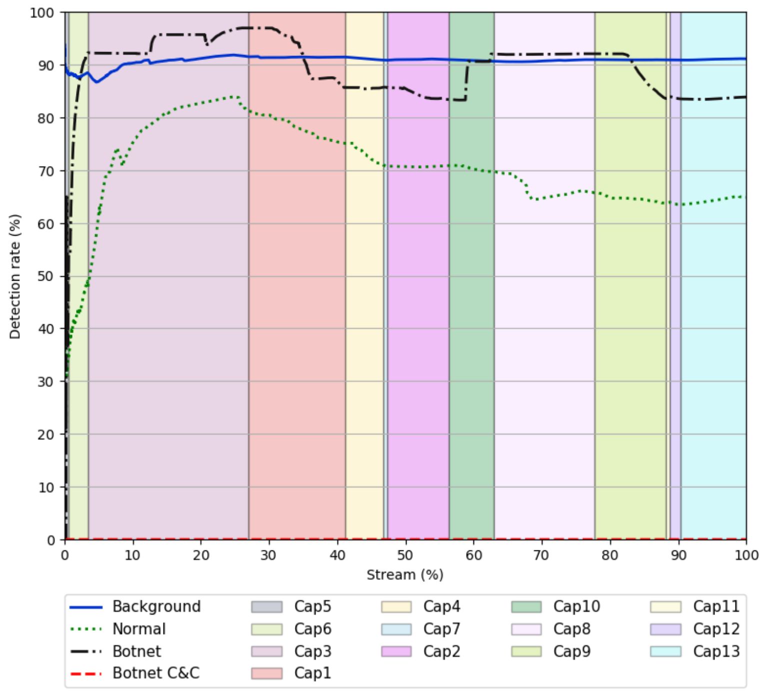


(a) GP–Archive (5.0%)

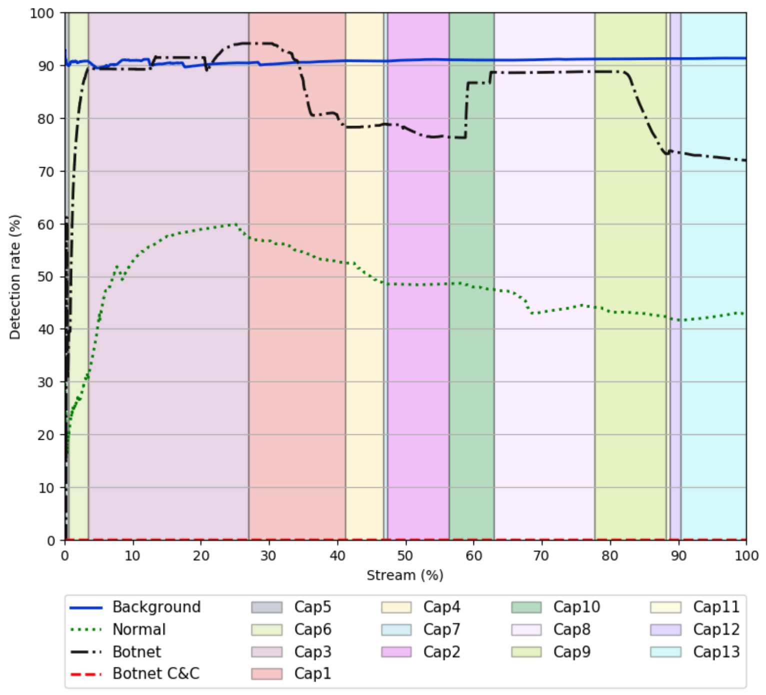


(b) GP–Both (5.0%)

Figure 5.33: CTU13-mixed Class-wise Detection rate through the stream at a 5% label budget based on the streaming AvDR metric (Average of 10 runs)

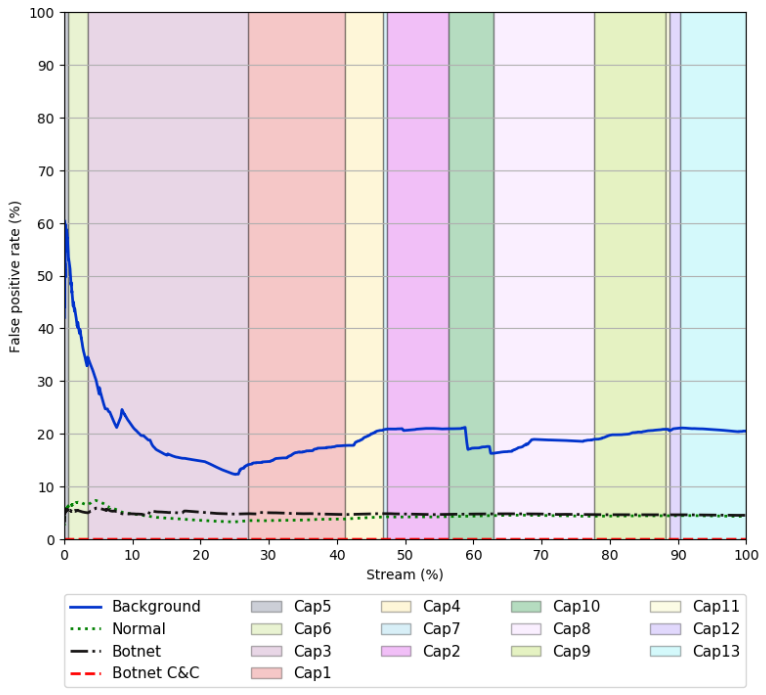


(a) Decision Tree (5.0%)

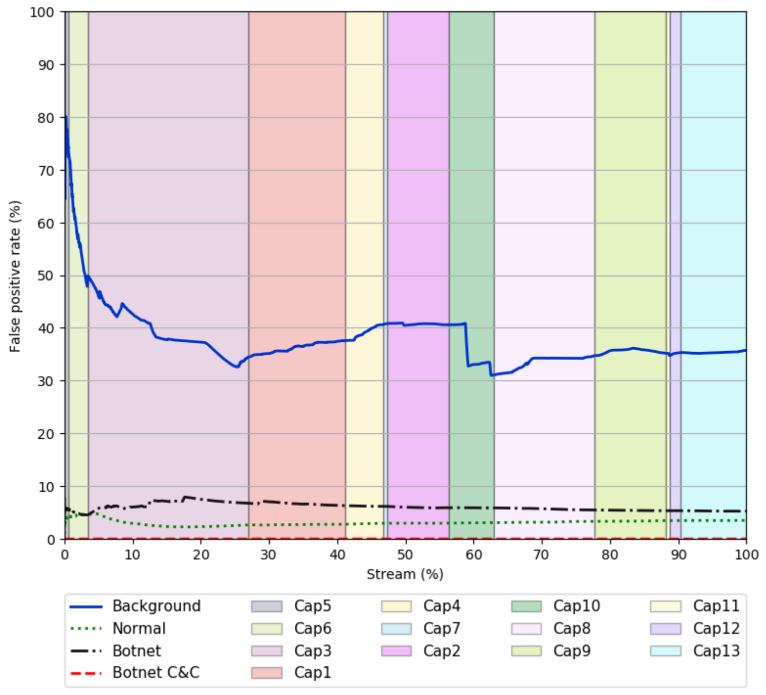


(b) Random Forest (5.0%)

Figure 5.34: CTU13_mixed class-wise Detection rate through the stream. Classic Scenario. 5.0% label budget (Average of 10 runs)



(a) Decision Tree (5.0%)



(b) Random Forest (5.0%)

Figure 5.35: CTU13_mixed class-wise False positive rate through the stream. Classic Scenario. 5.0% label budget (Average of 10 runs)

5.4.3 Capacity for Detecting Botnet Signals

The performance of the best algorithms for Stream-GP and its comparators, Apache Spark and MOA, on the specific case of botnet detection is reviewed in this section. Table 5.22 indicates the median AvDR of the runs for each algorithm. The results for the Apache Spark streaming-like scenario are obtained from the average of all splits. The result indicates that the Stream-GP and Apache Spark classic scenario is performing similarly on the botnet detection specifically. Although the classical Apache Spark algorithm is slightly better than Stream-GP, the fact that Stream-GP could keep up the same level demonstrates its effectiveness in minor class detection with limited access to the stream content.

Table 5.22: Median values for Botnet AvDR metric, under the 5% label budget. The best Stream-GP configuration (GP–Hybrid) and the best Apache Spark Algorithm (Decision Tree), *CTU13-mixed* dataset

Algorithms	AvDR Median (5%)
GP–Hybrid	87.5%
MOA–NB	84.2%
DT–Stream	59.0%
DT–Classic	89.3%

Table 5.23: p-value from the Mann-Whitney U test for the Stream-GP and Spark best algorithms based on Botnet median AvDR, for the 5.0% label budget, 95% confidence level

Algorithms	GP–Hybrid
MOA–NB	0.02
DT–Stream	1.8×10^{-5}
DT–Classic	1.8×10^{-5}

The pair-wise Mann-Whitney U test on the Stream-GP versus comparator algorithms is applied for a 5% label budget (Table 5.25). Thereafter, a Bonferrori-Dunn ad-hoc test is applied to find out if the Stream-GP algorithm is significantly different from all of the other algorithms overall. For this purpose, the α value ($\alpha = 0.05$) is divided into the number of comparisons ($k - 1 = 3$) in which the new α equals 0.017. With regard to the new alpha, the Stream-GP has been shown to be significantly different than Apache Spark algorithms in Botnet detection. The null-hypothesis is not rejected for the MOA–NB algorithm, which means that they are categorized in the same group in terms of botnet detection. This implies that with DT–Classic, which has an overall knowledge of the stream prior to the start of the stream, the botnets are better detected, which is not a realistic case in real-world streaming scenario. But even with that knowledge, Stream–GP is performing at a similar detection rate, and there is almost only a 1.5% difference between their median average detection rates. Figure 5.36 represents the wide range of the Apache Spark stream-based scenarios on Botnet detection, which is indication of the importance of the stream start point and the first exemplars entered.

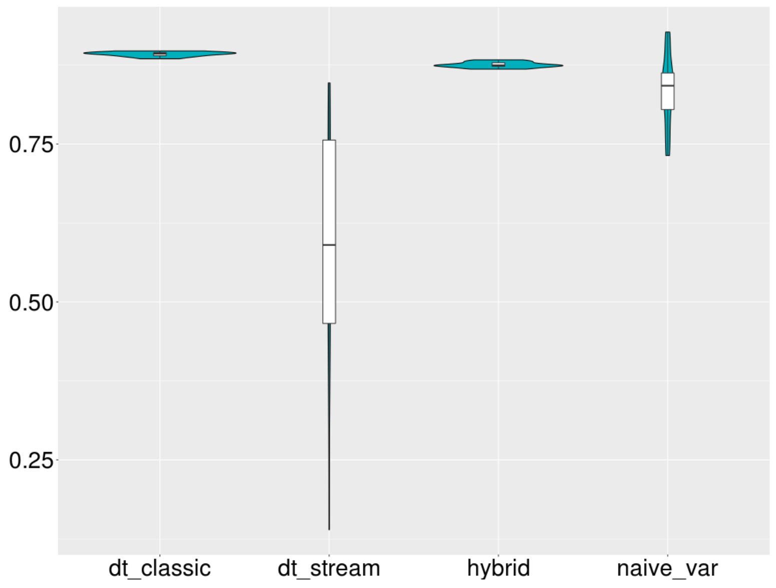


Figure 5.36: CTU13_mixed class-wise Detection rate through the stream. 5.0% label budget.

5.4.4 Capacity for Detecting Botnet C&C Signals

The best algorithms' performances of Stream-GP and its comparators, Apache Spark and MOA, are evaluated on the 5% label budget (Table 5.24). It is clearly shown that the Apache Spark algorithms are completely unable to detect the least frequent botnet C&C class throughout the course of the stream. In contrast, GP–Hybrid is consistently able to detect this scarce class with an AvDR median of 71.9%, and MOA–NB, with a detection rate of 57.7%, comes next. This means that Apache Spark is not able to detect the first indications of botnet attacks, which leads to more damage later in comparison to Stream-GP and MOA–NB. This demonstrates that classical offline solutions are not capable of detecting very rare classes in imbalanced data streams.

Table 5.24: Median values for the Botnet C&C AvDR metric under the 5% label budget. The best Stream-GP configuration (GP–Hybrid) and the best Apache Spark Algorithm (Decision Tree), *CTU13-mixed* dataset

Algorithms	AvDR Median (5%)
GP–Hybrid	71.9%
MOA–NB	57.7%
DT–Stream	0%
DT–Classic	0%

Table 5.25: p-value from the Mann-Whitney U test for the Stream-GP (GP–Hybrid) algorithm vs. the comparator Apache Spark and MOA algorithms based on Botnet C&C median AvDR, for the 5.0% label budget, 95% confidence level

Algorithms	GP–Hybrid
MOA–NB	0.001
DT–Stream	1.6×10^{-4}
DT–Classic	6.4×10^{-5}

The results of the pair-wise Mann-Whitney U test on the Stream-GP versus comparator algorithms demonstrate that Stream-GP is significantly better compared with the comparator algorithms on Apache Spark and MOA for the 5% label budget (Table 5.25). In other words, the null hypothesis is rejected for the 95% confidence threshold for each pair of comparisons. Afterwards, a Bonferrori-Dunn ad-hoc test is applied to find out if the Stream-GP algorithm is significantly better than all the other algorithms overall. For this purpose, the $\alpha(0.05)$ is divided into the number of comparisons ($k - 1 = 3$), and the new $\alpha = 0.017$. With regard to the new alpha, Stream-GP has been shown to be significantly better than all of the comparator algorithms in Botnet C&C detection.

5.4.5 Feature Selection by Decision Tree

A sample of Decision Tree in the stream simulation scenario on split 7 is illustrated in Figure 5.37. The graph represents the decision hierarchy based on specific feature values.

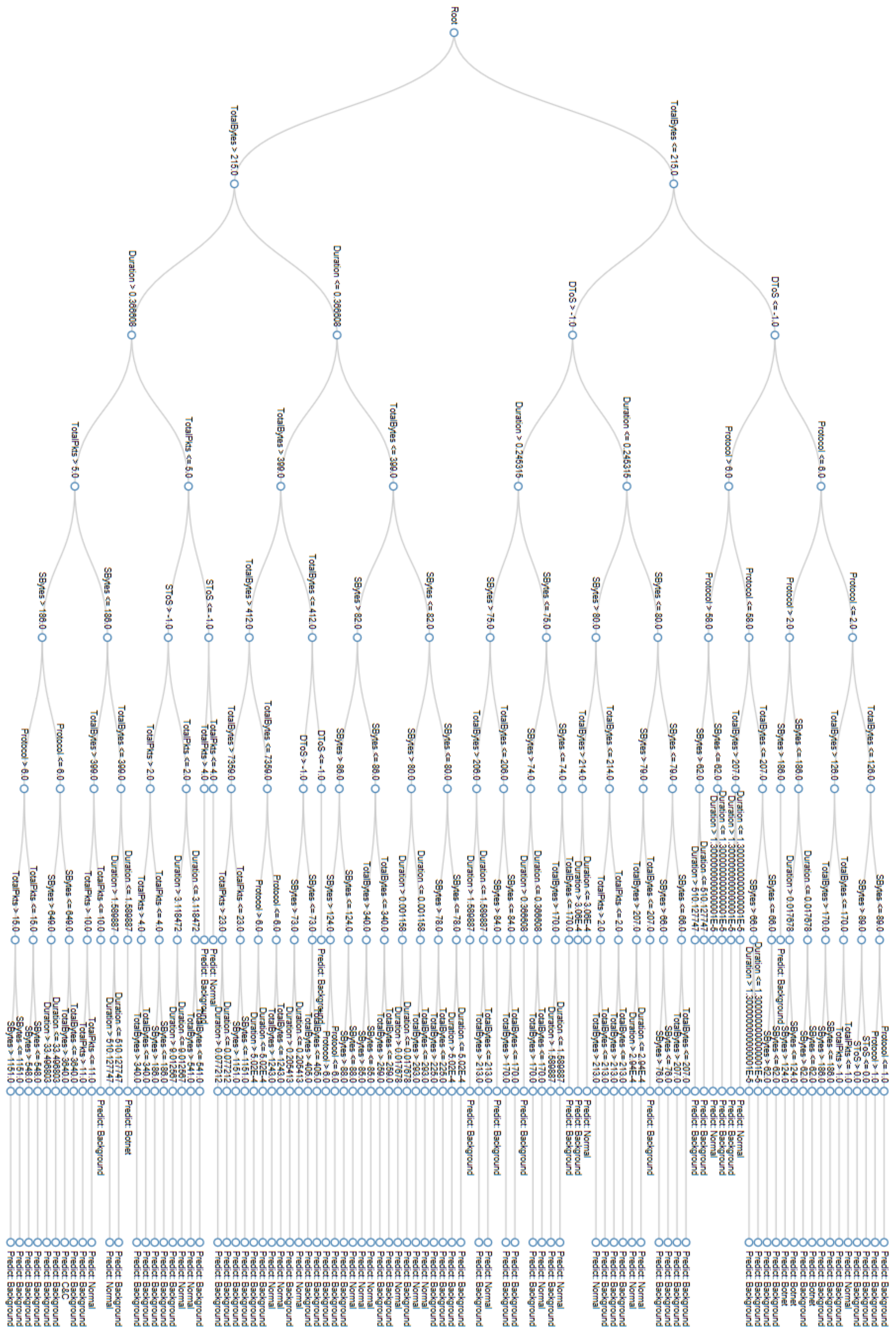


Figure 5.37: Overview of the Apache Spark Decision Tree in a stream simulation scenario for a run on split 7, 5% label budget

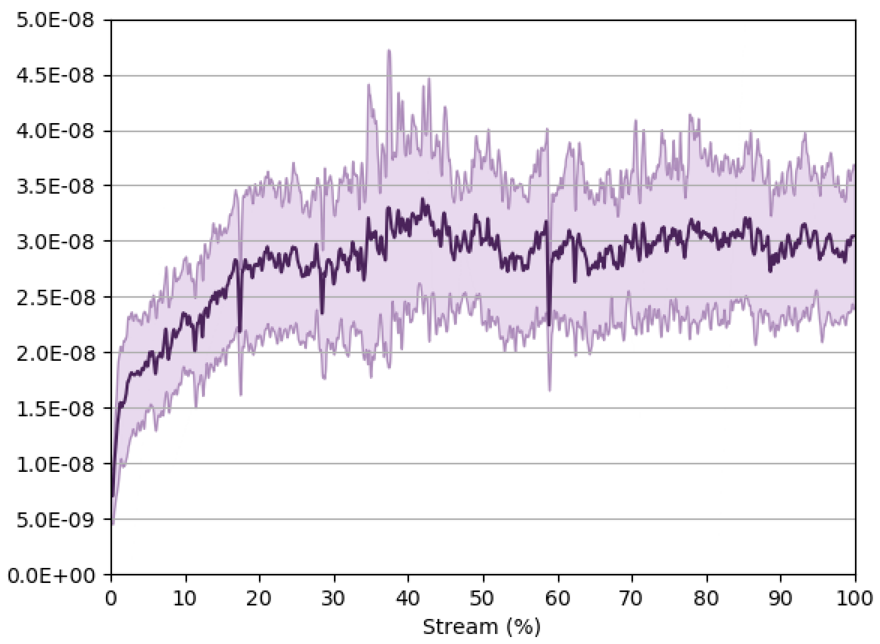
This represents a balanced binary tree illustrating that for any decision on the label prediction, the depth of the tree should be traversed at maximum. Seven features out of eight are selected in the tree for decision making (Direction is not used). This tree relates to the case Botnet C&C is detected unless any other runs could not detect it. One leaf is responsible for the Botnet C&C in this case. Most of the leaves are designated for Background (72) and Normal (24) detection, and only four leaves are assigned to the Botnet class. The lower number of Botnet nodes explains the low detection rate of the Botnet on split 7.

5.4.6 Computational Cost of Streaming Classifier Operation

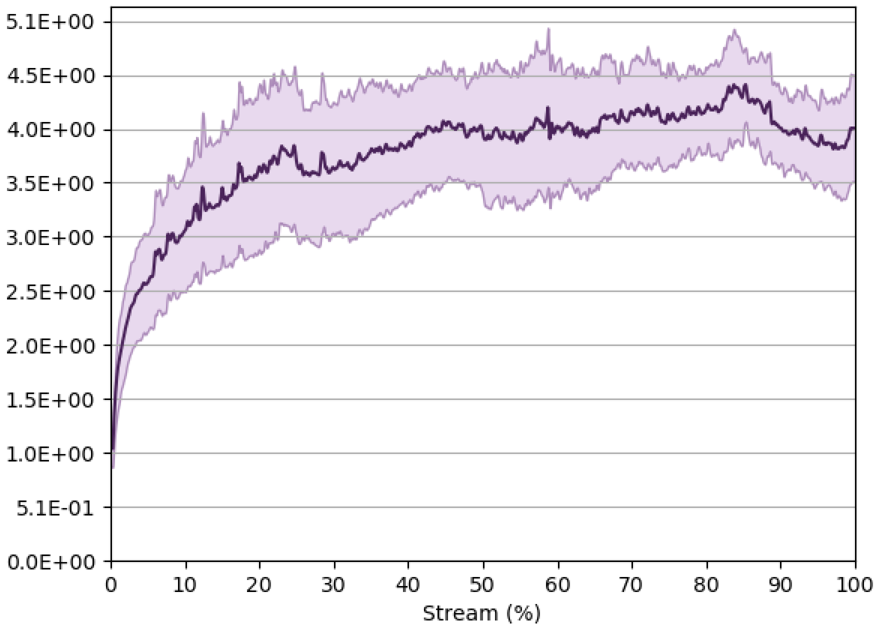
The computational cost for Stream-GP is calculated based on Section 5.2.6 and demonstrated in Figure 5.38, where it is the comparison counterpart in this section. Based on the illustration, it takes $\approx$ 30ns (Figure 5.38(a)) for the Stream-GP champion to predict labels for a window location under a 5% label budget. On the other hand, 4s is the average time to update the Stream-GP model.

The computation for Spark algorithms is done for the stream simulation scenario, but it also corresponds to the classical scenario as well, as the only difference between these two scenarios is that in the latter, the access to the stream is not limited. To this purpose, two phases are available to be computed: 1) Training and 2) Test. At the training phase, exemplars are sampled randomly based on the label budget, and then the model is trained based on that. This process is done completely off-line and the models are kept to be applied on the streaming data. On the other hand, for the test phase, the data are a stream of flow-based network traffic, and the model is applied on each exemplar to predict the label. To keep consistent with the Stream-GP results, the times are reported based on the same window size as Stream-GP.

Figure 5.39 shows the violin plots for Apache Spark algorithms' times for 10 runs on each split of the dataset as deployed on an Intel i5 CPU 16@2.67GHz and 48GB memory. The average time for the Apache Spark Decision Tree classifier model, the best-performing Spark algorithm, to label a record is $\approx$ 3ms (Figure 5.39(a)), whereas the Stream-GP classifier labels a window in $\approx$ 30ns. This indicates the notable difference between the execution times of the two methodologies. One possible explanation could be the programming languages they use, as Stream-GP is written in C/C++, whereas Apache Spark uses in Python code.

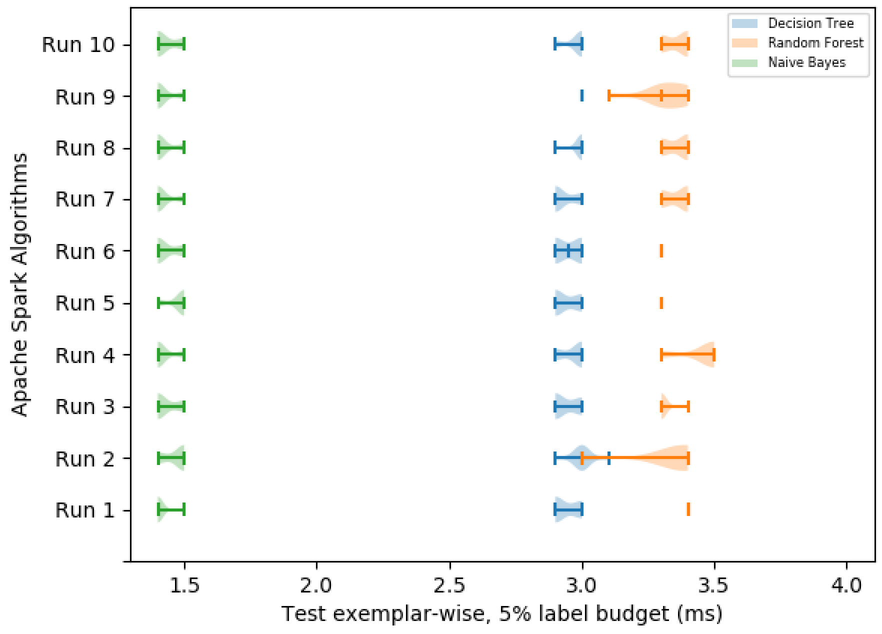


(a) Test Time

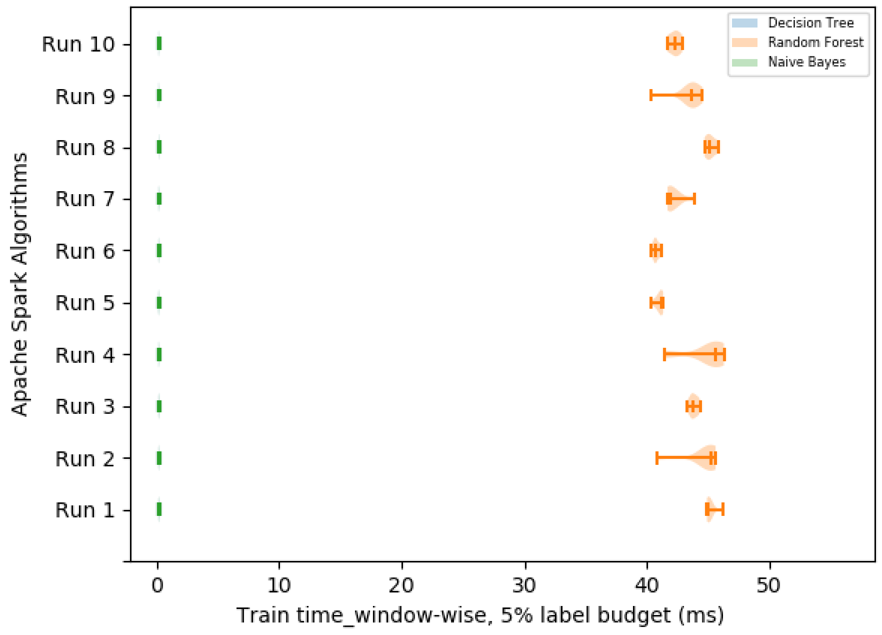


(b) Training Time

Figure 5.38: Wall clock time, GP–Archive (a) champion individual to make predictions and (b) fitness evaluation to update the content of the population on a new non-overlapping window location for the CTU13-mixed dataset on a 2.67GHz CPU.



(a) Test Time



(b) Training Time

Figure 5.39: Wall clock time (a) model to make predictions for each exemplar and (b) Off-line model that is trained on the training set for the CTU13-mixed dataset on a 2.67GHz CPU.

Although training is not in a streaming mode for Apache Spark algorithms, the total training time is divided by the number of windows to give us the opportunity to make the comparison. Stream-GP takes 4s to develop new models as opposed to the 45ms of models from Apache-Spark. However, this never has an impact on its ability to provide labels (as a champion classifier is always available, or a 30ns throughput), but reflects the cost of maintaining a population of models (which provides the ability to track multiple phenomena simultaneously).

5.4.7 Complexity of Algorithms

To compute the complexity of the algorithms, the number of executable code lines in Stream-GP is considered, whereas the number of nodes in the tree is conceived for the Apache Spark tree based algorithms.

GP—Archive is selected as the representative of the Stream-GP algorithms. Figure 5.40 represents the average number of executable code lines, both effective and introns, in a program based on the labels in the champion team, which is responsible for labelling the stream content. The following observations are concluded from this representation:

- The average number of total executable code lines for the GP–Archive champion team in all classes is around 42 line of codes.

- The average number of non-intron code lines that has an effect on the result of the program execution is around 12 for all classes.

The above statements mean that only 1/3 of the code lines in the programs of the champion team are effective and need to be executed to predict the label. So, on average, 12 code lines are executed in a single champion program to suggest a label at the end. Figure 5.41 represents the average champion team size (number of programs in a champion team) throughout the stream, which equals 18.8 on average. Therefore, the total number of code line executions is 4,800 code lines for a single exemplar (12 (champion executable code lines) * 18.8 (average champion team size throughout the stream for a 5% label budget) = 225.6). This is the complexity of the GP–Archive code for the champion label suggestion for a single exemplar. In addition, this gives us a clue on how to reduce the computational time in Figure 5.38(a) to 1/3, as these results are from the code with executing the introns as well.

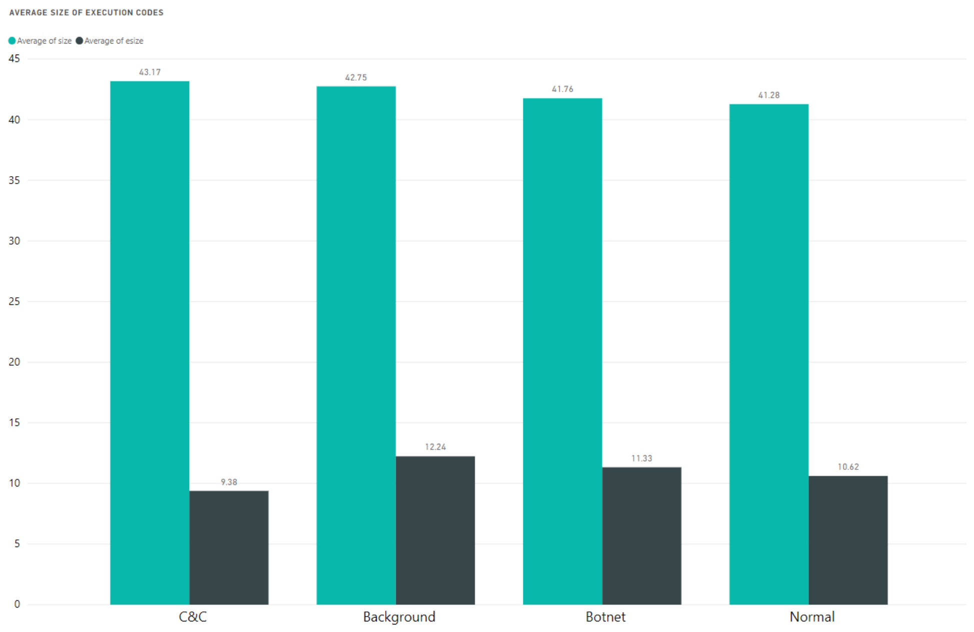


Figure 5.40: GP–Archive average number of execution codes, whole number and effective number, for the champion throughout the stream. 5% label budget.

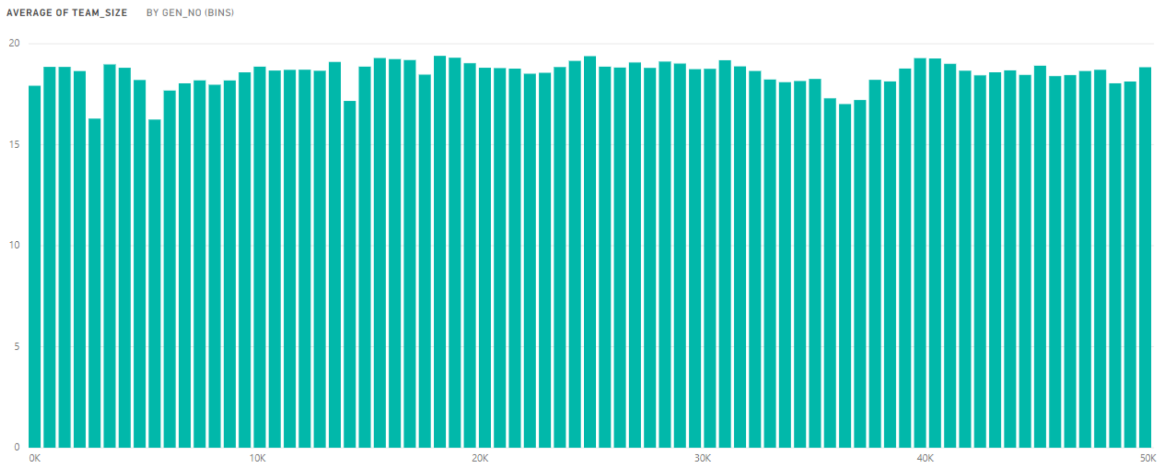


Figure 5.41: GP–Archive average champion team size throughout the stream. 5% label budget.

In Apache Spark, the decision tree is a balanced binary tree. Therefore, in the worst case, the maximum depth of the tree is traversed. In the case of Decision Tree, the maximum depth of the tree is 7, so for each exemplar, seven comparisons are needed to reach the final decision. So, the overall number of comparisons would be 7 (maximum depth of the tree) for a single exemplar. In the case of Random Forest, the maximum depth of the tree is 4, and 50 trees are there, so overall for a single exemplar 200, (4 (maximum depth of the tree) * 50 (number of trees) = 200).

Based on the computations of the complexity of the algorithms, Decision tree is 32 and Random Forest is 1.2 times less complex than Stream-GP. The complexity of the Decision Tree is less than Stream-GP, but the computational time shows that it takes more time than Stream-GP to label a window (Section 5.4.6). Factors contributing to this include: 1) the different languages used to implement each algorithm (Python versus C++) and, 2) Stream-GP solutions use arithmetic as well as conditional instructions as opposed to just conditionals (where arithmetic instructions execute faster than conditional instructions on modern CPUs due to more predicable behaviour under instruction level parallelism).

5.4.8 Summary

Stream-GP algorithms' performances are compared to the best state-of-the-art Apache Spark Streaming ML classification algorithms in a 5% label budget. Apache Spark does not provide a streaming learning process for its ML classification algorithms. Therefore, for evaluations, the training process happens in an off-line mode with two different scenarios. Later, the prepared trained model is used on the stream of data to predict labels. The two modes of training the models are: 1) Streaming simulation and 2) Classical operation. The observations specified that the Naive Bayes algorithm as an off-line model is completely incapable of being applied for the classification of the streaming data. In both streaming simulation and classical operation, Decision Tree and Random Forest had acceptable detection rates but still could not reach the Stream-GP algorithms, especially when non-stationary data are encountered. In the streaming mode, the point where the stream starts affects the detection rates because only limited behaviour is seen. Moreover, Decision Tree is shown to perform better than Random Forest. In all cases, Stream-GP algorithms are shown to perform significantly better than Apache Spark Streaming ML algorithms, which is due to their incapability to react to changes. In addition, they can not detect the important Botnet C&C class so they are not able to detect the botnet attack early in its fetus stages, while Stream-GP has been shown to be powerful in this. This indicates the weakness of the Apache Spark algorithms in dealing with the significant class imbalance. Furthermore, the labelling time is shown to be dramatically less in Stream-GP compared with the Apache Spark algorithms. However, the complexity of Stream-GP is 32 times more than Decision Tree and 1.2 times more than Random Forest. The time and complexity evaluations are done on the multi-bot stream with 5% label budget. The time and complexity to provide labels for the stream data increases gradually as it progresses through the stream due to the increased complexity of Botnet and Normal behaviours. Also, they would increase as more complicated behaviours are associated with the network traffic, which could be due to the difficulty of classes' detection. However, it should be noted here that the time and the complexity of the GP-Stream algorithms will decrease as the label budget available decreases.

5.5 Network Analytic Applications

The application of Stream-GP on network traffic data is quantified and compared to the available streaming and network tools in Section 5.1 to Section 5.3. In this section, another aspect of the stream-GP application in the network field is demonstrated. In Section 5.5.1, the utilization of Stream-GP to uncover the unknown behaviours of network traffic flows is discussed.

5.5.1 Traffic Analysis

Background traffic of the CTU13-mixed dataset is investigated by two algorithms: Stream-GP (GP–Archive) and Apache Spark Random Forest. In the case of Stream-GP, the model is built online through the exposure to the stream content, i.e. network traffic. However, Random Forest is learned off-line over the ground truth data where the true labels are known. Then, the built model is applied to the unknown Background traffic for predictions. In this section, some analysis is represented for both Ground Truth (GT) and Background Traffic.

Ground truth traffic refers to the portion of the CTU13-mixed dataset in which true labels are provided by a human expert based on known behaviours [49]. Like real world online ML scenarios, only a portion of the GT is used for learning purposes based on the label budget, i.e. 5% in this experiment, whereas for off-line methods, Random Forest uses all this information for training purposes.

There are some differences evident in Ground Truth and Background traffic. For example, some protocols and/or destination port numbers are available in Background traffic that are not seen in the Ground Truth information. An example is destination port 13363, which makes up a major part of the Background traffic but has no track in the Ground Truth traffic. In addition, there is more Botnet traffic than Normal traffic in Ground Truth traffic, which is unlikely to happen in real life, Figure 5.42.

The focus on malicious behaviours in the Ground Truth traffic, Figure 5.43, leads to the following observations:

- Most of the Internet Control Message Protocol (ICMP), Simple Mail Transfer Protocol (SMTP) and Secure Shell (SSH) network flows are labelled as malicious.
- Big portions of the Secure Hyper Text Transfer Protocol (HTTPS) and Domain Name System (DNS) traffic are also malicious.
- A part of HTTP traffic is also malicious. If the focus is only on Botnet C&C traffic, it is mostly HTTP rather than HTTPS. Correspondingly, one well-known way of establishing Botnet C&C connections in literature is using HTTP open port 80 [44].

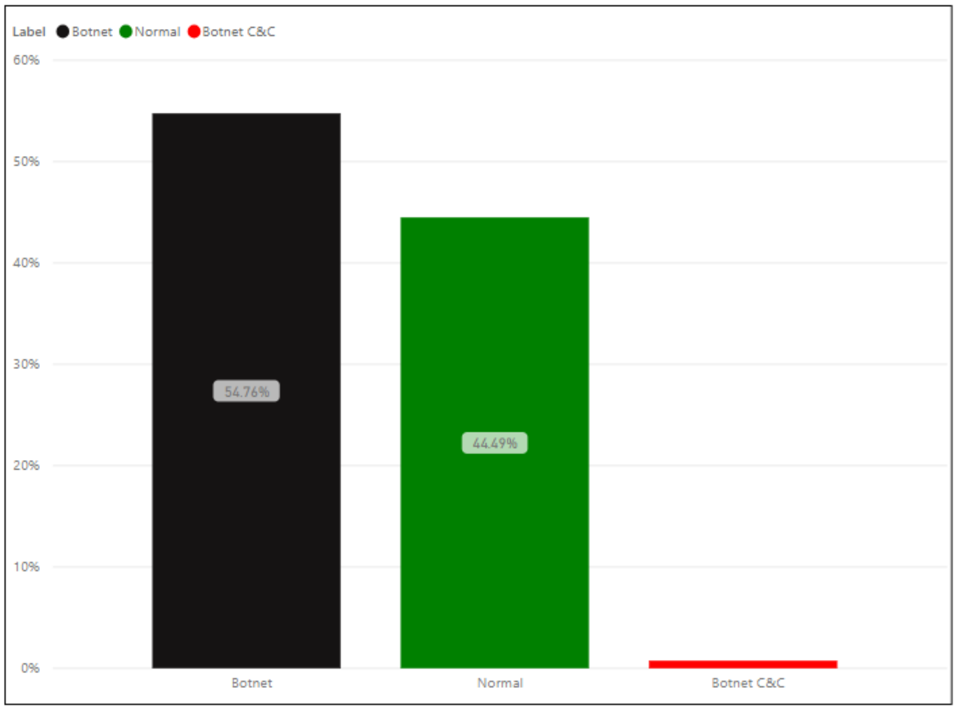


Figure 5.42: Distribution of Ground Truth traffic, CTU13-mixed dataset, Labels: Normal (1), Botnet C&C (2) and Botnet (3)

Botnets use omnipresent protocols like HTTP and DNS to hide their malicious activities within normal transactions [1]. DNS protocol is one indivisible part of Internet transactions that can not be filtered completely by firewalls, which makes it a safe place for malicious activities to happen within. The same relationship is also established for HTTP protocol. Intruders are taking advantage of these well-known, non-removable protocols to pretend their traffic is Normal and bypass firewalls. Another technique is encrypting their content to hide their identities [110].

Background traffic is investigated after the ground truth traffic is reviewed to find out how it is composed. This process of detecting the unknown behaviours of Background traffic is done by applying the algorithms on its content and analyzing their predictions.

Firstly, the Stream-GP outcome is investigated. Most Background traffic is detected as Normal, but a big part of it is recognized as malicious. The following analysis is done by focusing on the malicious activity traffic labelled by Stream-GP:

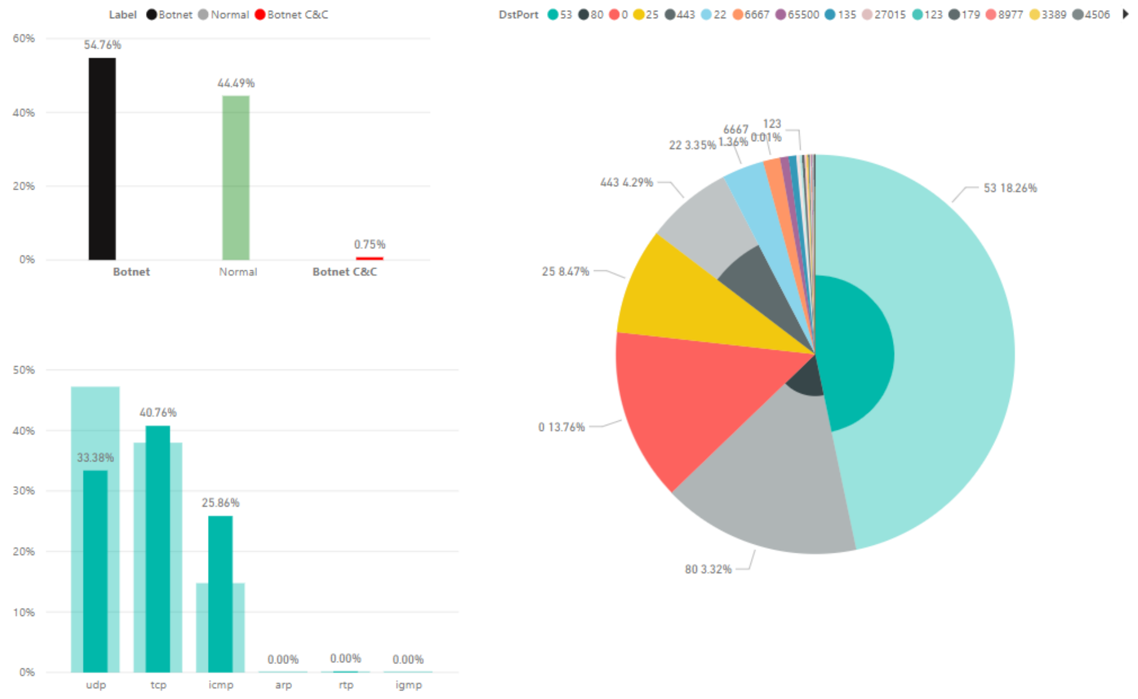


Figure 5.43: Botnet activities in ground truth traffic, CTU13-mixed dataset, Labels: Normal (1), Botnet C&C (2) and Botnet (3)

- The malicious activities are done mostly on the following protocol/destination port numbers: DNS (53), HTTP (80), HTTPS (443) and unassigned ports like 6881 (probable bittorrent) and 13363. The first three protocols are on the top list of well-known botnet communications [1]. Destination port – 6881 is an unknown port number but is mostly known to be used by bittorent protocol, which can be used to build the peer-to-peer botnet type [53].

- ICMP network flows are considered malicious by Stream-GP, given that these could be the first step in port scanning [28].

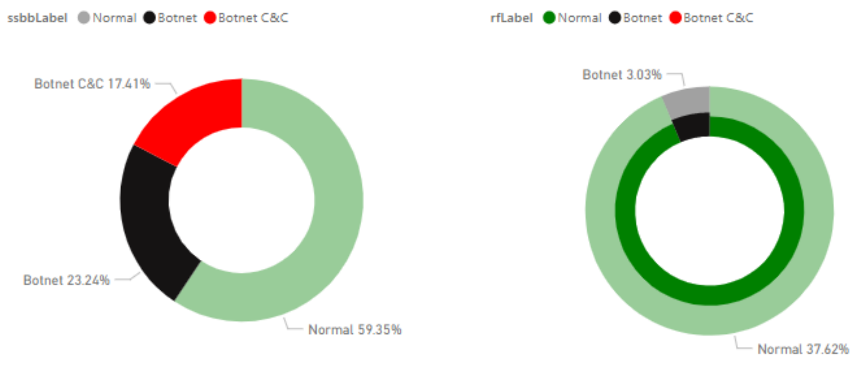
- Some DNS network flows are detected as malicious by Stream-GP, whereas Random Forest seems to miss them. It is shown that one popular form of botnet attacks is through the use of DNS protocol [125][35].

- Address Resolution Protocol (ARP) flows are considered normal in Random Forest but are considered malicious by Stream-GP. This could be happening because the majority of ARP flows are labelled as malicious, so Stream-GP seems to generalize this known behaviour.

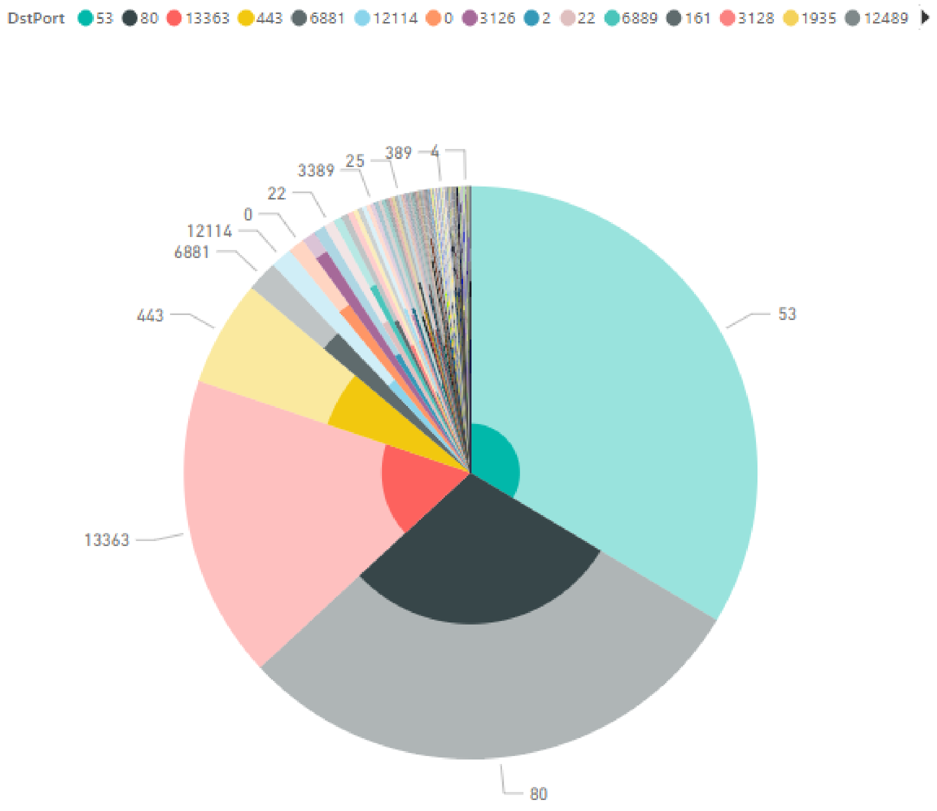
- Most malicious TCP connections are HTTP and HTTPS. Also, all network flows destined to port 8088, a HTTP proxy port, are detected as Botnet by Stream-GP and Random Forest.

5.5.2 Summary

The background traffic in the CTU13-mixed dataset is investigated by 1) Stream-GP (a streaming ML classifier) and 2) Apache Spark Random Forest (off-line classifier). The results from these two algorithms are then compared to the known Ground Truth traffic. It is demonstrated that there are known and unknown behaviours available in the CTU13-mixed Background traffic, which is not evident in the Ground Truth traffic provided with the dataset [49]. The known behaviours predicted by Stream-GP algorithms are distributed in the following categories: Normal (13%), Botnet (5%) and Botnet C&C (4%). The rest of the Background traffic (78%) is still considered “unknown”. Stream-GP could shed light on unknown traffic, and, based on observations, it could generalize the learned known behaviours of Ground Truth traffic. Therefore, Stream-GP could be used by human analysts in cases of huge volumes of network traffic in real scenarios to analyze upcoming traffic.



(a) Class distribution



(b) DST port

Figure 5.44: Stream-GP analytics on malicious botnet activities, CTU13-mixed dataset, Labels: Normal (1), Botnet C&C (2) and Botnet (3)

5.6 Summary

The active learning Stream-GP framework is evaluated under different network scenarios. Firstly, it was evaluated under the CTU13 dataset, which contains 13 different Botnet scenarios. Four different combinations of the sampling/archiving policies were examined under three different label budgets; 0.5%, 1% and 5%. The results demonstrate that the GP–Archive policy is effectively working in network applications with regard to the network stream challenges, e.g. class imbalance, label budget, partial observability, label budgets and anytime operation. GP–Both is the runner up in the results. The ability to detect Botnet and Botnet C&C is also investigated. It appears that even with the low rate of Botnet C&C, 1% of the stream, GP–Archive could manage to detect it under low label budgets as well.

Then, the best performing Stream-GP and MOA comparator algorithms based on the ranking tables in the CTU13 dataset are selected for benchmarking under a multi-bot scenario. The CTU13 dataset is concatenated in a way as to make the gap between similar botnets as wide as possible. A new algorithm, GP–Hybrid, is introduced, the its power of which relates to its greedy behaviour in picking up the very rare classes at the beginning of the stream and raising their detection rates, then continuing in a subtler way for the rest of the stream. Both algorithms, GP–Hybrid and GP–Archive, were ranked *first* and *second* in the multi-bot scenario. Moreover, GP–Hybrid showcased its power in Botnet C&C detection and got the top rank with a $\approx 15\%$ difference with the comparator MOA algorithm in both label budgets. The results show that Stream-GP is able to re-use previous learned behaviours when they occur again in the stream.

Later, a comparison of the top-ranked algorithms from Stream-GP, GP–Archive, and MOA was done under a 5% label budget on the network datasets while introducing two more datasets, ISOT and NSL-KDD. The evaluations once more identify the significant superiority of the GP–Archive over the comparator algorithms. The false positive rate is studied as one of the main evaluation metrics in the network domain.

Overall, the Stream-GP is compared to Apache Spark Streaming as the state-of-the-art framework as a network stream processing tool. The three best algorithms from the Apache Spark ML library are selected for the sake of comparison: Naive Bayes, Decision Tree and Random Forest. The Apache Spark Streaming ML library does not support streaming learning algorithms. Therefore, the comparison is done in two modes of operation: Streaming Simulation and Classical operation. The difference between these two modes are their access to training data, as streaming only has access to the beginning of the stream, whereas classical has no limitations. The results demonstrate that the Naive Bayes algorithm in Apache Spark was completely incapable of performing properly in the mentioned scenarios. It is interesting that this algorithm in the streaming learning mode in the MOA toolset was one of the best-performing algorithms when it uses the variable uncertainty policy for sampling. Among Decision Tree and Random Forest, Decision Tree has been shown to perform better, but it still cannot reach the Stream-GP algorithms. The significance test results demonstrate that the Stream-GP algorithm, GP–Hybrid, is significantly better than the Apache Spark and MOA comparator algorithms with more than a 15% difference on the AvDR median. This shows, however, that the access to the stream is not limited in one scenario, but the model still cannot use the knowledge to get a higher detection rate due to the non-stationary behaviour of the data.

Finally, another sample of the utilization of Stream-GP in the network area is demonstrated. In this sample, it is used to reveal more information on the Background traffic in the CTU13-mixed dataset.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Streaming processing is a new trend in today's computational world. Its necessity is clear in the network field, where there is a huge volume of data and low storage capacity. There are only a few streaming algorithms available, but their number is growing. However, in the network field, the focus is more on the framework to handle the data in a streaming way rather than learning simultaneously from the stream content. The potential challenges associated with the network stream are highly imbalanced data, costly labelling, non-stationary behaviour, etc. Existing streaming algorithms have not addressed the combination of streaming requirements in a holistic way before. This thesis for the *first* time provides an integrated solution for addressing all features of streaming data classification simultaneously. In addition, the proposed framework is based on network traffic flow analysis so it does not depend on a specific type of botnet architecture. This makes it widely applicable in real world problems.

The active learning approach in the proposed GP-based framework, Stream-GP, provides the opportunity to decouple the distribution of training data from that of the streaming data. The approach relies on keeping a subset of data from the stream. The subset is filled by a sampling policy that is subject to a label budget. Then, it becomes updated by the removal of some exemplars using an archiving policy and replacing afterwards. Proper sampling / archiving policies make the distribution of data subset more balanced.

In this thesis, combinations of sampling / archiving policies are benchmarked under label budget constraints to determine the best combination policy for network streaming applications. The benchmarking happened on several network security datasets where the ML models are mostly applied. The comparison of Stream-GP with the existing streaming frameworks in general and specifically in the network field demonstrates its effectiveness in overcoming the challenges associated with streaming classification problems. It is shown that the provided policies could help in balancing the training data subset. The significance test illustrates that the Stream-GP framework performs significantly better than comparator algorithms where the GP–Archive algorithm is always ranked *first* in Botnet detection in thirteen independent datasets.

GP–Archive and GP–Hybrid are the top-ranked Stream-GP algorithms in network security algorithms, specifically botnet detection. The GP–Archive combination policy is based on uniform sampling and biased archiving, which gently balances the content of the data subset and introduces new changes in the stream. To speed up the introduction of the minor classes, GP–Hybrid could be specifically used. It helps to increase the minor classes' detection rates to a predetermined threshold at the beginning of the stream. Based on the application, any of the Stream-GP combination could be used, but GP–Archive and GP–Hybrid are recommended due to their effectiveness despite their simplicity.

The Stream-GP evaluations on the multi-bot scenario demonstrates that Stream-GP variants can detect previously seen malicious botnet behaviours. The significance test against the best comparator algorithms, MOA and Apache Spark Streaming, demonstrates that the Stream-GP best algorithm, GP–Hybrid, performs significantly better with at least a 15% difference of AvDR median values for a 5% label budget. GP–Hybrid has shown to be significantly better than the comparator algorithms, with more than 14% for the 5% label budget in the early detection of the botnet indications in the stream. The early detection of botnet behaviour relies on detecting Botnet C&C signals that have a distribution lower than 0.05% over the whole stream. Apache Spark algorithms were completely dismissed in detecting this type of traffic, which is due to the very rare distribution of Botnet C&C and their inability to interact with the stream and detect changes during the learning phase.

Comparison against the state-of-the-art Apache Spark Streaming algorithms as a widely used network streaming tool suggests that there is a need for deploying streaming learning algorithms in network frameworks. The Apache Spark algorithms have been shown to be inapplicable in cases of non-stationary streaming data, as they cannot react to the stream changes. There was an overall 15% difference between the best Stream-GP algorithm, GP–Hybrid, and the best Apache Spark algorithm, DT–Classic. Unlike Apache Spark algorithms, Stream-GP algorithms do not need tuning, as they can adjust to the stream content automatically. This capability ensures that the settings are adapted based on the dynamic environment throughout the stream. Stream-GP has been shown to predict a record's label faster than Apache Spark algorithms.

The focus of the Stream-GP application in this thesis is on the network security field, but, it could be used for other streaming purposes as well. One sample of its utilization is for network analytic purposes. The results suggest that it could be applied to reveal more insights into the unknown traffic. Another application is for detecting insider threats in networks where data are gathered based on user behaviours and not network traffic [78].

6.2 Future Works

The following list represents the possible directions for future research:

1. Stream-GP could be extended to work in an exemplar-based mode (online mode). In this case, the non-overlapping window is omitted, and the decision is made upon each exemplar in the stream. All other things remain the same, and only the window limitation goes away.

2. The GP specific algorithm (SBB) used in this case could be replaced by its improved version, the Tangled Program Graph (TPG) [67]. The TPG approach uses a hierarchical SBB that has no limitation on the depth of the tree. This algorithm is an open-ended evolutionary approach.

3. It could be applied on other (network) streaming applications. This framework is benchmarked on the specific network (security) application, which inherits the most challenging streaming constraints. It is not limited to this field, however, and either could be used in other areas of network operation or other streaming applications.

4. More insights could be gained from the Stream-GP's programs in each generation to follow the effect of each class on specific pieces of code. This way, the definition of the class based on arithmetic calculations could be gained. So, the changes to the properties of the classes throughout the course of the stream in a non-stationary environment could be tracked in a clear way.

5. It could be implemented as a library on top of the strong existing streaming frameworks. By providing a library based on the Stream-GP, other people could apply it to their own area of interest.

Appendix A

Flow Features

A.1 ISOT Flow Features

List of the ISOT flow features extracted by the Tranalyzer flow exporter.

Dir, Duration, ETHVlanID, L4Proto, macPairs, numPktsSnt, numPktsRcvd, numBytesSnt, numBytesRcvd, minPktSz, maxPktSz, avePktSize, stdPktSize, pktps, bytps, pktAsm, ipMindIPID, ipMaxdIPID, ipMinTTL, ipMaxTTL, ipOptCnt, tcpPSeqCnt, tcpSeqSntBytes, tcpSeqFaultCnt, tcpPAckCnt, tcpFlwLssAckRcvdBytes, tcpAckFaultCnt, tcpInitWinSz, tcpAveWinSz, tcpMinWinSz, tcpMaxWinSz, tcpWinSzDwnCnt, tcpWinSzUpCnt, tcpOptPktCnt, tcpOptCnt, tcpMSS, tcpWS, tcpTmS, tcpTmER, tcpEcI, tcpBtm, tcpSSASAATrip, tcpRTTAckTripMin, tcpRTTAckTripMax, tcpRTTAckTripAve, tcpRTTAckTripJitAve, tcpRTTSseqAA, icmpTCcnt, icmpEchoSuccRatio, icmpPFindex, connSip, connDip, connSipDip, tCnt, MinPl, MaxPl, MeanPl, LowQuartilePl, MedianPl, UppQuartilePl, IqdPl, ModePl, RangePl, StdPl, RobStdPl, SkewPl, ExcPl, MinIat, MaxIat, MeanIat, LowQuartileIat, MedianIat, UppQuartileIat, IqdIat, ModeIat, RangeIat, StdIat, RobStdIat, SkewIat, Label

A.2 NSL-KDD Packet Features

A list of NSL-KDD packet features is provided in this section.

duration, protocol_type, service, flag, src_bytes, dst_bytes, land, wrong_fragment, urgent, hot, num_failed_logins, logged_in, num_compromised, root_shell, su_attempted, num_root, num_file_creations, num_shells, num_access_files, num_outbound_cmds, is_host_login, is_guest_login, count, srv_count, serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate, dst_host_srv_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate, label

Bibliography

[1] Http-botnets: the dark side of an standard protocol. `http://securityaffairs.co/wordpress/13747/cyber-crime/http-botnets-the-dark-side-of-an-standard-protocol.html`.

[2] Mohammad Alauthaman, Nauman Aslam, Li Zhang, Rafe Alasem, and M. A. Hossain. A p2p botnet detection scheme based on decision tree and adaptive multilayer neural networks. *Neural Computing and Applications*, 29(11):991–1004, Jun 2018.

[3] G. Apruzzese and M. Colajanni. Evading botnet detectors based on flows and random forest with adversarial samples. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8, Nov 2018.

[4] A. Atwater, M. I. Heywood, and A. N. Zincir-Heywood. GP under streaming data constraints: A case for Pareto archiving? In *ACM Genetic and Evolutionary Computation Conference*, pages 703–710, 2012.

[5] Arian Bär, Pedro Casas, Alessandro D'Alconzo, Pierdomenico Fiadino, Lukasz Golab, Marco Mellia, and Erich Schikuta. Dbstream: A holistic approach to large-scale network traffic monitoring and analysis. *Computer Networks*, 107:5–19, 2016.

[6] Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. D. Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, Nov 2010.

[7] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, ASIACCS '06, pages 16–25, New York, NY, USA, 2006. ACM.

[8] M. Behdad and T. French. Online learning classifiers in dynamic environments with incomplete feedback. In *IEEE Congress on Evolutionary Computation*, pages 1786–1793, 2013.

[9] R. K. Behera, S. Das, M. Jena, S. K. Rath, and B. Sahoo. A comparative study of distributed tools for analyzing streaming data. In *2017 International Conference on Information Technology (ICIT)*, pages 79–84, Dec 2017.

[10] Mustapha Belouch, Salah El Hadaj, and Mohamed Idhammad. Performance evaluation of intrusion detection based on machine learning using apache spark. *Procedia Computer Science*, 127:1 – 6, 2018. PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON INTELLIGENT COMPUTING IN DATA SCIENCES, ICDS2017.

[11] A. Bifet, S. Maniu, J. Qian, G. Tian, C. He, and W. Fan. Streamdm: Advanced data mining in spark streaming. In *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, pages 1608–1611, Nov 2015.

[12] A. Bifet and G. D. F. Morales. Big data stream learning with samoa. In *2014 IEEE International Conference on Data Mining Workshop*, pages 1199–1202, Dec 2014.

[13] A. Bifet, I. Žliobaitė, B. Pfahringer, and G. Holmes. Pitfalls in benchmarking data stream classification and how to avoid them. In *Machine Learning and Knowledge Discovery in Databases*, volume 8188 of *LNCS*, pages 465–479, 2013.

[14] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *J. Mach. Learn. Res.*, 11:1601–1604, August 2010.

[15] Mohamed-Rafik Bouguelia, Yolande Belaïd, and Abdel Belaïd. An adaptive streaming active learning strategy based on instance weighting. *Pattern Recognition Letters*, 70:38 – 44, 2016.

[16] M. Brameier and W. Banzhof. *Linear Genetic Programming.* Springer, 2007.

[17] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. classification and regression trees. *CRC Press*, 1984.

[18] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[19] Dariusz Brzezinski and Jerzy Stefanowski. Prequential auc for classifier evaluation and drift detection in evolving data streams. In Annalisa Appice, Michelangelo Ceci, Corrado Loglisci, Giuseppe Manco, Elio Masciari, and Zbigniew W. Ras, editors, *New Frontiers in Mining Complex Patterns*, pages 87–101. Springer International Publishing, 2015.

[20] S. Burschka and B. Dupasquier. Tranalyzer: Versatile high performance network traffic analyser. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, Dec 2016.

[21] P. Casas, F. Soro, J. Vanerio, G. Settanni, and A. D'Alconzo. Network security and anomaly detection with big-dama, a big data analytics framework. In *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*, pages 1–7, Sep. 2017.

[22] M. Cermak, M. Laštovička, and T. Jirsik. Real-time pattern detection in ip flow data using apache spark. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 521–526, April 2019.

[23] A. Cervantes, P. Isasi, C. Gagné, and M. Parizeau. Learning from non-stationary data using a growing network of prototypes. In *IEEE Congress on Evolutionary Computation*, pages 2634–2641, 2013.

[24] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[25] Z. Chen, H. Zhang, W. G. Hatcher, J. Nguyen, and W. Yu. A streaming-based network monitoring and threat detection system. In *2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 31–37, June 2016.

[26] L. Chi, B. Li, X. Zhu, S. Pan, and L. Chen. Hashing for adaptive real-time graph stream classification with concept drifts. *IEEE Transactions on Cybernetics*, 48(5):1591–1604, May 2018.

[27] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. One trillion edges: Graph processing at facebook-scale. *Proc. VLDB Endow.*, 8(12):1804–1815, August 2015.

[28] A. Dainotti, A. King, K. Claffy, F. Papale, and A. Pescapé. Analysis of a "/0" stealth scan from a botnet. *IEEE/ACM Transactions on Networking*, 23(2):341–354, April 2015.

[29] H. H. Dam, C. Lokan, and H. A. Abbass. Evolutionary online data mining: An investigation in a dynamic environment. In *Studies in Computational Intelligence*, volume 51, chapter 7, pages 153–178. Springer, 2007.

[30] L. De Carli, R. Torres, G. Modelo-Howard, A. Tongaonkar, and S. Jha. Botnet protocol inference in the presence of encrypted traffic. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, May 2017.

[31] S. J. Delany, P. Cunningham, A. Tsymbal, and L. Coyle. A case-based technique for tracking concept drift in spam filtering. *Know.-Based Syst.*, 18(4-5):187–195, August 2005.

[32] Ian Dempsey, Michael O'Neill, and Anthony Brabazon. *Foundations in Grammatical Evolution for Dynamic Environments.* Springer Publishing Company, Incorporated, 1st edition, 2009.

[33] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7(1):1–30, 2006.

[34] K. S. Desale, C. N. Kumathekar, and A. P. Chavan. Efficient intrusion detection system using stream data mining classification technique. In *2015 International Conference on Computing Communication Control and Automation*, pages 469–473, Feb 2015.

[35] C. J. Dietrich, C. Rossow, F. C. Freiling, H. Bos, M. v. Steen, and N. Pohlmann. On botnets that use dns for command and control. In *2011 Seventh European Conference on Computer Network Defense*, pages 9–16, Sept 2011.

[36] G. Ditzler and R. Polikar. Incremental learning of concept drift from streaming imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 25(10):2283–2301, Oct 2013.

[37] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar. Learning in non-stationary environments: A survey. *IEEE Computational Intelligence*, 10(4):12–25, 2015.

[38] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 71–80. ACM, 2000.

[39] R. Dubin, A. Dvir, O. Pele, and O. Hadar. I know what you saw last minute—encrypted http adaptive video streaming title classification. *IEEE Transactions on Information Forensics and Security*, 12(12):3039–3049, Dec 2017.

[40] R. Dubin, O. Hadar, I. Richman, O. Trabelsi, A. Dvir, and O. Pele. Video quality representation classification of safari encrypted dash streams. In *2016 Digital Media Industry Academic Forum (DMIAF)*, pages 213–216, July 2016.

[41] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2Nd Edition)*. Wiley-Interscience, New York, NY, USA, 2000.

[42] K. B. Dyer, R. Capo, and R. Polikar. Compose: A semisupervised learning framework for initially labeled nonstationary streaming data. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):12–26, Jan 2014.

[43] R. Elwell and R. Polikar. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, 22(10):1517–1531, 2011.

[44] G. Fedynyshyn, M. C. Chuah, and G. Tan. Detection and classification of different botnet c&c channels. In *Autonomic and Trusted Computing*, pages 228–242. Springer Berlin Heidelberg, 2011.

[45] G. Folino and G. Papuzzo. Handling different categories of concept drifts in data streams using distributed GP. In *European Conference on Genetic Programming*, volume 6021 of *LNCS*, pages 74–85, 2010.

[46] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. Mawilab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 8:1–8:12, New York, NY, USA, 2010. ACM.

[47] A. Frank and A. Asuncion. UCI machine learning repository [Online]. Available at `http://archive.ics.uci.edu/ml`, 2010.

[48] J. Gama. A survey on learning from data streams: current and future trends. *Progress in AI*, 1(1):45–55, 2012.

[49] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. An empirical comparison of botnet detection methods. *Computers & Security*, 45:100–123, 2014.

[50] Shree Garg, Sateesh Kumar Peddoju, and Anil K. Sarje. Scalable P2P bot detection system based on network data stream. *Peer-to-Peer Networking and Applications*, 9(6):1209–1225, 2016.

[51] C. Gautam, A. Tiwari, S. Suresh, and K. Ahuja. Adaptive online learning with regularized kernel for one-class classification. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pages 1–16, 2019.

[52] A. Ghazikhani, R. Monsefi, and H. S. Yazdi. Recursive least square perceptron model for non-stationary and imbalanced data stream classification. *Evolving Systems*, 4(2):119–131, 2013.

[53] Julian B. Grizzard, Vikram Sharma, Chris Nunnery, Brent ByungHoon Kang, and David Dagon. Peer-to-peer botnets: Overview and case study. In *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07. USENIX Association, 2007.

[54] G. P. Gupta and M. Kulariya. A framework for fast and efficient cyber security network intrusion detection using apache spark. *Procedia Computer Science*, 93:824 – 831, 2016. Proceedings of the 6th International Conference on Advances in Computing and Communications.

[55] H. L. Hammer, A. Yazidi, and B. J. Oommen. On using novel "anti-bayesian" techniques for the classification of dynamical data streams. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1173–1182, June 2017.

[56] M. B. Harries, C. Sammut, and K. Horn. Extracting hidden context. *Machine Learning*, 32(2):101–126, 1998.

[57] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., 2001.

[58] J. L. Hennessy and D. A. Patterson. *Computer Architecture a quantitive approach*. Morgan Kaufmann, 2nd edition, 1996.

[59] M. I. Heywood. Evolutionary model building under streaming data for classification tasks: opportunities and challenges. *Genetic Programming and Evolvable Machines*, 16(3):283–326, 2015.

[60] M. I. Heywood and P. Lichodzijewski. Symbiogensis as a mechanism for building complex adaptive systems: a review. In *European Conference on Genetic Programming*, volume 6024 of *LNCS*, pages 51–60, 2010.

[61] Mohammad Javad Hosseini, Ameneh Gholipour, and Hamid Beigy. An ensemble of cluster-based classifiers for semi-supervised classification of non-stationary data streams. *Knowledge and Information Systems*, 46(3):567–597, Mar 2016.

[62] IBM, Paul Zikopoulos, and Chris Eaton. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill Osborne Media, 1st edition, 2011.

[63] E. Ikonomovska, J. Gama, and S. Džeroski. Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery*, 23(1):128–168, 2011.

[64] N. Japkowicz and M. Shah. *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, 2011.

[65] Sachini Jayasekara, Shanika Karunasekera, and Aaron Harwood. Enhancing the scalability and performance of iterative graph algorithms on apache storm. *2018 IEEE International Conference on Big Data (Big Data)*, pages 3863–3872, 2018.

[66] M. Kampouridis and E. Tsang. EDDIE for investment opportunities forecasting: Extending the search space of the GP. In *IEEE Congress on Evolutionary Computation*, pages 2019–2026, 2010.

[67] Stephen Kelly and Malcolm I. Heywood. Emergent solutions to high-dimensional multitask reinforcement learning. *Evolutionary Computation*, 26(3), 2018.

[68] S. Khanchi, M.I. Heywood, and N. Zincir-Heywood. On the impact of class imbalance in GP streaming classification with label budgets. In *European Conference on Genetic Programming*, volume 9594 of *LNCS*, pages 35–50, 2016.

[69] S. Khanchi, M.I. Heywood, and N. Zincir-Heywood. Properties of a GP active learning framework for streaming data with class imbalance. In *ACM Genetic and Evolutionary Computation Conference*, pages 945–952, 2017.

[70] Sara Khanchi, Ali Vahdat, Malcolm I. Heywood, and A. Nur Zincir-Heywood. On botnet detection with genetic programming under streaming data label budgets and class imbalance. *Swarm and Evolutionary Computation*, 39:123–140, 2018.

[71] Sara Khanchi, A. Nur Zincir-Heywood, and Malcolm I. Heywood. Streaming botnet traffic analysis using bio-inspired active learning. In *2018 IEEE/IFIP Network Operations and Management Symposium, NOMS 2018, Taipei, Taiwan, April 23-27, 2018*, pages 1–6, 2018.

[72] Sara Khanchi, A. Nur Zincir-Heywood, and Malcolm I. Heywood. Network analytics for streaming traffic analysis. In *IFIP/IEEE International Symposium on Integrated Network Management, IM 2019, Washington, DC, USA, April 09-11, 2019.*, pages 25–30, 2019.

[73] H. Kim, S. Madhvanath, and T. Sun. Hybrid active learning for non-stationary streaming data with asynchronous labeling. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 287–292, Oct 2015.

[74] Nicolas Kourtellis, Gianmarco De Francisci Morales, and Albert Bifet. *Large-Scale Learning from Data Streams with Apache SAMOA*, pages 177–207. Springer International Publishing, Cham, 2019.

[75] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.

[76] Bartosz Krawczyk, Leandro L. Minku, João Gama, Jerzy Stefanowski, and Michał Woźniak. Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132 – 156, 2017.

[77] Paweł Ksieniewicz, Michał Woźniak, Bogusław Cyganek, Andrzej Kasprzak, and Krzysztof Walkowiak. Data stream classification using active learned neural networks. *Neurocomputing*, 353:74 – 82, 2019. Recent Advancements in Hybrid Artificial Intelligence Systems.

[78] Duc C. Le, Sara Khanchi, A. Nur Zincir-Heywood, and Malcolm I. Heywood. Benchmarking evolutionary computation approaches to insider threat detection. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 1286–1293, New York, NY, USA, 2018. ACM.

[79] P. Lichodzijewski and M. I. Heywood. Managing team-based problem solving with Symbiotic Bid-based Genetic Programming. In *ACM Genetic and Evolutionary Computation Conference*, pages 363–370, 2008.

[80] P. Lichodzijewski and M. I. Heywood. Symbiosis, complexification and simplicity under GP. In *ACM Genetic and Evolutionary Computation Conference*, pages 853–860, 2010.

[81] P. Lindstrom, B. MacNamee, and S. J. Delany. Drift detection using uncertainty distribution divergence. *Evolving Systems*, 4(1):13–25, 2013.

[82] A. Loginov, M. I. Heywood, and G. Wilson. Benchmarking a coevolutionary streaming classifier under the individual household electric power consumption dataset. In *IEEE-INNS Joint Conference on Neural Networks*, pages 1–8, 2016.

[83] Matthew V. Mahoney and Philip K. Chan. An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection. In *Recent Advances in Intrusion Detection*, pages 220–237, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[84] M. A. Manzoor and Y. Morgan. Real-time support vector machine based network intrusion detection system using apache storm. In *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 1–5, Oct 2016.

[85] M.M Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham. *A Multi-partition Multi-chunk Ensemble Technique to Classify Concept-Drifting Data Streams*, pages 363–375. Springer Berlin Heidelberg, 2009.

[86] Mohammad M. Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham. Classification and novel class detection in data streams with active mining. In Mohammed J. Zaki, Jeffrey Xu Yu, B. Ravindran, and Vikram Pudi, editors, *Advances in Knowledge Discovery and Data Mining*, pages 311–324, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[87] V. Metsis, I. Androutsopoulos, and G. Paliouras. Spam filtering with naive bayes, which naive bayes? In *Third Conference on Email and Anti-Spam (CEAS)*, pages 125–134, 2006.

[88] L. L. Minku, A. P. White, and X. Yao. The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering*, 22(5):730–742, May 2010.

[89] B. Mirza, Z. Lin, and K.-A. Toh. Weighted online sequential extreme learning machine for class imbalance learning. *Neural Process Letters*, 38(3):465–486, 2013.

[90] J. Morgan, A. N. Zincir-Heywood, and J. T. Jacobs. *A Benchmarking Study on Stream Network Traffic Analysis Using Active Learning*, pages 249–273. Springer International Publishing, 2016.

[91] G. Mylavarapu, J. Thomas, and A. K. TK. Real-time hybrid intrusion detection system using apache storm. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pages 1436–1441, Aug 2015.

[92] R. Nossenson and S. Polacheck. On-line flows classification of video streaming applications. In *2015 IEEE 14th International Symposium on Network Computing and Applications*, pages 251–258, Sep. 2015.

[93] R. Polikar, L. Upda, S. S. Upda, and V. Honavar. Learn++: an incremental learning algorithm for supervised neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 31(4):497–508, Nov 2001.

[94] P. M. Pondkule and B. Padmavathi. Botshark — detection and prevention of peer-to-peer botnets by tracking conversation using cart. In *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, volume 1, pages 291–295, April 2017.

[95] M. A. M. Raja and S. Swamynathan. Ensemble learning for network data stream classification using similarity and online genetic algorithm classifiers. In *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1601–1607, Sep. 2016.

[96] S. Ren, Y. Lian, and X. Zou. Incremental naïve bayesian learning algorithm based on classification contribution degree. *Journal of Computers*, 9(8):1967–1974, 2014.

[97] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. v. Steen. Prudent practices for designing malware experiments: Status quo and outlook. In *2012 IEEE Symposium on Security and Privacy*, pages 65–79, May 2012.

[98] J. C. Schlimmer and R. H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986.

[99] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, 31(3):357 – 374, 2012.

[100] W. N. Street and Y. Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 377–382, 2001.

[101] M. Sugiyama and M. Kawanabe. *Machine Learning in non-stationary environments.* MIT Press, 2012.

[102] Y. Sun, K. Tang, L. L. Minku, S. Wang, and X. Yao. Online ensemble learning of data streams with gradually evolved classes. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1532–1545, June 2016.

[103] Géza Szabó, Dániel Orincsay, Szabolcs Malomsoky, and István Szabó. On the validation of traffic classification algorithms. In Mark Claypool and Steve Uhlig, editors, *Passive and Active Network Measurement*, pages 72–81. Springer Berlin Heidelberg, 2008.

[104] H. Tajalizadeh and R. Boostani. A novel stream clustering framework for spam detection in twitter. *IEEE Transactions on Computational Social Systems*, 6(3):525–534, June 2019.

[105] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6, July 2009.

[106] I. Žliobaitė, A. Bifet, B. Pfahringer, and G. Holmes. Active learning with drifting streaming data. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):27–54, 2014.

[107] A. Vahdat, J. Morgan, A. R. McIntyre, M. I. Heywood, and A. N. Zincir-Heywood. Evolving GP classifiers for streaming data tasks with concept change and label budgets: A benchmarking study. In *Handbook of Genetic Programming Applications*, chapter 18, pages 451–480. Springer, 2015.

[108] Ali Vahdat, Aaron Atwater, Andrew R. McIntyre, and Malcolm I. Heywood. On the application of gp to streaming data classification tasks with label budgets. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO Comp '14, pages 1287–1294. ACM, 2014.

[109] K. Vimalkumar and N. Radhika. A big data framework for intrusion detection in smart grids using apache spark. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 198–204, Sep. 2017.

[110] G. Vormayr, T. Zseby, and J. Fabini. Botnet communication patterns. *IEEE Communications Surveys Tutorials*, 19(4):2768–2796, 2017.

[111] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 226–235. ACM, 2003.

[112] S. Wang, L. L. Minku, and X. Yao. Resampling based ensemble methods for online class imbalance learning. *IEEE Transactions on Knowledge and Data Engineering*, 27(5):1356–1368, 2015.

[113] Z. Wang, J. Yang, H. Zhang, C. Li, S. Zhang, and H. Wang. Towards online anomaly detection by combining multiple detection methods and storm. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 804–807, April 2016.

[114] M. Wielgosz, M. Pietroń, and K. Wiatr. Using spatial pooler of hierarchical temporal memory for object classification in noisy video streams. In *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 271–274, Sep. 2016.

[115] Wikipedia contributors. Botnet — Wikipedia, the free encyclopedia, 2019.

[116] Michał Woźniak, Paweł Ksieniewicz, Bogusław Cyganek, Andrzej Kasprzak, and Krzysztof Walkowiak. Active learning classification of drifted streaming data. *Procedia Computer Science*, 80:1724 – 1733, 2016. International Conference on

Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA.

[117] M. Wozniak. Accuracy based weighted aging ensemble (ab-wae) — algorithm for data stream classification. In *2017 IEEE 4th International Conference on Soft Computing Machine Intelligence (ISCMI)*, pages 21–24, Nov 2017.

[118] Reynold S. Xin, Joseph E. Gonzalez, Michael J. Franklin, and Ion Stoica. Graphx: A resilient distributed graph system on spark. In *First International Workshop on Graph Data Management Experiences and Systems*, GRADES '13, pages 2:1–2:6. ACM, 2013.

[119] W. Yang, X. Yin, and G. Xia. Learning high-level features for satellite image classification with limited labeled samples. *IEEE Transactions on Geoscience and Remote Sensing*, 53(8):4472–4482, Aug 2015.

[120] G. Zhang and S. Li. Research on differentially private bayesian classification algorithm for data streams. In *2019 IEEE 4th International Conference on Big Data Analytics (ICBDA)*, pages 14–20, March 2019.

[121] Harry Zhang. The optimality of naive bayes. In *In FLAIRS2004 conference*, 2004.

[122] L. Zhang, S. Yu, D. Wu, and P. Watters. A survey on latest botnet attack and defense. In *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 53–60, Nov 2011.

[123] David Zhao, Issa Traore, Bassam Sayed, Wei Lu, Sherif Saad, Ali Ghorbani, and Dan Garant. Botnet detection based on traffic behavior analysis and flow intervals. *Computers & Security*, 39:2 – 16, 2013. 27th IFIP International Information Security Conference.

[124] X. Zhu, P. Zhang, X. Lin, and Y. Shi. Active learning from stream data using optimal weight classifier ensemble. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 40(6):1607–1621, 2010.

[125] Z. Zhu, G. Lu, Y. Chen, Z. J. Fu, P. Roberts, and K. Han. Botnet research survey. In *2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 967–972, July 2008.