STREAMING NETWORK TRAFFIC ANALYSIS USING ACTIVE LEARNING

by

Jillian Morgan

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

at

Dalhousie University
Halifax, Nova Scotia
March 2015

Table of Contents

List of Tables

List of Figures

Abstract

The aim of this thesis is to evaluate the performance of different budgeting strategies, as well as an Adaptive Neural Network, in analyzing streaming network traffic, specifically, for the purpose of detecting malicious/botnet activity. In previous works, researchers have generally measured the classification performance by the overall accuracy of their strategy. However, this method of analyzing performance is not necessarily the most effective. Thus, in addition to accuracy, performance is measured by analyzing detection rate, prequential accuracy, and prequential detection rate. Measuring the detection rate of a strategy provides a performance metric that is not biased in terms of class distribution. The prequential accuracy and prequential detection rates offer additional performance analysis in that these performance metrics present the changes of accuracy and detection rate throughout the network stream.

In a real life scenario network traffic is unending and constantly being streamed, resulting in large datasets that require a large number of resources to train a classifier on. Thus, budgeting strategies that select a small portion of data instances on which to train on have been developed. In this thesis, five budgeting strategies are evaluated; Random, Fixed Uncertainty, Variable Uncertainty, Random Variable Uncertainty, and Select Sampling. Performance of the budgeting strategies is measured at budgets of 10% and 100%. The aforementioned strategies are tested in conjunction with two different classifiers; Naive Bayes and Hoeffding Tree. In addition to the budgeting strategies, an adaptive Neural Network Strategy is also evaluated. The proposed strategies are applied to six different streaming network traffic datasets that include different malicious or botnet activity.

The results demonstrate that all of the budgeting strategies (with the exception of the fixed uncertainty strategy) are suitable candidates for classification of streaming network traffic where some of the state-of-the-art classifiers achieved accuracies in the range of 90% or higher. Furthermore, limiting labeling budgets to 10% does not affect performance negatively, thus its use is recommended as to save computing resources.

List of Abbreviations Used

ACC Accuracy.

ACK Acknowledgement Flag.

ANN Artificial Neural Network.

API Application Programming Interface.

C&C Command and Control.

CTU Czech Technical University.

DDoS Distributed Denial of Service.

DR Detection Rate.

FIN End Flag.

HT Hoeffding Tree.

HTTP HyperText Transfer Protocol.

IP Internet Protocol.

IRC Internet Relay Chat.

ISOT Information Security and Object Technology.

MOA Massive Online Analysis.

NB Naive Bayes.

NIMS Network Management Information and Security.

P2P Peer-to-Peer.

PSH Push.

QBC Query-by-Committee.

REJ Rejection.

ROC Receiver Operator Characteristic.

RST Reset.

SVDD Support Vector Domain Description.

SVM Support Vector Machine.

SYN Synchronize.

TCP Transmission Control Protocol.

UDP User Datagram Protocol.

URG Urgent Flag.

Acknowledgements

This research is supported by Raytheon SAS. The research is conducted as part of the Dalhousie NIMS Lab at https://projects.cs.dal.ca/projectx/

Chapter 1

Introduction

In recent years, malicious network activity, such as viruses, denial of service attacks, man in the middle attacks, and botnets, has become a growing concern for businesses and the general public alike. Botnets, in particular, are a serious threat as they provide a platform for malicious activities to be performed in high volumes in a distributed fashion. Botnets are made up of a collection of infected hosts, called bots. These bots are controlled by a botmaster using Command and Control (C&C) channels that allow for the botmaster to remain anonymous and have the bots perform tasks automated by the botmaster [17]. Computers can be infected by a botnet by a number of different means. These methods include but are not limited to: viruses, software and web browser vulnerabilities, social engineering and Trojan horses.

A computer infected by a botnet can now be controlled by the botmaster in order to perform a number of malicious activities. Botnets can be used to collect personal information from infected machines, such as passwords and credit card information, by way of keyloggers (i.e. malicious software that records the keystrokes performed on a keyboard). The collected information is sent to the botmaster for the purpose of financial gain, identity theft, or even blackmailing. Botnets can also be used as computing resources for the botmaster. On the less threatening end of the spectrum, botmasters may use these resources to sell unique views to web pages or videos, or ad clicks which a consumer may purchase in order to garner ad revenue. These resources can also be used to generate vast amounts of spam via email. These emails can be used as phishing scams that attempt to gather personal information from the recipient, or to harvest email addresses to later sell to groups such as marketing firms. These emails can be used to spread malware and even the botnet itself to other machines.

On the more threatening end of the spectrum, the botmaster may use the host resources to perform Distributed Denial of Service (DDoS) attacks. A DDoS attack is performed with the intention to interrupt the services of a host or hosts on the Internet. A botmaster may choose to attack a particular host in order to hinder competition, political reasons (disagree with an organization's views), blackmail an organization, or even just for their own amusement. A botnet usually performs a DDoS by issuing commands to its bots to attempt to overwhelm the target system by forcing the system to utilize all its resources (processing power, memory, network connections, bandwidth, etc.). Botnets are able to consume these resources by way of a number of different methods: ping Floods, User Datagram Protocol (UDP) Floods, Smurf Attacks, Transmission Control Protocol (TCP) Synchronize (SYN) Attacks, Tear Drop Attacks, and Land Attacks.

Unfortunately, most hosts that are infected with a botnet are not aware. Detection of malicious network activity, such as botnets, as it occurs is important since it can assist the network management teams in preventing further damage on their systems and networks, as well as the systems and networks of others. Therefore, it is of interest to aim to classify network activity as it is being streamed. Many researchers propose the use of different learning techniques in order to accurately analyze and classify the network traffic in streaming environments [32][31] in order to detect botnet activity [27][30][26].

Even when using a machine learning algorithm, classification within a streaming environment poses many challenges. One of the main challenges is that one cannot have access to all available data in a streaming scenario at once, as in the case of non-streaming environments such as offline streaming scenarios. Thus, classification of streaming data can be quite costly in terms of resources as datasets can grow to be quite large. Furthermore, because a complete set of data cannot be viewed at once it is difficult to determine if given data instance attributes (features) are representative of others in the same class. Additionally, network traffic patterns can slowly change over time or even immediately. Indeed, such problems are also exhibited in other streamed datasets. To this end, active learning has often been implemented to alleviate these issues [34][33]. Active learning is the task of selecting a data instance on which to query the true classification label and retrain the learning algorithm. Selection of labels is not a simple task as one must consider how many and which labels will represent the entire dataset and allow for the most accurate prediction of future data instances.

Many researchers in the literature [30][34][33][21] determine the success of classification by measuring the overall accuracy (the number of successful classifications over the total amount of classifications made) of the chosen classification algorithm and active learning strategy. This is not necessarily the best solution in determining classification success as it does not account for class distribution. If a dataset has an unbalanced distribution of classes then the resulting prediction accuracy may not be representative of the actual performance of the given classification strategy. Thus, a performance metric that factors class distribution into account is necessary.

In this research, I aim to benchmark the performance of previously existing active learning and query budgeting strategies as well as an Adaptive Artificial Neural Network approach when performed on network traffic flows, specifically in order to detect malicious network activity, such as botnets. In evaluating the performance of these strategies, I include two performance measures; (i) prequential accuracy and (ii) prequential detection rate. My new contribution is analyzing and classifying malicious behaviour among network streaming traffic using various budgeting and active learning strategies while using prequential accuracy and prequential detection rate as a performance metric.

The rest of the thesis is organized as the following: Chapter 2 discusses the related work in this field. The methodology employed in this research is detailed in Chapter 3. Evaluations are presented and the results are given in Chapter 4. Finally, the conclusions are drawn and the future research directions are discussed in Chapter 5.

Chapter 2

Literature Review

There are many works on the detection of malicious activity within network traffic, classification of streaming data, and active learning, as separate topics. However, to the best of my knowledge there are no works that combine all of these to evaluate and analyze their performances on the detection of malicious behaviours on network traffic. It should be noted that, some techniques have been proposed that utilize a subset of these ideas. The works most relevant to the study are described in detail below. In this case, the related works are discussed under three categories; active learning, streaming data classification, and detection of malicious activity among network traffic.

2.1 Active Learning

S. Liu et al. [24] propose a method of Peer-to-Peer (P2P) traffic identification that utilizes active learning in conjunction with a Support Vector Machine (SVM). In this work, the authors use a partially labelled dataset for their training set. Instances are first filtered using a technique called Support Vector Domain Description (SVDD) that filters the dataset to include the instances that are most useful to learning. Once the dataset has been filtered the active learning portion of the authors methodology is implemented. The active learning strategy proposed in this work chooses instances on which to train on by maintaining two principles: (i) the entirety of the data selected to train on represents a balanced dataset, and (ii) query the label with the highest certainty. The instance that had been queried for its true label is then used to train the SVM. Thus the entire proposed process for classification of peer-to-peer traffic is as follows. First the SVDD is invoked. All labelled instances are then used to train the SVM. Next the active learning strategy is used with a portion of the unlabelled data and produces the instance on which to train the SVM. The active learning strategy is then invoked on the other subsets of the dataset until all data has been processed. The authors use a benchmarking dataset that includes ten different types of network traffic. Performance was measured by accuracy. The researchers concluded that overall their strategy achieved higher performance accuracy than when using a similar strategy without balancing or a similar strategy without using SVDD for filtering instances.

R. Wang et al. [32] focus on a strategy utilizing computational intelligence in the form of fuzzy rough sets combined with a SVM. This strategy is performed on both binary and multi-class benchmarking datasets where the performance is evaluated by measuring accuracy, time costs for labeling new examples from the data, and paired Wilcoxon rank-sum tests. The results of this strategy are compared to other existing strategies such as Random Sampling, SVM Active, and Query-by-Committee (QBC). The researchers conclude that this new strategy is generally successful when compared to other tested strategies in terms of accuracy and the paired Wilcoxin rank-sum tests, however, this comes at a higher time cost as labeling instances takes longer when using this strategy.

Zliobaite et al. [34], also endeavour to produce new active learning strategies. However, the active learning strategies they designed are created with the intention of being used on drifting streaming data. These methods focus on retraining a learning algorithm when confidence of successful prediction of an entity falls below a set threshold and randomly selecting entities. The selection of entities to train on was limited by a set budget for querying new labels on which to train the model. The strategies that were developed were tested on a series of publicly available big data sets, which were categorized as either being a prediction or textual dataset. Prediction datasets required a prediction from the classifier and textual datasets required a recommendation from the classifier. Performance of the active learning strategies was evaluated by applying these strategies to the Naive Bayes algorithm and the Hoeffding Tree algorithm. Accuracy of these techniques using different datasets and labeling budgets (10 and 100 percent) was measured. The researchers concluded that the strategies are effective for reducing computation costs while maintaining performance.

Like the previous researchers, Zhu et al. [33] proposed another active learning strategy for the implementation of streamed datasets. The strategy they designed features a weighted classifier ensemble framework with an emphasis on reducing variance. The researchers reported that by decreasing the classifier ensemble variance, the error rate of the classifier ensemble would decrease as well. Thus, a minimum-variance principle was introduced whereas labels were queried for instances that produced a high ensemble variance. The combination of a weighted classifier ensemble and a minimum variance principle were employed over three publicly available prediction datasets. Performance of this strategy was evaluated by determining the accuracy and runtime when using this strategy on various data chunks and data chunk sizes. These results were then compared to the results of simpler solutions. The researchers concluded that their strategy was effective at dealing with multi-class problems in a streaming environment.

2.2 Streaming Data Classification

Dalal et al. [15] demonstrated various data mining prediction techniques used to predict user-perceived streamed media quality. The researchers proposed the use of a nearest neighbour algorithm, in which unlabelled instances in a stream are given the label (i.e. the normalized user-perceived quality rating) of the instance in the training set of the instance within the closest distance. The researchers tested this algorithm using two different types of distance metrics; summary statistics and dynamic time warping. The algorithm was employed over three different data streams; a commercial, a movie trailer, and a news segment. The authors measured performance using hit rates, that is, the percentage of predictions within 0.8 standard deviations of the normalized user quality rating for the given stream. The researchers concluded that the chosen techniques performed effectively. They expanded upon their work further [16] by performing similar tests in real time using TCP-based streams rather than offline using UDP-based streams. They conclude that the performance accuracy was hopeful (falling between 75 to 87 percent accuracy) but could be improved further.

Moreover, Cunha et al. [14] evaluated the performance of Naive Bayes and C4.5 Decision Trees algorithms in classifying different failure states when streaming video data. Specifically, the researchers wanted to be able to predict whether a server failure was a performance anomaly or was caused by overloading produced by clients.

Performance was measured by; True Positive Rate, False Positive Rate, Precision, Recall, F-measure, Receiver Operator Characteristic (ROC) Area, and Root Mean Squared Error. The researchers concluded that both algorithms were adequate but C4.5 performed slightly better than Naive Bayes.

F. Stahl et al. [29] take an evolving rule based approach, named eRules, in order to make classifications on a streaming dataset in a data mining scenario. Interestingly, with this approach the authors do not desire to make classifications on instances that a set of rules does not currently exist for. This means that at the end of the classification process, not all instances will be classified. The researchers argue this is beneficial in some scenarios where an incorrect classification is more than just a nuisance. The authors name medical applications as such a scenario. The eRules technique involves the use of *if x then y* type rule set where the class of an instance is determined by what rule the instance falls into based on its attributes. The eRules approach first takes a subset of incoming data instances and trains the classifier to create an initial set of rules. Once the first set of rules has been generated instances are streamed through the eRules algorithm one by one and are processed as follows. If an instance is not covered by any rules then it is added to a buffer of unclassified instances and remains unclassified. Once the buffer is full a new set of rules is generated based on these unclassified instances and the buffer is cleared. Instances that will fall under existing rules will be classified, with rules being updated to reflect whether or not the classification made was correct. Finally, if the number of instances that are unclassified becomes too high then the classifier will be retrained on a subset of the incoming data. The proposed eRules technique was tested on three software generated data streams. Performance of the eRules approach was measured by the total accuracy, the percentage of instances that were not classified, and the percentage of correct classified instances. The researchers conclude that the classification performance performed well in most cases but could very well be improved by changing parameter settings. It should be noted that in most cases the number of classifications that were not made are fairly high (over 50% in most cases), thus this is not a useful technique if one aims to classify most instances.

Vahdat et al. [31] designed and developed a framework for employing genetic programming in order to perform classification on streamed data while maintaining a labeling budget. They employed artificially generated, as well as publicly available, datasets. They measured the performance not only by the aforementioned performance metrics but also by prequential accuracy. They conclude that genetic programming with labeling budgets is an effective method for making classifications on streaming data.

2.3 Detection of Malicious Behaviour Among Network Traffic

The use of flow-based network traffic in detecting malicious activity among network traffic appears to be quite popular within existing literature. A network traffic flow is a sequence of network traffic packets with 5-tuple information over a specific period defined by the Internet Engineering Task Force [13]. This 5-tuple information includes; the source/destination IP addresses, source/destination port numbers and the protocol. In Stevanovic et al's [30] study, network traffic was converted into network flows in which to be classified. To evaluate the validity of using such a technique to detect malicious activity, a number of classifiers were tested; Naive Bayes, Bayesian Network, Logistic Regression, Artificial Neural Networks, Support Vector Machines with a linear kernel, C4.5 decision tree, Random Tree, and Random Forest. The proposed technique was implemented on a combination of datasets featuring traffic from Storm and Waledec botnets and normal traffic. In order to measure performance, they employed precision, recall, F-measure, and a correlation coefficient. Additionally, they measured the training and classification time when using each classifier. The researchers concluded that C4.5, Random Tree and Random Forest were the most successful algorithms for their task.

Similarly, Nogueira et al. [26] proposed the use of a flow based system in order to detect botnet activity among network traffic. They employed a Neural Network model in conjunction with a flow-based system. However, the employed system also features a user interface to visualize illicit activity that was detected for further action by an administrator. In identifying botnet activity a feed-forward propagation neural network with three layers was implemented. Performance was evaluated by testing the framework on traffic generated by known safe applications such as Skype. Malicious activity was artificially generated. The authors concluded that the detection of the botnet activity using their methodology was quite successful.

Hsiao et al. [21] also proposed the use of flow-based network traffic for the purpose of detecting malicious behaviour amongst said traffic. Flows were generated from network flows collected by the researchers. What differentiates this study from others is that the authors varied the number of flow attributes and which flow attributes were presented between experiments. Thus, they created four sets of attributes to be tested; NetFlow variables, Temporal Variables, Spatial Variables, and a combination of Temporal and Spatial variables. In these experiments, the classification algorithms chosen to employ on the flows were as follows; Naive Bayes, Decision Tree and SVM algorithms. The results showed that using a combination of temporal and spatial attributes provided the best prediction accuracy.

On the other hand, Saad et al. [27] implement a slightly different approach to detecting malicious botnet behavior than the aforementioned studies that share the same goal. They used not only flow based attributes but also used host-based attributes (i.e. attributes that are exhibited in communications between hosts). They employed the following classification algorithms: Nearest Neighbor, Linear SVM, Artificial Neural Network, Gaussian Based Classifier, and Naive Bayes. With these methods, the researchers aimed to meet three botnet detection requirements; adaptability, novelty detection, and early detection. The authors used a combination of three datasets for their experiments. The first two datasets were generated by a botnet infected machine where all packets incoming and outgoing were captured. Each machine was infected with a different botnet; Storm or Walodec. The third dataset was made up of normal traffic. These datasets were then combined into one larger dataset in order to simulate a real world scenario. To determine the performance of the selected methodology the researchers adopted four performance metrics: training time, classification time, accuracy, and classification error. The researchers concluded that the selected methodology did not sufficiently satisfy the three stated requirements for effectively detecting botnets.

P. Narang et al. [25] argue that it is of more use to limit the amount of features (attributes) used when making classifications on flow based network traffic. The researchers contend that some of the attributes in a flow are redundant or irrelevant, thus these attributes should be removed. The researchers state that by removing these features that training time for the model will be reduced as well as reduce over fitting. Three methods of selecting which features will be removed are proposed and are listed as follows: Correlation-based feature selection, Consistency based subset evaluation and Principal component analysis. The effectiveness of the chosen methods was determined by utilizing these methods with three different machine learning techniques: Decision tree, Naive Bayes, and Bayesian Network Classifier. The strategies were tested on a dataset of network traffic containing botnet activity that was generated by the researchers. Performance of the chosen methods was measured by classification accuracy, and the rate at which botnet behavior is correctly detected. The number of features extracted by each method was also noted. The researchers conclude that using smaller amounts of features gave similar results when compared to the results when using all features of the dataset. However, the researchers note that the training models using the smaller feature sets were much faster to build, thus feature selection is beneficial.

In my previous work [19], we proposed a new framework to detect HyperText Transfer Protocol (HTTP) based botnet activity based on botnet behavior analysis. To achieve this, we employed machine learning algorithms on flow-based network traffic utilizing NetFlow (via Softflowd). The proposed botnet analysis system was implemented by employing two different learning algorithms, namely C4.5 and Naive Bayes. For this work we had used False Positive and True Positive rates, complexity (i.e. computation time, the number of attributes that are used, and tree size when using the C4.5 algorithm) as my performance metrics. My results showed that the C4.5 learning algorithm based classifier obtained very promising performance on detecting HTTP based botnet activity. However, that work did not employ any streaming or budgeting strategies.

On the other hand, in this thesis, I aim to apply and benchmark existing active learning strategies on network streamed traffic in order to make classification predictions for malicious network behavior. This approach differs from the aforementioned related work as the previous work has not combined active learning strategies with the streaming data classification on network traffic. In this thesis, I employ such an approach, specifically to detect botnet activities under a streaming scenario. I also aim to compare the performance of these strategies with an adaptive Artificial Neural Network Approach and determine which is more effective in performing the desired task.

Last but not least, I also introduce the use of performance metrics; prequential accuracy and prequential detection rate. Prequential detection rate has not been used to measure performance under Massive Online Analysis scenarios. Prequential detection rate is a useful metric when unbalanced distributions of classes are present in a given dataset, because unlike accuracy, prequential detection rate can reflect the difference of correctly classifying data instances of the smaller classes in the data. To give an example, if a data set has 99% of class-normal and 1% class-malicious, by classifying everything as class-normal, a classifier can reach 100% accuracy! Even though false positive rates may show the picture a bit clearer, in streaming environments this kind of metric can be ineffective and cannot measure the performance correctly. This is important to consider as looking at accuracy alone can skew how one perceives the performance of the algorithms on an unbalanced dataset. I also analyze the prequential values for both accuracy and detection rate as this allows for one to see how these values change over time.

Chapter 3

Methodology

In this research, the goal is to utilize the algorithms as discussed in [34], and determine the success of these algorithms when classifying streamed network traffic data for detecting malicious botnet behavior. In order to achieve this, I enacted three major steps: data collection, implementation of learning algorithms in conjunction with various active learning budgeting strategies, and performance analysis.

3.1 Datasets and Features Employed

In this research, six datasets are employed in the evaluations. These are:

1. **KDD Cup 1999:** The KDD Cup 1999 dataset is a simple to classify dataset that contains malicious and normal network traffic flows, where each instance to be classified is a connection record [6]. The malicious traffic is broken into four types; denial of service, unauthorized remote access, unauthorized access to root commands, and probing. For my purposes, I only aimed to detect whether or not a connection is malicious, thus, I combined the four attack types into one class. Even though, it is an old dataset, it is chosen to provide a baseline and reference point for the other results. This dataset is suitable for this purpose as it is one of the first datasets that was made publicly available for benchmarking computational intelligence techniques for network security purposes.

2. **NIMS1:** The NIMS1 dataset can be retrieved from the Network Management Information and Security (NIMS)[9]. The dataset is a collection of network traffic flows. Unlike the other datasets, where I aim to detect malicious behavior among network traffic, with this dataset, I aim to classify application type. Thus, this dataset is chosen to provide a comparison of results when aiming to classify different applications on streaming network traffic as opposed to making classifications between normal and malicious network behaviours.

3. **ISOT:** The Information Security and Object Technology (ISOT) dataset is a collection of publicly available malicious and normal datasets [5]. These traffic datasets were generated by using a series of machines with different MAC addresses and IP addresses. The traffic generated was captured by the open source packet capturing tool, Wireshark[1], in order to combine the smaller datasets into the larger ISOT dataset. Thus, I employed this dataset to predict whether or not a connection was malicious.

4. **Zeus vs. Alexa:** This dataset is generated by the NIMS Lab to be used for botnet detection purposes in 2014. To this end, I generated a traffic dataset that exhibited an approximate balance of malicious and normal network traffic. In order to generate this dataset, lists of valid malicious and non-malicious domain names were obtained [19]. I obtained the list of non-malicious (normal) domain names from Alexa, a website that ranks the top 500 websites on the Internet according to page views [2]. Because the domains listed are some of the most popular domains on the Internet, it is fair to assume that the traffic generated by accessing these domains is representative of normal, everyday, network traffic. For the malicious domains, I obtained a list of domain names that are known to belong to the Zeus botnet [1][3]. To simulate web traffic to these domains, a script was written to randomly connect to either a normal or malicious domain using the wget command in Linux. These steps are detailed in Fig.1.

5. **CTU-2 Dataset:** This dataset is one of thirteen different datasets within the CTU-13 dataset generated by Czech Technical University (CTU) in Czech Republic in 2011 [18][10]. The dataset is referred to as Scenario 2 or CTU-Malware-Capture-Botnet-43 in the literature. However, for ease of differentiation between the other datasets used in this research, it will be referred to as CTU-2 Dataset. The malicious traffic in this scenario was generated by a computer infected with the Neris botnet. This dataset, in particular, is a collection of flows that include normal, background, and malicious traffic. However for the purposes of this thesis research, normal and background traffic were combined into one class as the aim was to detect malicious traffic among the remaining

[1]Wireshark: https://www.wireshark.org/

traffic.

6. **CTU-11 Dataset:** This dataset is another one of thirteen different datasets within the CTU-13 dataset generated by CTU in Czech Republic in 2011 [18][10]. The dataset is referred to as Scenario 11 or CTU-Malware-Capture-Botnet-52 in the literature. However, for ease of differentiation between the other datasets used in this research, it will be referred to as CTU-11 Dataset. The malicious traffic in this scenario was generated by a number of computers infected with the Rbot botnet. Rbot is an older botnet that uses Internet Relay Chat (IRC) to perform DDoS or spam attacks [4]. What is most worrisome about this botnet is that its code has been made easily available to the public [11]. The flows in this dataset consisted of three classes but were combined into two classes as is described for CTU-2 Dataset.

Unlike the other datasets, ISOT and Zeus vs. Alexa datasets had not been preprocessed into flows when retrieved from their sources. Thus traffic from these datasets was replayed on a test bed network to emulate a real life scenario of streaming traffic. The streaming traffic is then converted to flows as the traffic runs and then, the network flows are used as input for the streaming classifiers. Usually, in a real life scenario, the router (such as a Cisco router with NetFlow) will do this on the flow. To emulate such a scenario, the following open source tools were employed to convert the packets into flows:

- **Softflowd:** Softflowd[2] is an open source tool that accepts network packets and exports them into NetFlow[3] flows.
- **Nfcapd:** Nfcapd[4] captures the exported flows and stores them for further processing. The flow data that Nfcapd records are not in a human readable format, thus, further processing is required.
- **Nfdump:** NfDump[5] takes the recorded flow data and converts it into a human readable format.

[2]http://www.mindrot.org/projects/softflowd/

[3]Netflow:http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html

[4]Nfcapd: http://nfdump.sourceforge.net/

[5]NfDump: http://nfdump.sourceforge.net/

Table 3.1: Summary of the datasets

Dataset	Instances	Attributes	Class Distribution	Total Number of Classes
KDD Cup 1999	494021	41	Normal: 97278 Malicious: 396743	2
ISOT	2084216	15	Normal: 2020948 Malicious: 63268	2
NIMS	713851	22	Telnet: 1251 FTP: 4140 HTTP: 11904 DNS: 38016 Lime: 646271 Local Forwarding: 2557 Remote Forwarding: 2422 SCP: 2444 SFTP: 2412 X11: 2355 Shell:2491	11
Zeus vs. Alexa	11468	15	Normal: 7877 Malicious: 3591	2
CTU-2	6351188	7	Normal: 6296755 Malcious: 54433	2
CTU-11	408835	7	Normal: 130943 Malcious: 277892	2

The streaming datasets were converted into flows by different sources where different types of attributes are chosen to represent a flow, thus the number of attributes for each dataset varies. A summary of the datasets is described in Table 3.1 while a more detailed description of the attributes and the classes used for each dataset is listed in Tables 3.2, 3.3, 3.4 and 3.5. Table 3.6 shows the attributes that are common among all of the datasets and what the attribute name is refereed to in each dataset. It should be noted that for the purposes of this thesis source and destination port numbers and Internet Protocol (IP) addresses have been removed from the datasets.

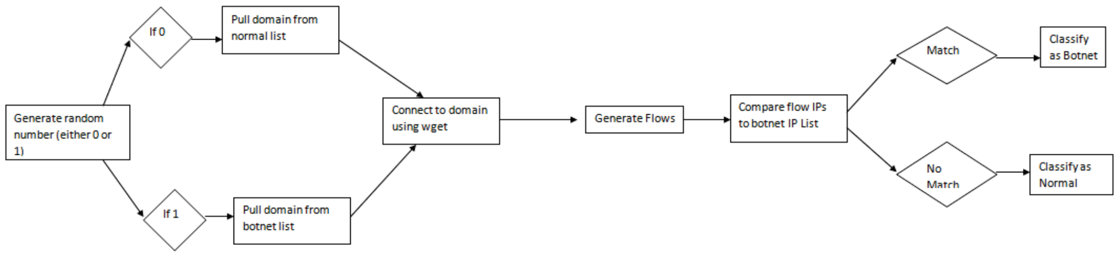
Generate random number (either 0 or 1)
If 0
Pull domain from normal list
If 1
Pull domain from botnet list
Connect to domain using wget
Generate Flows
Compare flow IPs to botnet IP List
Match
Classify as Botnet
No Match
Classify as Normal

Figure 3.1: Script functionality for generating web traffic.

Table 3.2: Dataset Attributes and Descriptions for the KDD Cup 1999 Dataset [6] based on descriptions given in [7] and [23]

Attribute	Description
duration	Duration of the connection.
protocol_type	Type of protocol used in the connection.
service	Type of service used in the connection.
flag	Indicates whether or not the connection has a normal or error status.
src_bytes	Total number of bytes moving from source to destination during the connection.
dst_bytes	Total number of bytes moving from destination to source during the connection.
land	Indicates whether or not the connection indicates a land attack (i.e. the connection destination and source are the same host or port).
wrong_fragment	Total amount of fragments marked as wrong.
urgent	Total number of urgent packets in during a connection.
hot	Total number of hot indicators found during a connection.
num_failed_logins	Total number of failed logins made during a connection.
logged_in	Indicates whether or not the connection was considered to be logged in or not.
num_compromised	Indicates the number of compromised conditions exhibited within the connection.
root_shell	Indicates whether or not the root shell was being used or not during the connection.
su_attempted	Indicates whether or not the su root command was initiated during a connection.
num_root	Total number of times root has been accessed during the connection.
num_file	Total number of files created during the connection.

Continued on next page

Table 3.2 – continued from previous page

Attribute	**Description**
num_shells	Total number of shell prompts exhibited during a connection.
num_access_files	Total number of operations that have been performed on access controlled files during a connection.
num_outbound_cmds	Number of outbound commands exhibited during an ftp connection.
is_host_login	Indicates whether or not the current login information provided belongs to the host or not.
is_guest_login	Indicates whether or not the current login information provided belongs to the guest or not.
count	Total number of connections made to the current host within the last two seconds.
srv_count	Total number of connections made to the current service within the last two seconds.
serror_rate	The percentage of connections that have exhibited synchronization SYN errors.
srv_serror_rate	The percentage of connections that have exhibited SYN errors.
rerror_rate	The percentage of connections that have exhibited SYN errors.
srv_rerror_rate	The percentage of connections that have exhibited Rejection (REJ) errors.
same_srv_rate	The percentage of connections that have exhibited REJ errors.
diff_srv_rate	The percentage of connections made to different services than the current service being used is this connection.
srv_diff_host_rate	The percentage of connections made to different hosts than the current service being used is this connection.
dst_host_count	Total number of connections made that have the same destination host as the current connection.
dst_host_srv_count	Total number of connections made that have the same destination host and are using the same service as the current connection.
dst_host_same_srv_rate	Percentage of connections made that have the same destination host and are using the same service as the current connection.
dst_host_diff_srv_count	Total number of different services running on the current host.
dst_host_same_src_port_rate	The percentage of connections on the current host thatare using the same source port.

Continued on next page

Table 3.2 – continued from previous page

Attribute	Description
dst_host_srv_diff_host_rate	The percentage of connections that are using the same source port on the current host.
dst_host_serror_rate	The percentage of connections that exhibit an S0 error on the current host.
dst_host_srv_serror_rate	The percentage of connections using the current service with the current host.
dst_host_rerror_rate	The percentage of connections to the current host that exhibit a Reset (RST) error.
dst_host_srv_rerror_rate	The percentage of connections to the current host that are using the current service that are exhibiting a RST error.

Table 3.3: Dataset Attributes and Descriptions for the NIMS1 dataset [9].

Attribute	Description
min_fpktl	The size of smallest packet moving forward in a flow.
mean_fpktl	The mean size of all packets moving forward in a flow.
max_fptkl	The size of the largest packet moving forward in the flow.
std_fpktl	The standard deviation from the mean size of the packets moving forward in a flow.
min_bpktl	The size of the smallest packet moving backwards in the flow.
mean_bpktl	The mean size of all packets moving backward in a flow.
max_bpktl	The size of the largest packet moving backward in the flow.
std_bpktl	The standard deviation from the mean size of the packets moving backward in a flow.
min_fiat	The smallest amount of time elapsed between two packets moving forward in a flow.
mean_fiat	The mean amount of time elapsed between two packets moving forward in a flow.
max_fiat	The largest amount of time elapsed between two packets moving forward in a flow.
std_fiat	The standard deviation from the mean of time elapsed between two packets moving forward in a flow.
min_biat	The smallest amount of time elapsed between two packets moving backward in a flow.

Continued on next page

Table 3.3 – continued from previous page

Attribute	Description
mean_biat	The mean amount of time elapsed between two packets moving backward in a flow.
max_biat	The largest amount of time elapsed between two packets moving backward in a flow.
std_biat	The standard deviation from the mean of time elapsed between two packets moving backward in a flow.
duration	Overall duration of a flow.
proto	The protocol used.
total_fpackets	The total number of packets moving forward in a flow.
total_fvolume	The total number of bytes moving forward in a flow.
total_bpackets	The total number of packets moving backwards in a flow.
total_bvolume	The total number of bytes moving backwards in a flow.

Table 3.4: Dataset Attributes and Descriptions for the ISOT and Zeus vs. Alexa datasets.

Attribute	Description
Duration	Duration of a flow.
Protocol	Protocol used in a flow.
FlagsU	Indicates whether or not the Urgent Flag (URG) has been used in a flow.
FlagsA	Indicates whether or not the Acknowledgement Flag (ACK) has been used in a flow.
FlagsP	Indicates whether or not the Push (PSH) flag (PSH) has been used in a flow.
FlagsR	Indicates whether or not the RST flag has been used in a flow.
FlagsS	Indicates whether or not the SYN flag has been used in a flow.
FlagsF	Indicates whether or not the End Flag (FIN) has been used in a flow.
ToS	Type of service used in a flow
Packets	The number of packets in a flow.
Bytes	The number of bytes in a flow.
PPS	The average number of packets per second in a flow.
BPS	The average number of bytes per second in a flow.
BPP	The average number of bytes per packet in a flow.
Flows	The number of flows used (should always be 1).

Table 3.5: Dataset Attributes and Descriptions for the CTU-13 datasets (CTU-2 dataset and CTU-11 dataset).

Attribute	Description
Duration	The total duration of the flow.
Protocol	The protocol used in the flow.
Flags	Indicates the flags that are used during the flow.
ToS	Indicates the type of service used during the flow.
Packets	Indicates the number of packets moving in the flow.
Bytes	Indicates the number of bytes in the flow.
Flows	Indicates the number of flows used (should always be 1).

Table 3.6: Attributes that are common between datasets and their given attribute names.

Attribute Name	KDD 1999 Cup	NIMS1	ISOT	Zeus vs. Alexa	CTU-2 Dataset	CTU-11 Dataset
Duration	duration	duration	Duration	Duration	Duration	Duration
Protocol	protocol_type	proto	Protocol	Protocol	Protocol	Protocol

3.2 Learning Algorithms and Budgeting Strategies

As mentioned previously, the goal of this study is to benchmark the performance of existing active learning strategies on streamed network traffic flows. Thus, in this section the data stream mining tools, the budgeting strategies, and performance metrics I employed in my evaluations are presented.

3.2.1 Massive Online Analysis

Massive Online Analysis (MOA)[6] is an open source tool for data stream mining [8]. It has proved very useful for this study as it is able to simulate a data stream with a provided input. Furthermore, MOA provides users the ability to implement the use of various machine learning algorithms and active learning strategies on the data as it is being streamed. Additionally, MOA includes an Application Programming Interface (API) suite that allows for users to create and modify the functionality of existing code to suit their own evaluation needs.

[6]http://moa.cms.waikato.ac.nz/

3.2.2 Labels, Budgeting, and Active Learning

In a real world streaming network traffic environment, it is assumed that the amount of incoming data is infinite and dynamic. This means that the data attributes and how they relate to one another can change over time either slowly (concept drift) or suddenly (concept shift). Thus it can be assumed that it is of more use to train a classifier on incoming data than to use a preexisting model. In this scenario, a classifier predicts the class of an instance based on the attributes of instances received prior. Once the prediction has been made, the classifier will query the actual class from a human provided label. The classifier will then train on the current instance with the intention of increasing prediction accuracy for future oncoming instances. In a network streaming environment where one aims to classify between normal or malicious behavior this would mean that for every flow that arrived at the network the classifier would have to be provided with its true classification label (i.e. whether the flow was normal or malicious network traffic behavior). As mentioned previously, attempting to perform classification tasks on such a large dataset can be quite costly in terms of human effort (providing true classification labels) time efficiency and hardware required to handle such large datasets. Thus, the concept of budgeting is introduced. Budgeting involves limiting the amount of queries that can be made to retrieve the true classification label of an instance in a data stream [34]. Active learning incorporates this idea of budgeting but adds a learning aspect in which the system makes an educated guess on which classifications labels are most useful to query.

3.2.3 Budgeting and Active Learning Strategies

The budgeting strategies that were chosen for this benchmarking study were chosen based on the study performed in [34]. I chose to implement the same strategies for a multitude of reasons. Firstly, work in [34] focuses on developing strategies for streams with drifts, which is relevant for my study as most network streamed data will exhibit drifts. Secondly, using the same active learning strategies gives an opportunity to compare how the strategies perform on streamed network traffic datasets as compared to the general prediction and textual datasets used within Zliobaite et al.'s study [34]. Because I will be comparing the results with the results of [34], I will also be using budgets of 10 percent and 100 percent. The active learning strategies that were used in the study are described below.

- **Random:** This strategy randomly chooses data instances to query for the true label [34]. No active learning occurs with this simple budgeting strategy, so it provides an effective baseline for the evaluations.
- **Fixed Uncertainty:** Queries the true labels of the data instances with a confidence below a given threshold [34].
- **Variable Uncertainty:** Queries the true labels of the data instances with the lowest confidence within a variable time interval [34].
- **Random Variable Uncertainty:** This is a combination of the Random and Fixed Uncertainty budgeting strategies [34].
- **Select Sampling:** Queries the true labels randomly with a changing probability bias [12].

It is important to note that if a query for a true label is necessary then the training model will be trained on the instance that was queried.

3.2.4 Learning Algorithms

For this study, I selected three different algorithms to accompany the chosen active learning strategies for streaming classification. The algorithms chosen are: (i) Naive Bayes, (ii) Hoeffding Tree, and (iii) Adaptive Artificial Neural Networks.

Naive Bayes

Naive Bayes is a simple probabilistic classifier that is known to perform quite well considering its simplicity [28]. The classifier makes predictions by assuming that all attributes of a given instance do not correlate to each other in the probability of a label being of a given class. Predicting a class using this algorithm is performed by determining which class (C_1, C_2, C_k where k is the total number of classes) has the highest posterior probability based on the input x:

$$P(C_i \mid x) > P(C_j \mid x) for 1 \leq j \leq k, j \neq i \tag{3.1}$$

where:

$$P(C_i \mid x) = \frac{P(C_i \mid x)P(C_i)}{P(x)} \tag{3.2}$$

where $P(C_i|x)$ is the priori probability and $P(C_i)$ is the likelihood.

Hoeffding Tree

The Hoeffding Tree algorithm, also known as a Very Fast Decision Tree, is a more complicated algorithm that incorporates the use of decision trees. It was designed to be used on large data streams where only a subset of the data that passes through is used to find the best split for the tree. The number of samples included in this subset to achieve the desired confidence threshold is determined by a dynamic threshold called a Hoeffding Bound. Hoeglinger et al. [20] describe the Hoeffding Bound as a principle that says with a probability of $1 - \delta$, the true mean of a variable is at least $\bar{r} - \varepsilon$ where ε is the desired error and is described as follows:

$$\varepsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}} \tag{3.3}$$

where l is the current leaf in the decision tree, R is the range of random variables, r. n is the number of independent observations made so far, and $1 - \delta$ is the error probability. The described Hoeffding Bound is then used within a decision tree to determine on which attribute to split. This is done by determining the largest gain between two attributes. If the largest calculated gain is greater than the ε then the Hoeffding Tree algorithm states that this attribute is the best attribute to split on with a probability of $1 - \delta$ [20].

Neural Networks

I also employ a well-known bio-inspired computational intelligence technique in my evaluations in order to systematically benchmark different learning techniques. To this end, I specifically use adaptive Artificial Neural Network (ANN). ANNs are learning algorithms that are designed to imitate real-world biological neural networks.

In this work, I use the Pattern Recognition network with a Multi-layer Perceptron within Matlab's[7] Neural Network Toolbox. In order for the network to work properly with streaming data I implement the use of the *adapt* function within Matlab. The *adapt* function, as I used it in these experiments, allowed for the neural network to adapt as data was being streamed. In other words, instead of training the network on a training set, the neural network would be trained on each data instance as the data (traffic) arrives. This means that a labeling budget of 100 percent is used.

It should be noted that when using the chosen learning algorithms, only the default parameters were utilized within MOA and Matlab.

[7]Matlab: http://www.mathworks.com/products/matlab/

Chapter 4

Evaluation and Results

For the purpose of evaluating the performance of the chosen machine learning algorithms and budgeting strategies on network datasets, two performance metrics are employed: prequential accuracy and prequential detection rate.

Accuracy of a classifier is described as the total number of correct classifications over all the classification predictions made (n), that is:

$$Accuracy = \frac{tp}{tp + fn} \tag{4.1}$$

where tp indicates the number of true positives, and fn indicates false negatives.

Similarly, prequential accuracy is the total number of correct classifications over the total number of classifications made at a given point in time, that is;

$$preqACC_t = \frac{(t-1)preqACC_{t-1} + C_t}{t} \tag{4.2}$$

where t indicates a given time point, $t-1$ indicates the previous time point, and C indicates whether or not the classification at the given time point was successful ($C = 1$ if the classification was correct, or $C = 0$ if the classification was incorrect).

Although accuracy is used to measure the performance in some works [32][27][34] [33][21][15][16], its use could be problematic. With the use of unbalanced datasets, where the number of instances belonging to each class is significantly different, using accuracy as a measure of classification performance can be misleading. For example, if a classifier is presented with a dataset that consists of 98% normal activity and 2% malicious activity and the classification model predicts that all activity is normal then achieve 98% prediction accuracy is achieved. However, this result does not indicate successful classification, as no malicious activity was detected. Therefore, I want to use a performance metric that accounts for class imbalance in addition to false positive rates. Thus, the use of prequential detection rate is introduced as a performance metric for these experiments.

In this research, detection rate is calculated using Eq. 4.3 and Eq. 4.4:

$$DR = \frac{1}{Q}\sum_{q=1}^{Q} DR_q \tag{4.3}$$

where:

$$DR_q = \frac{tp_q}{tp_q + fn_q} \tag{4.4}$$

where Q is the number of classes, q denotes a particular class, tp indicates true positives, and fn indicates false negatives. Similarly, to find the detection rate at any given time, Prequential detection rate is calculated below, where t denotes the given point in time.

$$preqDR_t = \frac{(t-1)preqDR_{t-1} + DR_t}{t} \tag{4.5}$$

where:

$$DR(t) = \frac{1}{Q}\sum_{q=1}^{Q} DR_q(t) \tag{4.6}$$

where:

$$DR_q(t) = \frac{tp_q(t)}{tp_q(t) + fn_q(t)} \tag{4.7}$$

4.1 Results of Using Learning Algorithms Together with Budgeting Strategies

When first employing the described methodology, default parameters were employed on the budgeting strategies within MOA. However, this resulted in the Fixed Uncertainty strategy achieving a near 0% performance in terms of both accuracy and detection rate. With the Fixed Uncertainty strategy, a query for the true label of an instance is only made when the confidence falls below a given threshold. If the confidence never falls below the threshold then active learning stops and the classifier is not retrained. Due to this aforementioned problem with the Fixed Uncertainty threshold some parameter tuning was necessary. Thus the confidence threshold was raised from the default 90% to 95%.

The computation cost of the Naive Bayes and Hoeffding Tree algorithms was quite small in terms of processing with the Naive Bayes algorithm taking only seconds to complete and the Hoeffding Tree algorithm taking approximately a minute to process the largest dataset.

Table 4.1, 4.2, 4.3, 4.4, 4.5 and 4.6 show the overall results for each labeling strategy when using different budgets with different classification algorithms (Naive Bayes and Hoeffding Tree) over the four chosen datasets employed in this research. Classifications predictions made on the KDD 1999 Cup dataset (Table 6) generally appear to perform the same with a high level of performance (approximately 99%) regardless of the budget or the learning algorithm chosen. However, an exception is observed with the Hoeffding Tree Algorithm using the Fixed Uncertainty Strategy where fewer correct classifications are made (approximate accuracy of 80%). Furthermore, the detection rate makes a dramatic drop from the other detection rates and accuracies presented here. This is believed to be due to the differences between the malicious and non-malicious behaviours being easily separable as discussed in [22].

Table 4.1: Overall prediction ACC and DR and of different budgeting strategies using Kdd 1999 Cup dataset. The algorithms NB and HT are represented.

	Performance Metric	**Random**	**Fixed Uncertainty**	**Variable Uncertainty**	**Random Variable Uncertainty**	**Select Sampling**
NB 100%	ACC	99.87	94.21	99.73	99.84	99.81
	DR	99.70	92.64	99.62	99.79	99.75
NB 10%	ACC	99.90	97.05	99.54	99.47	99.47
	DR	99.14	97.12	99.37	99.29	99.38
HT 100%	ACC	99.90	80.46	98.37	99.77	99.86
	DR	99.87	50.38	96.06	99.59	99.81
HT 10%	ACC	99.46	81.88	99.36	99.56	99.54
	DR	99.15	53.99	98.84	99.38	99.38

On the NIMS1 dataset (Table 4.2), all strategies that use Naive Bayes as the classification algorithm perform similarly well (approximately 80-90%) in terms of accuracy. In terms of detection rate, an even higher overall accuracy (approximately 92-95%) is observed among strategies implemented with Naive Bayes. The exception of the Naive Bayes algorithms detection rate success in this scenario can be observed when using the Fixed Uncertainty labeling strategy where a detection rate of only 55% is achieved.

A significant bump in performance in accuracy (approximate overall accuracy of 95%) is observed when using the Hoeffding Tree algorithm on the NIMS1 dataset as compared to the use of the Naive Bayes algorithm. The exception to this is when using the Fixed Uncertainty strategy, where there is a large drop in accuracy performance. Interestingly, with the increase of prediction accuracy a decrease in detection rate performance is observed when using the Hoeffding Tree algorithm. The decrease in detection rate is fairly significant (from approximately 92-95% when using the Naive Bayes algorithm to approximately 70-88% when using the Hoeffding Tree algorithm) but is even more noticeable when using the Fixed Uncertainty strategy where detection rates as low as 8.96% are observed. However, it should be noted that in this scenario an even lower accuracy performance is achieved with only 0.33% of classifications made being correct.

Table 4.2: Overall prediction ACC and DR and of different budgeting strategies using NIMS1 dataset. NB indicates Naive Bayes and HT indicates Hoeffding Tree.

	Performance Metric	**Random**	**Fixed Uncertainty**	**Variable Uncertainty**	**Random Variable Uncertainty**	**Select Sampling**
NB 100%	ACC	88.73	87.20	90.41	90.72	90.47
	DR	96.42	55.00	96.41	96.20	95.63
NB 10%	ACC	82.16	89.73	89.17	91.11	89.31
	DR	92.24	49.17	92.79	91.70	92.85
HT 100%	ACC	96.38	0.33	95.33	95.30	95.54
	DR	88.05	9.08	79.74	80.70	81.96
HT 10%	ACC	93.69	0.33	94.25	94.56	94.50
	DR	71.00	8.96	74.87	75.24	75.92

On the ISOT dataset (Table 4.3), Random, Fixed Uncertainty, Random Variable Uncertainty and Select Sampling perform roughly the same among all budgets and classification algorithms (excluding the fixed uncertainty strategy used in conjunction with the Hoeffding Tree algorithm). However, there are some interesting results when using the Variable Uncertainty strategy on this dataset. Although very poor performance (results $< 60\%$) are observed when using budgets of 100% with variable uncertainty, when the budget is changed to 10%, an increase in performance is achieved, bringing the results on par with the other strategies. This is important to note, as it shows that even though training is performed on less information, training with less information can be noticeably more effective in some cases.

Table 4.3: Overall prediction accuracy and detection rate of different budgeting strategies using the ISOT dataset. NB indicates Naive Bayes and HT indicates Hoeffding Tree.

	Performance Metric	**Random**	**Fixed Uncertainty**	**Variable Uncertainty**	**Random Variable Uncertainty**	**Select Sampling**
NB 100%	ACC	99.99	90.27	19.48	99.99	99.99
	DR	99.94	94.96	58.45	99.96	99.93
NB 10%	ACC	99.98	92.71	89.71	99.99	99.98
	DR	99.84	96.18	99.77	99.88	99.88
HT 100%	ACC	99.99	3.03	12.94	99.99	99.99
	DR	99.99	49.98	55.10	99.99	99.99
HT 10%	ACC	99.99	3.03	99.72	99.99	99.99
	DR	99.94	49.99	99.79	99.96	99.96

Results for the experiments performed on the Alexa vs. Zeus dataset are available in Table 4.4. Again, a similar performance is observed among all strategies, budgets, and classification algorithms except for when using the Fixed Uncertainty Strategy with the Hoeffding Tree machine learning algorithm where performance is significantly lower.

On the CTU-2 Dataset: Neris a variety of results are observed over all of the implemented strategies, budgets, and machine learning algorithms. When using the Naive Bayes algorithm the results when using the Random, Random Variable Uncertainty, and Select Sampling are fairly similar (approximately 77% accuracy and approximately 85% detection rate) when using a budget of 100%. However, when using a budget of 10% a bump in accuracy performance (approximately 86%) occurs when using the Random labeling strategy.

Table 4.4: Overall prediction accuracy and detection rate of different budgeting strategies using Alexa vs. Zeus dataset. NB indicates Naive Bayes and HT indicates Hoeffding Tree.

	Performance Metric	**Random**	**Fixed Uncertainty**	**Variable Uncertainty**	**Random Variable Uncertainty**	**Select Sampling**
NB 100%	ACC	98.44	97.62	97.52	98.16	97.50
	DR	99.96	96.21	97.18	97.73	97.14
NB 10%	ACC	95.80	97.50	96.15	95.40	93.85
	DR	95.23	96.01	95.60	95.61	94.47
HT 100%	ACC	98.32	31.29	93.69	97.96	97.95
	DR	97.94	49.97	94.38	97.46	97.66
HT 10%	ACC	96.47	31.21	94.84	95.22	94.34
	DR	95.26	49.84	94.89	95.30	94.51

On the other hand, a drop in detection rate performance occurs among the Random and Select Sampling Strategies. When using the Fixed Uncertainty labeling and Variable Uncertainty strategies with the Naive Bayes algorithm, detection rate is fairly low (<50%). In terms of accuracy, interestingly it is observed that performance accuracy more than doubles when using a budget of 10% rather than a budget of 100%. When implementing the Hoeffding Tree algorithm accuracy performance is very high in all cases (approximately 99%) except when using the Variable Uncertainty strategy with a 100% labeling budget where accuracy is low (44%) compared to the aforementioned accuracy performance results.

On the other hand, there is much more variation in detection rate performance. The best detection rate performance (70%) when using the Hoeffding Tree algorithm occurs when using a labeling budget on 100%, with a Random or Select Sampling labeling strategy. However, the lowest detection rate (24%) when using the Hoeffding Tree algorithm also is observed when using a labeling budget of 100%, although this occurs when utilizing the Variable Uncertainty strategy.

Finally, when viewing the CTU-11 Dataset: Rbot dataset the most consistent results are achieved compared to the previously discussed datasets. The Random, Variable Uncertainty, Random Variable Uncertainty, Select Sampling strategies perform similarly in that accuracy and detection rate is approximately 99% among all aforementioned strategies. However, as is the case with the previously discussed

Table 4.5: Overall prediction accuracy and detection rate of different budgeting strategies using CTU-2 Dataset: Neris. NB indicates Naive Bayes and HT indicates Hoeffding Tree.

	Performance Metric	**Random**	**Fixed Uncertainty**	**Variable Uncertainty**	**Random Variable Uncertainty**	**Select Sampling**
NB 100%	ACC	78.69	96.48	44.49	77.18	76.16
	DR	85.30	48.69	24.21	84.48	85.08
NB 10%	ACC	86.00	99.14	99.14	74.92	79.54
	DR	79.41	50.0	50.00	79.01	85.13
HT 100%	ACC	99.38	99.83	44.46	99.25	99.38
	DR	72.84	49.85	24.21	64.93	70.31
HT 10%	ACC	99.17	99.14	99.14	99.18	99.26
	DR	57.17	50.00	50.0	62.66	62.60

datasets, Fixed Uncertainty results vary from the other strategies. When using the Fixed Uncertainty labeling strategy the same results (32% accuracy and 50% detection rate) are achieved for all machine learning algorithms and budgets used.

Table 4.6: Overall prediction accuracy of different budgeting strategies using CTU-11 Dataset: Rbot. NB indicates Naive Bayes and HT indicates Hoeffding Tree.

	Performance Metric	**Random**	**Fixed Uncertainty**	**Variable Uncertainty**	**Random Variable Uncertainty**	**Select Sampling**
NB 100%	ACC	99.47	32.3	99.47	99.49	99.46
	DR	99.43	50.0	99.41	99.48	99.41
NB 10%	ACC	98.57	32.03	98.59	98.42	97.68
	DR	98.84	50.0	98.69	98.64	98.24
HT 100%	ACC	99.97	32.03	99.76	99.95	99.95
	DR	99.63	50.0	99.68	99.93	99.95
HT 10%	ACC	99.89	32.03	99.75	99.15	99.90
	DR	99.87	50.0	99.65	99.07	99.88

In the cases where the fixed uncertainty strategy produces low rates of correct classifications, it can be assumed that this may be because the confidence of each instance when streaming is never low enough to invoke training. Thus, no learning is performed. This happens with the fixed uncertainty strategy in particular as the confidence threshold is fixed and never changes to adjust to incoming data.

In Figures 4.1 through 4.49, the prequential accuracy and prequential detection rates are presented for each instance (time point) when using machine learning algorithms (Naive Bayes and Hoeffding Tree) in conjunction with the selected budgeting strategies.

When the selected budgeting experiments were performed on the KDD 1999 (Figures 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8) Cup dataset, accuracy and detection rate increase rapidly at the beginning of the stream and then maintains a consistent performance throughout the streaming process. This statement is true for all budgeting strategies used here except for the fixed uncertainty strategy, which exhibits a different pattern. In terms of prequential accuracy, a slower rise is exhibited when using the fixed uncertainty strategy in all cases on this dataset, except when using the Naives Bayes algorithm with a budget of 10%. In this case, the prequential accuracy initially increases rapidly but drops soon after in a concave formation. The prequential detection rate when employing the fixed uncertainty strategy remains around 50% in all cases except when using Naives Bayes with a budget of 10%. In this case, the prequential detection rate initially increases rapidly, but then drops soon after in a concave formation.

When evaluating the employed machine learning algorithms in conjunction with the budgeting strategies on the NIMS1 dataset (Figures 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, 4.15, and 4.16), prequential accuracy tends to rise quickly and then maintains its performance. However, the detection rate appears to increase in performance more slowly, almost in a step pattern. These are consistent in all cases when using this dataset except when the fixed uncertainty strategy is utilized. In that case, the prequential accuracy rises quickly and then drops back down quickly to a very low accuracy for the rest of the stream. When measuring the prequential detection rate, under the fixed uncertainty strategy, it is observed that it either remains low (when using the Hoeffding Tree Algorithm, Figures 10, 11, 12 and 13) or follows the trend of the rest of the strategies but at a lower accuracy (when using the Naive Bayes Algorithm, Figures 14, 15, 16, and 17) throughout the stream. On the other hand, when these strategies are employed on the ISOT dataset (Figures 4.17, 4.18, 4.19, 4.20, 4.21, 4.22, 4.23 and 25), the Random, Random variable Uncertainty, and Select Sampling strategies, the prequential detection rate and accuracy remain at approximately 100% throughout the streaming process. However, with the Fixed Uncertainty strategy and the Variable Uncertainty strategy, interesting behavior presents itself. When using a budget of 10% with the variable uncertainty strategy, the trends exhibited follows the patterns of the Random, Random Variable Uncertainty, and Select Sampling on this dataset. However when using a budget of 100% with either machine learning algorithm, a significant drop in the performance occurs in terms of prequential detection rate and even greater loss in terms of prequential accuracy. When using the fixed uncertainty strategy with the Hoeffding Tree Algorithm, a downwards slope in the prequential accuracy occurs, while the prequential detection rate maintains a steady performance around 50%. When this strategy is used in conjunction with the Naives Bayes Algorithm an interesting pattern is exhibited where the prequential accuracy and the prequential detection rate show a sharp rise and then a slight fall and then a steady climb upwards again.

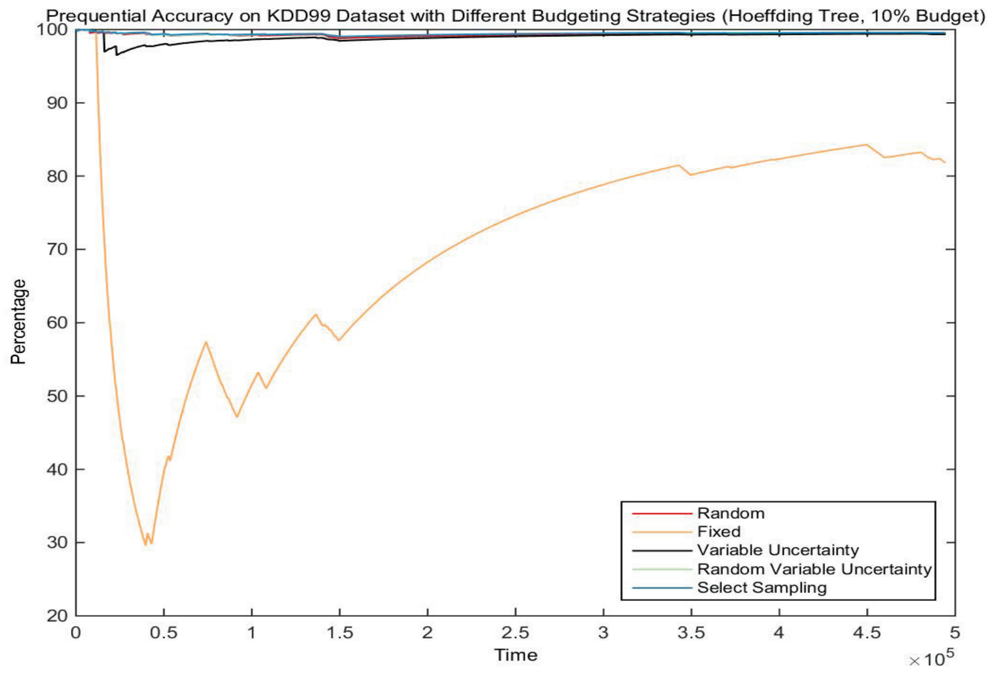

Figure 4.1: Prequential accuracy on KDD 1999 Cup Dataset using the Hoeffding Tree Algorithm with a 10% budget.

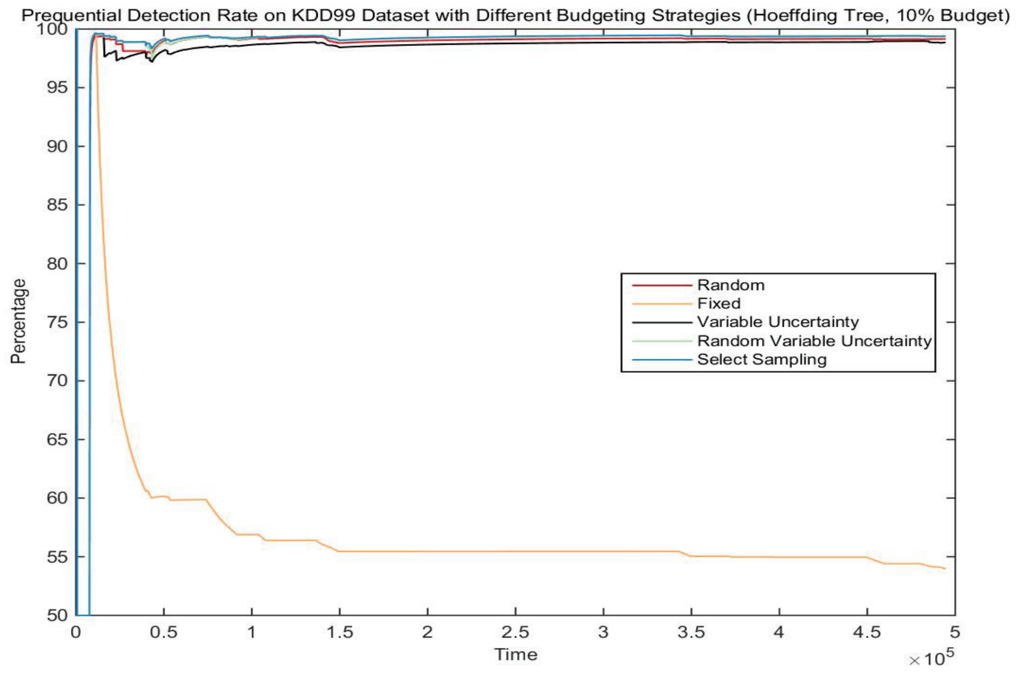

Figure 4.2: Prequential detection rate on KDD 1999 Cup Dataset using the Hoeffding Tree Algorithm with a 10% budget.

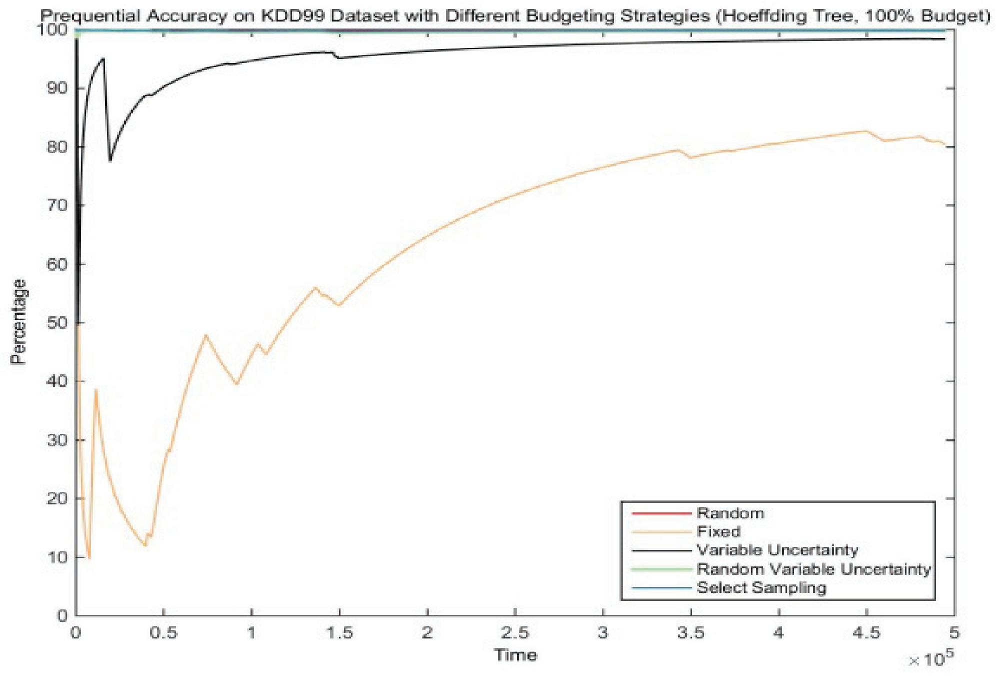

Figure 4.3: Prequential accuracy on KDD 1999 Cup Dataset using the Hoeffding Tree Algorithm with a 100% budget.

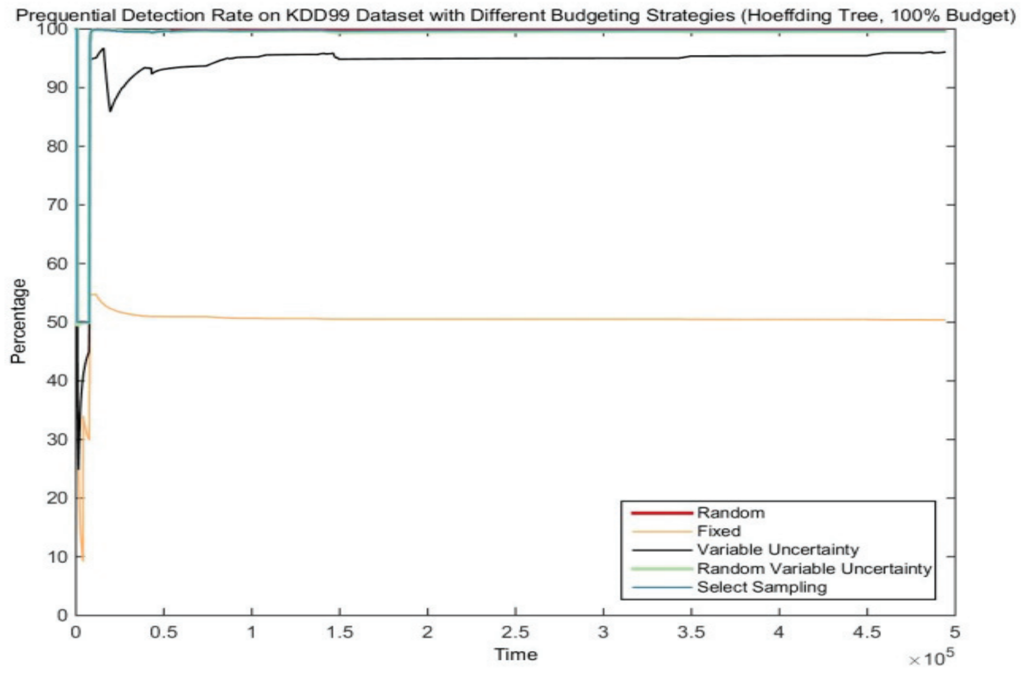

Figure 4.4: Prequential detection rate on KDD 1999 Cup Dataset using the Hoeffding Tree Algorithm with a 100% budget.

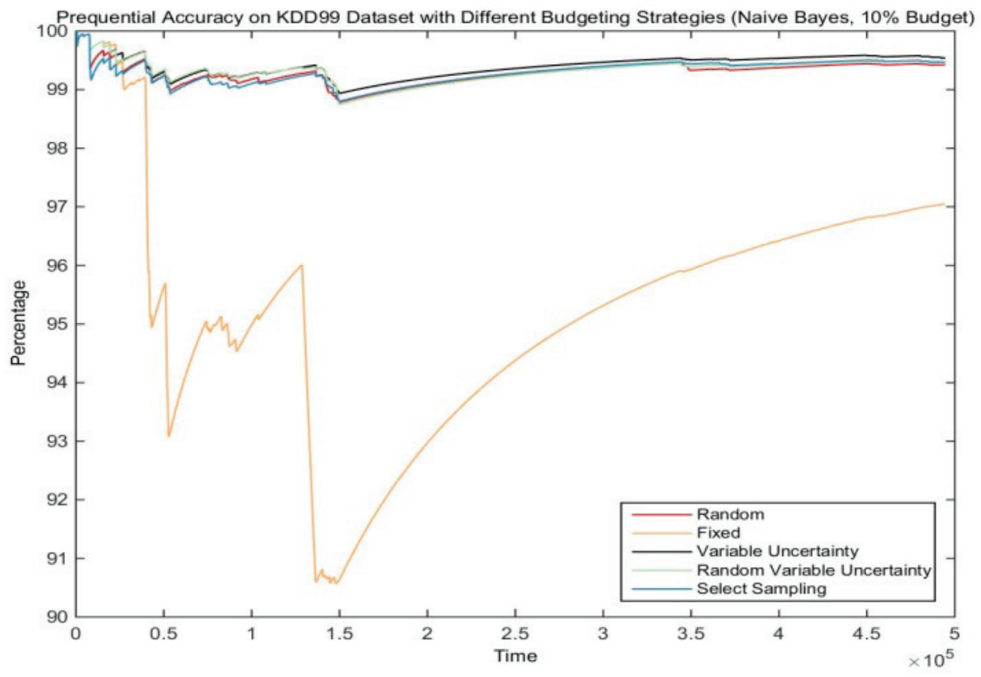

Figure 4.5: Prequential accuracy on KDD 1999 Cup Dataset using the Naive Bayes Algorithm with a 10% budget.

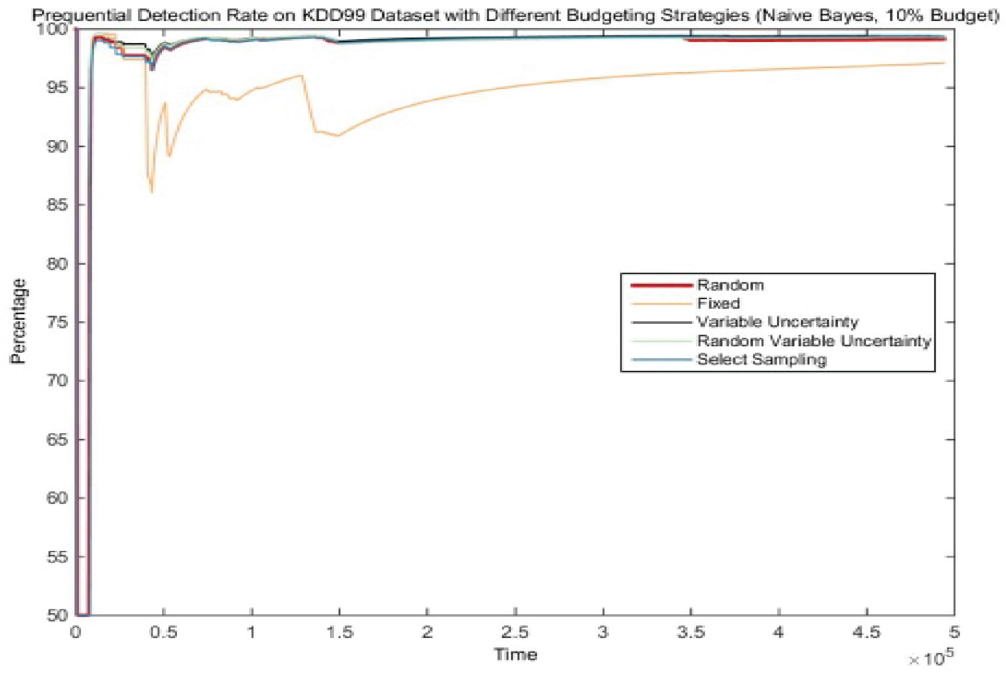

Figure 4.6: Prequential detection rate on KDD 1999 Cup Dataset using the Naive Bayes Algorithm with a 10% budget.

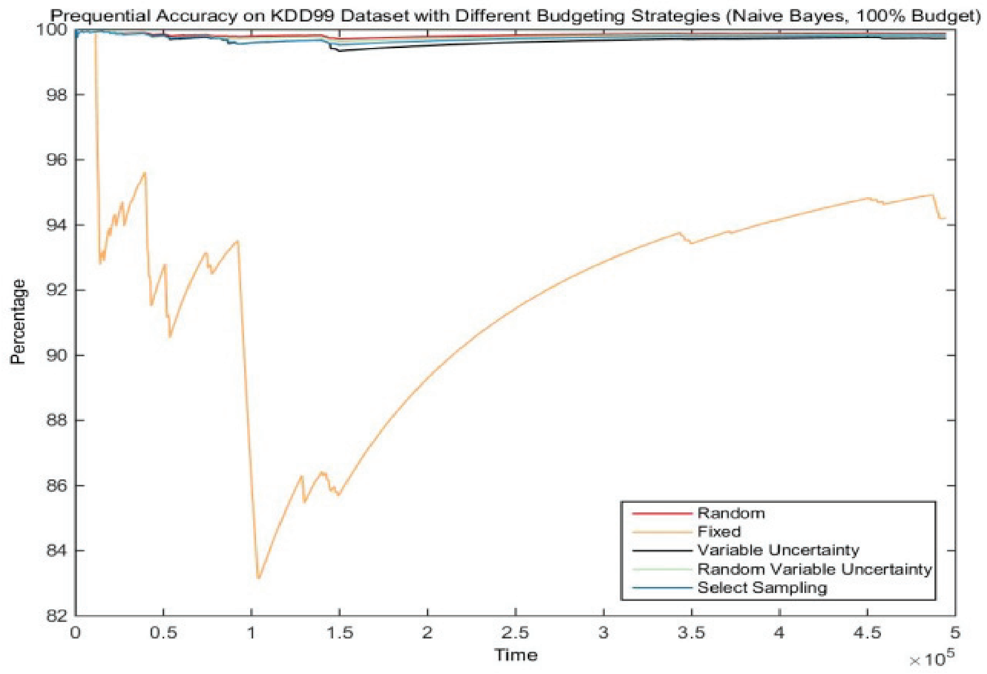

Figure 4.7: Prequential accuracy on KDD 1999 Cup Dataset using the Naive Bayes Algorithm with a 100% budget.

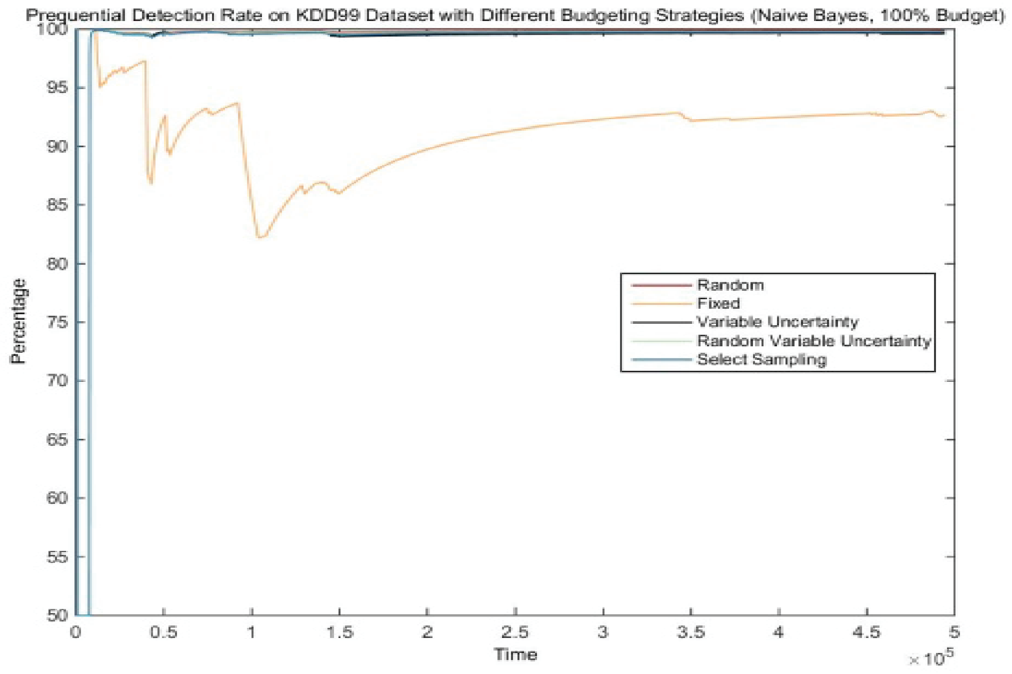

Figure 4.8: Prequential detection rate on KDD 1999 Cup Dataset using the Naive Bayes Algorithm with a 100% budget.

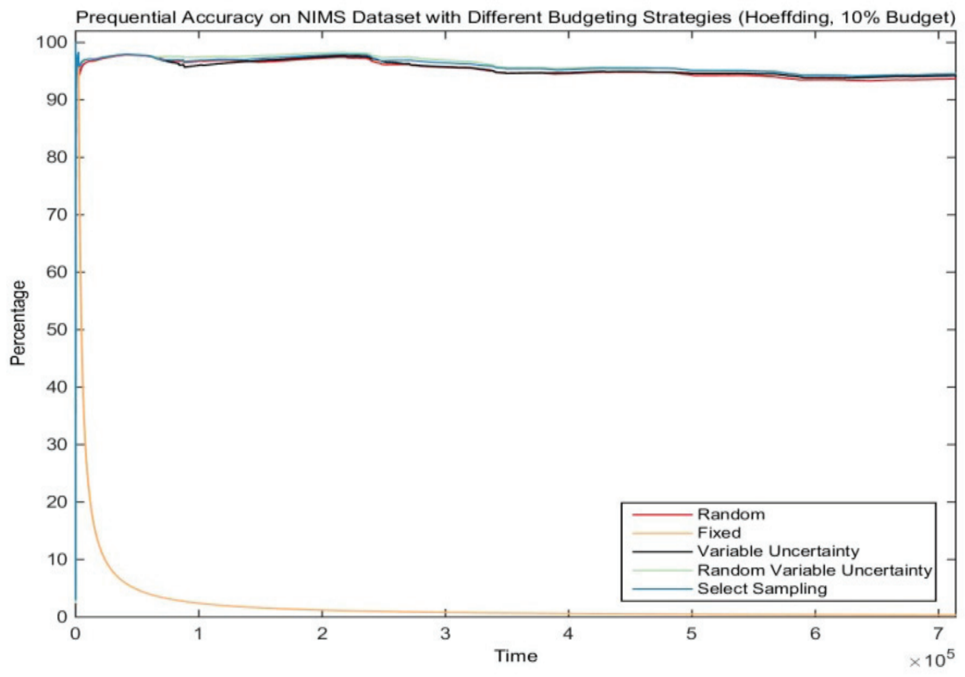

Figure 4.9: Prequential accuracy on NIMS Dataset using the Hoeffding Tree Algorithm with a 10% budget.

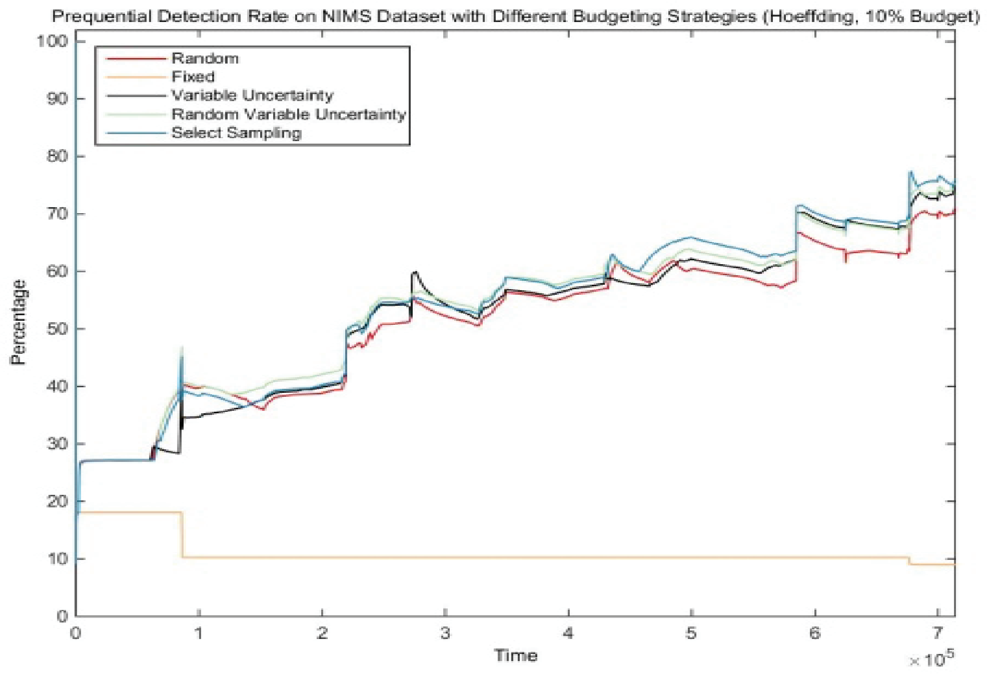

Figure 4.10: Prequential detection rate on NIMS Dataset using the Hoeffding Tree Algorithm with a 10% budget.

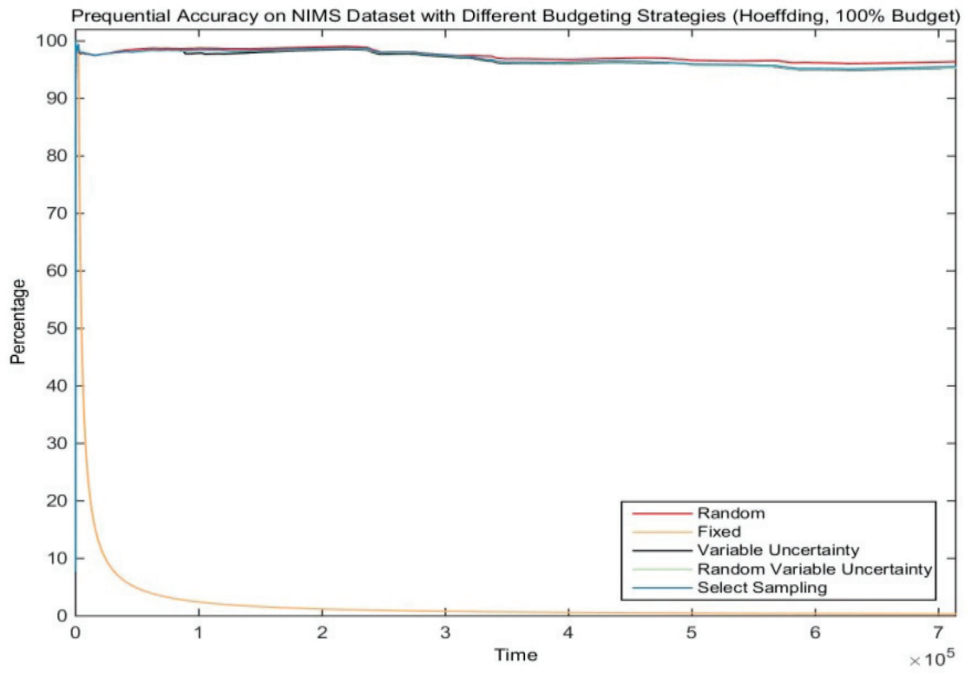

Figure 4.11: Prequential accuracy on NIMS Dataset using the Hoeffding Tree Algorithm with a 100% budget.

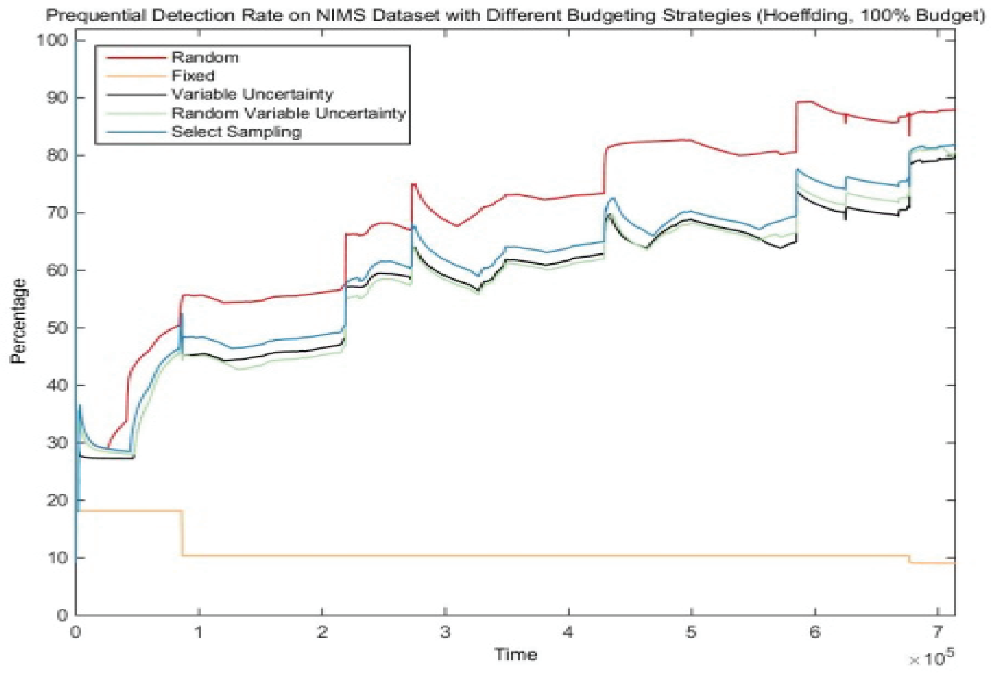

Figure 4.12: Prequential detection rate on NIMS Dataset using the Hoeffding Tree Algorithm with a 100% budget.

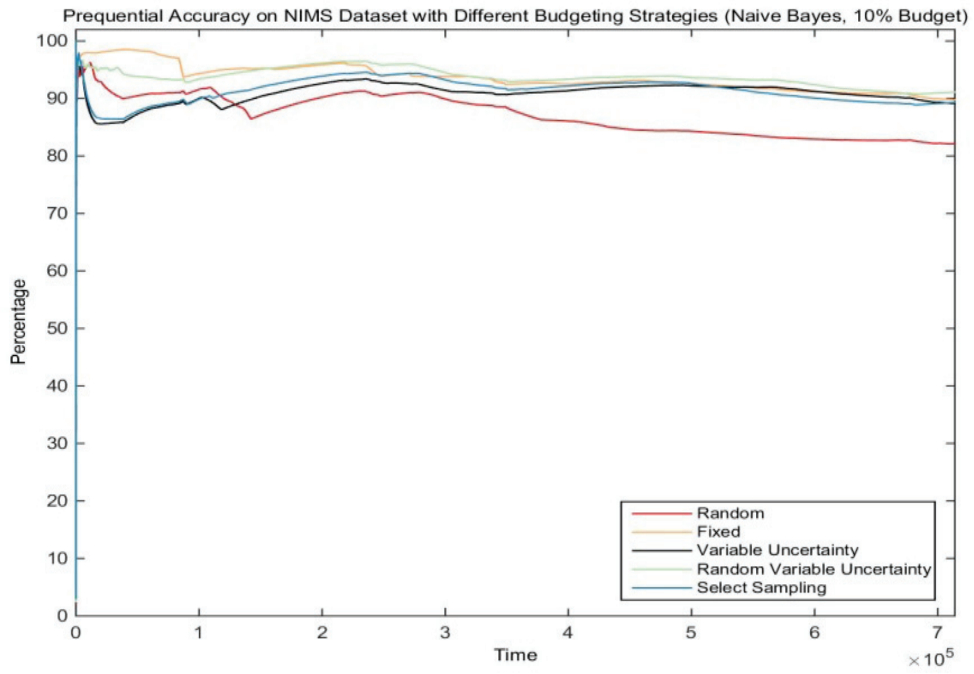

Figure 4.13: Prequential accuracy on NIMS Dataset using the Naive Bayes Algorithm with a 10% budget.

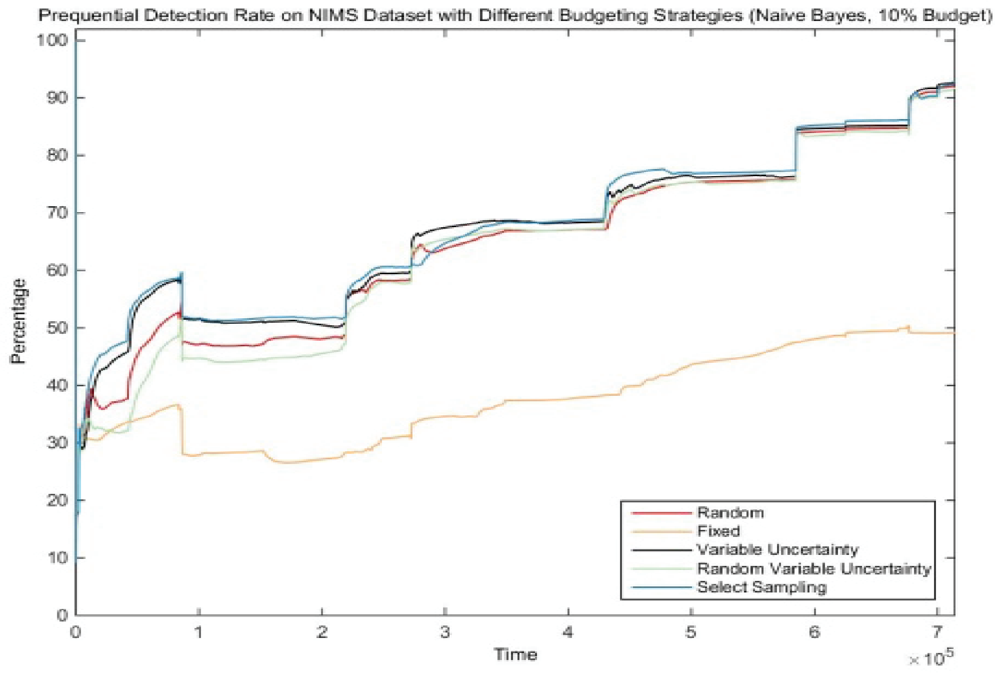

Figure 4.14: Prequential detection rate on NIMS Dataset using the Naive Bayes Algorithm with a 10% budget.

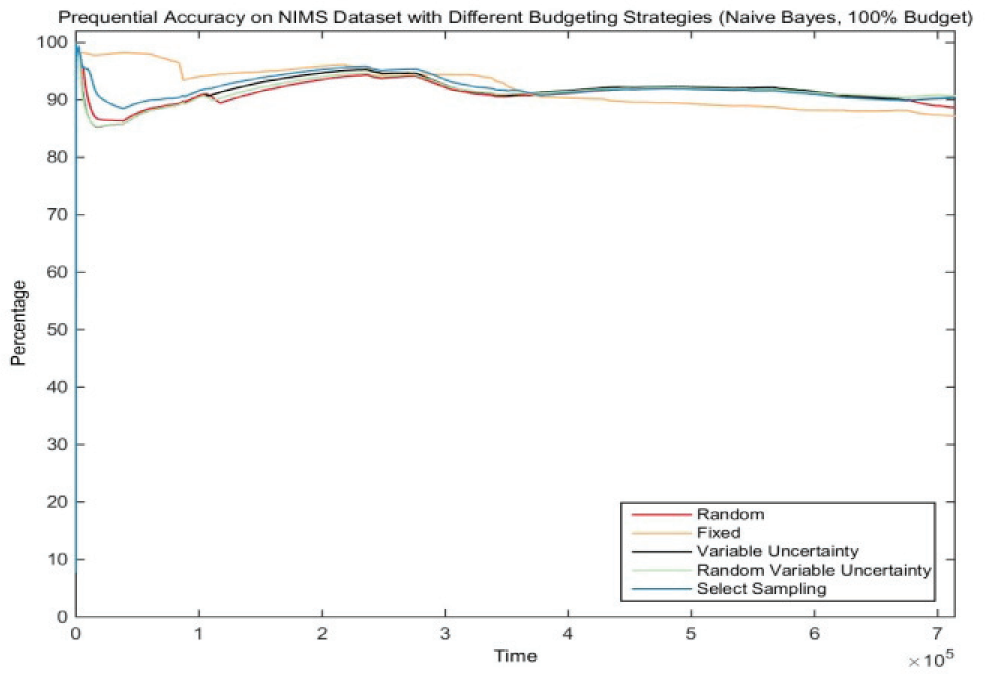

Figure 4.15: Prequential accuracy on NIMS Dataset using the Naive Bayes Algorithm with a 100% budget.

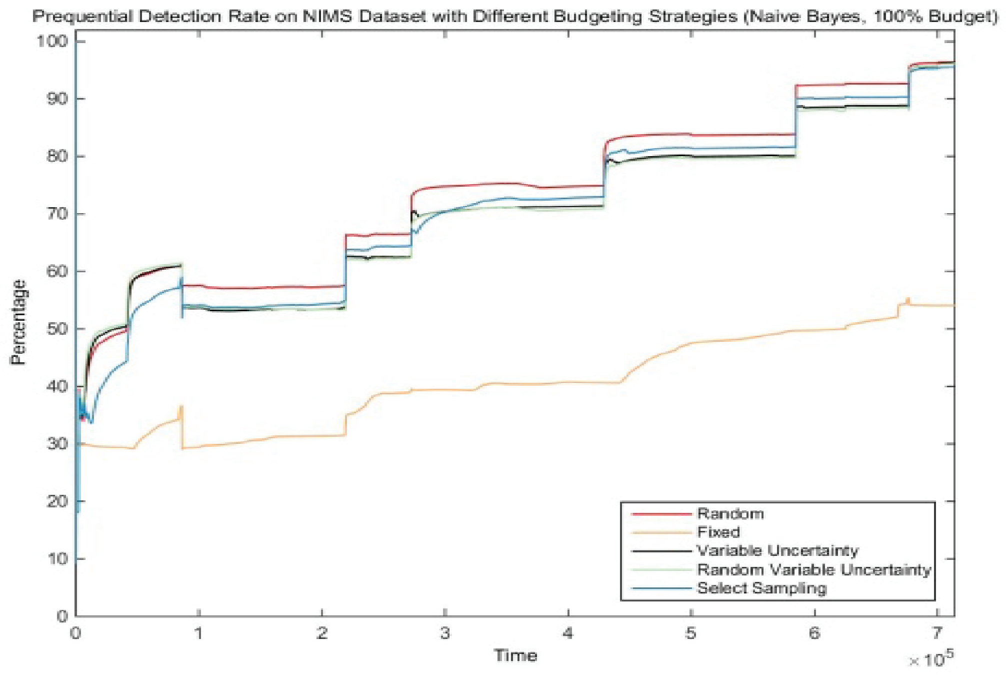

Figure 4.16: Prequential detection rate on NIMS Dataset using the Naive Bayes Algorithm with a 100% budget.

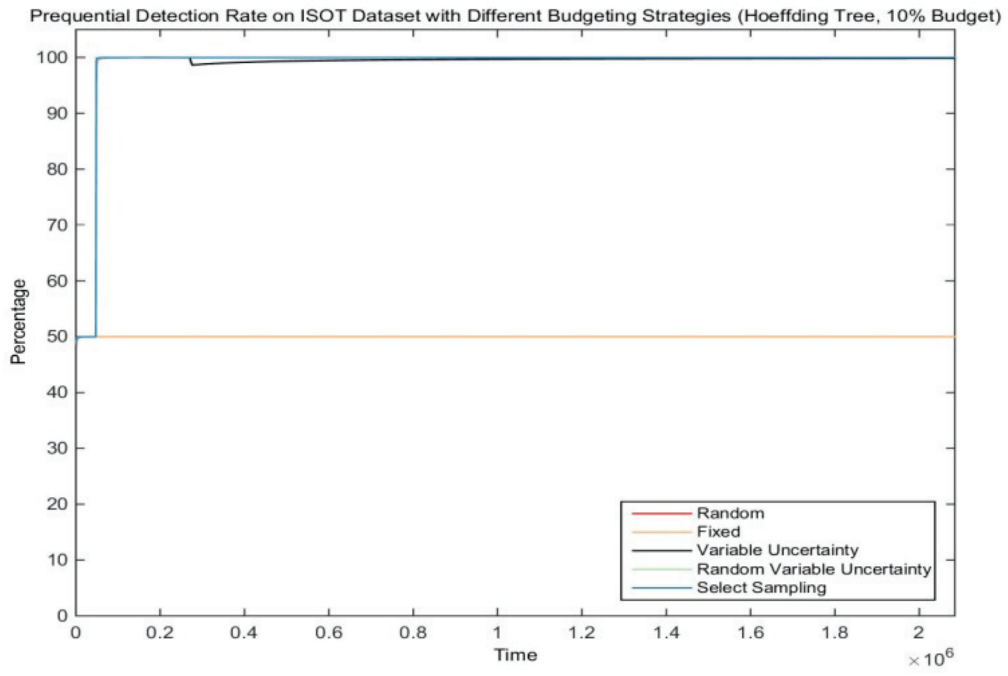

Figure 4.17: Prequential accuracy on ISOT Dataset using the Hoeffding Tree Algorithm with a 10% budget.

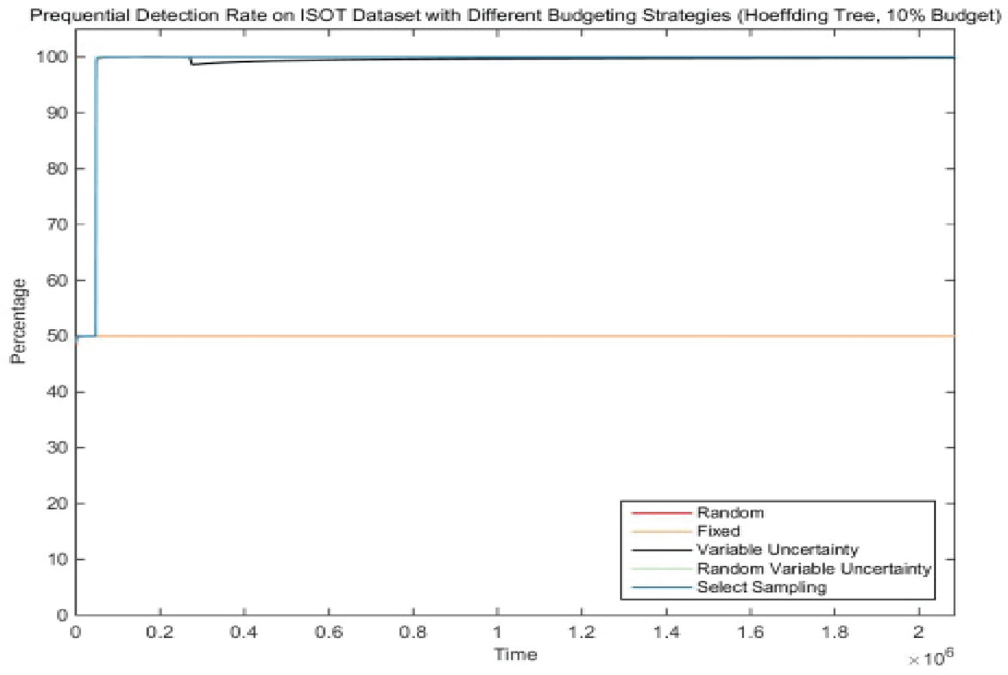

Figure 4.18: Prequential detection rate on ISOT Dataset using the Hoeffding Tree Algorithm with a 10% budget.

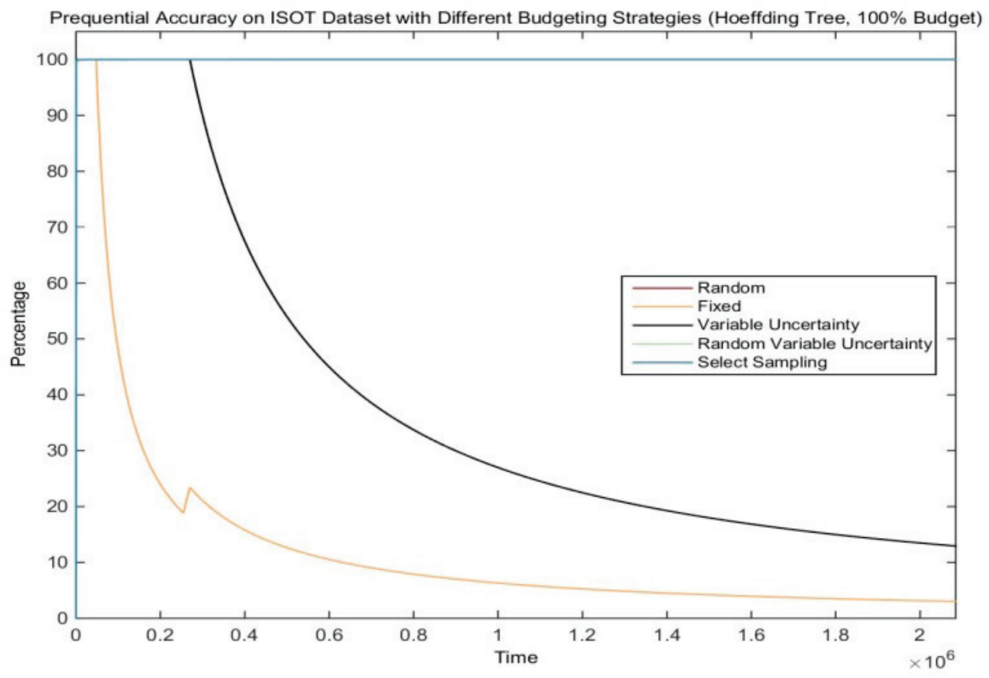

Figure 4.19: Prequential accuracy and prequential detection rate (right) on ISOT Dataset using the Hoeffding Tree Algorithm with a 100% budget.

When looking at the trends when using the selected strategies and chosen machine learning algorithms on the Alexa vs. Zeus dataset (Figures 4.25, 4.26, 4.27, 4.28, 4.29, 4.30, 4.31, and 4.32). Random, Random Variable Uncertainty, and Select Sampling strategies in conjunction with any budget and machine learning algorithm maintain high prequential accuracy and prequential detection rates throughout this streaming dataset. The Variable Uncertainty strategy performs similarly to the previous cases except when using the Hoeffding Tree Algorithm at a 10% budget. In this case, the prequential accuracy and the prequential detection rate drop in performance.

As with the previous experiments, the uncertainty strategy performs differently than the others. When using the Hoeffding Tree algorithm, the prequential detection rate and prequential accuracy performances remain fairly low whereas when the Naive Bayes Algorithm is used, the performance begins to go upwards again.

When viewing the prequential accuracy when using the Hoeffding Tree algorithm with a labeling budget of 10% (Figure 4.33) on the CTU-2 Dataset: Neris, the performance of each budgeting strategy remains fairly steady though out the stream. In this scenario performance hovers at approximately 99%. A similar result

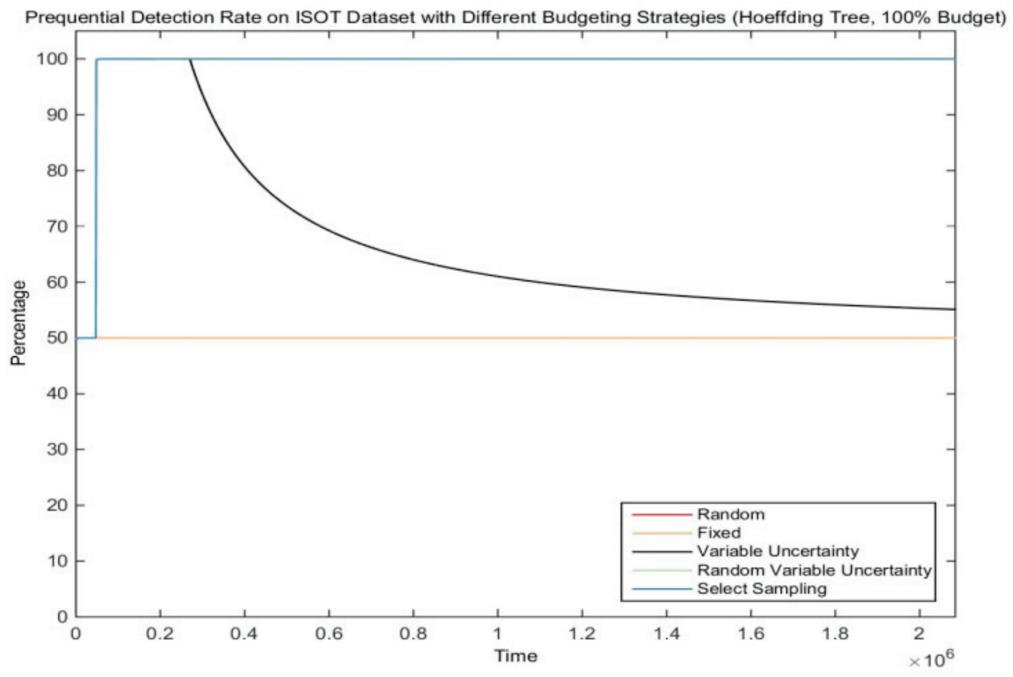

Figure 4.20: Prequential detection rate on ISOT Dataset using the Hoeffding Tree Algorithm with a 100% budget.

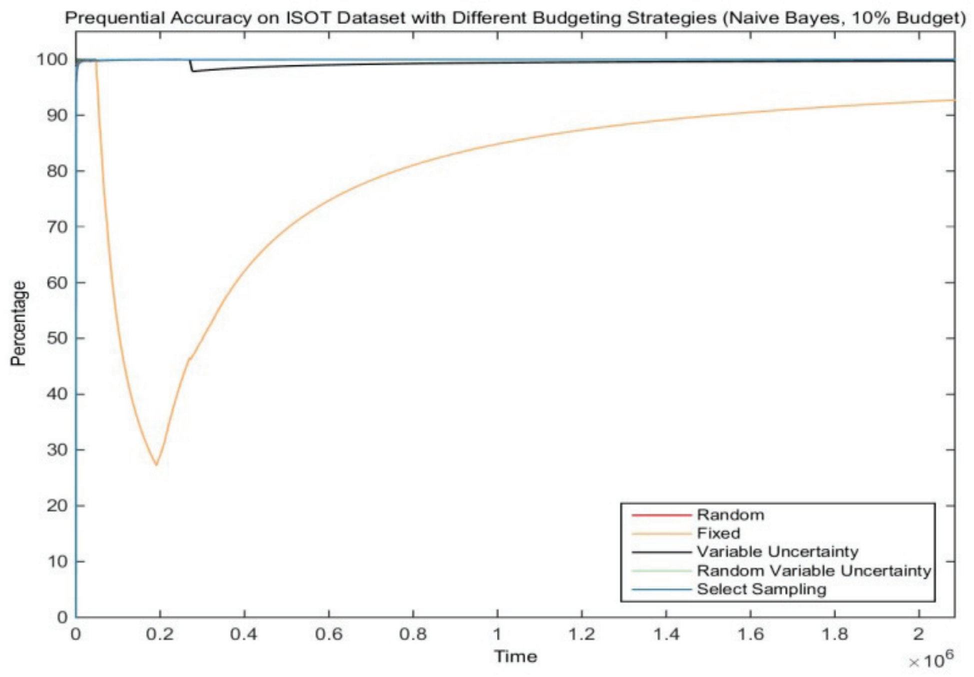

Figure 4.21: Prequential accuracy on ISOT Dataset using the Naive Bayes Algorithm with a 10% budget.

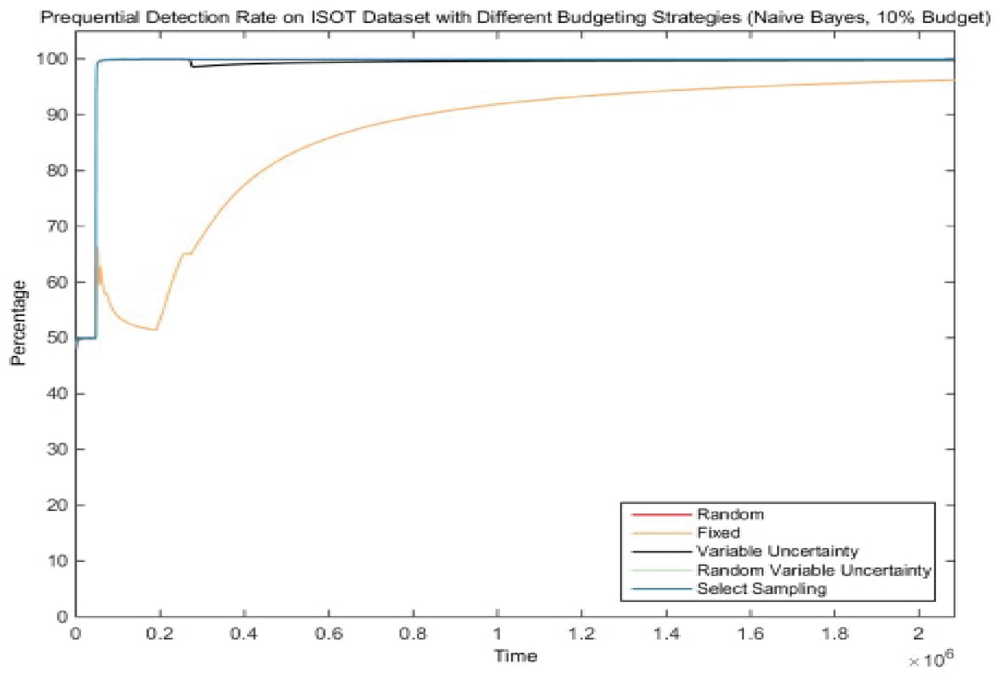

Figure 4.22: Prequential detection rate on ISOT Dataset using the Naive Bayes Algorithm with a 10% budget.

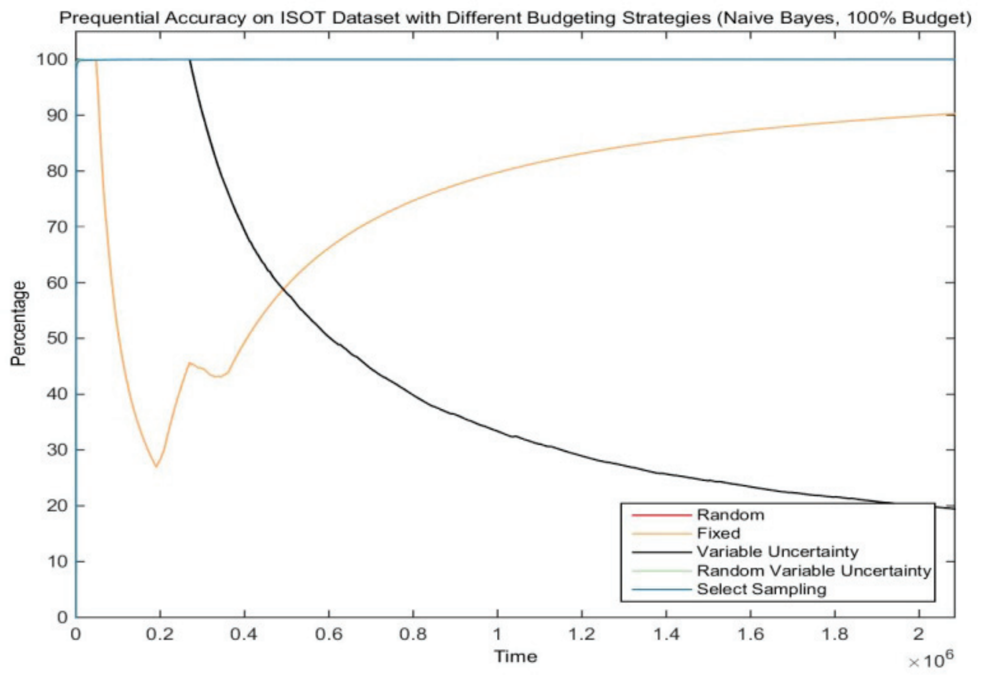

Figure 4.23: Prequential accuracy on ISOT Dataset using the Naive Bayes Algorithm with a 100% budget.

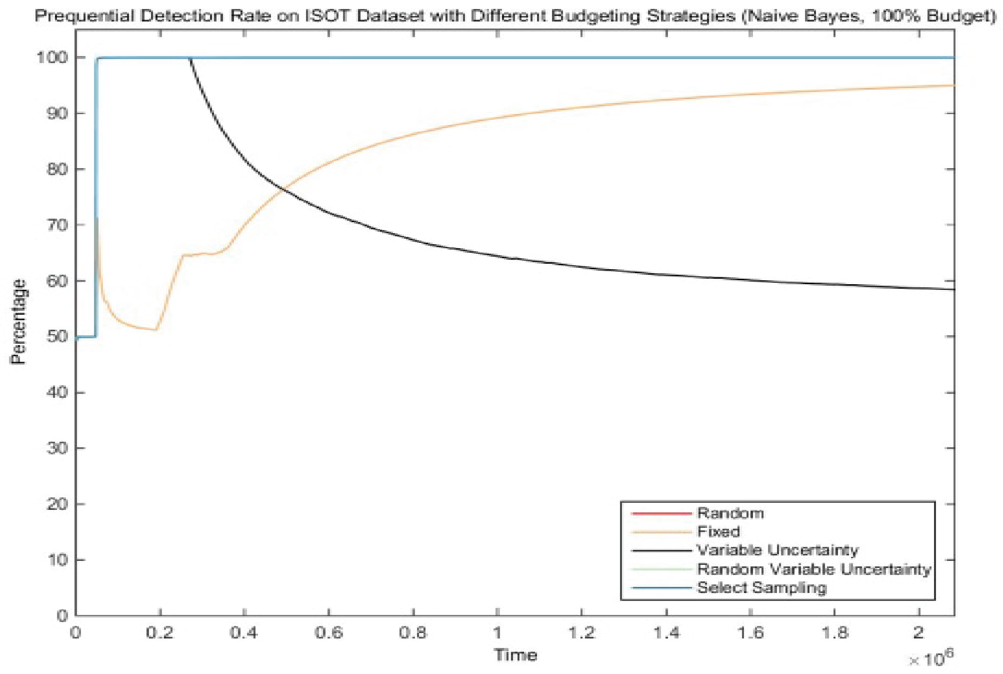

Figure 4.24: Prequential detection rate on ISOT Dataset using the Naive Bayes Algorithm with a 100% budget.

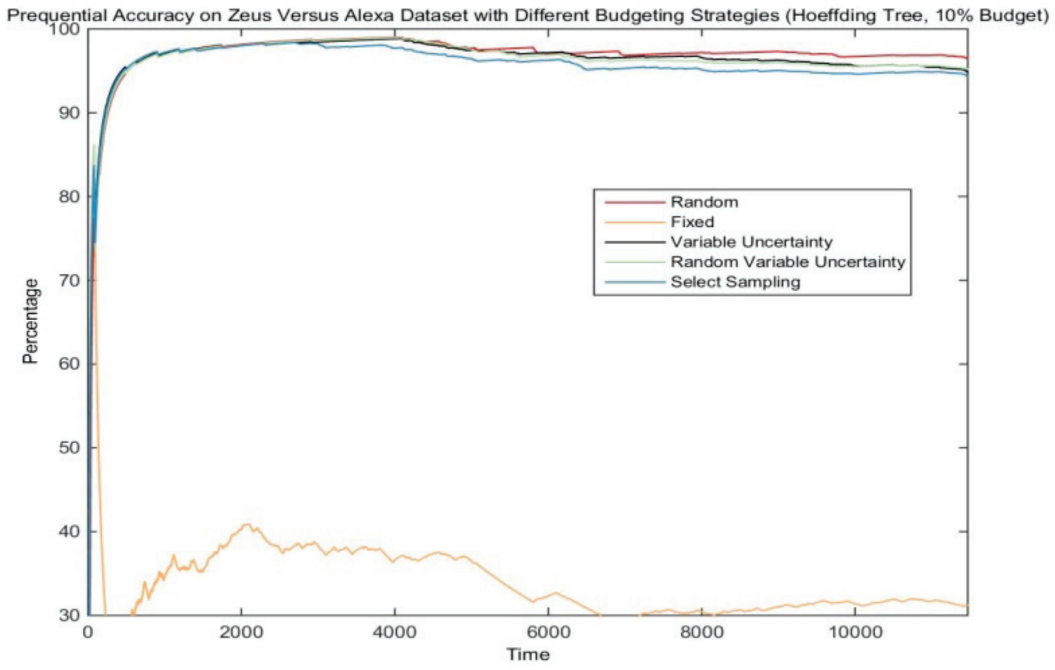

Figure 4.25: Prequential accuracy on Alexa vs. Zeus Dataset using the Hoeffding Tree Algorithm with a 10% budget.

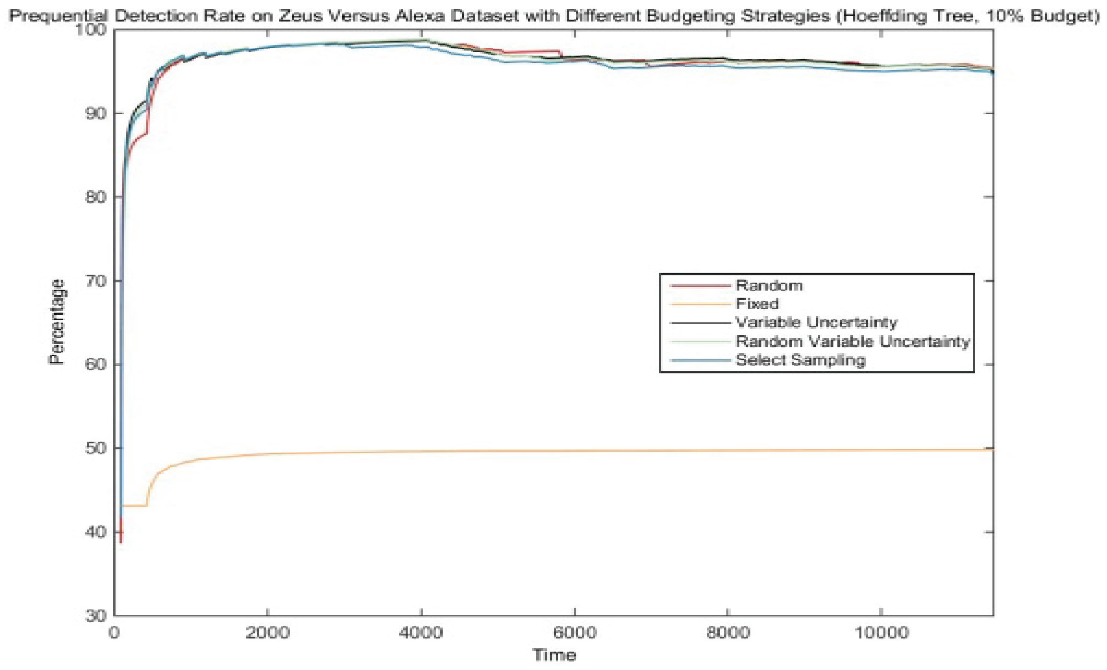

Figure 4.26: Prequential detection rate on Alexa vs. Zeus Dataset using the Hoeffding Tree Algorithm with a 10% budget.

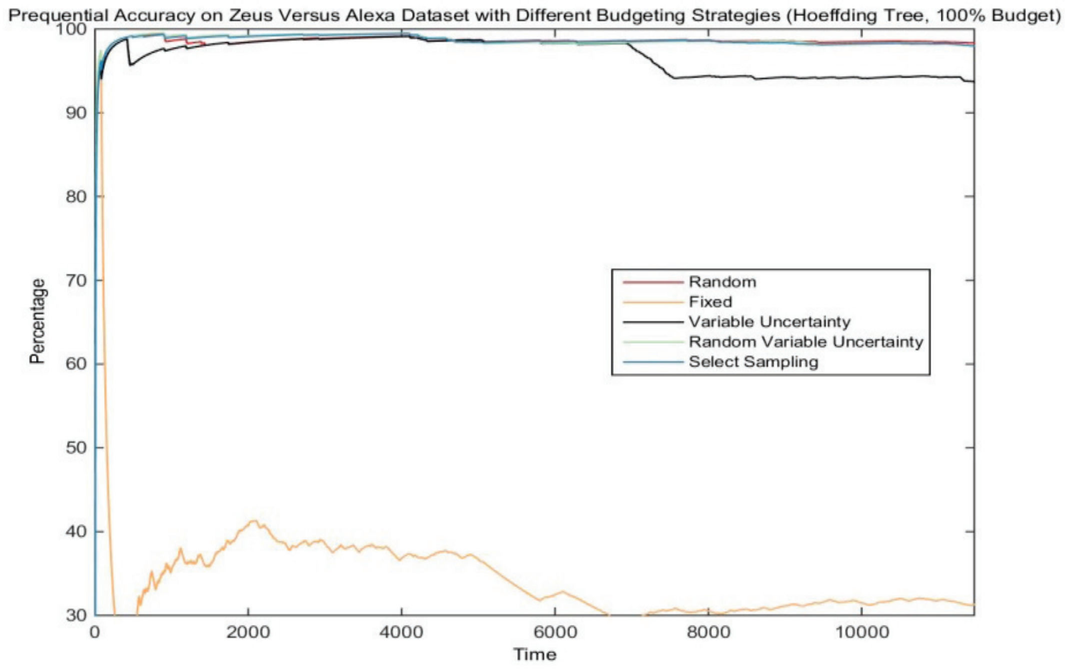

Figure 4.27: Prequential accuracy on Alexa vs. Zeus Dataset using the Hoeffding Tree Algorithm with a 100% budget.

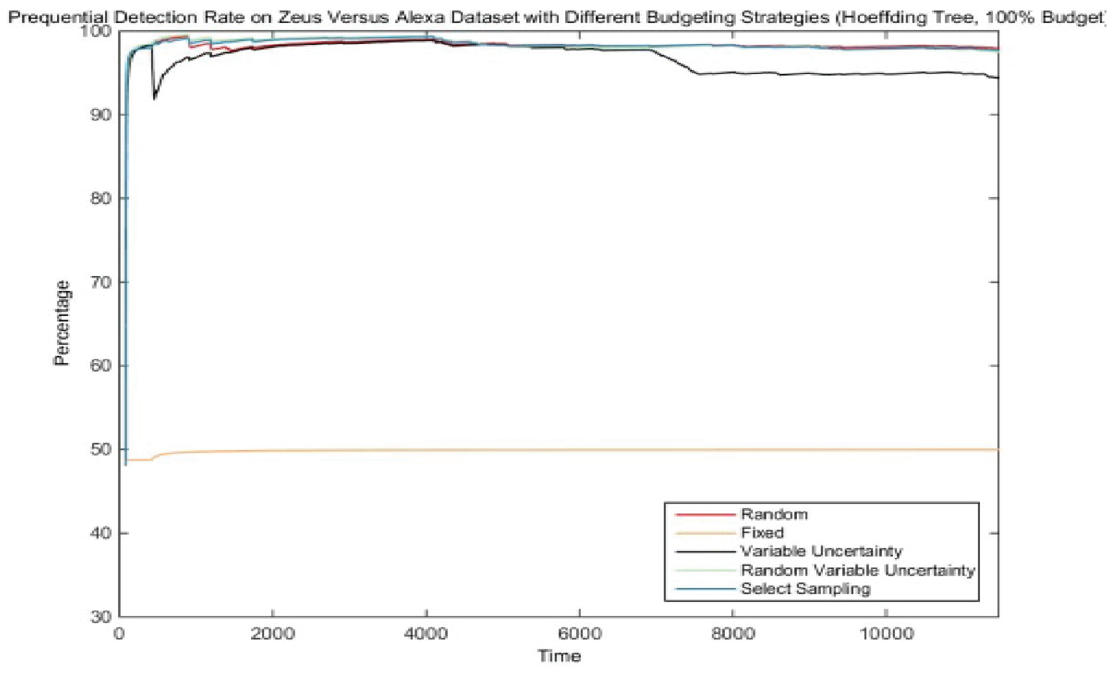

Figure 4.28: Prequential detection rate on Alexa vs. Zeus Dataset using the Hoeffding Tree Algorithm with a 100% budget.

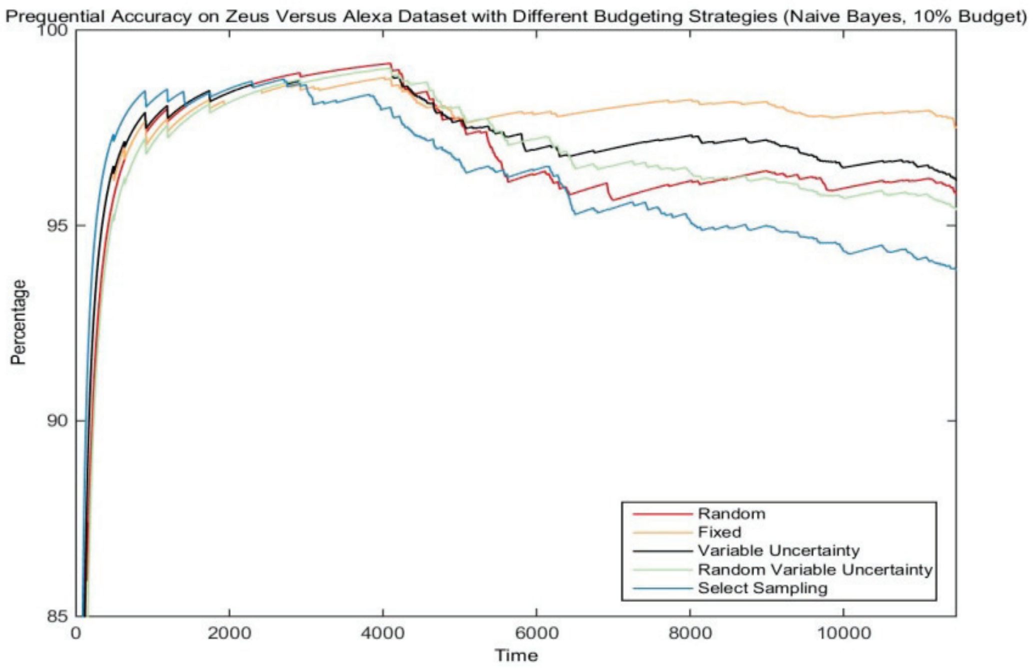

Figure 4.29: Prequential accuracy on Alexa vs. Zeus Dataset using the Naive Bayes Algorithm with a 10% budget.

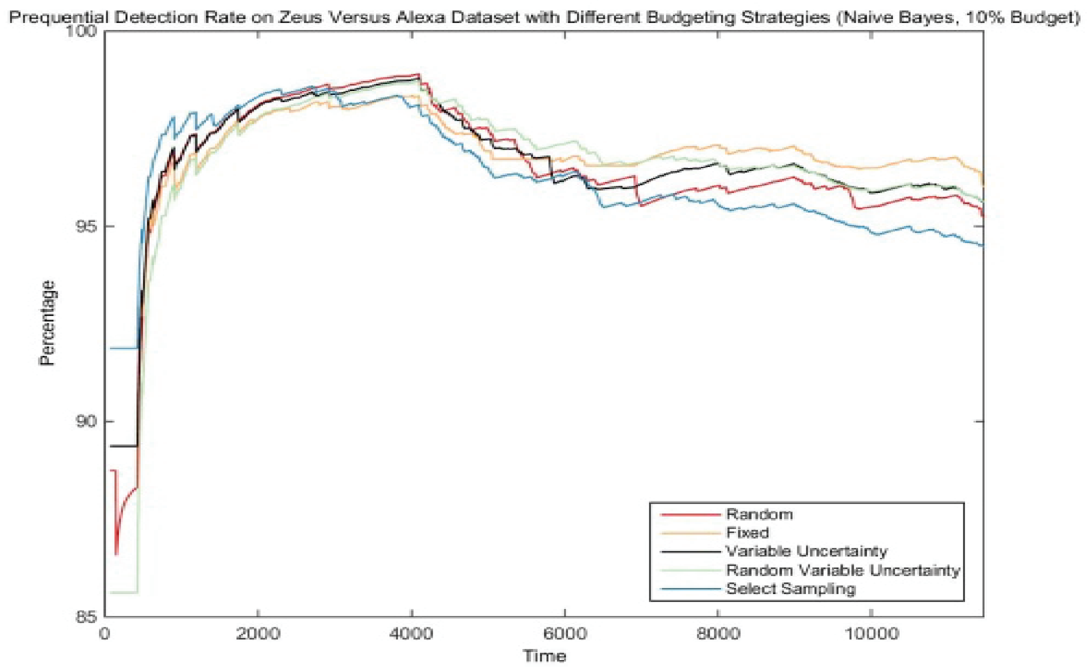

Figure 4.30: Prequential detection rate on Alexa vs. Zeus Dataset using the Naive Bayes Algorithm with a 10% budget.

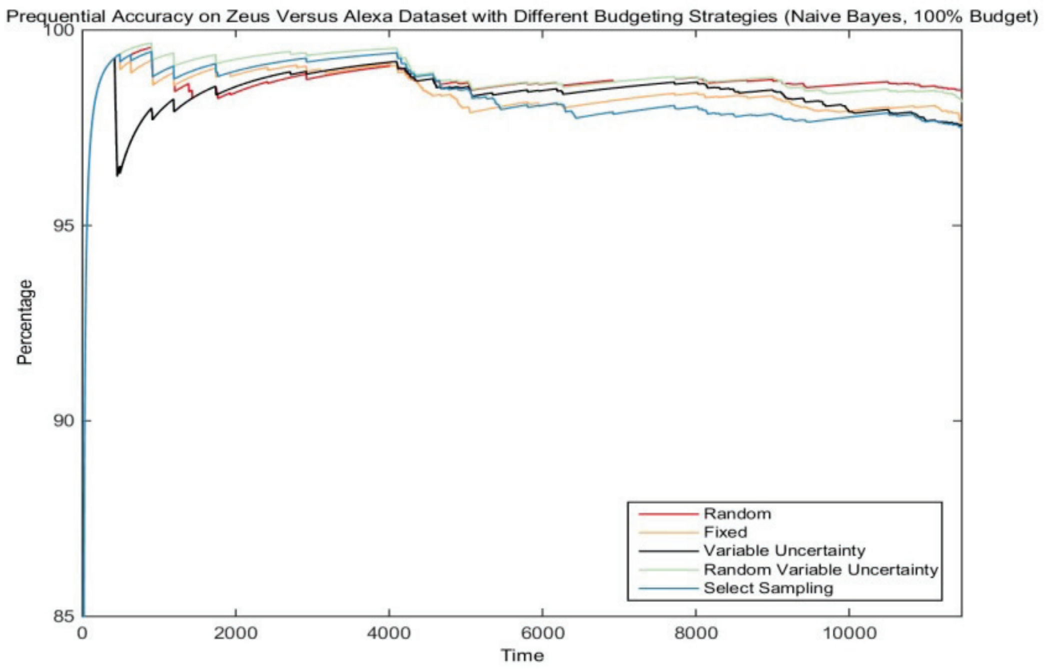

Figure 4.31: Prequential accuracy on Alexa vs. Zeus Dataset using the Naive Bayes Algorithm with a 100% budget.

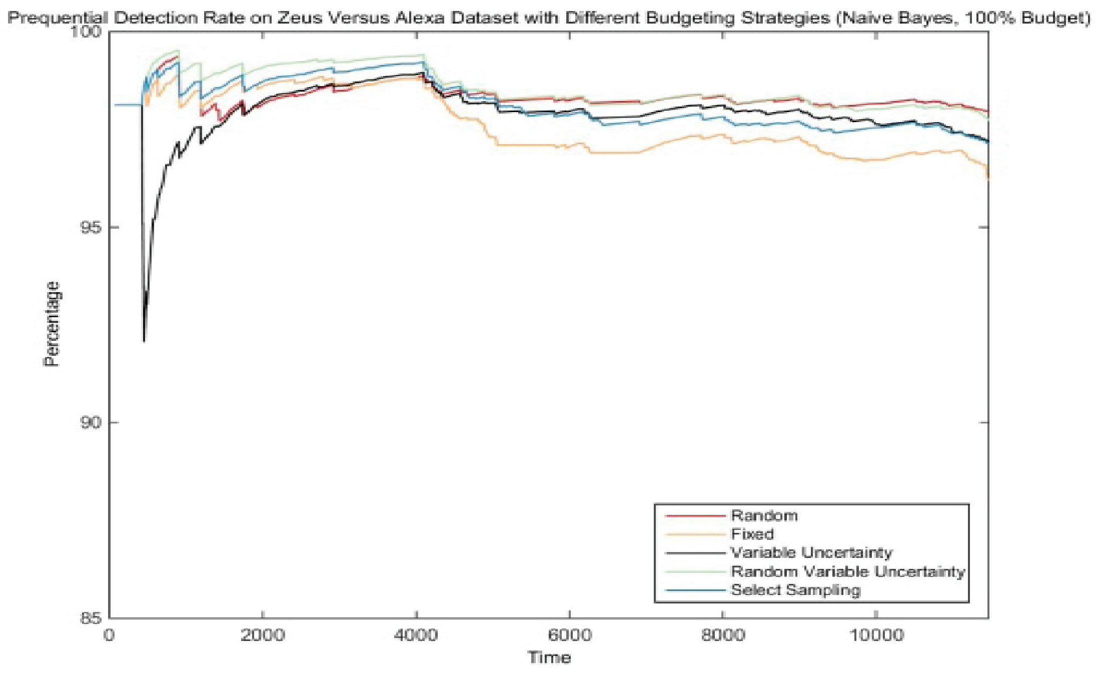

Figure 4.32: Prequential detection rate on Alexa vs. Zeus Dataset using the Naive Bayes Algorithm with a 100% budget.

is displayed when implementing the Hoeffding Tree algorithm with a budget of 100% (Figure 4.34). However, early on in the stream the prequential accuracy when using the Variable Uncertainty labeling strategy begins to gradually fall to approximately 45%. Thus, a lower performance is presented when using this strategy with a 100% labeling budget as compared to using a 10% labeling budget.

Observing the prequential detection rates when using the Hoeffding Tree algorithm on CTU-2 Dataset: Neris yields a more interesting series of prequential accuracy and detection rate performance patterns. Prequential detection rate, when using the Hoeffding Tree algorithm with a 10% budget (Figure 4.34), begins at a performance of 100% but declines sharply to approximately 50% early on in the stream for all labeling strategies. However, the Random, Random Variable, and Select Sampling strategies see a spike back up in performance to approximately 87%. On the other hand, Fixed and Variable Uncertainty strategy prequential detection rate performances remain at 50% for the rest of the stream. Random Variable Uncertainty and Select sampling prequential detection rate performance gradually fall to approximately 64% while the

Random strategy performance takes more of a sharp fall to this performance rate before gradually falling to approximately 57%. The prequential detection rate pattern when using the Hoeffding Tree algorithm with a budget of 100% (Figure 4.36) yields similar results, however there are two notable differences. The first difference is that the Random strategy performance pattern follows the patterns of the Random Variable Uncertainty and Select Sampling strategies rather than taking a sharp fall in its decline in performance. The second and more notable difference when using a budget of 100% rather than 10% is that after reaching a performance of 50%, the Variable Uncertainty strategy's performance continues to gradually fall to approximately 24%, thus resulting in lower performance than when using a labeling budget of 10%.

In terms of the prequential detection rate when using the Naive Bayes algorithm on the CTU-2 Dataset (Figures 4.38 and 4.40), the patterns displayed in the graphs follow the same patterns as exhibited when using the Hoeffding Tree algorithm. However, the Random, Random Variable Uncertainty. and Select Sampling labeling strategies perform significantly better when using the Naive Bayes algorithm as opposed to the Hoeffding tree algorithm in this scenario.

When using a 10% labeling budget in conjunction with the Naive Bayes machine learning algorithm on the CTU-2 Dataset (Figure 4.37) all strategies begin with a 100% prequential accuracy performance. Early in the stream, however, a divergence occurs in performance among the different labeling strategies. The Fixed and Variable Uncertainty strategies tend to stay around 99% prequential accuracy while the other strategies slowly decline to approximately 75-85%. When using a 100% labeling budget (Figure 4.39), the Random, Random Variable Uncertainty, and Select Sampling labeling strategies follow a similar performance pattern to when using a labeling budget of 10%. However, the other strategies show differences between the two chosen budgets. The Fixed strategy exhibits a slight decline before gradually climbing back up to approximately 96%. The Variable Uncertainty strategy prequential accuracy performance begins at 100% but declines to approximately 44%.

Finally, the results when implementing the chosen methods on the CTU-11 Dataset are discussed. In Figures 4.41 and 4.43, results are approximately the same between budgets of 10% and 100% budgets, in terms of prequential accuracy performance, when using the Hoeffding Tree machine learning algorithm. Even individual

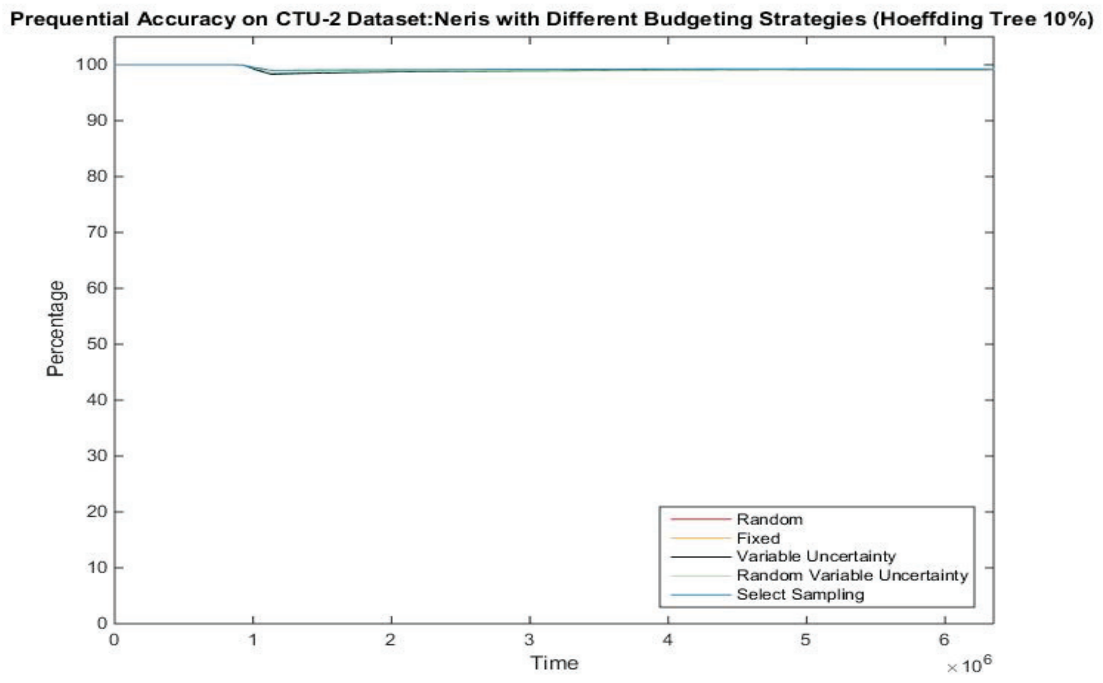

Figure 4.33: Prequential accuracy on CTU-2 Dataset: Neris using the Hoeffding Tree algorithm with a 10% budget.

labeling strategies perform similarly in this scenario with all strategies, except the Fixed labeling strategy that gradually declines to a performance of approximately 32%, maintaining a prequential accuracy performance of nearly 100% throughout the stream.

Prequential detection rate results when using the Hoeffding Tree algorithm are fairly similar between budgets of 10% (Figure 4.42) and 100% (Figure 4.44) as well. For both budgets, Random, Random Variable Uncertainty, and Select Sampling strategies achieve a high performance early on, however while the jump from approximately 50% prequential detection rate to over 90% is fairly abrupt when using a labeling budget of 100%, the change is slightly more gradual when using a budget of 10%. It should be noted that for both budgets, the Fixed strategy performs the same, maintaining a prequential detection rate of approximately 50% throughout the stream.

The aforementioned result patterns when applying the Hoeffding Tree machine learning algorithm are approximately the same when using the Naive Bayes algorithm on the CTU-11 Dataset.

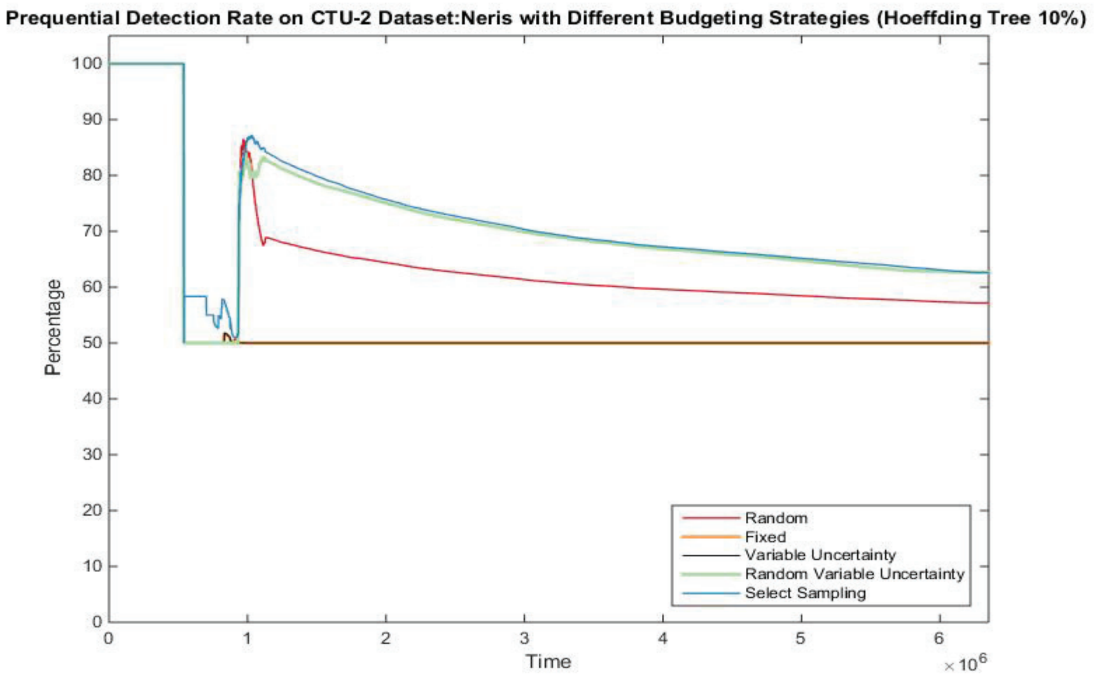

Figure 4.34: Prequential detection rate on CTU-2 Dataset: Neris using the Hoeffding Tree algorithm with a 10% budget.

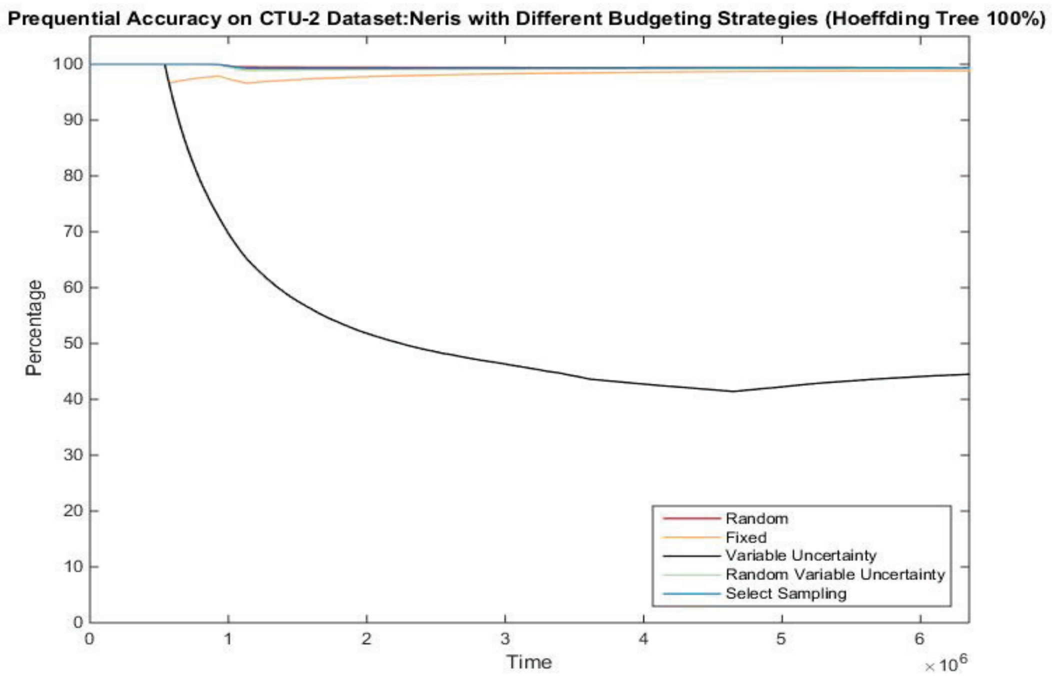

Figure 4.35: Prequential accuracy on CTU-2 Dataset: Neris using the Hoeffding Tree algorithm with a 100% budget.

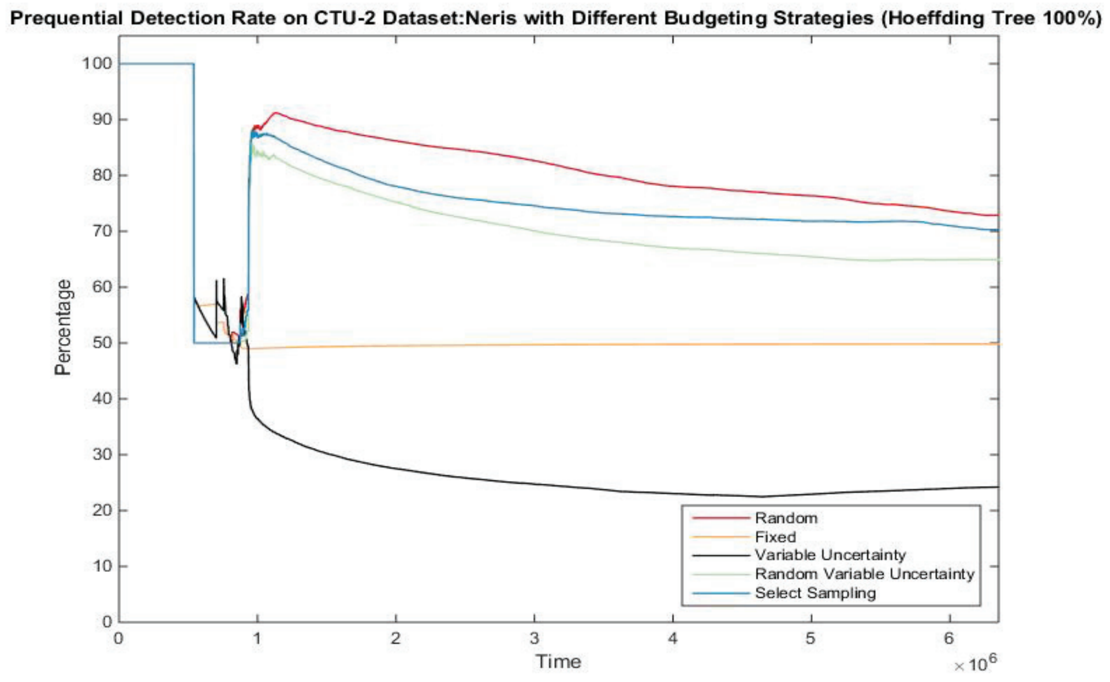

Figure 4.36: Prequential detection rate on CTU-2 Dataset: Neris using the Hoeffding Tree algorithm with a 100% budget.

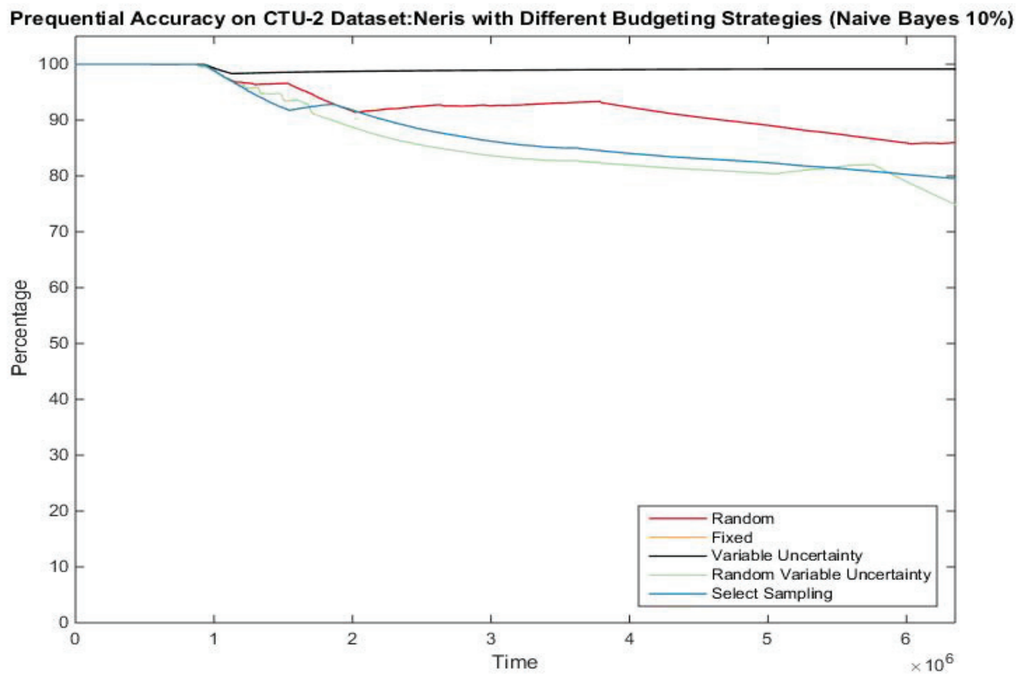

Figure 4.37: Prequential accuracy on CTU-2 Dataset: Neris using the Naive Bayes algorithm with a 10% budget.

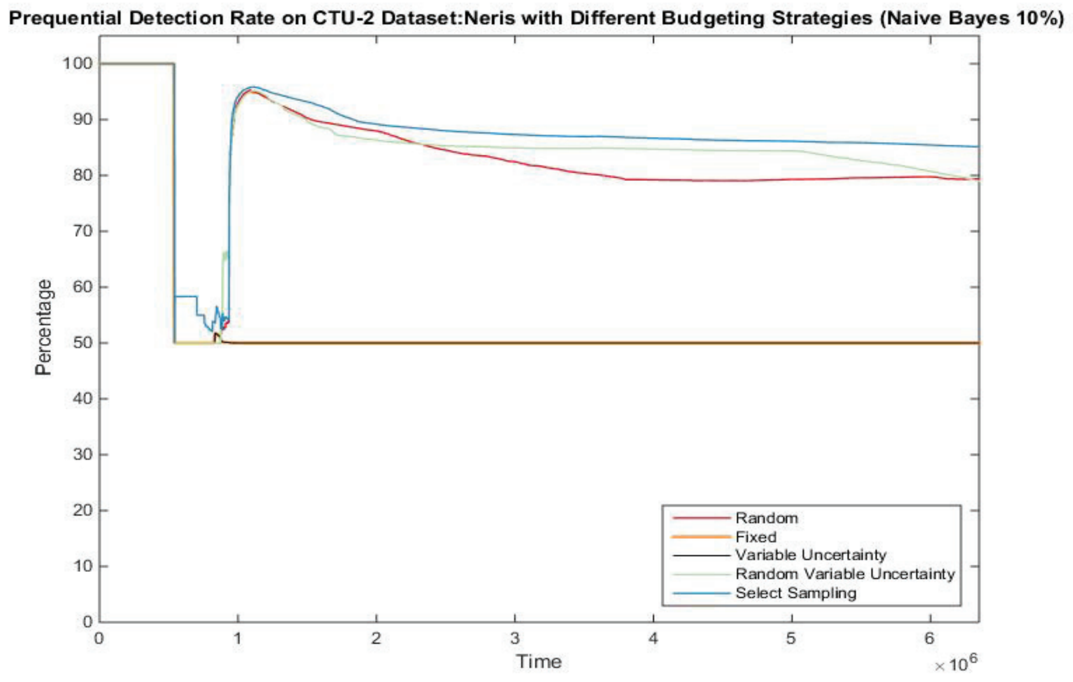

Figure 4.38: Prequential detection rate on CTU-2 Dataset: Neris using the Naive Bayes algorithm with a 10% budget.

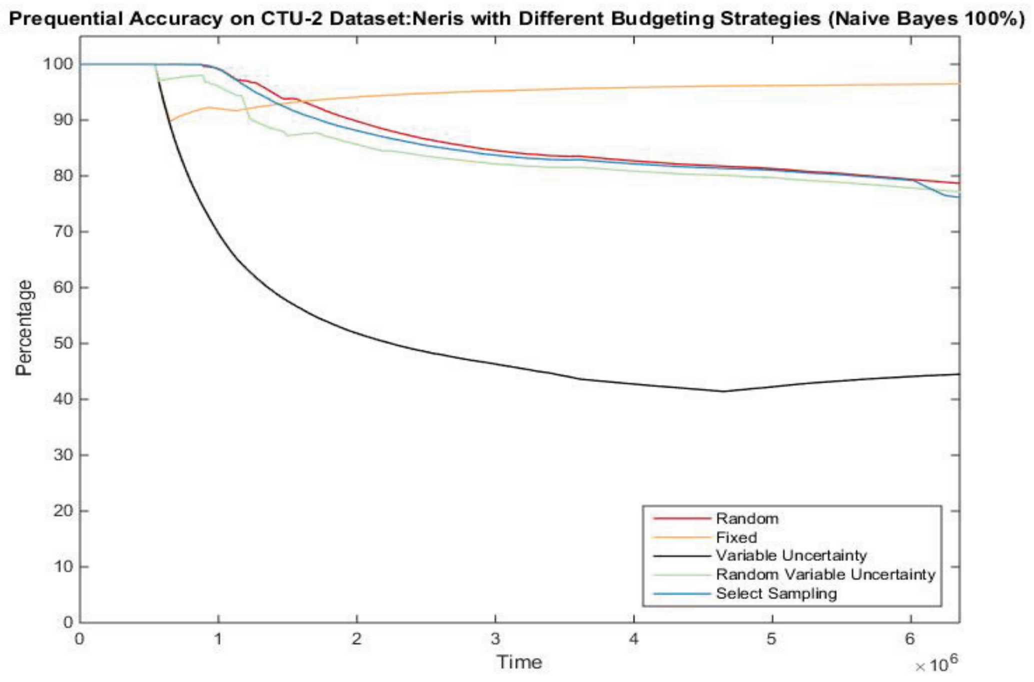

Figure 4.39: Prequential accuracy on CTU-2 Dataset: Neris using the Naive Bayes algorithm with a 100% budget.

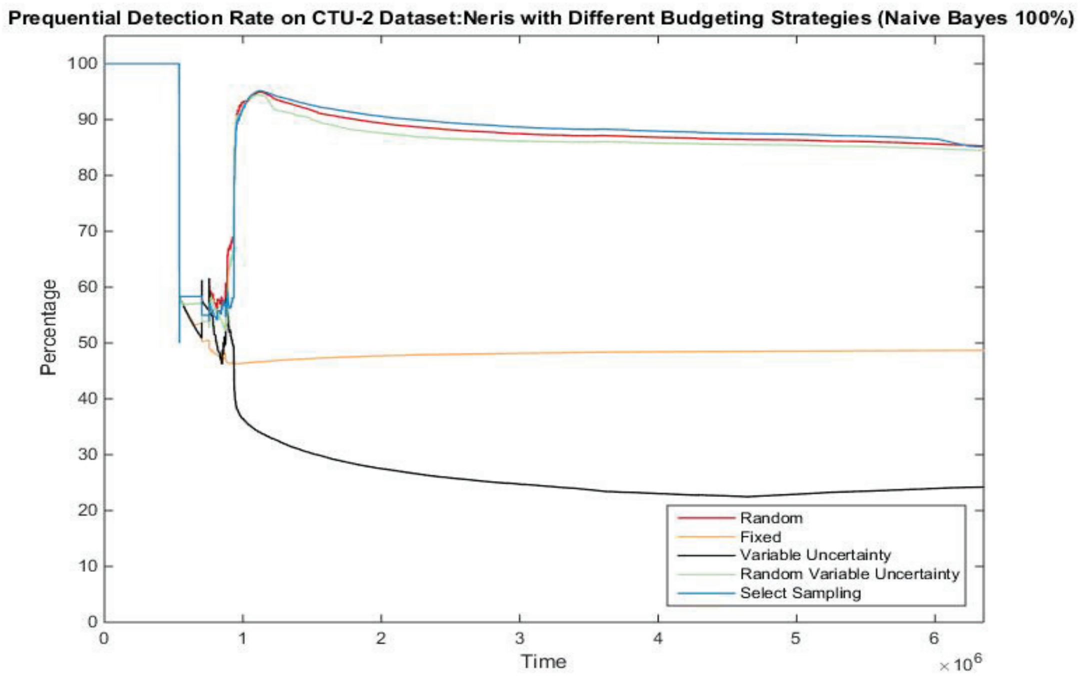

Figure 4.40: Prequential detection rate on CTU-2 Dataset: Neris using the Naive Bayes algorithm with a 100% budget.

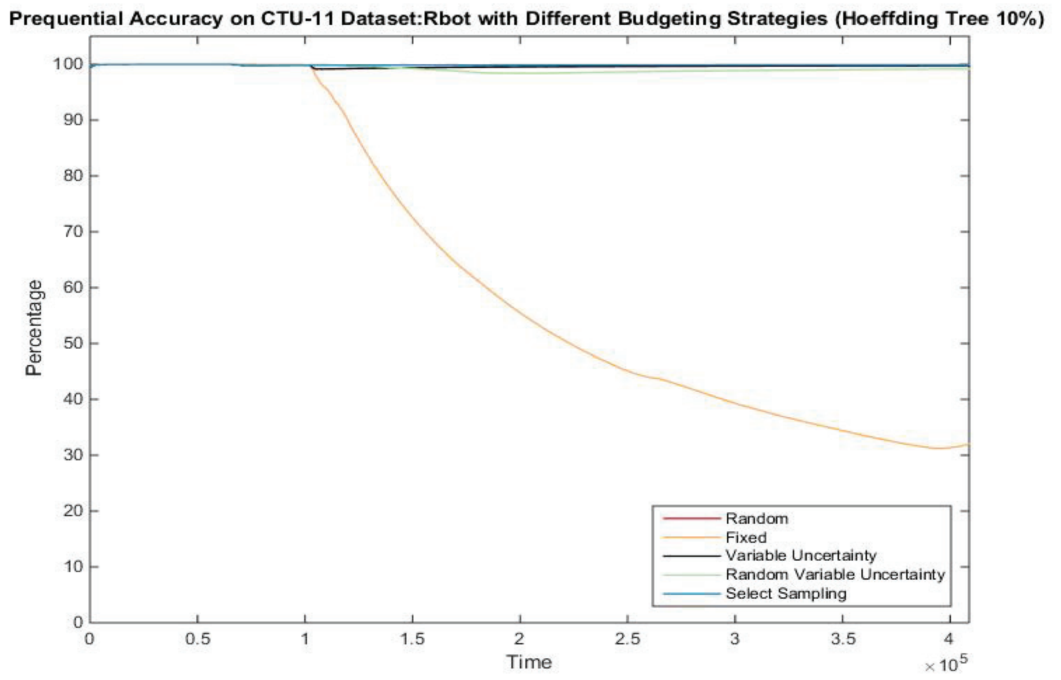

Figure 4.41: Prequential accuracy on CTU-11 Dataset: Rbot using the Hoeffding Tree algorithm with a 10% budget.

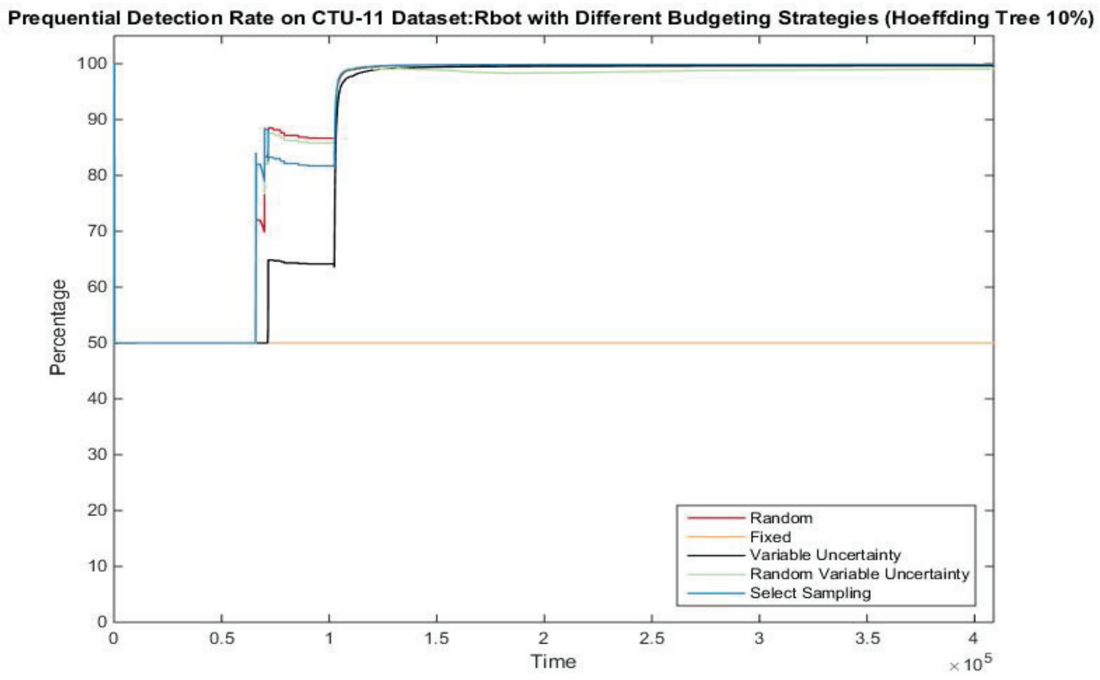

Figure 4.42: Prequential detection rate on CTU-11 Dataset: Rbot using the Hoeffding Tree algorithm with a 10% budget.

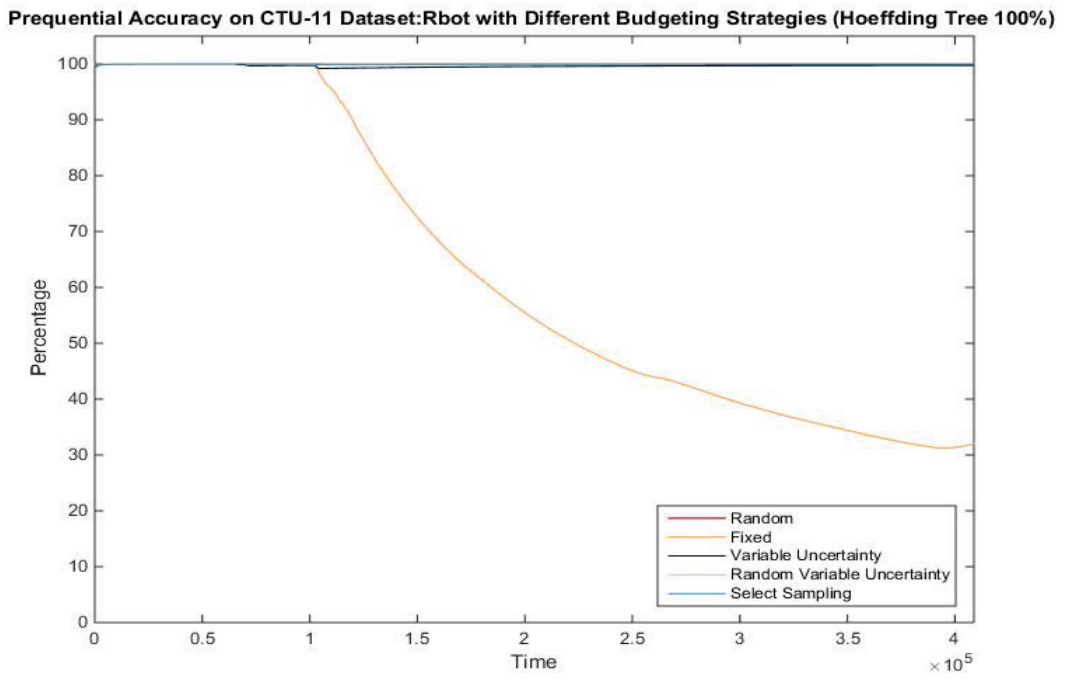

Figure 4.43: Prequential accuracy on CTU-11 Dataset: Rbot using the Hoeffding Tree algorithm with a 100% budget.

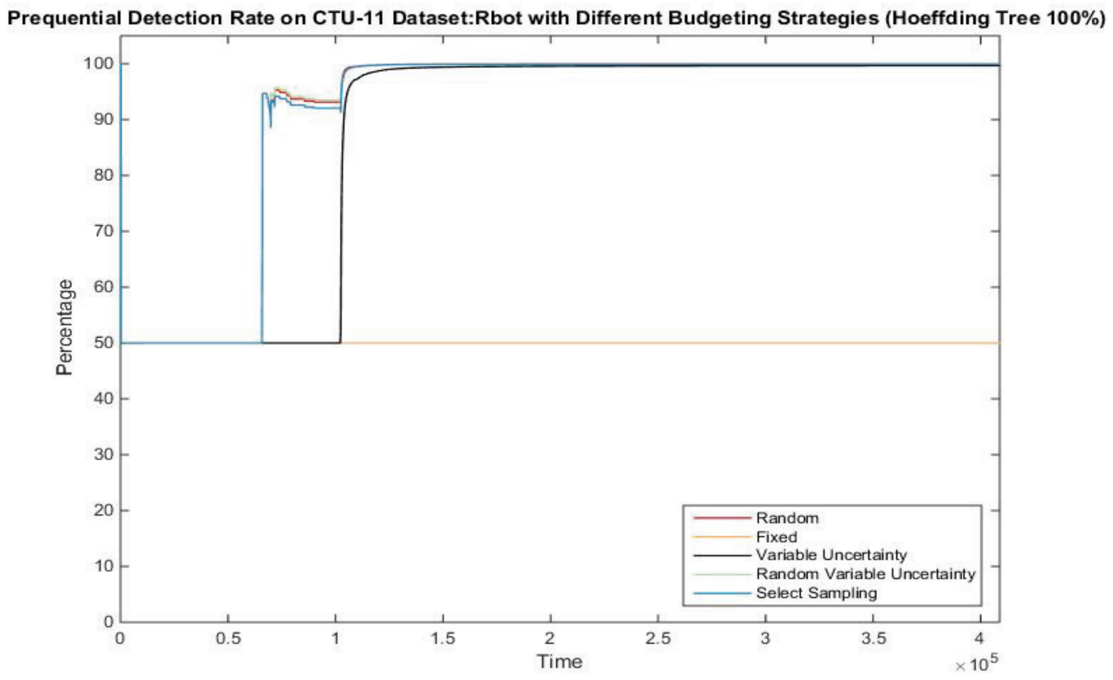

Figure 4.44: Prequential detection rate on CTU-11 Dataset: Rbot using the Hoeffding Tree algorithm with a 100% budget.

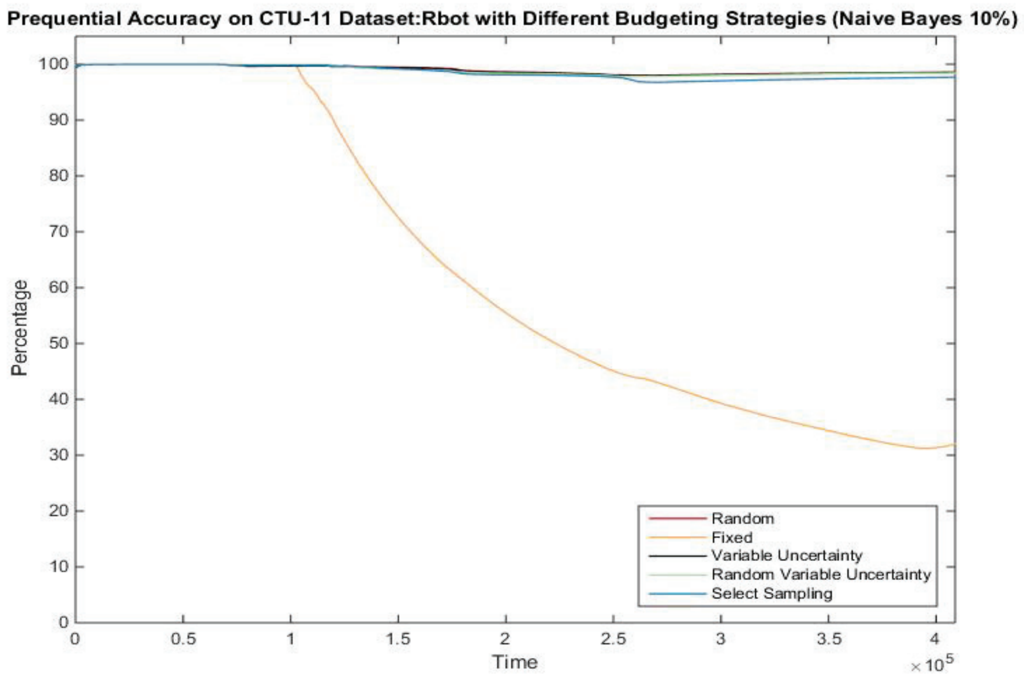

Figure 4.45: Prequential accuracy on CTU-11 Dataset: Rbot using the Naive Bayes algorithm with a 10% budget.

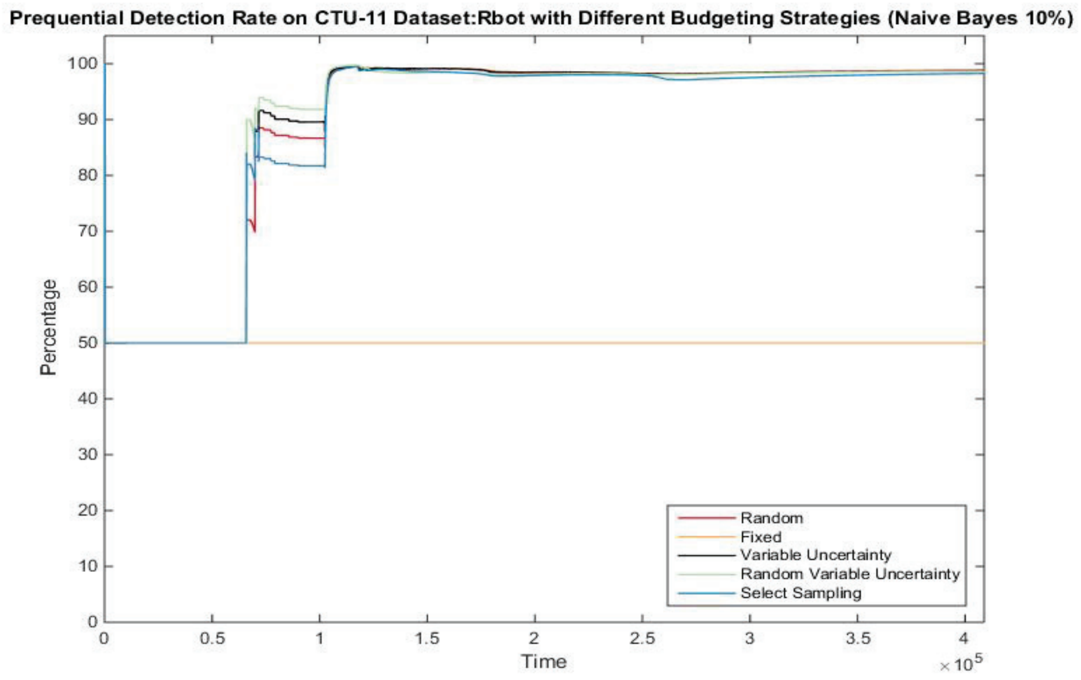

Figure 4.46: Prequential detection rate on CTU-11 Dataset: Rbot using the Naive Bayes algorithm with a 10% budget.

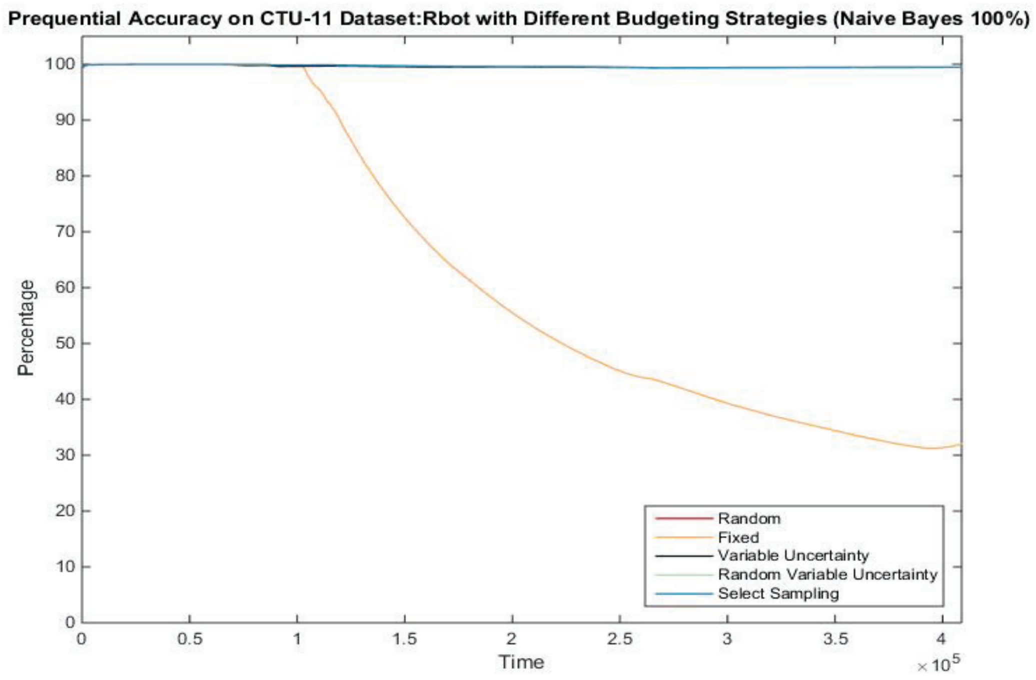

Figure 4.47: Prequential accuracy on CTU-11 Dataset: Rbot using the Naive Bayes algorithm with a 100% budget.

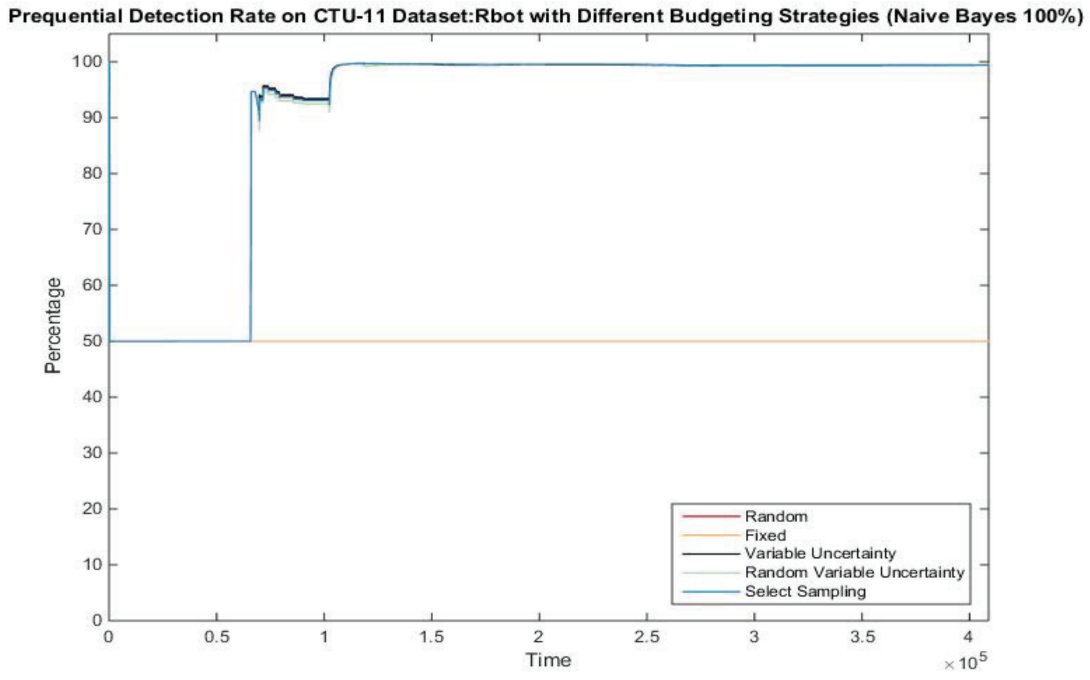

Figure 4.48: Prequential detection rate on CTU-11 Dataset: Rbot using the Naive Bayes algorithm with a 100% budget.

Based on the similarity in performances between using 10% and 100% budgets in all experiments, the results indicate that it is well worth it to use a low budget as there appears to be little to no negative effect on the overall performance. In some cases, using a 10% yielded higher performance than when using a budget of 100%. This may be surprising to some as one may assume that with less information provided the classifier should have a lower performance. This is not the case, however, as machine learning algorithms can often fall victim to over fitting. Over fitting occurs when a classifier fits its model too closely to the instances that have already been seen during the training rather than creating a model that effectively predicts new incoming instances. This is especially problematic when the classifier begins to fit its model to instances that are outliers and do correlate to other instances of the same class. Using a lower budget reduces the problem of overfitting, as we are more selective with what instances to train on.

One may recall from Table 3.1, that distribution of classes varied somewhat among the datasets. However, the distribution of classes did not appear to have an impact on performance as performance on all datasets was quite high. On the other hand, concept shift appeared to have a impact on performance in terms of prequential accuracy and prequential detection rate as we often see a dip in performance once a large burst of instances of a new class is introduced.

Lastly, it should be mentioned that the overall accuracy and detection rates are usually quite high when paired with the Random, Variable Uncertainty, Random Variable Uncertainty, or Select Sampling strategies with performances averaging in the 90%s while the fixed uncertainty strategy performed poorly in most cases.

4.2 Adaptive Artificial Neural Network Results

The overall accuracy and detection rate when applying the Adaptive Neural Network approach on the same datasets used above are presented in Table 4.7. Here, an overall detection rate of approximately 50% on all datasets with two classes is observed. Some very low prediction accuracies are also observed. Furthermore, in this case, the highest achieved overall accuracy and overall detection rate are 68% and 50%, respectively. These results are significantly lower than most of the results presented in the previous section. Furthermore, application of the Adaptive Artificial Neural Network was quite costly in terms of time as datasets could take hours or even days to process.

Table 4.7: Overall accuracy and detection rate when using Adaptive Artificial Neural Networks on various datasets.

	KDD 1999 Cup	**NIMS1**	**ISOT**	**Zeus vs. Alexa**	**CTU-2 Dataset**	**CTU-11 Dataset**
Accuracy	19.69	1.71	6.84	68.32	99.05	31.98
Detection Rate	49.97	16.73	48.48	49.74	50.00	49.98

The trend shown when looking at the prequential detection rate using this learning technique on the datasets seems to remain at approximately 50% throughout the streaming process (Figures 4.49, 4.50, 4.52, 4.53, 4.54). This is consistent for tests on all datasets except for the NIMS1 dataset (Figure 4.52) where detection rate is at 16.73 and with the CTU-2 Dataset where the detection rate remains at 100% for a short time. This is to be expected however, as there are more classes in this dataset.

On the other hand, a rapid rise in prequential accuracy is observed at the at the

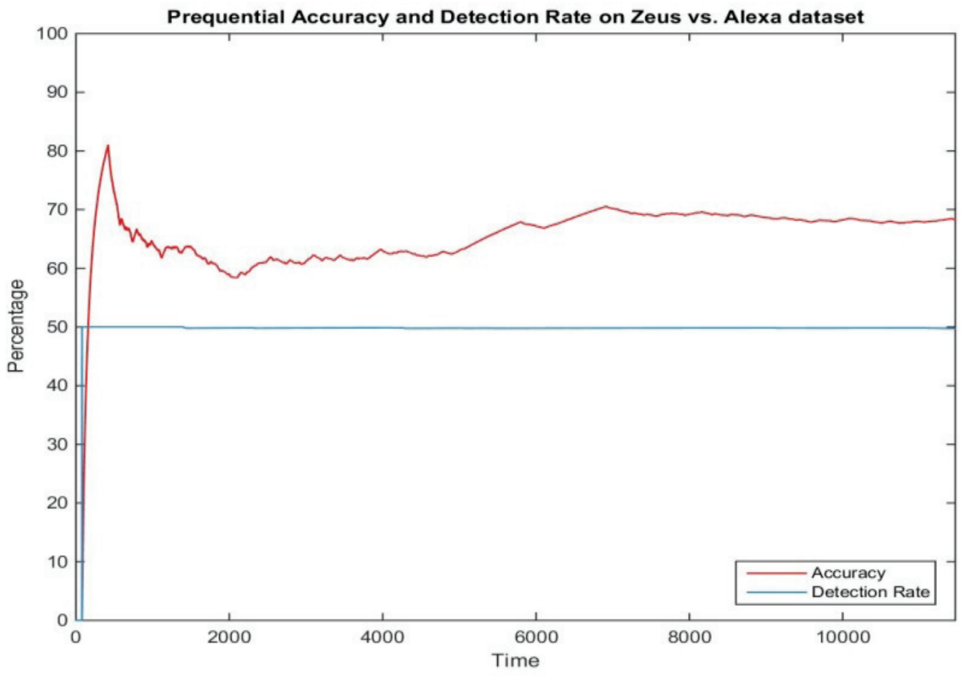

Figure 4.49: Prequential accuracy vs. prequential detection rate on the Zeus and Alexa dataset.

beginning, followed by either a drop or a steady state for the rest of the stream in most of the datasets (Figures 4.49, 4.50, 4.51, 4.53, 4.54 and 4.55). This seems to indicate that the learning algorithm is not able to detect drifts in the behaviours.

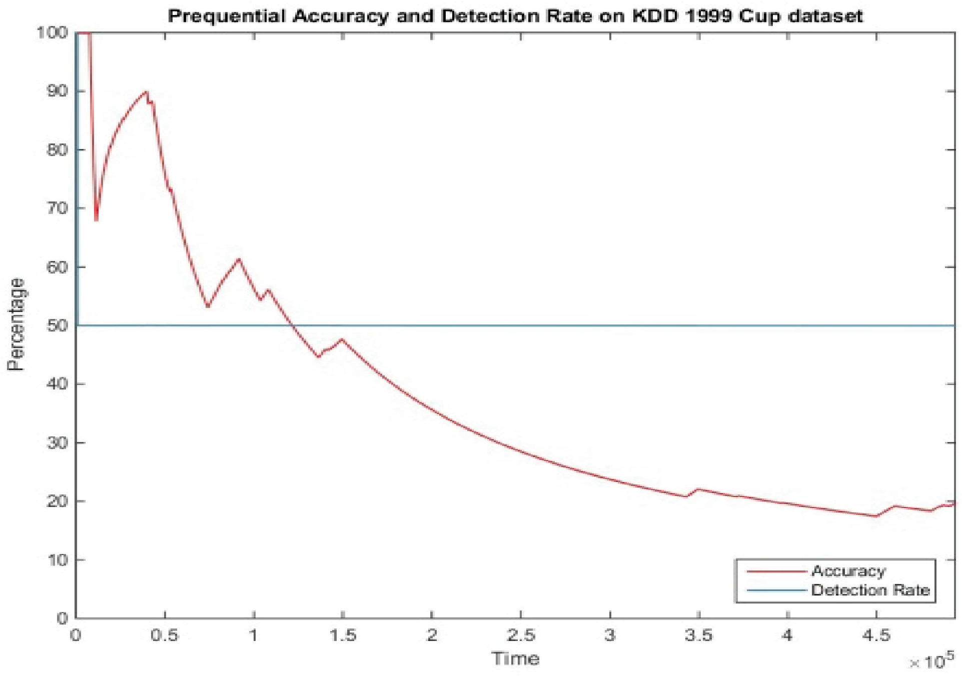

Figure 4.50: Prequential accuracy vs. prequential detection rate on the KDD 1999 Cup dataset.

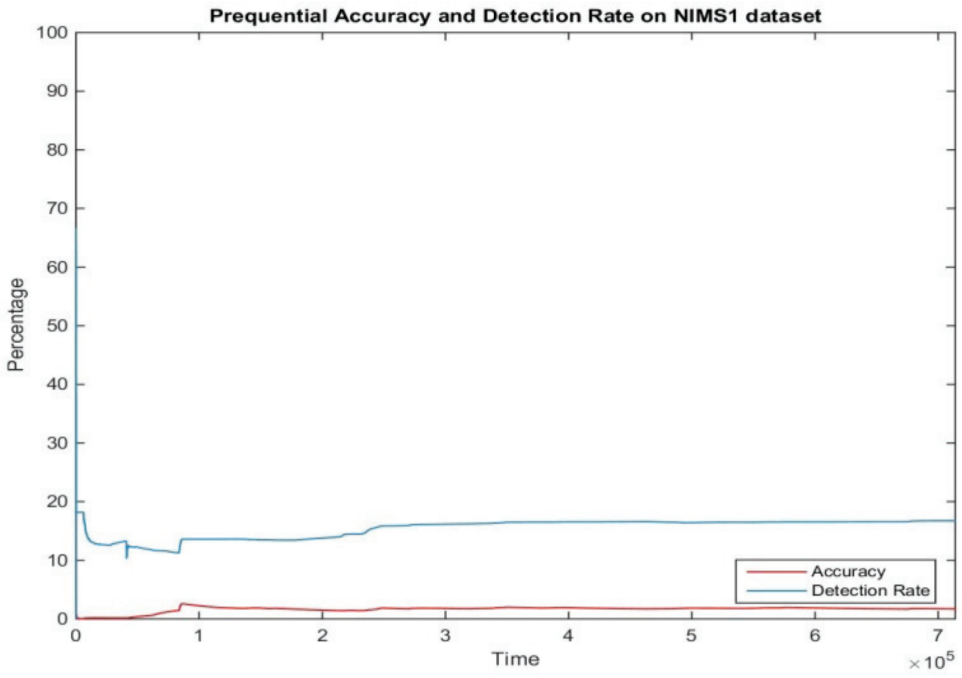

Figure 4.51: Prequential accuracy vs. prequential detection rate on the NIMS1 dataset.

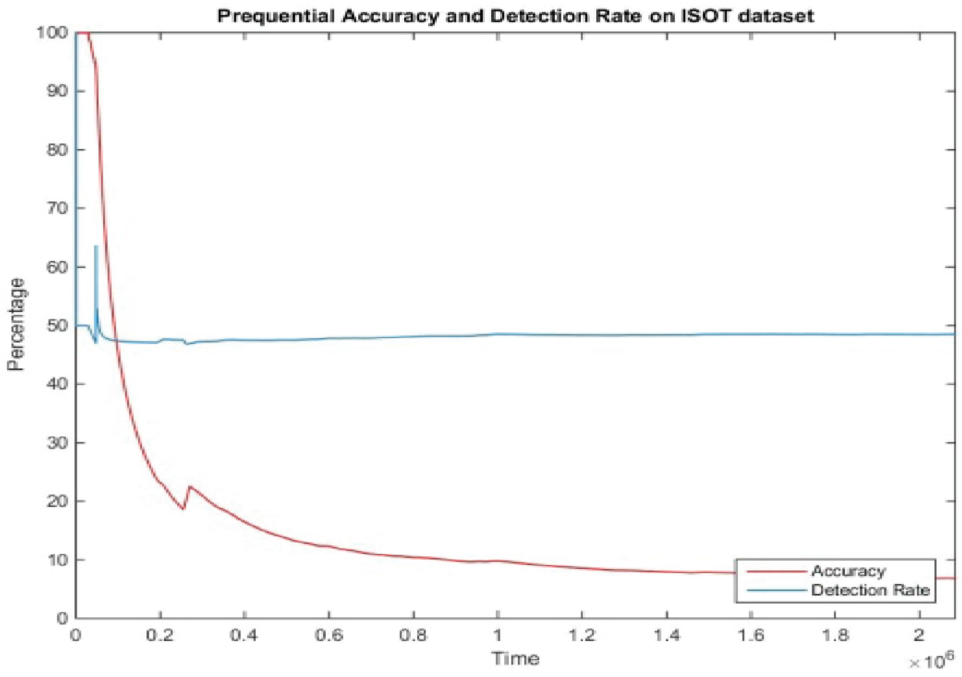

Figure 4.52: Prequential accuracy vs. prequential detection rate on the ISOT dataset.

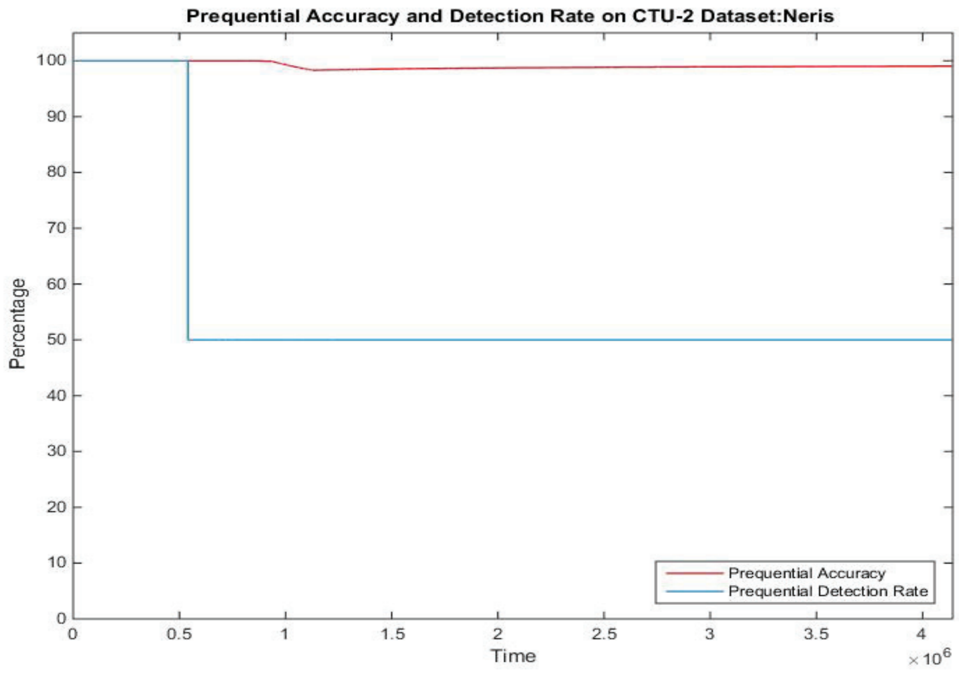

Figure 4.53: Prequential accuracy vs. prequential detection rate on the CTU-2 Dataset:Neris.

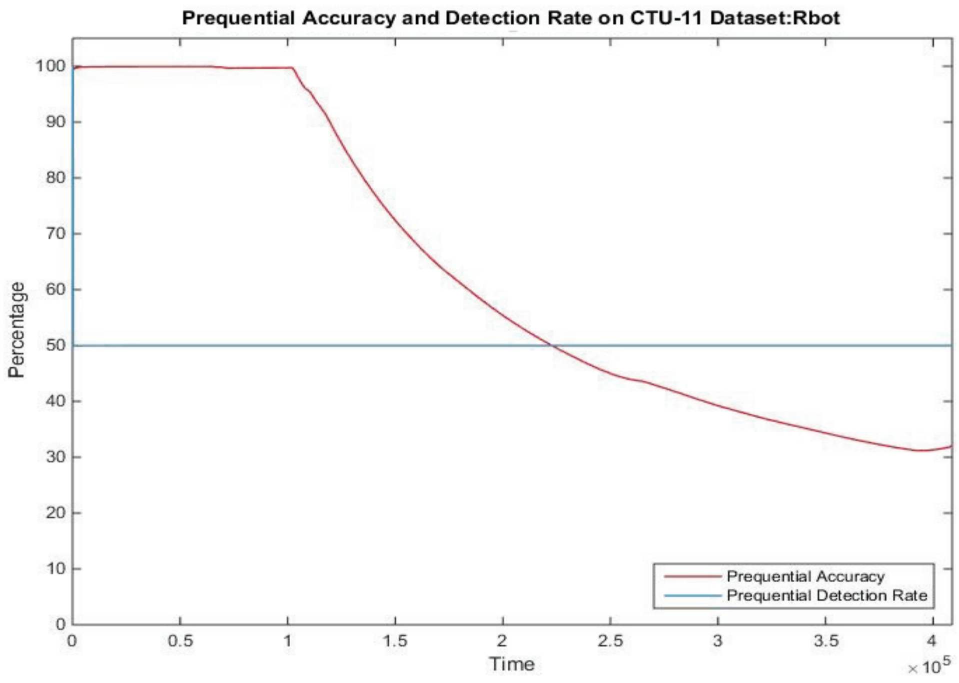

Figure 4.54: Prequential accuracy vs. prequential detection rate on the CTU-11 Dataset:Rbot.

Chapter 5

Conclusion

In this research, I study how to classify (analyze) streaming network traffic using different machine learning algorithms under different training (budgeting) strategies. As mentioned previously, this is an important task as it allows for one to detect malicious behaviour as it occurs among streaming network traffic in order to prevent further damage from the malicious activity to occur. Furthermore, by introducing budgeting strategies and active learning, the amount of resources needed to achieve the successful detection of malicious activity is reduced greatly. To achieve the aforementioned tasks, I analyzed the traffic using flow type features with Adaptive Artificial Neural Network, Naive Bayes and Hoeffding Tree stream classifiers under 10% and 100% training scenarios with five different budgeting strategies to train. Furthermore, I evaluated the performance of the different combinations of these algorithms and strategies using both the standard accuracy and detection rate as well as the prequential accuracy and detection rate. The evaluations show that all the tested budgeting strategies perform relatively similarly (with the exclusion of the fixed uncertainty strategy) on the network datasets employed regardless of the number of different classes in the datasets (NIMS1-application versus KDD Cup 1999, ISOT, Alexa vs. Zeus, CTU-2, and CTU-11 datasets). The results are generally quite high, averaging in the 90%'s. This indicates that any of these strategies could be used successfully in classifying network traffic and detecting malicious activity.

Furthermore, it is observed that changing the budget to 10% does not affect the performance of the selected strategies negatively, and can actually increase the performance of a given strategy in some cases.

When comparing these results to the adaptive Artificial Neural Network, it is observed that this method is not effective at classifying malicious activity among streamed network traffic. Thus it is recommended that an active learning approach is used instead.

Although the Hoeffding Tree algorithm has temporal dependencies, its use did not seem to have a large impact in terms of the performance on datasets, which featured more temporal variables. In fact, the Naive Bayes algorithm, which does not have temporal dependencies, performed similarly to the Hoeffding Tree algorithm in most cases with the exception of the NIMS and CTU-2 datasets where the Hoeffding Tree algorithm achieves a higher accuracy but a lower detection rate.

Future research would explore these strategies on more datasets as this will give a better idea how these methods would perform under other real world scenarios. Additionally, it would be interesting to look other budgeting strategies evaluated on such datasets since in this work only two different budgets are studied. This would allow for one to investigate what is the most effective labelling budget when factoring in both the cost and the performance. Due to results being very similar between strategies, future research should also include statistical analysis of results to determine how significant of a performance difference there is between results. Furthermore, it would be of interest to investigate the complexity of the trees generated by the Hoeffding tree as well as exact processing times of the learning algorithms to gain a better idea of the efficiency of their use. Finally, even though the performance of the Adaptive Artificial Neural Network used in this research was very low, it would still be valuable to conduct a parameter sensitivity analysis using this method.

Bibliography

[1] Abuse: Zeus tracker. [Online] https://zeustracker.abuse.ch/.

[2] Alexa. [Online] http://alexa.com/topsites.

[3] Dns-bh-malware domain blocklist. [Online] http://www.malwaredomains.com.

[4] ebot the ruby irc bot. [Online] http://www.darkreading.com/the-worlds-biggest-botnets-/d/d-id/1129117?

[5] Isot botnet data. [Online] http://www.uvic.ca/engineering/ece/isot/datasets/.

[6] Kdd cup 1999 data. [Online] http://kdd.ics.uci.edu/databases/kddcup99/ kddcup99.html.

[7] Kdd cup 1999 task description. [Online] http://kdd.ics.uci.edu/databases/kddcup99/task.html.

[8] Moa (massive online analysis). [Online] http://moa.cms.waikato.ac.nz/.

[9] Nims1 dataset. [Online] https://projects.cs.dal.ca/projectx/Download.html.

[10] The ctu-13 dataset. a labelled dataset with botnet, normal and background traffic. [Online] http://mcfp.weebly.com/the-ctu-13-dataset-a-labeled-dataset-with-botnet-normal-and-background-traffic.html.

[11] The ctu-13 dataset. a labelled dataset with botnet, normal and background traffic. [Online] http://ruby-rbot.org/.

[12] Nicolo Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. Worst-case analysis of selective sampling for linear classification. In *Journal of Machine Learning Research*, volume 7, pages 1205–1230, 2006.

[13] B. Claise. Specification of the ip flow information export (ipfix) protocol for the exchange of ip traffic flow information. rfc 5101. 2008. [Online] http://www.rfc-editor.org/info/rfc5101.

[14] C.A. Cunha and L.M.E Silva. Separating performance anomalies from workload-explained failures in streaming servers. In *Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on Cluster*, pages 292–299. IEEE, 2012.

[15] A. Csizmar Dalai, D.R. Musicant, J. Olson, B. McMenamy, S. Benzaid, B. Kazez, and E. Bolan. Predicting user-perceived quality ratings from streaming media data. In *2007. IEEE International Conference on Communications*, pages 65–72. ieee, 2007.

[16] A.C. Dalal, A.K. Bouchard, S. Cantor, Y. Guo, and A. Johnson. Assessing qoe of on-demand tcp video streams in real time. In *2012 IEEE International Conference on Communications (ICC)*, pages 1165–1170. IEEE, 2012.

[17] M. Feily, A. Shahrestani, and S. Ramadass. A survey of botnet and botnet detection. In *2009 Third International Conference on Emerging Security Information, Systems and Technologies*, pages 268–273. IEEE, 2009.

[18] S. Garcia, M. Grill, H. Stiborek, and A. Zunino. An empirical comparison of botnet detection methods. In *Computers and Security Journal, Elsevier*, volume 45, pages 100–123. ACM, 2014.

[19] F. Haddadi, J. Morgan, E.G. Filho, and A.N. Zincir-Heywood. Botnet behaviour analysis using ip flows: with http filters using classifiers. In *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, pages 7–12. IEEE, 2014.

[20] S. Hoelinger and R. Pears. Use of hoeffding trees in concept based data stream mining. In *Third International Conference on Information and Automation for Sustainability, 2007. ICIAFS 2007*, pages 57–62. IEEE, 2007.

[21] Han-Wei Hsiao, Deng-Neng Chen, and Tsung Ju Wu. Detecting hiding malicious website network traffic mining approach. In *2010 2nd International Conference on Education Technology Computer (ICETC)*, volume 5, pages 276–280. IEEE, 2010.

[22] G. H. Kayacik, A.N. Zincir-Heywood, and M.I. Heywood. On the capability of an som based intrusion detection system. In *IEEE International Joint Conference on Neural Networks*, pages 1808–1813, 2003.

[23] H. Gunes Kayacik, A.N. Zincir-Heywood, and M.I. Heywood. Selecting features for intrusion detection: A feature relevance analysis on kdd 99 intrusion detection datasets. 2006. [Online] https://web.cs.dal.ca/ zincir/bildiri/pst05-gnm.pdf.

[24] San-Min Liu and Zhi-Xin Sun. *Active learning for P2P traffic identification.* Springer US, 2014.

[25] Pratik Narang, Jagan Mohan Reddy, and Chittaranjan Hota. Feature selection for detection of peer-to-peer botnet traffic. In *Compute 13 Proceedings of the 6th ACM India Computing Convention.* ACM, 2013.

[26] A. Nogueira, P. Salvador, and F. Blessa. A botnet detection system based on neural networks. In *2010 Fifth International Conference on Digital Telecommunications*, pages 57–62. IEEE, 2010.

[27] S. Saad, I. Traore, A. Ghorbani, D. Zhao B. Sayed, J. Felix W. Lu, and P. Hakimian. Detecting p2p botnets through network behavior analysis and machine learning. In *Proceedings of 9th Annual Conference on Privacy, Security and Trust (PST2011)*, pages 174–180. IEEE, 2011.

[28] Alex Smola and S.V.N. Vishwanathan. *Naive Bayes*. Cambridge University Press, 2008.

[29] Frederic Stahl, Mohamed Medhat Gaber, and Manuel Martin Salvador. erules: A modular adaptive classifications rule learning algorithm for data streams. In *Research and Development in Intelligent Systems XXIX*, pages 65–78. Springer US, 2012.

[30] M. Stevanovic and J.M. Pedersen. An efficient flow-based botnet detection using supervised machine learning. In *2014 International Conference on Computing, Networking and Communications (ICNC)*, pages 797–801. IEEE, 2014.

[31] 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 *GECCO Comp 14 Proceedings in the 2014 conference companion on Genetic ad evolutionary computation companion*, pages 1287–1294. ACM, 2014.

[32] Ran Wang, Sam Kwong, Degang Chen, and Qiang He. Fuzzy rough sets based uncertainty measuring for stream based active learning. In *2012 International Conference on MachineLearning and Cybernetics (ICMLC)*, volume 1, pages 282–288. IEEE, 2012.

[33] Xingquan Zhu, Peng Zhang, Xiaodong Lin, and Yong Shi. Active learning from stream data using optimal weight classifier ensemble. In *IEEE Transactions on Systems, Man, and Cybernetics,Part B: Cybernetics*, volume 40, pages 1607–1621. IEEE, 2010.

[34] I. Zliobaite, A. Bifet, B. Pfahringer, and G. Holmes. Active learning with drifting streaming data. In *IEEE Transactions on Neural Networks and Learning Systems*, volume 25, pages 27–39. IEEE, 2013.