Nova Southeastern University
NSUWorks

CEC Theses and Dissertations

College of Engineering and Computing

2016

Evaluation of Supervised Machine Learning for Classifying Video Traffic

Farrell R. Taylor
Nova Southeastern University, ftaylor@mitre.org

This document is a product of extensive research conducted at the Nova Southeastern University College of Engineering and Computing. For more information on research and degree programs at the NSU College of Engineering and Computing, please click here.

Follow this and additional works at: http://nsuworks.nova.edu/gscis_etd

Part of the Artificial Intelligence and Robotics Commons

Share Feedback About This Item

NSUWorks Citation

Farrell R. Taylor. 2016. *Evaluation of Supervised Machine Learning for Classifying Video Traffic*. Doctoral dissertation. Nova Southeastern University. Retrieved from NSUWorks, College of Engineering and Computing. (972) http://nsuworks.nova.edu/gscis_etd/972.

This Dissertation is brought to you by the College of Engineering and Computing at NSUWorks. It has been accepted for inclusion in CEC Theses and Dissertations by an authorized administrator of NSUWorks. For more information, please contact nsuworks@nova.edu.

Evaluation of Supervised Machine Learning for Classifying Video Traffic

by

Farrell Taylor

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

in

Information Systems

Graduate School of Computer and Information Sciences

Nova Southeastern University

2016

We hereby certify that this dissertation, submitted by Farrell Taylor, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

Sumitra Mukherjee, Ph.D. Chairperson of Dissertation Committee	Date
Michael J. Laszlo, Ph.D. Dissertation Committee Member	Date
Gregory E. Simco, Ph.D. Dissertation Committee Member	Date

Approved:

Ronald J. Chenail, Ph.D. Interim Dean, College of Engineering and Computing	Date

SIGNED ORIGINAL ON FILE AT NSU

An Abstract of a Dissertation Submitted to Nova Southeastern University
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Evaluation of Supervised Machine Learning for Classifying Video Traffic

by
Farrell Taylor
May 2016

Operational deployment of machine learning based classifiers in real-world networks has become an important area of research to support automated real-time quality of service decisions by Internet service providers (ISPs) and more generally, network administrators. As the Internet has evolved, multimedia applications, such as voice over Internet protocol (VoIP), gaming, and video streaming, have become commonplace. These traffic types are sensitive to network perturbations, e.g. jitter and delay. Automated quality of service (QoS) capabilities offer a degree of relief by prioritizing network traffic without human intervention; however, they rely on the integration of real-time traffic classification to identify applications. Accordingly, researchers have begun to explore various techniques to incorporate into real-world networks. One method that shows promise is the use of machine learning techniques trained on sub-flows – a small number of consecutive packets selected from different phases of the full application flow. Generally, research on machine learning classifiers was based on statistics derived from full traffic flows, which can limit their effectiveness (recall and precision) if partial data captures are encountered by the classifier. In real-world networks, partial data captures can be caused by unscheduled restarts/reboots of the classifier or data capture capabilities, network interruptions, or application errors. Research on the use of machine learning algorithms trained on sub-flows to classify VoIP and gaming traffic has shown promise, even when partial data captures are encountered. This research extends that work by applying machine learning algorithms trained on multiple sub-flows to classification of video streaming traffic.

Results from this research indicate that sub-flow classifiers have much higher and more consistent recall and precision than full flow classifiers when applied to video traffic. Moreover, the application of ensemble methods, specifically Bagging and adaptive boosting (AdaBoost) further improves recall and precision for sub-flow classifiers. Findings indicate sub-flow classifiers based on AdaBoost in combination with the C4.5 algorithm exhibited the best performance with the most consistent results for classification of video streaming traffic.

Acknowledgements

It has been a long journey to this point, filled with numerous challenges; however, looking back, the endeavor was well worth the effort. When I first entered the program, I was told that a PhD is a personal journey that should not be undertaken for career, financial gain, nor ego. This was sage advice.

Several friends have been supportive of my efforts to attain a PhD. Dr. Dennis Bauer, a dear friend and colleague who provided an emotional push to move me from the point of indecision to commitment. Also, Dr. Dolly Mastrangelo, who provided encouragement at a key point in this journey when I strongly considered stopping my pursuit of a PhD. I owe Dr. Mastrangelo a great deal of thanks. To Zachary Parker, who spent his time discussing and providing reviews on a subject that had very little interest to him, and simply being there when needed, thank you. I would like to thank Dr. Stephen Dinkle for his advice and mentorship from the beginning of my PhD program until its completion. Dr. Dinkel's reviews and advise throughout this process was instrumental to my success.

To my dissertation committee, Dr. Sumitra Mukherjee, Dr. Michael Laszlo and Dr. Gregory Simco, I would like to extend appreciation and gratitude for their guidance and support. A very special thanks to Dr. Sumitra Mukherjee. Dr. Mukherjee was and still is a motivating force for me undertaking and completing my dissertation. Dr. Mukherjee is simply one of the best professors, more importantly, "teachers", I have ever had, and as he's done for me, will continue to inspire other students to achieve.

Finally, I would like to thank my wife, Toni, who has supported me throughout this process. Many nights, weekends and in some cases mini-vacations where sacrificed for this work. I owe her a sincere debt of gratitude. And to my children; the greatest achievement I have ever made is being a father. Thank you Mackenzie, Elyse and Samantha for calling me "Dad".

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Background

Internet Protocol (IP) network traffic classification is a key objective of internet service providers (ISPs) and network administrators supporting decisions related to quality of service (QoS), security, traffic shaping and overall network management (Dainotti, Pescape, & Claffy, 2012; Nguyen & Armitage, 2008). Traffic classification is the practice of correlating network flows to the applications that generated them (Mu & Wu, 2011). Initially, IP traffic classification was accomplished through the examination of common characteristics of network packets such as IP address, well-known ports and payload inspection (Karagiannis, Papagiannaki, & Faloutsos, 2005). Well-known ports were the preeminent means of identifying traffic (i.e. traffic classification) based on the Internet Assigned Numbers Authority (IANA) application port registration and were integrated into network monitoring tools such as NetFlow and sflow (Zander, Nguyen, & Armitage, 2005). Payload inspection, also referred to as deep-packet inspection, was a complementary technique, based on content analysis of the data portion of an IP packet (Bernaille, Teixeira, Akodkenou, Soule, & Salamatian, 2006). Both methodologies produced early success in classifying network flows to the applications that originated the traffic (Bernaille et al., 2006; Moore & Papagiannaki, 2005).

Although techniques based on well-known ports and payload inspection realized a level of success, today’s network applications, especially peer to peer (P2P), have become more sophisticated and the reliance on these characteristics to identify specific application protocols is suspect (Soysal & Schmidt, 2010; Yuan, Li, Guan, & Xu, 2010).

P2P applications (e.g. gaming, video streaming, voice over IP (VoIP)) may use a variety of ports to communicate between end user devices and servers, and payload inspection can be computationally expensive, infringe on privacy laws by revealing user content and could be rendered ineffective if encryption is used (Karagiannis, Broido, Faloutsos, & claffy, 2004; Yibo, Dawei, & Luoshi, 2013). Moreover, users have begun to purposely evade detection using encryption, tunnels, and ephemeral ports (Karagiannis et al., 2004).

To address deficiencies associated with using port and payload inspection for traffic identification researchers have applied machine learning techniques – based on network flow statistics – to support classification of IP traffic (Callado et al., 2009; Zander et al., 2005). Generally, a flow is defined by a sequence of five-tuples: source IP, destination IP, source port, destination port, and protocol (Dainotti et al., 2012; Hu, Chiu, & Lui, 2009). Overall results have been promising; however, several research worthy areas remain; in particular, research on the operational deployment of classifiers in real-world networks to identify P2P interactive traffic (Li, Springer, Bebis, & Hadi Gunes, 2013; Nguyen & Armitage, 2008). Deploying classifiers into real-world networks is a key aspect of automating QoS decisions to enable immediate, without the need for human intervention, reprioritization of network traffic to support real-time Internet applications (McGregor, Hall, Lorier, & Brunskill, 2004).

Operational deployment of machine learning (ML) based classifiers have several challenges: timely and continuous classification, directional neutrality, efficient use of memory, portability and robustness (Nguyen & Armitage, 2008). Nguyen, Armitage, Branch, and Zander (2012) developed a means to address a key challenge associated with real-time classifiers, specifically, the challenges associated with timeliness and continuous classification of traffic flows. Nguyen et al. (2012) methodology uses sub-flows – fragments of full traffic flows containing some number of contiguous packets – for identification of IP flows that addressed timeliness and continuous classification challenges. Prior to this work, the majority of the research on IP traffic classification used statistics derived from the entire traffic flow (Nguyen & Armitage, 2006). However, real-time classifiers may encounter partial, incomplete, traffic flows for a number of reasons: unscheduled shutdown/reboots of packet capture capabilities, network interruptions, or application errors (Nguyen & Armitage, 2006). Nguyen et al. (2012) found that classifiers trained on statistics from full flows, and used to identify flows from partial, incomplete network traffic captures where initial packets are missing, exhibited degraded performance in terms of recall and precision. Conversely, classifiers trained on multiple sub-flows across the entire life of the application performed well -- better than 95% for both recall and precision – even if the data being analyzed did not represent complete captures of the entire application session. Additionally, sub-flows represent a small portion of the entire flow of traffic, consequently less processing is needed to generate flow statistics, train, and perform classification of the target network traffic. Although Nguyen et al. (2012) were successful in applying this methodology, their work focused on the identification of two specific applications: Wolfenstein: Enemy Territory and VoIP. This research extends (Nguyen et al., 2012) work by evaluating the performance, in terms of recall and precision, of supervised machine learning algorithms trained on sub-flows in identifying video streaming traffic (i.e. YouTube and Netflix).

Problem Statement

Deployment of traffic classifiers in real-world networks has several challenges: timely and continuous classification, directional neutrality, efficient use of memory, and portability and robustness (Nguyen & Armitage, 2008). Of particular interest to this research is the challenge associated with *timely and continuous* classification of IP traffic. “A *timely* classifier should reach its decision using as few packets as possible from each flow (rather than waiting until each flow completes before reaching a decision)” (Nguyen & Armitage, 2008, p. 63). Additionally, it is not adequate to require the beginning packets of a traffic flow to produce high recall and precision– good classifier performance. In reality, network flows captured from real-world networks may be incomplete, due to unscheduled restarts of monitoring capabilities, network interruption, or application errors (Nguyen & Armitage, 2006; Nguyen et al., 2012; Zander, Nguyen, & Armitage, 2012). Moreover, packet statistics may change over the lifetime of an application’s flow, e.g. initial client server negotiation vice established connection between client and server. Accordingly, classifiers must be able to *continuously classify* traffic throughout the lifetime of the application’s flow (Nguyen & Armitage, 2008).

The problem studied for this research effort is the timely and continuous classification of video streaming traffic using ML based classifiers trained on multiple sub-flows, when partial, incomplete data sets are encountered.

Dissertation Goal

The goal of this research is to evaluate the effectiveness, specifically recall and precision, of ML techniques trained on sub-flows to classify video streaming traffic. Three ML algorithms are used – C4.5, Naïve Bayes, and Support Vector Machine (SVM)

– to address this goal. C4.5 and Naïve Bayes were used as part of the original work by Nguyen et al. (2012) and Nguyen and Armitage (2006) with good results; thusly, these methods are expected to be well suited to support this research effort. SVM has also been applied successfully in previous work for classification of network traffic (Este, Gringoli, & Salgarelli, 2009; Yuan et al., 2010). Additionally, ensemble techniques were considered, combining the outputs of each ML algorithm in order to enhance the performance of any single classifier (Dong & Han, 2005; Jianli & Yuncai, 2012). This research effort expands knowledge on using ML techniques to classify IP network traffic toward enabling the timely and continuous classification in real-world network environments.

Research Questions

This research answers the following questions:

1) What recall and precision can be attained using ML algorithms trained on multiple sub-flows in classifying video streaming traffic?
2) What sub-flow sized is needed to train, test and classify video traffic to attain high recall and precision?
3) What features, sub-flow attributes, are required to enable classification of video traffic?
4) What is the effect of different sub-flow sizes, number of packets per sub-flow, on ML recall and precision?
5) How effective are ML algorithms trained on multiple sub-flows in classifying video streaming traffic from disparate data sets containing packets captured from different network environments?

Relevance and Significance

In the early days of the Internet, data was transmitted on the basis of best effort (Xipeng & Ni, 1999). Nowadays, the Internet has become a platform for provisioning complex multimedia application services such as online gaming, e-commerce, video (streaming and interactive), VoIP, Internet radio, and large-scale file sharing (Roughan, Sen, Spatscheck, & Duffield, 2004). Additionally, with the advent of mobile devices, which ushered in the era of ubiquitous network access, the Internet has seen exponential growth (Roughan et al., 2004). “At the current pace of growth, Internet traffic is doubling approximately every two years, leading to a factor of 1000 growth in the next two decades” (Saleh & Simmons, 2011, p. 132).

As demand for Internet services has steadily increased, so has ISPs desire for detailed understanding of the various applications traversing their networks to support real-time network management (Jin et al., 2012). Content providers, understanding the importance of provisioning high-quality application services, are keenly interested in assured services to support a competitive advantage in their respective markets (Meddeb, 2010). The confluence of these challenges has provided ISPs with a new business opportunity where differentiated services, in the form of QoS guarantees, can be offered individualistically at varying price-points leading to new sources of revenue (Meddeb, 2010). Moreover, given the open nature of the Internet, a variety of legitimate and malicious users exist. ISPs and content providers are examining various technologies to support both QoS requirements and security (Saleh & Simmons, 2011). “In order to prioritize, protect, or prevent certain traffic, providers need to implement technology for traffic classification: associating traffic flows with the applications — or application types — that generated them" (Dainotti et al., 2012, p. 35). As such, research on traffic classification methodologies has steadily grown over the past decade (Li et al., 2013). Both offline forensic analysis, and more recently, online, real-time capabilities have been explored to support QoS and security.

Although offline traffic classification has shown good results, the need for real-time traffic classification for deployment in real-world networks is critical to make timely decisions regarding network management, particularly as it relates to automated QoS capabilities that prioritize IP traffic (Li et al., 2013; Roughan et al., 2004). Network administrators need to make decisions on QoS well before the flow of traffic has completed (Nguyen & Armitage, 2006, 2008; Nguyen et al., 2012). This is especially true for applications that are sensitive to jitter and delay such as VoIP and video (Dehghani, Movahhedinia, Khayyambashi, & Kianian, 2010).

Security also motivates the need for deployment of traffic classification in operational networks. In terms of security, IP classification can be used to support lawful intercept based on malicious traffic that is linked to systems and users (Baker, Foster, & Sharp, 2004). Anomaly detection and Botnet detection are other areas where IP classification can be used to identify inconsistencies in traffic patterns that may be indicative of malware on end user systems (Feily, Shahrestani, & Ramadass, 2009). Security administrators can also use these techniques to profile traffic between clients and servers on the network in order to make decisions on bandwidth allocation and to block illicit traffic (Hu et al., 2009; Zhao et al., 2013).

Based on these drivers, operational deployment of machine learning base IP network classifiers has become a meaningful area of research.

Barriers and Issues

Several barriers and issues affected this research effort. First, the acquisition of the appropriate data was required for this research; second, selections of the right number of sub-flows and associated features was challenging; third, selection of a suitable ensemble techniques toward enhancing recall and precision of individual classifiers was not straight forward; and finally, the robustness of the classifier as it relates to disparate data sets was a challenge that needed to be addressed.

- Acquiring the Right Data – Although there are publicly accessible data sets, it was difficult to acquire traces of the right applications, such as Netflix or YouTube traces, to support this work. Additionally, lab generated traffic may not be as realistic since the traffic may be so well contained within a segment of the network that classifiers trained on this type of data set may not be generalizable to traffic from an entirely different network. Some congruence between benchmark and lab generated data must exist to support the generalizability of the ML based classifiers. Additionally, it was important that the labeled training data sets represent ground truth, i.e. the label on the traffic flows are truly correct.
- Sub-flow and Feature Selection – Selecting the optimum sub-flows and associated features was challenging. Video traffic data did not exhibit sufficient differences across entire network flows to generate clusters of sub-flows and features to alleviate the need for manually inspection of the data set. Accordingly, examination of training and test datasets manually as well as

repetitive preliminary experimentation was needed to select features used to train and test classifiers for experimentation.

- Applying Ensemble Techniques – Based on this research, selection of an ensemble technique that is most suitable for enhancing the ML classifiers used in this research will be a key goal (Fern & Givan, 2003). Although, ensemble techniques may not be appropriate to support optimizing the classifiers used in this work.
- Robustness of the ML Classifiers – Robustness within the context of this research refers to the generalizability of the classifier. Although the use of lab captured data from different networks was be used, this may not fully validate classifiers robustness across all network environments. In all cases, data used in this research was captured from real networks and was not artificially generated.

Definition of Terms

Table 1 Definition of Terms

Term	**Definition**
Machine Learning	A discipline within the field of artificial intelligence concerned with the use of algorithms that allow computers to learn (improve their performance) based on previous experience, in the form of data, to address a specified task (Abu-Mostafa, Magdon-Ismail, & Lin, 2012; Flach, 2012; Mitchell, 1997).

Instance/Observation	Instance or Observation, within the context of this paper, is synonymous and refers to a tuple of attributes for an individual data point within a given input dataset.
Attribute/Feature	For this research, attribute and feature are synonymous and refer to one or more measured characteristics of an instance of the input dataset.
Traffic Classification	Describes the process of correlating network traffic to its associated protocol or application (Mu & Wu, 2011).
Flows	Refers to a five-tuples: source IP, destination IP, source port, destination port, and protocol of network traffic (Dainotti et al., 2012).
Sub-flow	A fragment of “n” contiguous packets of a particular traffic flow (Nguyen & Armitage, 2006).
Quality of Service (QoS)	Relates to the prioritization of specific network traffic types.
Discriminative Learning	Discriminative algorithms estimate the direct posterior probability between the input vector X, and a target class Y, $P(Y\|X)$, without any understanding of any of the underlying probability distributions that may exist (Ng & Jordan, 2002).

Generative Learning	Generative algorithms model the joint conditional probability distribution between the target class Y and the input vector X, succinctly $P(X, Y)$, accounting for the underlying probabilities, likelihood and prior probability of the target class (Ng & Jordan, 2002)
Information Gain	Information gain measures the relative importance of an individual attribute for classification of an instance (Quinlan, 1986).
Entropy	Entropy, within the context of information theory, is a measure of impurity or uncertainty of a given dataset (Mitchell, 1997).

Summary

As the Internet expands to support growing demands for P2P traffic, social media, online commerce, and gaming, the need to control, secure, and proactively manage network traffic, will increase accordingly. Consequently, traffic classification based on machine learning has become an important area of research with an emphasis on real world application of these techniques. This research is focused on supporting these goals by addressing gaps associated with timeliness and continuous classification of video traffic. In the following section, literature related to this effort and a description of machine learning algorithms used to pursue the objectives of this research is provided.

Chapter 2

Review of the literature

Introduction

There are two main themes of this chapter: a discussion of related research literature on the use of ML techniques for classifying IP traffic and a discussion of the specific supervised ML algorithms used for this research effort. Although not exhaustive, the review of literature related to IP classification is focused on the use of both supervised and unsupervised methods; albeit, the emphasis was on supervised efforts, which is the predominant type of ML algorithm used and the primary focus of this research. The ML algorithms that are discussed in the latter segment of this chapter include C4.5, Naïve Bayes and Support Vector Machines. Finally, ensemble techniques, specifically bagging and boosting, are also be detailed.

Initial Approaches to IP Traffic Classification

Early incarnations of application classification were based on well-known port and payload inspection. One of the initial works detailing the use of port numbers for application classification was performed by Schneider (1996). Schneider (1996) proposed the use of well-known port numbers registered in IANA. Ports below 1024 are documented in the registry in terms of the applications that use them; although, not required, the Request for Comment (RFC) 4632 also lists the use of ports beyond 1024 for convenience (Reynolds, Postel, & Group, 1994; Schneider, 1996). While Schneider (1996) stated the benefits of using well-known ports, the paper also recommended the use of additional traffic characteristics, especially in the case of ports above 1024, where port registration was not required by the RFC.

Another means of classifying network traffic was based on packet inspection. Sen, Spatscheck, and Wang (2004) evaluated the use of deep packet inspection to determine application signatures for reliable and accurate identification of applications traffic flows. Sen et al. (2004) work proved that packet inspection had advantages over port based classification with false positive and negative rates below 5%; however, with the advent of encryption and the increased density and diversity of traffic across the Internet, the benefits of deep packet inspection became computationally costly when compared to the use of flow statistics (Li et al., 2013; Raineri & Verticale, 2009).

IP Classification using Unsupervised ML

Nearly two decades ago Cisco patented NetFlow – a capability to derive statistical information on network traffic flows (Li et al., 2013). Since that time, research has evolved to leverage network flow statistics for a variety of activities such as application identification, host/user profiling, anomaly detection, and intrusion detection (Li et al., 2013). McGregor et al. (2004) were early adopters of flow statistics to support IP classification. McGregor et al. (2004) used unsupervised machine learning techniques, in particular expectation maximization (EM), for coarse grain clustering of traffic flows. Although McGregor et al. (2004) work was effective, specific identification of traffic was not possible; nevertheless, McGregor et al. (2004) research gave insight into the use of flow statistics for probability clustering. Another unsupervised approach, termed Autoclass, used a Bayesian classifier pioneered by Zander et al. (2005) for traffic classification. Using Autoclass, better results were realized in terms of clustering applications; although the authors stated that some clusters contained multiple application flows, which could not be discerned by this method. As a follow-on to Zander et al. (2005), Erman, Arlitt, and Mahanti (2006) compared the performance of Autoclass to two other clustering algorithms, K-Means and density-based spatial clustering of applications with noise (DBSCAN). Results indicated that both K-Means and DBSCAN had significantly lower classifier build time than Autoclass, while Autoclass had the best overall accuracy. The small difference in accuracy of Autoclass over DBSCAN and K-Means was offset by the latter two algorithms' ability to generate small, tight clusters, indicating the overall classification power for identifying unlabeled instances. K-Means was also used by Grimaudo, Mellia, Baralis, and Keralapura (2014) to develop a self-learning unsupervised classifier named SeLeCT. SeLeCT used an iterative approach to increase the fidelity of clustering ML techniques, specifically, pure clusters. Results from Grimaudo et al. (2014) indicated that SeLeCT could semi-automatically classify traffic, with the use of seed data derived from filtering previously identified traffic flows. Moreover, in combination with supervised methods, SeLeCT's iterative and adaptive process generated homogenous cluster that predominantly contain only a single traffic flow. Although clustering techniques show promise, sole use of these techniques to support on-line traffic classification still presents challenges given the requirement to positively identify traffic in real-world networks for decision-making purposes.

Clustering, or unsupervised techniques, are key foundational elements to support IP classification (Erman, Mahanti, Arlitt, Cohen, & Williamson, 2007; Marnerides, Schaeffer-Filho, & Mauthe, 2014). Initially, clustering was focused on crude groupings of similar traffic as a precursor for processing unlabeled data instances, however, clustering techniques has served as the basis for more sophisticated approaches to traffic classification that combine both supervised and unsupervised hybrid methods (Dainotti et al., 2012).

IP Classification using Supervised ML

Supervised methods have shown a great deal of promise and have become the predominant approach used for traffic classification (Nguyen & Armitage, 2008). Moore and Zuev (2005) used Naïve Bayes techniques to categorize network traffic. Unlike unsupervised methods, Moore and Zuev (2005) required training on traffic that was in some way, manually or otherwise, labeled with the correct application classification for each flow. In their work on classification of IP traffic using Naïve Bayes, Moore and Zuev (2005) showed that classification accuracy could be improved significantly (65 – 95% accuracy) by employing kernel density estimation to calculate required probability distributions and enhancing the quality of discriminators for the input data. Although their work did not address real-time classification, it provided insights on the use of Naïve Bayes in terms of its efficiency and accuracy for classifying IP flows. Este et al. (2009) adapted a SVM based algorithm to perform multi-class traffic categorization. In this work, Este et al. (2009) demonstrated both the usefulness of SVM as a multi-class traffic classification technique and its application to real-time traffic identification by only leveraging a small number of the first few packets of the application flow.

Soysal and Schmidt (2010) evaluated three ML algorithms, Bayesian Networks, decision trees, and multilayer perceptron, ability to classify six different types of P2P traffic. The key objective of this work was determining if ML based classifiers are affected by the amount and breadth of training data used. Furthermore, Soysal and

Schmidt (2010) evaluated the impact of incorrectly labeled training data on classifier performance. Soysal and Schmidt (2010) concluded that the amount of data processed by ML classifiers – in their case over one million flows – can have impact on accuracy of classification. Moreover, their results also strongly encouraged the use of correctly labeled instances to reduce error rates. An important aspect of Soysal and Schmidt (2010) work are the insights into real-world application of classifiers, in relationship to the amount of data used to train ML based classifiers. Another comparative analysis by Singh and Agrawal (2011) used five of ML algorithms, multilayer perceptron, radial basis function, C4.5 decision tree, Bayesian network, and Naïve Bayes. Each algorithm was exposed to approximately two minutes of Internet data, which constitutes a large and diverse sample set. Additionally, the feature set used was incrementally reduced to determine the effects on classifier performance. Results indicate that C4.5 and Bayesian network performed best. More importantly, the study called for further research to reduce the sample and feature size to make the ML algorithms more compatible with real-time classification problems.

In concert with the findings of Singh and Agrawal (2011), Singh, Agrawal, and Sohi (2013) researched the application of the same five ML algorithms to real-time IP traffic classification. In particular, their work refined the approach in described in Singh and Agrawal (2011) by capturing only two sec intervals of Internet traffic packets and deeply examining the elimination of attributes using feature selection algorithms. Results indicate that this approach effectively reduce training and classification time. Moreover, there was a strong dependency between the reduction of sample data and feature space in relation to classifier suitability to near real-time implementation of classifiers (Singh et al., 2013). As to the efficacy of the various ML algorithms, Bayesian network proved to be most effective within the context of the research methodology used.

IP Classification using Semi-Supervised (Hybrid) ML

Hybrid solutions have also shown some promise in terms IP classification, where both unsupervised and supervised methods are combined. Erman et al. (2007) used labeled training data to perform classification and clustering to aggregate traffic that was unknown (not labeled). This combination allowed for a more robust capability that could react to both known and unknown application traffic. Shrivastav and Tiwari (2010) research used a similar thesis; however, clustering was used first on traffic data, then the traffic was labeled, and finally the labeled data was used to train supervised classification algorithms. Callado, Kelner, Sadok, Alberto Kamienski, and Fernandes (2010) combined the output of multiple supervised machine learning techniques, e.g. Naïve Bayes, J48, SVM and others, in different ways as an approach to improve classification of IP traffic. Multiple algorithms were applied to the output of the classifiers, such as random selection of classifier’s outputs, maximum likelihood, Dempster-Shafer theory, and an enhanced version of Dempster-Shafer (Callado et al., 2010). Follow-on work was recommended to understand the optimal combination of machine learning techniques along with other combinatorial methods for aggregating the output of multiple algorithms to improve classification recall and precision.

Operationalizing ML Classifiers

While the offline research on unsupervised and supervised ML classifiers has shown significant progress, the need to operationally deploy classifiers in real world networks has grown (de A Ribeiro, Filho, & Maia, 2011; Nguyen et al., 2012). As the Internet evolves, the growth in online multimedia traffic, gaming, interactive P2Ps, and video has driven the need for automated traffic management to ensure the quality of these services (Nguyen et al., 2012). Consequently, research on real-time deployment of ML classifiers has become an area of increased focus within the field of IP traffic classification (Dehghani et al., 2010; Nguyen & Armitage, 2008). One of the earlier efforts to address the challenges of real-time classification was undertaken by (Bernaille et al., 2006). The methodology proposed by Bernaille et al. (2006) relies on capturing the first few packets of network traffic and applying ML algorithms for classification. Though this method produced some level of success, the requirement to always capture the initial packets for target flows may not be reasonable in real-world environments. Haffner, Sen, Spatscheck, and Wang (2005) provided another approach to real-time traffic identification based on the use of ML classifiers to automatically recognize a target application by its payload signature. As is the case with Bernaille et al. (2006), Haffner et al. (2005) relies on capturing the initial packets of traffic flows.

Of particular interest to this work is Nguyen and Armitage (2006) research that devised a method using sub-flows to train ML algorithms and classify traffic. A sub-flow is a traffic flow fragment of some number of contiguous packets taken from an application's full flow (Nguyen & Armitage, 2008; Nguyen et al., 2012). Statistics from multiple sub-flows selected from various phases of the application's flow can be used to train the classifier (Nguyen et al., 2012). Once trained, the classifier can be used to examine traffic at any point in the traffic flow, irrespective of incomplete data captures.

Generally, the predominance of the work discussed in this section that uses unsupervised, supervised or hybrid methods relies on statistics from full traffic flows.

This presupposes that full flows can always be obtained, which may not always be the case (Nguyen et al., 2012). This research effort extended Nguyen et al. (2012) work by applying their methodology to classifying video streaming traffic. As such, the following sub-section provides a more in-depth discussion of the ML algorithms that were be used to pursue this research goal.

ML Techniques Applied in this Research

Machine Learning is a discipline within the field of artificial intelligence concerned with the use of algorithms that allow computers to learn based on previous experience, in the form of data, to perform a specified task (Abu-Mostafa et al., 2012; Flach, 2012). In general, there are three fundamental forms of machine learning: supervised, unsupervised, and reinforcement. Supervised learning entails learning from data that is labeled, i.e. a priori knowledge of the actual classification of the input data is known (Mitchell, 1997). Conversely, for unsupervised learning, no a priori knowledge of the class of the input data is provided; thus, the data is unlabeled and the ML algorithm must deduce natural groupings, clusters, without any insight of underlying patterns within the dataset (Mitchell, 1997). Reinforcement learning takes a different tack, whereby automated computational decision-making is performed through application of a reward system based on feedback from trial-and-error (Sutton & Barto, 1998). Since the primary focus of this work is supervised learning, the discussion that follows is scoped accordingly.

Data is the key element needed to apply ML algorithms to any given task (Abu-Mostafa et al., 2012). The learning process is based on previously gathered data, whether unlabeled or labeled, to support prediction of future outcomes, modeling of patterns in the form of natural clusters, or classification of new instances. Depending on the problem space, the input data may undergo some degree of preprocessing such as feature selection, generation of statistics, and formatting in order to use a particular ML algorithm (Abu-Mostafa et al., 2012). Furthermore, the input data may be separated into a training and validation set. Figure 1 provides a generalized depiction of machine learning along with some of the terms that are commonly used in this section.

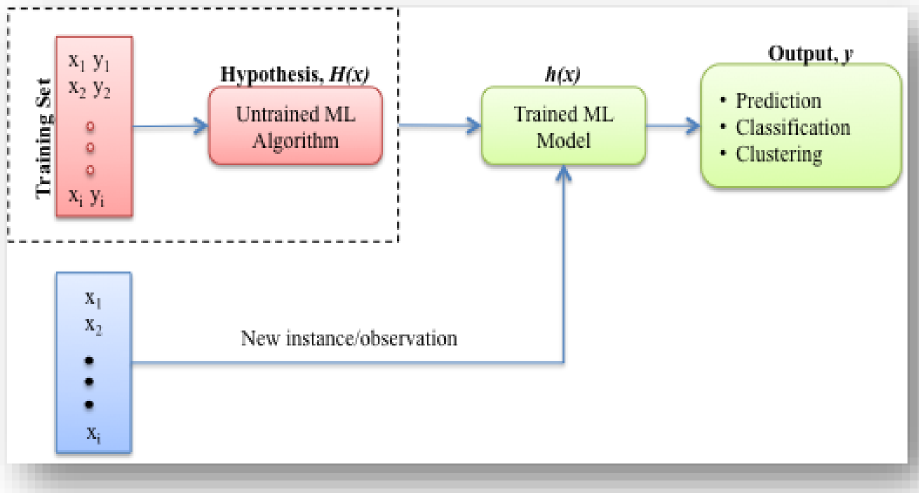


Figure 1 Generalized Depiction of Machine Learning

As the name implies, the training set is use to select the optimum hypothesis *h(x)*, from the space of hypothesis, *H(x)*. Succinctly, training data is used to build a model that can used to predict, cluster, or classify new instances. The hypothesis is in fact a function that maps the input vector *X* to an output *Y*; written formally, $F: X \rightarrow Y$. The function, *h(x)*, is representative of the particular ML algorithm used. In Figure 1 the input data has a single feature, *x*; however, in practice the input feature space may be very large, as in the case of classifying photos of common objects where a single picture may have 256x256 pixels. Selection of a particular ML algorithm, e.g. linear regression, logistic regression, perceptron, etc. is a function of the data, task, and the preference of the analyst.

Finally, the validation data set is use to evaluate the trained ML algorithm, *h(x)*. A well accepted method for evaluating the quality of a ML model is to measure recall and precision. Recall and precision are defined as follows:

- *Recall* represents the proportion of all the instances of a particular class that are correctly classified as that class (Blair & Maron, 1985; Flach, 2012; Hand, 2009). Concisely, did the classifier correctly classify all the instances of a particular class. To calculate recall the following expression is used:

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

- *Precision* represents the proportion of instances that were classified as a particular class that are actually classified correctly (Blair & Maron, 1985; Flach, 2012; Hand, 2009). In short, out of the instances classified, what percentage of them are correct. Precision is calculated using the following expression:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Both recall and precision are important for assessing classifier performance. If a classifier has high precision – indicating that the majority of observations classified were classified correctly – and the classifier failed to classify many of the target instances (i.e., poor recall), then the overall performance cannot be considered good. The converse is also true, where recall is high and precision is low. In the following section the three ML algorithms used in this research, SVM, Naïve Bayes, and C4.5, is described in more detail.

Support Vector Machine

Support Vector Machine (SVM) has become one of the most popular supervised ML algorithms and is applied to a wide range of tasks within the field of genetics, medical science, security, and network analysis (Burges, 1998). Although SVM is based on a linear classification model, its ability to be extended to tasks with high dimensional features, with a relatively small training set, has only widened its use across a variety of problem sets (Burges, 1998; Yuan et al., 2010). Moreover, SVM can be applied to binary, multi-class, and non-linear classification problems, while still maintaining a high degree of efficiency (Chang & Lin, 2011; Chih-Wei & Chih-Jen, 2002).

SVM is considered a large margin classifier since it constructs a hyperplane (decision boundary) that offers the greatest separation between the different classes of data under analysis (Muller, Mika, Ratsch, Tsuda, & Scholkopf, 2001; Tsochantaridis, Joachims, Hofmann, Altun, & Singer, 2005). Since the hyperplane has a large margin between positive and negative classes, SVM mitigates issues associated with misclassification of new unlabeled data instances; succinctly, the trained classifier is more generalizable to new instances of the data than a basic linear classification model (Smola & Schölkopf, 2004). In the following sub-section, a discussion of SVM, along with an overview of its mathematical underpinnings, is detailed initially from the perspective of a generic linear classification task, followed by an overview of a nonlinear case.

Linear SVM (LSVM)

LSVM is the most basic SVM model that supports binary classification of data into negative and positive classes, assuming the input data is linearly separable. For example, given a data set *D* defined by the following

$$D = \{(x_i, y_i) \mid \boldsymbol{x} \in \mathbb{R}^d, y \in \{1, -1\}\}, i = 1 \ldots n, \qquad (1)$$

where the vector $\boldsymbol{x}$ represents a set of scalar data points $x_1 \ldots x_n$ that can be used to train and test a function that maps the input data to the output y. The dependent variable y will be either 1 or -1 for positive and negative classes, respectively. Since SVM is a supervised learning algorithm, all training data instances were labeled with either a 1 or -1 when training the classifier. Furthermore, given this is a linear classification task, the SVM function to be trained with dataset *D* can be described by the following expression

$$y = h(x) = \boldsymbol{w} \cdot \boldsymbol{x} + b; \;\; y \in \{1, -1\} \qquad (2)$$

where $\boldsymbol{w}$ is the normal vector to the decision plane, $\boldsymbol{x}$ is the input vector, and b is the bias or offset. Additionally, equation 2 specifies the dot product of vector $\boldsymbol{w}$ and $\boldsymbol{x}$ which is defined as

$$\boldsymbol{w} \cdot \boldsymbol{x} = \sum_{i=1}^{n} w_i \, x_i \qquad (3)$$

To gain better intuition of the details regarding SVM, Figure 2, based on Flach (2012), is used as a reference for a generalized LSVM and the discussion that follows.

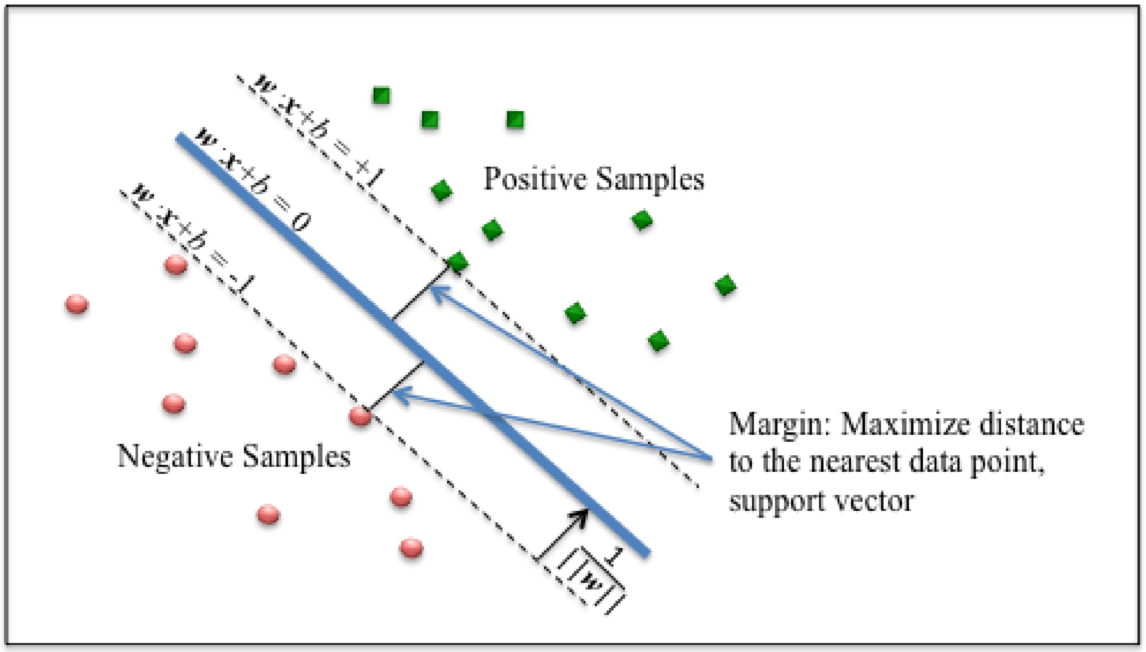


Figure 2 Linear Support Vector Machine

As depicted in Figure 2, the decision boundary hyperplane, is specified by

$$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0 \qquad (4)$$

and the maximum margin hyperplanes are defined by

$$\boldsymbol{w} \cdot \boldsymbol{x} + \mathrm{b} = 1 \;\; \boldsymbol{and} \;\; \boldsymbol{w} \cdot \boldsymbol{x} + b = -1 \qquad (5)$$

separating positive and negative values, respectively. Constructing the maximum margin hyperplanes (dashed lines) for both positive and negative classes is based on the data instances nearest to the decision boundary hyperplane, which are referred to as support vectors. The Euclidean distance from the maximum margin hyperplanes defined by equation (5) to the decision boundary hyperplane, equation (4), can be determined using the following $1/|(|\boldsymbol{w}|)|$, where ||***w***|| is the norm of the vector $\boldsymbol{w}$. Intuitively, minimizing ||***w***|| will maximize the distance between the nearest positive or negative sample to the decision boundary hyperplanes, which implies the following constraint optimization problem.

$$\min \ \frac{1}{2} \parallel w \parallel^2 \quad \text{subject to} \quad y_i(\boldsymbol{w} \cdot \boldsymbol{x} + b) \geq 1, \ \ \forall\, i, \ \ i \in \{1 \ldots n\} \qquad (6)$$

Extending LSVM

Two key limitations arise from the constraint optimization problem expressed in (6), its ability to deal with input data that is not linearly separable (non-linear feature space), as well as a high dimensional input vector space (Flach, 2012). In order to addresses these issues, the introduction of a soft margin constraint, Lagrange multiplier, and a Kernel function will be explored (Flach, 2012).

First, the addition of slack variables to the objective function and constraint in equation (6) will relax the constraint and introduce the concept of a soft margin (Tsochantaridis et al., 2005). The addition of slack variables allows some degree of misclassification, which assumes that the data may not perfectly satisfy the linear constraint that was imposed in equation (6). Concretely, if the input data is noisy or not linearly separable, then the constraint $y_i(\boldsymbol{w} \cdot \boldsymbol{x} + b) \geq 1$ will not be met. By applying a slack variable ξ to the constraint, some degree of margin violation is allowed, which begins the process of addressing non-linearly separable data (Tsochantaridis et al., 2005). Accordingly, slack variables are added to both the objective function and the constraint for the SVM. Moreover, a penalization parameter C is introduced to balance the effects of slack variables on the objective function. Therefore, equation (6) takes the form

$$\min \ \frac{1}{2} \parallel \boldsymbol{w} \parallel^2 + C \sum_{i=1}^{n} \xi_i$$

$$\text{subject to} \quad y_i(\boldsymbol{w} \cdot \boldsymbol{x} + b) \geq 1 - \xi_i \ and \ \ \xi_i \geq 0 \ \ \forall\, i, \ \ i \in \{1 \ldots n\} \qquad (7)$$

where the parameter C is used to minimize the effects of the sum of the slack variable ξ on the objective function.

By convention, a linear optimization problem of the form specified in (7) can be approached using Lagrange multiplier α to find the extrema of the objective function under the specified constraint (Cortes & Vapnik, 1995). Furthermore, by devising the dual form of the Lagrange function the SVM optimization problem, equation (7) can be expressed as follows,

$$\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \, (\boldsymbol{x_i} \cdot \boldsymbol{x_j})$$

$$\text{subject to } \sum_{i=1}^{n} \alpha_i y_i \text{ and } 0 \leq \alpha_i \leq C, i \in \{1 \ldots n\} \qquad (8)$$

Completing the process of making SVM applicable to non-linear problem sets requires the addition of Kernel methods to equation (8). Kernel methods are functions that can be applied to various ML algorithms to address non-linearity of input data and has proven to be well suited for SVM (Burges, 1998; Cortes & Vapnik, 1995; Howley & Madden, 2005). By replacing the dot product in the optimization in (8) with a Kernel function, the equation takes the form

$$\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \, K(\boldsymbol{x_i} \cdot \boldsymbol{x_j})$$

$$\text{subject to } \sum_{i=1}^{n} \alpha_i y_i \text{ and } 0 \leq \alpha_i \leq C, i \in \{1 \ldots n\} \qquad (9)$$

where $K(\boldsymbol{x_i} \cdot \boldsymbol{x_j})$ represents the application of a Kernel function to the SVM (Flach, 2012). This approach allows the algorithm to fit a non-linear input data set to a large margin hyperplane decision boundary in a high dimensional feature space. There are several Kernel functions that can be used to support this transformation; although, Gaussian Kernel is one of the more common methods used across a large spectrum of problem sets (Chang & Lin, 2011; Keerthi & Lin, 2003).

Finally, as stated previously, SVM can be applied to multi-class problem sets. For multi-class systems, the most rudimentary method used is the principle of one-against-all, whereby multiple SVM algorithms are independently trained to identify a particular class of the data, say red, blue or green, and then applied against new instances (Weston & Watkins, 1998). As expected, each classifier identifies the input data it was trained on for a given instance, providing the effect of a multi-class classifier system.

Naïve Bayes

In general, probabilistic ML algorithms can be characterized as either discriminative or generative. Discriminative algorithms estimate the direct posterior probability between the input vector X, and a target class Y, $P(Y|X)$, without any understanding of the underlying probability distributions that may exist (Ng & Jordan, 2002). Generative algorithms model the joint conditional probability distribution between the target class Y and the input vector X, succinctly $P(X,Y)$, accounting for the underlying probabilities, likelihood, and prior probability of the target class (Ng & Jordan, 2002). Although Naïve Bayes ML algorithms are comparatively less complex than other supervised learning models, it has been empirically proven to be effective across a variety of problem sets (Soria, Garibaldi, Ambrogi, Biganzoli, & Ellis, 2011).

From Bayes Rule to Naïve Bayes

Fundamentally, Naïve Bayes is simplified form of Bayes rule, with the inclusion of a key assumption that allows its practical application to ML tasks. Any discussion of Naïve Bayes, must begin with Bayes Rule, which is defined as

$$P(Y|X) = \frac{P(X|Y)\,P(Y)}{P(X)} \qquad (10)$$

where $P(Y|X)$ is the posterior joint conditional probability of class Y given the input X and is computed using the product of $P(X|Y)$, termed the likelihood, and the prior probability for the class Y, $P(Y)$ (Friedman, Geiger, & Goldszmidt, 1997; Lewis, 1998). The denominator, $P(X)$, is used to normalize the resulting posterior probability to a value less than or equal to 1.

To begin extending Bayes Rule to the Naïve Bayes algorithm, the focus is on maximizing $P(Y|X)$, as expressed by

$$Class\ of\ X = max\ P(Y|X) \qquad (11)$$

or stated more explicitly,

$$\text{Class of } X = \max \frac{P(X|Y)\ P(Y)}{P(X)} \qquad (12)$$

which indicates that the classification of X for a target class is a function of the largest joint posterior probability (Mitchell, 1997; Seeger, 2011). Understanding that X is a vector that is comprised of a set of features, $x_1 \dots x_i$, a key assumption can be introduced to simplify this formulation to reduce the complexity of calculating the likelihood when using data with a high dimensional feature space and a large number of samples. Specifically, it can be proposed that the likelihood value $P(X|Y)$ can be expressed as the combination of individual and independent probabilities of each input feature with respect to a given class, i.e. $P(x_1 \dots x_i|y_{j=target\ class})$. This postulation constitutes the "Naïve" assumption for Bayes Rule and is referred to as conditional independence (Koc, Mazzuchi, & Sarkani, 2012). Written generically,

$$P(x_1 \dots x_i|y_j) = P(x_1|y_j) \cdot\ P(x_2|y_j) \dots \cdot P(x_i|y_j) \qquad (13)$$

represents the product of the independent conditional probabilities of x given a class y. This significantly simplifies the calculation of $P(X|Y)$. Furthermore, the denominator for the Bayes Rule, $P(X)$, can be dropped since its value is constant for the entire input dataset (Mitchell, 1997). Consequently, *P(X)* does not affect the resultant joint posterior probability and is in accord with the assumption that each feature is conditionally independent across the entire feature set and sample space. Thus, the final form of the equation for Naïve Bayes can be expressed as follows

$$Class\ of\ X = \max\ P(Y|X) = P(y_j)\prod_{i=1}^{n} P(x_i|y_j) \tag{14}$$

where the class of a new observation is the product of independent likelihoods, multiplied by the prior probability $P(y_j)$ for a specified class.

Estimating Probability Distributions for Naïve Bayes

Generating the required probability distributions for the Naïve Bayes classifier can be performed using maximum likelihood estimates (McCallum & Nigam, 1998). Concretely, the training set is used to estimate $P(X|Y)$ and $P(Y)$ by examining relative frequencies for each class and attribute in the dataset. First, the probability of a class, y_j, within a given dataset can be estimated by the following

$$P(y_j) = \frac{|y_j|}{|D|} \tag{15}$$

where $|y_j|$ is the number of occurrences of a specific class normalized against the total number of instances, $|D|$; and to determine likelihood, the following formulation can be used

$$P(x_i|y_j) = \frac{\#x_i\ for\ class\ y_j}{\sum_{\forall\, v} \#x\ for\ class\ y_j} \tag{16}$$

where the numerator represents the frequency that the attribute x_i occurs for the specified class, y_j, normalized by the count of all of attributes within the training set that have the class y_j (McCallum & Nigam, 1998).

Since maximum likelihood is used to determine component probability distributions for Naïve Bayes, in real-world problems certain distributions of an instance's feature may be equal to zero for a given class. Simply stated, the training set may not have an occurrence of a particular attribute-class pair, while a new observation may in fact represent such an attribute-class relationship. Based on equation (14), which specifies the class of a new instance is a product of independent probability distributions, a zero probability can in effect lead to an unknown classification – zero for *P(Y|X)*. In order to address this issue, Laplace smoothing can be used (F. Peng, Schuurmans, & Wang, 2004). In its most basic form, Laplace smoothing can be implemented by adding one (add-one-smoothing) to both counts in equation (16) as follows:

$$P(x_i|y_j) = \frac{\#x_i\ for\ class\ y_j + 1}{\sum_{\forall\ v} \#x\ for\ class\ y_j + |D|} \qquad (17)$$

where the value $|D|$ is a more compact form for adding one to each occurrence of an attribute of class y_j. The result of add-one-smoothing is to ensure that missing attribute-class pairs in the training set do not impair the ability for the algorithm to classify new, unknown instances.

While the formulation of Naïve Bayes is based on a simplifying premise, it has exhibited excellent performance in terms of computation time and classification results despite the assumption of conditional independence (Rish, 2001; Yuguang & Lei, 2011). In fact, Naïve Bayes has become the de facto standard for text classification and sentiment analysis where it is used in conjunction with ensemble techniques (Rennie, 2001).

C4.5

One of the more practical inductive machine learning methods are decision trees (Safavian & Landgrebe, 1991). A decision tree represents a classification task as a structure containing a root, branches, and leafs. The root of the tree, which itself is an attribute (feature), is the starting point of the structure, with each associated branch representing a decision point based on testing the value of an attribute, and each leaf equating to a specific classification of the input data under analysis. Decision tress can also be represented as a series on conditional statements (if-then), a sequence of rules that illustrates the testing of an attribute value to determine the final classification of an instance. Objectively, it is important to select the most appropriate root attribute and subsequent branch attributes to reduce decision tree complexity, computation time and overfitting (Quinlan, 1986). Quinlan (1986) and Quinlan (1993) developed two methods, ID3 (Iterative Dichotomiser 3) and C4.5, respectively, to optimize the building of a decision tree classifier.

ID3

Simplistically, a decision tree can be created based on randomly and continuously generating individual trees from sample data, with the hope of building an optimal classifier that can be generalized to new instances. However, depending on the size of the training data in terms of the various classes and attributes, this approach can be time consuming to generate a viable decision tree. Moreover, the selected decision tree may in fact be one that is overly complex, as well as computationally expensive when used to classify new instances. ID3 is a top-down, greedy methodology for inducing an optimal decision tree with less computational overhead for both the generation of the tree and the classification of new observations. Considering ID3's top-down approach, it is critical for the algorithm to select an attribute for the root of the tree that ultimately minimizes complexity (number of nodes and branches), yet is efficient at performing classifications of new observations. One means for determining the root and subsequent descendant branch nodes is to use a statistical based methodology referred to as *information gain* that measures the relative importance of an individual attribute for classification of an instance (Mitchell, 1997; Quinlan, 1986). In order to calculate information gain, two values are needed: the entropy of the entire dataset and the normalized entropy after the dataset has been split using an attribute (Quinlan, 1993). Entropy, within the context of information theory, is a measure of the impurity or uncertainty of a given dataset (Mitchell, 1997). An examination of how the entropy of a data set is calculated is first described, followed by a discussion of the normalized entropy after segmenting the input data using a selected attribute.

Given a training data set, *D,* which has two distinct classes (positive and negative, denoted by P and *N*, respectively), the probability of positive and negative instances is calculated by the following

$$p_{\oplus} = \frac{\hat{p}}{p+n} \;\; and \; p_{\ominus} = \frac{\hat{n}}{p+n} \qquad (18)$$

where $\hat{p}$ and $\hat{n}$ represent the number of positive and negative instances within the dataset normalized over all instances in the dataset (Quinlan, 1986). Since a decision tree returns a single class for any instance evaluated, it can be considered as a message source for each class, *P* or *N*, contained in the dataset (Quinlan, 1986). Accordingly, principles related to information theory can be applied to determine the information needed to generate a message for *P* or *N*. Based on this precept, the probability equations in (18) can be used to evaluate the entropy of the system and is specified by the formula

$$Entropy(D) = \; -p_{\oplus} \; \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus} \qquad (19)$$

where $p_{\oplus}$ and $p_{\ominus}$are the proportion of positive and negative instances of the dataset (Fayyad & Irani, 1992). Note that if the input space *D* only contains a single class, then equation (19) for the entropy of the system evaluates to 0. Units for the output of equation (19) are in bits and range from 0 to 1, indicating the amount of information required to generated a message related to the class of an instance. The restriction of the dataset to a boolean classification is done for simplicity, and is not indicative of a limitation for ID3 or C4.5. Input datasets may contain significantly more classes than two. As such, equation (19) can generalized to the following

$$Entropy(D) = \sum_{i=1}^{c} -p_i \log_2 p_i \qquad (20)$$

where c is the total number of distinct classes and p_i represents the proportion of each class within the input space *D*.

Determining the entropy after splitting on an attribute follows a similar approach as the entropy for the entire dataset prior to dividing. However, the scope of the evaluation pertains to a single attribute and includes a normalization factor. More explicitly, if an attribute *A* with values $\{a_1 \ldots a_v\}$ is used as the root of tree, it partitions the input space *D* into a subset of branches and associated classes, using each attribute value. That is, for each attribute *A*, and its associated values a_v, a subset of the objective decision tree can be formed by testing the different values for *A*. Accordingly, the entropy for the sub-tree generated from this activity can be evaluated with respect to the particular attribute under test. Written formally,

$$Entropy(A) = \sum_{i=1}^{v} \frac{\hat{p}_i + \hat{n}_i}{p + n} Entropy(D) \tag{21}$$

where $\hat{p}_i \ and \ \hat{n}_i$ represent the number of positive and negative classes related to the attribute A_i being evaluated, normalized by the total number of positive and negative instances for the entire input data space *D*.

Now that both the entropy for the entire input data set *D* and the normalized entropy for each attribute can be determined, information gain can be used as a measure of the effectiveness of an individual attribute for classifying data. Stated differently, information gain for an attribute is a measure of the reduction of entropy for classifying the dataset when a particular attribute is used to partition the data (De Mántaras, 1991; Mitchell, 1997). Written formally

$$Inf\ Gain\ (A) = Entropy(D) - \ Entropy(A) \tag{22}$$

represents the information gain for an attribute used to classify (partition) the data (Quinlan, 1986). In building the decision tree, the attribute that generates the largest information gain for the initial segmentation should be selected as the root. Subsequent descendant nodes are recursively generated in the same manner until either the all classes or attributes within the dataset are exhausted.

Extending ID3 to C4.5

In practice several challenges arise that impact the performance of ID3, some of which are related to the data used to train, while others are inherent to the algorithm itself. Quinlan (1993) implemented several enhancements to ID3 that were codified within the C4.5 algorithm. The discussion that follows provides an overview of four of the key challenges encountered in real-world application of ID3 and provides a synopsis of the method used to address the issue in C4.5.

- Managing Complexity – As with all machine learning algorithms, the optimum balance between complexity and simplicity of an algorithm can be a difficult objective to attain. If a model is complex it may fit the training data extremely well but do poorly when generalize to new instances, an effect referred to as overfitting the data (Schaffer, 1993). Simplicity is always desired, although it may lead to higher error rates. To address this challenge, C4.5 employs post-pruning of a decision tree. Principally, the decision tree is generated using the ID3 algorithm without regard to overfitting issues and subsequently pruned to reduce the number of branches of the tree (Breslow & Aha, 1997). In order to perform post-pruning with some level of assurance that the loss branch does not increase error rates, the input dataset is disjunctively separated into a training and validation subset. Once post-pruning begins, the validation set is used to verify that error rates have not increase as a result eliminating a branch. If error rates increase, the branch is restored. Note, post-pruning can be efficiently applied to decision tree rules instead of the tree structure. The effect of post-pruning is to reduce the overall number of branches, thereby reducing complexity, while maintaining good classifier performance.

- Attributes with Continuous Values – Originally, the ID3 algorithm focused on attributes with discrete rather than continuous values (Quinlan, 1996). In reality, attributes with continuous values occur often in real-world applications of classifiers. Moreover, decision trees must deal with both discrete and continuous values within the same decision tree. *Length* is an example of an attribute with continuous values that can take on a variety of measures. An approach to using this type of attribute within the decision tree is to first sort the values and then identify where changes in attribute values cause subsequent changes in the class of the instance (Fayyad & Irani, 1992). Inherently, thresholds can be identified that align with the transition from one class to another, e.g. positive to negative. These thresholds can be used to test an attribute to determine branching or leaf nodes within the tree. For each threshold of the attribute length associated with change in the output class, information gain can be evaluated in the same manner as any discrete value attributes to determine its place in the decision tree hierarchy.
- Attributes with Missing Values – Although it is optimum to have data that has a value for each attribute to efficiently induce a decision tree, in practice attributes

may be missing values. Though attributes with missing values may introduce errors, the instance may still be of some importance. C4.5 employs probabilities for instances with missing values for attributes (Grzymala-Busse & Hu, 2001). Plainly stated, the frequency of attributes for fully populated instances and their associated class are used to calculate probability for the instance with missing attribute values. The derived probabilities are used instead of assigning the most frequent value to an instance. Once probabilities of attributes with missing values are calculated, they can be used in the evaluation of information gain (Quinlan, 1993).

- Attributes with Different Costs – Certain machine learning problem sets may involve attributes with associated cost. Within this context, cost can be considered explicit, i.e. monetary or inherent such as the importance of one attribute with respect to another. C4.5 employs a weighting factor to information gain that reduces the effect of one attribute vice others (Quinlan, 1996). Side effects include the possible generation of a less optimal decision tree that exhibits bias to a certain classes; although, to some extent bias is the desired effect.

With the enhancements to ID3, C4.5 has become a common classifier algorithm used on a broad range of problems to include data mining tasks. C4.5 has also been enhanced to improve speed, optimize memory usage, incorporate boosting, among other refinements which are embedded in C5.0.

Ensemble Techniques used to Improve ML Performance

Ensembles attempt to find the best result, whether for prediction or classification, from the space of trained hypothesis to reduce misclassification error (Seni & Elder, 2010). Fundamentally, model ensembles follow two principles:

- Generate multiple trained hypothesis, classifiers, that are as diverse as possible
- Use various techniques to leverage the output of the set of diverse classifiers, in such a way that they reduce the overall errors associated with any single classifier

Use of ensemble methods has seen steady growth in both academia and the commercial sector (Rokach, 2010). Accordingly, the number of methods that fall within the category of ensembles has also experience significant growth. For this research effort, two common ensemble methods were used, bagging and boosting.

Bagging

As with all ensemble methodologies, creating diversity amongst the classifiers used is a key objective. Breiman (1996) Bagging, short for “bootstrap aggregating”, creates diversity by manipulating the training dataset. More precisely, given a training set, *D*, bagging entails random sampling of the dataset, with replacement, generating *n* number of bootstrap samples that are used to train individual classifiers (Breiman, 1996). Since sampling is performed with replacement, each bootstrap sample has some number of duplicate instances. However, the probability that a particular training instance is not part of a bootstrap, given *n* samples can be estimated by $\left(1-\frac{1}{n}\right)^n$, which implies that approximately a third of the instances (as *n* gets very large) are omitted from each sample (Flach, 2012). The expectation is that each bootstrap sample induces some level of diversity among the various classifiers of the ensemble. When evaluating the classification of new instances using the bagging ensemble method, a plurality vote is used to select the target class from the output of the various classifiers (Oza & Tumer, 2008). For problems involving prediction, averaging the outputs of the classifier is typically used to determine the target value (Seni & Elder, 2010).

Bagging implies that averaging outputs from the committee of classifiers inevitably limits the effects of noisy data and, to some degree, issues associated with overfitting, since it is unlikely that all the ensemble classifiers respond to the data in the same way (Rokach, 2010). Unlike AdaBoost (described below), bagging does not require weak learners to provide good results; however, learners sensitive to changes in the input data set tend to receive the greatest benefit (Mordelet & Vert, 2014).

Boosting

Similar to bagging, Adaptive Boosting (AdaBoost) attempts to improve the performance of an individual classifier by manipulating training sets; however, bagging depends on replacement sampling of the input data to generate multiple classifiers. In contrast, AdaBoost applies weights, recursively, to instances of the training set to improve the performance of classifiers that are part of an ensemble. That is, the training data for each classifier within the ensemble is modified to account for weights derived from misclassifications errors (Freund & Schapire, 1997). Larger weights are assessed to misclassified instances, while smaller weights are given to correctly classified instances per iteration. The effect of this process is to focus each successive classifier on the misclassified observations, increasing the likelihood of eliminating incorrectly classified instances. The final hypothesis is a weighted combination of classifiers and is expected to produce higher recall and precision— classifier performance. Equation (21) formalizes the objective function for the AdaBoost algorithm

$$y = h(x) = sign\left(\sum_{t=1}^{T} \alpha h_t(x)\right) \tag{23}$$

where $h(x)$ is the signed output of the strong classifier that is generated from the weighed linear combination parameter, α, times the set of hypothesis $h_t(x)$. The process for generating the objective function, based on Flach (2012), in (23) follows the generalized steps outlined below for a given dataset $D = \{(x_i, y_i) \mid \boldsymbol{x} \in \mathbb{R}^d, y \in \{1, -1\}\}, i = 1 \ldots n$, with a specified number of training hypothesis, *T*, and a learning algorithm, $h_t(x)$.

1. An initial weight vector, $\boldsymbol{w} = 1/(|D|)$, is calculated and applied uniformly to all instances of the training dataset.
2. For each iteration from $t = 1\ to\ T$ do the following:
    - Train the target classifier $h_t(x)$, using the weight calculated in step 1 uniformly distributed across each instance of the input data set *D*.
    - Calculate the weighted misclassification error for $h_t(x)$: $\epsilon_t = w_{ti}(h_t(x_i) - y_i)$.
    - Check if error, $\epsilon_t \leq .5$. If so, exit the loop.
    - Calculate the confidence value, $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$, which is used to update weights for misclassified and correctly classified instances. The final value for, α, is used to proportionally combine members of the ensemble in step 3.
    - Update weights for misclassified instances using the following: $w_{(t+1)i} = \frac{w_t}{Z_t}\exp(-\alpha_t \cdot y_t\, h_t(x_i))$, where Z_t is a normalization constant.

Depending on the classification of a target instance, the exponent will be either positive or negative. A positive exponent has the effect of increasing the weight for that instance; a negative exponent has the opposite effect.

3. The output is the objective function in equation (24).

$$h(x) = sign\left(\sum_{t=1}^{T} \alpha h_t(x)\right) \qquad (24)$$

Boosting requires the use of weak classifiers that are slightly better than random guessing (Seni & Elder, 2010). Strong candidates for developing weak classifiers are decision trees that are one level deep, referred to as *stumps* (Rodríguez & Maudes, 2008). However, Adaboost has been combined with other ML algorithms, such as SVM and Naïve Bayes (Kim, Pang, Je, Kim, & Yang Bang, 2003; Korada, Kumar, & Deekshitulu, 2012).

Summary

In this chapter, an overview of supporting literature, ML algorithms and ensemble techniques that used in this research have been provided. Pertaining to supporting literature, several examples of unsupervised, supervised and hybrid ML were presented to provide context for this work. The particular emphasis has been on supervised models, in particular SVM, Naïve Bayes, and C4.5, which constitute the focus of this research effort on classification of video traffic. In addition, ensemble techniques, i.e. bagging and boosting, were described as means to improve recall and precision for each model. In the following chapter on methodology, specifics on how each ML algorithm and ensemble technique support experimentation are described.

Chapter 3

Methodology

Introduction

The methodology used to perform experimentation involved capturing real network traffic to train, test and compare performance, recall, and precision, of three ML classifiers, C4.5, Naïve Bayes, and SVM, to identify a video streaming traffic when partial, incomplete data traces are encountered. Waikato Environment for Knowledge Analysis (Weka) implementation of C4.5, Naïve Bayes and SVM was used for all experimentation (Witten et al., 1999). Experiments confirmed that ML classifiers trained on statistics derived from full traffic flows exhibited degraded recall and precision as the number of missing beginning packets increases. Conversely, ML classifiers trained on multiple sub-flows selected from all phases of the application produced much higher and consistent recall and precision, despite the presence of incomplete traffic traces. In the process of comparing the outcome from testing classifiers trained with full flows and sub-flows against partial data captures, research question 1 was addressed. Figure 3 represents a generalized overview of methodology used in this research. A description of each step of the process follows.

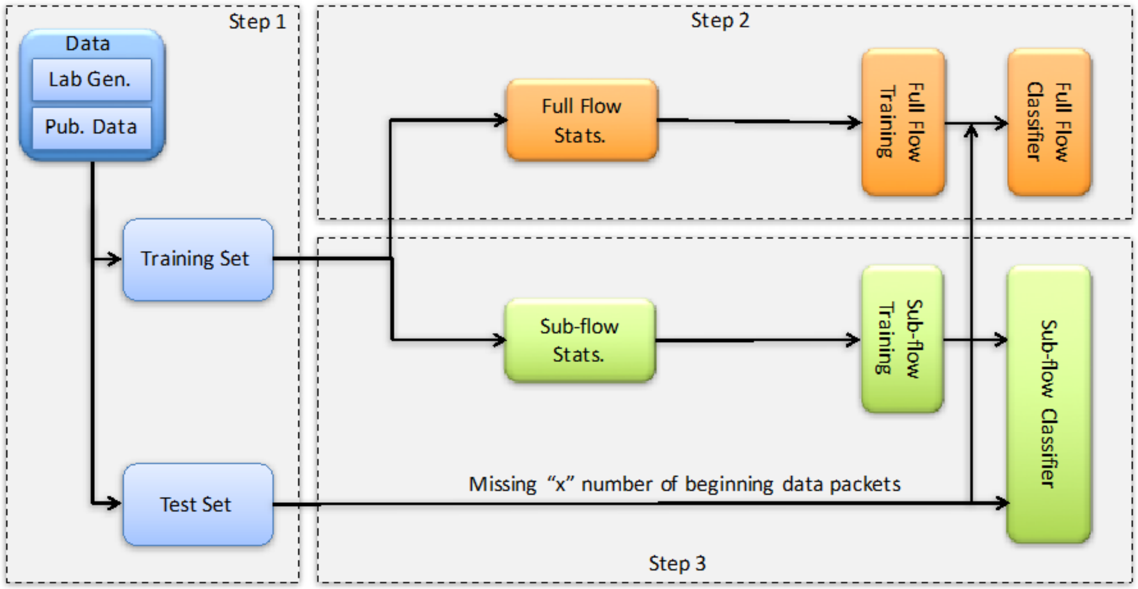


Figure 3 Overview of Research Methodology

Step 1 – Data Collection

Traffic data used for this research included target video application flows as well as interfering traffic. Interfering traffic, within the context of this study, is any traffic flow that is not the target application to be classified (Karam & Tobagi, 2000). The methodology used to secure data for use in this research effort was internet traffic collected in a controlled lab environment.

Step 1a – Capturing Live Traffic

To capture video traffic data, Wireshark was used. Wireshark is a freely available, publicly accessible, network analysis application that is used to collect information about data packets (Dabir & Matrawy, 2007; Lamping & Warnicke, 2004). To capture packet data, each target application (YouTube and Netflix) along with a separate instance of Wireshark, was run on a virtual machine (VM) instance on a laptop. Additionally, interfering traffic was also generated at the same time as target application data on each VM instance. Figure 4 depicts the environment used to capture network traffic.

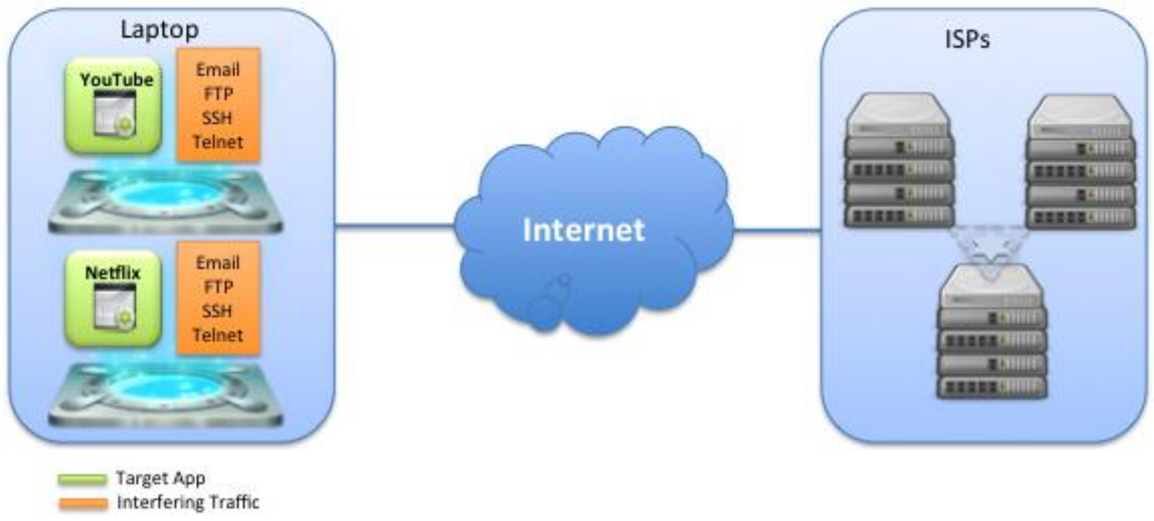


Figure 4 Environment used to Capture Traffic

The use of a multiple VMs running each target application along with interfering traffic simplifies the process of determining the "ground truth" for the class of each application flow within training and test data set. Specifically, to the greatest extent possible, a limited and deterministic set of application traffic was generated that can be readily identified using source and destination IP address, port and protocol to enable proper labeling of network traffic (Pascoal et al., 2012; Piraisoody, Changcheng, Nandy, & Seddigh, 2013). The process for generating known traffic types within a lab environment to improve the likelihood of correctly labeling training and test sets is a common approach (Alshammari & Zincir-Heywood, 2008; L. Peng, Zhang, Yang, Chen, & Wu, 2014). Note in Figure 4 that interfering traffic is comprised of email, FTP, Secure Shell, and telnet. These are applications that have well-known ports officially registered in IANA. Accordingly, this increases the probability of distinguishing target from interfering traffic. The size of the files containing captured packets were limited to make processing of the files more efficient. A portion of the captured traffic was used exclusively for training and the rest used for testing both the full flow and sub-flow classifiers. The ratio of training to testing data set was ~60/40.

Step 1b – Generating Full Flow and Sub-flow Feature Sets

Once Wireshark has captured the requisite sets of packet capture (pcap) data for each target application, steps must be taken to extract statistical information and create an attribute-relation file format (ARFF) file for use as input to the Weka's ML application. Pcap files produced by Wireshark cannot be used directly with Weka for training or testing ML classifiers.

Two types of Weka input files containing statistics for both target applications were produced: full flow and sub-flow ARFF files. For both types of ARFF files, scripts were built for processing Wireshark output. Wireshark is integral to the generation of both full flow and sub-flow statistics since it performs the duty of capturing network traffic as well as text exports of pcap data. Figure 5 provides a generalized depiction of the process for generation of full and sub-flow statistics. The following sections provide additional details related to each element of the process.

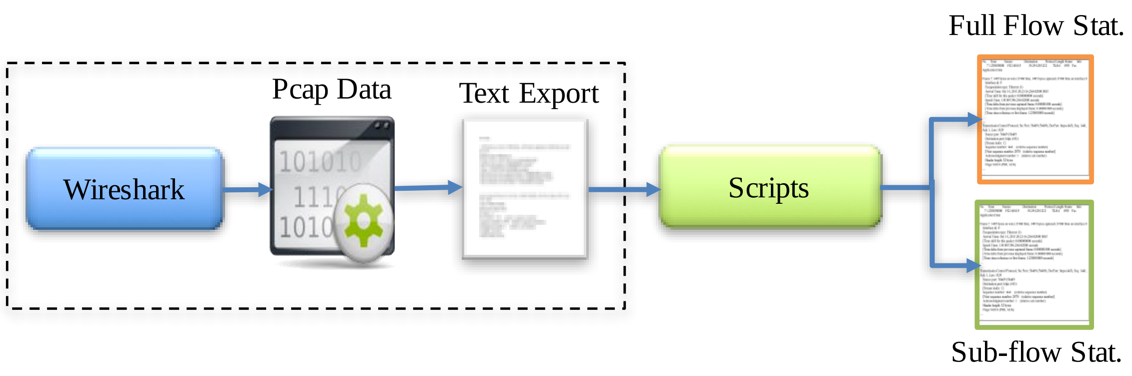


Figure 5 Generating Full and Sub-flow Statistics

Full Flow Statistics

While Wireshark is proficient at capturing network traffic, to generate full flow and sub-flow statistics it was necessary to develop specific scripts to generate a more robust set of stats. Wireshark can provide some full flow statistics; however, the depth, breadth and type of statistics needed to support this research necessitated additional preprocessing. Multiple approaches were attempted to discern the correct set of attributes. However, the selection of features is primarily based on work by Alshammari and Zincir-Heywood (2011) and Nguyen and Armitage (2006). As experimentation progressed, features were pruned and added to improve classification accuracy. Table 2 provides the final set of features that were used for training and classification experimentation.

Table 2 Full Flow and Sub-flow Statistics

Attribute	Description
total_packets	Total number of packets
total_volume	Total number of bytes
min_pktl	Minimum packet length
mean_pktl	Mean packet length
max_pktl	Maximum packet length
std_pktl	Standard deviation of packet length
min_iat	Minimum inter-arrival time of packets
mean_iat	Mean inter-arrival time of packets

max_iat	Maximum inter-arrival time of packets
std_iat	Standard deviation for inter-arrival time of packets
total_headl	Total header length
min_tcphl	Minimum TCP header length
mean_tcphl	Mean TCP header length
max_tcphl	Maximum TCP header length
std_tcphl	Standard deviation for TCP header length
total_intframe	Total inter-packet length between packets of the same flow
min_intframe	Minimum inter-packet length between packets of the same flow
mean_intframe	Mean inter-packet length between packets of the same flow
max_intframe	Maximum inter-packet length between packets of the same flow

Wireshark has the ability to generate information about each layer of the TCP/IP protocol stack (physical, link, network, transport and application layer) for a given packet (Lamping & Warnicke, 2004). Using Wireshark, a text file export of protocol attributes gathered from pcap files, such as frame length, time delta, IP length, TCP header length, etc. can be produced. Figure 6 represents a sample of a Wireshark text export file containing a subset of packet attributes.

```
No.    Time          Source              Destination          Protocol Length Frame
Info
     7 1.230858000   192.168.0.9         54.244.245.212       TLSv1    1495   Yes
Application Data

Frame 7: 1495 bytes on wire (11960 bits), 1495 bytes captured (11960 bits) on
interface 0
    Interface id: 0
    Encapsulation type: Ethernet (1)
    Arrival Time: Oct 14, 2013 20:23:16.256442000 MST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1381807396.256442000 seconds
    [Time delta from previous captured frame: 0.000001000 seconds]
    [Time delta from previous displayed frame: 0.000001000 seconds]
    [Time since reference or first frame: 1.230858000 seconds]
…
…
Transmission Control Protocol, Src Port: 56469 (56469), Dst Port: https (443),
Seq: 1441, Ack: 1, Len: 1429
    Source port: 56469 (56469
    Destination port: https (443)
    [Stream index: 1]
    Sequence number: 1441    (relative sequence number)
    [Next sequence number: 2870    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Header length: 32 bytes
    Flags: 0x018 (PSH, ACK)
…
…
```

Figure 6 Sample Wireshark Text Export

Once the Wireshark text export file is created, it can be processed to generate the statistics listed Table 2 in addition to ARFF formatted files for sub-flows. The requisite scripts were developed to perform the processing of Wireshark text export files.

Step 1c – Creating Training and Test Sets

To properly train and test the classifier, a mixture of the target application and interfering traffic must be part of the same respective data file prior to use with Weka ML classifiers. In previous steps, a mixture of target and interfering traffic is captured and then used to generate ARFF files for use with Weka. Prior to using the ARFF file for testing or training, the class of each flow must be determined and labeled as either the target (YouTube or Netflix) or interfering traffic (all other traffic). To determine the "ground truth" class for captured flows, a two-step process was used that employs both Wireshark and the output from the scripts. First, Wireshark was used to examine captured traffic in order to understand key attributes such IP address, port, protocols, start time of various flows in the traffic. Since the traffic generated and captured in Step 1a is fixed to the greatest extent possibly, use of IP address, port, protocol, and start time provided strong evidence as to class of the traffic. This is especially true for traffic that use IANA registered ports below 1023, which was the objective. Secondly, once ARFF files are generated using scripts, the same attributes, IP address, port, protocol, and start time were used to label flows in the ARFF file. For files containing YoutTube traffic, the class labels was YT or OTHER; and for files containing Netflix traffic, labels were NF or OTHER. When creating training and test data sets, maintaining a ratio close to 1:1 between target application and interfering traffic was the objective.

Step 2 – Classification Based on Full Flows

Evaluating the effectiveness of ML classifiers trained on statistics from full flows on partial data sets, specifically, flows that are missing the beginning packets of the traffic, is an important first step. In addition to testing data sets that are missing the initial set of packets, test datasets also contained varying sub-flow sizes to simulate the effect of partial flows. This initial experiment is required to confirm the degradation of performance, recall and precision, for ML algorithms trained on full flows for identifying target traffic when used with incomplete data captures. Recollect that captured data in real-world networks may be incomplete due to network and application perturbations. C4.5, Naïve Bayes, and SVM ML algorithms were trained on statistics from full flows then tested with data sets that have the first 10, 20, 30, 40, 50, 60, 100 and 200 packets missing from the traffic flow. Note that flow statistics used for training were not used for test purposes; once the required number of flows were collected, separate data sets for both training and testing were generated. For each test data set containing missing packets, an ARFF file was constructed that had both the target and interfering traffic. Input files for Weka were created using the process described in step 1, to generate data sets with missing packets. Recall and precision were calculated and graphed as a function of the missing packets to illustrate the effects of partial data captures on full flow classifiers. It was expected that recall and precision would reveal a significant degradation in performance as the number of missing beginning packets increases. These results were used as a reference for comparison against ML classifiers trained on multiple sub-flows.

Step 3 – Classification Based on Sub-flows

Initially, sub-flows of 25 consecutive packets were selected from the full flow data, with a sliding window of 10 packets between each sub-flow. In previous work by Nguyen et al. (2012), sub-flows of 25 packets provided good results in classifying Wolfenstein and VoIP traffic. However, it is unknown what the optimum number of packets per sub-flow should be to produce high, above 92% or better, recall and precision; accordingly, 25 packets constituted a starting point for sub-flow classification. Multiple sub-flow sizes were attempted with the objective of maintaining high recall and precision. Adjusting the number of packets per sub-flow, while recording the effects on recall and precision addressed research question 4. ARFF files were created to evaluate sub-flows of differing sizes (number of packets) to execute this portion of the experiment. The initial set of features used were based on features listed in Table 2. Some degree of experimentation with various combinations of features was undertaken to find the minimum number of features to gain high recall and precision, addressing research question 3.

Evaluating Ensemble Techniques: Bagging and Boosting

Once the analysis in steps 2 and 3 were completed, an evaluation of ensemble techniques was performed. As outlined in section 2, bagging and boosting were the ensemble techniques evaluated for this research effort. Weka has the capability to perform bagging and boosting using a variety of base ML algorithms (Bouckaert et al., 2013; Elovici, Shabtai, Moskovitch, Tahan, & Glezer, 2007).

Testing with Weka can be performed using a command line interface (CLI), explorer or the experimenter. Both the explorer and experimenter Weka application can be used for training and testing ML applications. The key difference is the experimenter allows the training and testing of ML algorithms side-by-side for direct comparison. This provides an effective means of comparing results of ML algorithms on the same data set to determine which method preforms best.

Using the explorer, bagging in combination with base implementations of Naïve Bayes, C4.5, and SVM were trained and tested using the same data set in step 3 to determine if there was any improvement to performance, specifically, recall and precision. It was expected that bagging would have a greater effect on C4.5, since decision trees are more sensitive to changes in the training and test data set (Galar, Fernandez, Barrenechea, Bustince, & Herrera, 2012). Moreover, the effect of bagging may be less pronounced on the performance of Naïve Bayes and SVM, since in general these ML algorithms are less sensitive to variances in the training and test data (Yuan et al., 2010). In the case of boosting, the same process was followed. Boosting was performed using each base ML algorithm and compared to outcomes from the execution of step 2. Boosting was also expected to be less effective on Naïve Bayes and SVM, since both of these algorithms are considered to be strong learners (Hall, Witten, & Frank, 2011).

Format for Results

Generally, tables and figures containing text and graphs are used to display results from experimentation.

Pcap data files were not presented in this dissertation as these files would be too large and would not add value to communicating the results of this work. Additionally, ARFF files containing features and associated statistics were also omitted from this document based on the same rational.

Outputs from testing classifiers on full flows are depicted in tables and graphs as a function of the number of packets missing from the beginning of each network flow, to recall and precision of the classifier tested. In the same manner, graphs of recall and precision per missing packets for classifiers tested with sub-flows were also be presented in the results section with Table containing detailed information and plots used as a graphical depiction of the data.

Resource Requirements

This research effort required a variety of resources to perform required experimentation as specified in Table 3.

Table 3 Required Resources

Type	Resource	Purpose
Data	Lab Captured Data	Using Wireshark, pcap data was captured from test systems (laptop/PCs/desktops) to support training, validation and testing of ML algorithms.
Software	Microsoft Office	MS Office (Word, PowerPoint and Excel) is a general purpose document, presentation and spreadsheet software package that was used throughout this research activity. Excel was also used for generating statistics and manipulating data sets.
	Perl	Perl was used to develop scripts to manipulate files and to generate sub-flow statistics in ARFF format.
	VM Fusion	This software package is used to create virtual machines.
	Wireshark	This software was used to capture IP traffic for both target and interfering application flows.
Hardware	PC and Laptops	General purpose PCs, Macs, and desktops for data capture, manipulating data and developing documents and presentations.

Summary

In this section, the methodology used to execute the experiments was described fully. Concepts regarding the gathering of network traffic, generation of statistics for both full flow and sub-flows were outlined. The use of lab captured traffic thoroughly supports this research effort. While the use of benchmark data would support assessment of the generalizability of the classifier, significant challenges were experienced in collecting "ground truth" benchmark data for both YouTube and Netflix traffic that could be used for testing the generalizability of each classifier (Caiyun, Lizhi, Bo, & Zhenxiang, 2012). Accordingly, benchmark data was not used in this research.

Evaluation of ensembles was undertaken as the final stage of experimentation to assess if bagging or boosting can improve the performance of the sub-flow classifier. Evaluation of ensembles for this research effort is important given the increase use of ensembles across the spectrum of ML applications (Galar et al., 2012).

Chapter 4

Results

In this chapter the results of experimentation are presented. Prior to describing results, a short discussion on data preprocessing and class imbalance is provided. Next, results on the effects of partial flows on a classifier trained on full flow statistics is examined. Then, an evaluation on the effectiveness of classifiers trained on sub-flows is examined using traffic flows with missing packets to determine if performance is improved. Finally, the use of ensemble techniques, Boosting and Bagging, is examined to discern if these algorithms improve sub-flow classifier performance. In all cases, performance of classifiers was judged based on recall and precision.

Data Preprocessing

Basic preprocessing of data was required to ensure proper training and building of classifiers. Weka provides filters, methods for manipulating data, as a means to preform data preprocessing, prior to training and testing classifiers. Additionally, Weka filters, were used to split data into training and testing datasets. Resampling without replacement was used to partition the data into training and testing datasets. Finally, class imbalance was addressed using the Weka filter synthetic minority over-sampling technique (SMOTE). Table 4 and 5 provides the breakout of traffic classes for both YouTube and Netflix (bold print), respectively, prior to preprocessing. All subsequent data files used in experimentation were derived from these two foundational data sets.

Table 4 YouTube Dataset

Class of Traffic	Sum of Packets	Sum of Bytes
aggregate	80	20862
akamai Tech	19438	19115955
amazon	214171	253509968
apple	66	6089
avast	554	111371
criteo	698	425408
DNS	21659	18236174
doubleclick	6081	2632553
doubleverify	126	38484
edgecast	26	4019
email	48412	43023115
facebook	38	8670
footprint	236	78593
ftp	258500	260123595
google	73624	40540232
imdb	834	215953
motocast	1738	275806
pki	46	9900
spyware	162	42997
twitter	3837	979977
unknown	3884	2615064
yahoo	1627	512630
YouTube	**1143792**	**1063013067**
Grand Total	1799629	1705540482

Table 5 Netflix Dataset

Class of Traffic	Sum of Packets	Sum of Bytes
akamai Tech	14134	12457513
apple	1164	270255
avast	234	42385
bright tag ad	185	65738
DNS	4423	1128399
doubleclick	662	237501
ftp	1211092	1354307441
google	45	11512
mawi	301015	386171493
Netflix	**1233909**	**1395827732**

Class of Traffic	Sum of Packets	Sum of Bytes
Nova University	42	2947
twitter	117	24721
yahoo	849	279204
Yahoo email	137748	154893713
Grand Total	2905619	3305720554

Addressing Class Imbalance

Class imbalance for this research effort was an outcome of two key factors: 1) the nature of the data used and 2) the manipulation of the data to derive sub-flows for training and testing classifiers. Class imbalance is exhibited by a significant difference between the majority and minority classes of a given dataset. For example, the majority class may be 2 to 3 times larger, in terms of the number of class instances represented in the dataset.

Video streaming traffic is typically long-lived flows comprised of large numbers of packets as compared to interfering traffic. Since video traffic consists of long-lived flows, the total number flows may be substantially less than those of interfering traffic, which generally has relatively small numbers of packets but repeats often within the captured dataset. Accordingly, the predominance of network flows for training are interfering traffic over the total capture time for the data. Inherently this leads to class imbalance for datasets used to train and generate full flow classifiers as well as any associated full flow test data sets.

The second factor that causes class imbalance is the generation of sub-flows for experimentation. While the predominance of full flows are interfering traffic, the total number of packets is disproportionately associated with video traffic, since video traffic tends to be long-lived flows with 100s to 1000s of packets. As a consequence, the generation of sub-flow instances, as a function of the sub-flow size selected, can create class imbalance for datasets used for experimentation, since sub-flow generation divides full flows into subsets. This is especially true with small sub-flows sizes of 25 and 100 packets. In general, video streams generated significantly more sub-flows due to the total number of packets per flow in contrast to interfering traffic. For both types of class imbalance, SMOTE was used to address this condition.

SMOTE is a long-standing and accepted means to address class imbalance by oversampling the minority class of a particular dataset ((Sáez, Luengo, Stefanowski, & Herrera, 2015)). More specifically, SMOTE works within the feature space not with the instance space (deleting instances from the majority class) to synthetically generate a new minority class instance based on two sample classes within the original dataset (Chawla, Bowyer, Hall, & Kegelmeyer, 2002). Weka supplies a SMOTE implementation that was used to address class imbalance issues and is used throughout this research. SMOTE reduces the effect of class imbalance while maintaining the integrity of the dataset used for training classifiers ((Chawla et al., 2002; J. Wang, Xu, Wang, & Zhang, 2006)).

Full Flow Trained Classifier Applied to Partial Flows with Missing Packets

For this experiment, data was captured for YouTube and Netflix along with interfering traffic. Target traffic was considered positive instances, and interfering traffic was designated as negative instances. Full flow statistics were used to train a C4.5, Naïve Bayes and SVM full-flow classifier. Table 6 provides key information associated with building full flow classifiers to execute this portion of the test.

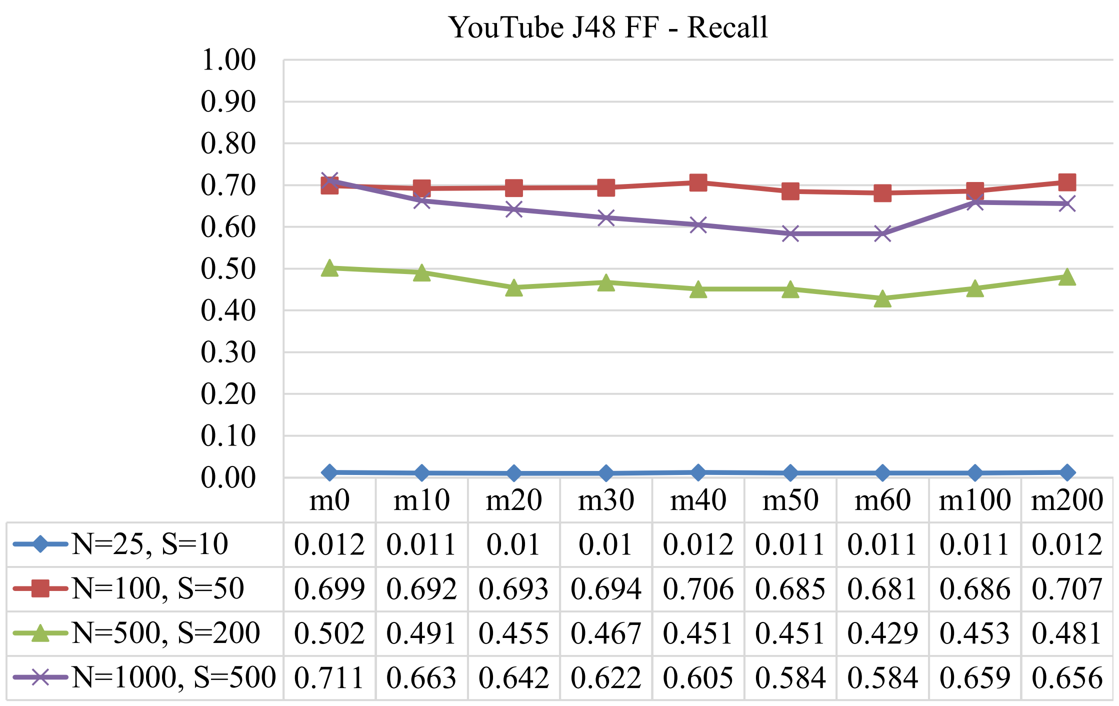
Table 6 Full Flow Training Stats

Traffic	Algorithm	Positive	Negative	Precision	Recall
YouTube	J48-C4.5	2455	3423	0.974	0.983
	Naïve Bayes	2455	3423	0.969	0.952
	SMO SVM	2455	3423	0.868	0.829
Netflix	J48-C4.5	808	492	0.967	0.979
	Naïve Bayes	873	427	0.905	0.99
	SMO SVM	833	467	0.879	0.917

Once the full flow classifiers were built, each full flow classifier was then tested against files with a select number of packets missing as well as different sub-flow sizes to assess performance.

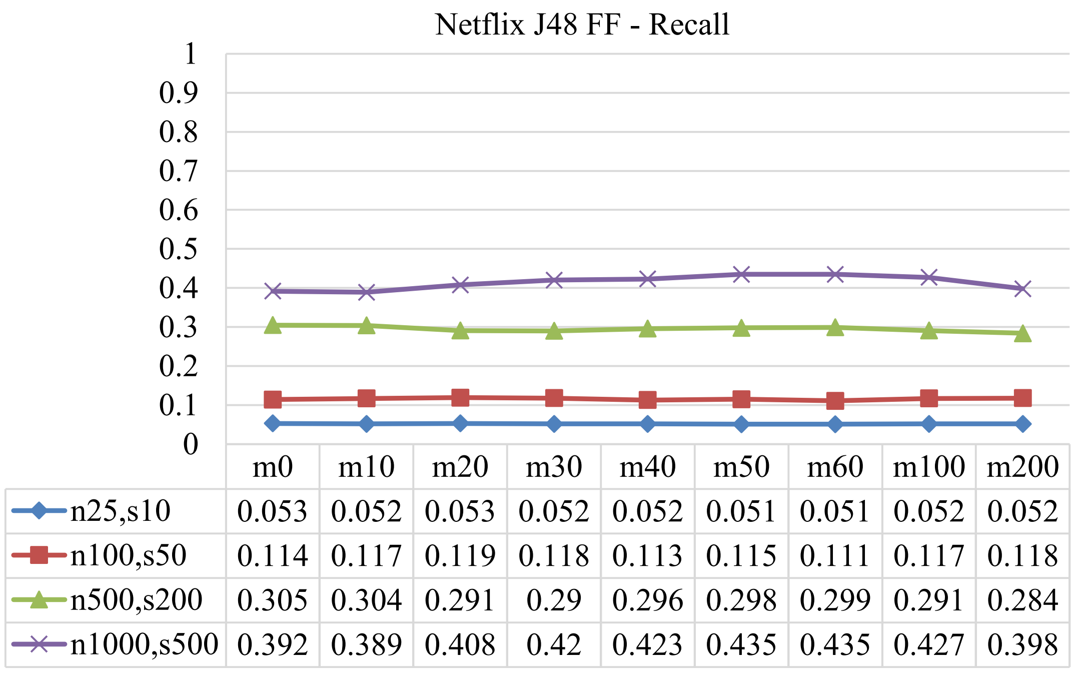
J48 C4.5 Full Flow Classifier Performance

Figure 7 and 8 represents recall for YouTube and Netflix J48 full-flow classifiers, respectively, tested against datasets with missing packets and varying sub-flow sizes. J48 is Weka’s implementation of C4.5 and is used interchangeably throughout the rest of this document. Weka default settings were used for the J48 decision tree algorithm.



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ N=25, S=10	0.012	0.011	0.01	0.01	0.012	0.011	0.011	0.011	0.012
■ N=100, S=50	0.699	0.692	0.693	0.694	0.706	0.685	0.681	0.686	0.707
▲ N=500, S=200	0.502	0.491	0.455	0.467	0.451	0.451	0.429	0.453	0.481
× N=1000, S=500	0.711	0.663	0.642	0.622	0.605	0.584	0.584	0.659	0.656

Figure 7 Recall for YouTube J48 Full-flow Classifier Tested with Partial Flows



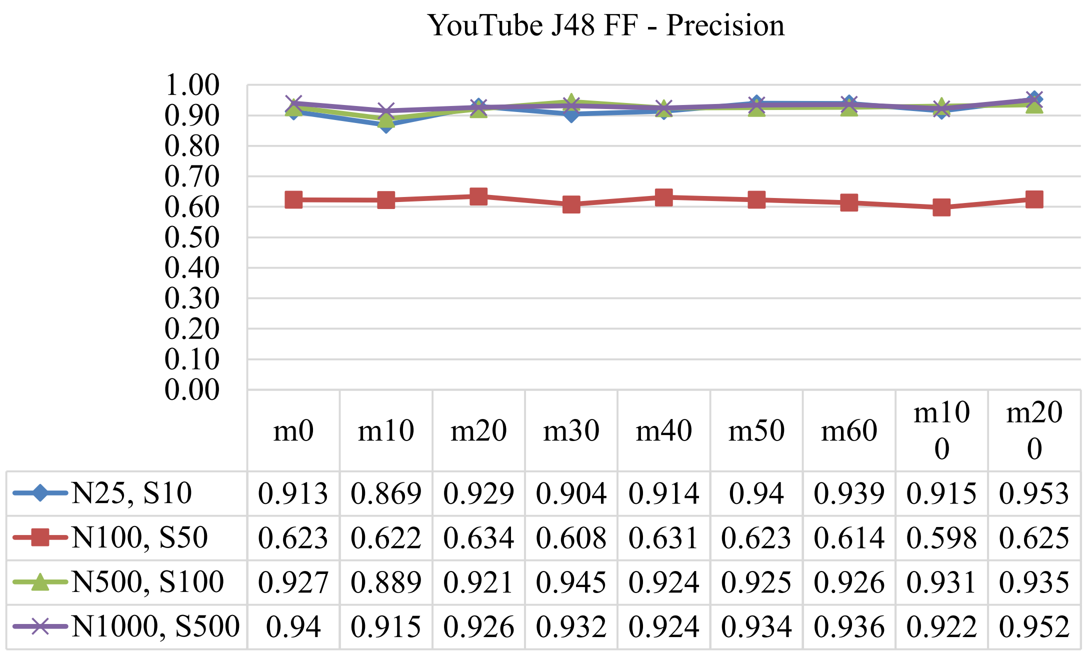
	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ n25,s10	0.053	0.052	0.053	0.052	0.052	0.051	0.051	0.052	0.052
■ n100,s50	0.114	0.117	0.119	0.118	0.113	0.115	0.111	0.117	0.118
▲ n500,s200	0.305	0.304	0.291	0.29	0.296	0.298	0.299	0.291	0.284
× n1000,s500	0.392	0.389	0.408	0.42	0.423	0.435	0.435	0.427	0.398

Figure 8 Recall for Netflix J48 Full-flow Classifier Tested with Partial Flows

The y-axis indicates recall for the J48 decision tree for a given dataset, with a specific number of missing packets, “m0 – m200” on the x-axis. Each point on a plot for a given line (color coded with different markers) represents a particular dataset tested against a J48 full-flow classifier. For example, the first point on the “m0,n,=25,s=10” plot represents a data file that contains instances that are 25 packets long (sub-flow size), with a step size (skipped packets) of 10 packets with no missing packets (m = 0). This data file would be representative of a partial flow that, although it has no missing packets, contains partial flows of 25 packets. The second point on the same line plot (m10,n=25,s=10) represents a data file that is missing the first 10 packets from each flow and has partial flows of 25 packets. Each line plot on the graph represents 9 separate datasets, points, for a particular sub-flow size of 25, 100, 500 and 1000 packets, missing 0 to 200 packets (m0 – m200) from the start of the flow. The use of sub-flows in this test illustrates the impact of partial flows on a classifier trained on full flow statistics.

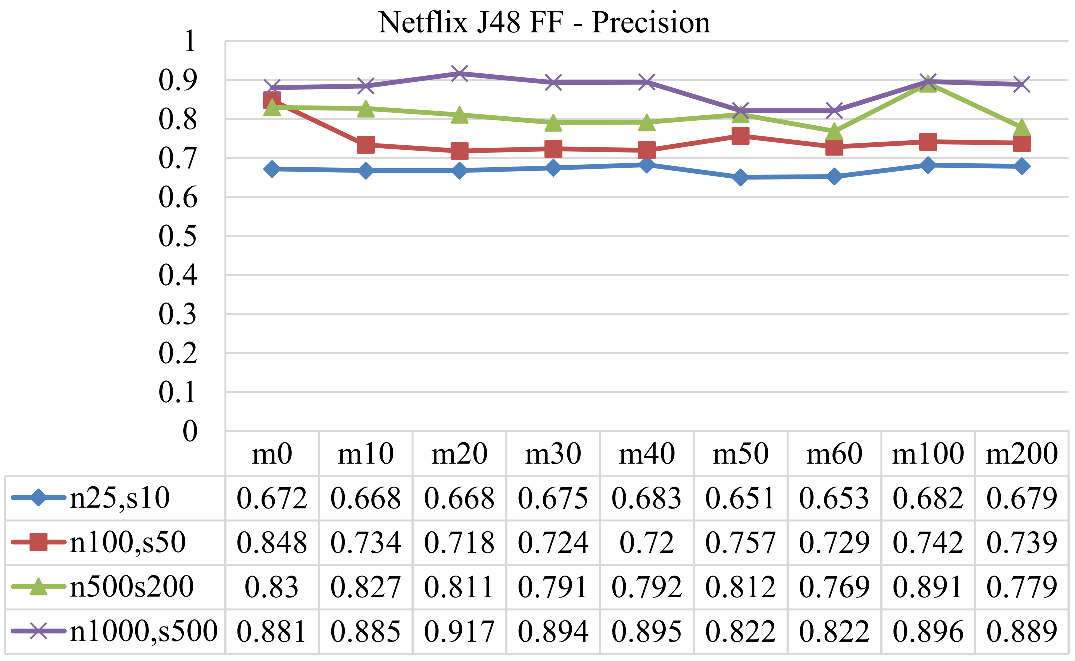
As evidenced, using a full-flow classifier to classify partial flows leads to average performance for YouTube traffic with a maximum recall of ~0.70 for “m40,n100,s50” and poor performance for Netflix traffic with a maximum of ~0.43 for “m50,n1000,s=500”. Good performance is not attained even in the case of larger sub-flow sizes, such as 1000 packets; recall is relatively low as packets are removed, which indicates the full-flow classifier has limited accuracy and in fact misclassifies a significant portion of the target class as the negative classes (false negatives). This was expected given the difference in statistics calculated over the life of a flow can vary dramatically with respect to the type of traffic.

Figure 9 and 10 depicts precision for the same J48 full-flow classifier tested with the same dataset for YouTube and Netflix, respectively. In certain cases, with partial flows of sub-flow size of 25, 500 or 1000 packets, the full-flow classifier exhibits consistently high precision -- above 0.90. However, given the poor recall for the same classifier – indicating large numbers of false negatives – high precision is of little benefit. Both precision and recall need to exhibit good performance to assess classifier performance as excellent.



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ N25, S10	0.913	0.869	0.929	0.904	0.914	0.94	0.939	0.915	0.953
■ N100, S50	0.623	0.622	0.634	0.608	0.631	0.623	0.614	0.598	0.625
▲ N500, S100	0.927	0.889	0.921	0.945	0.924	0.925	0.926	0.931	0.935
× N1000, S500	0.94	0.915	0.926	0.932	0.924	0.934	0.936	0.922	0.952

Figure 9 Precision for YouTube J48 Full-flow Classifier Tested with Partial Flows



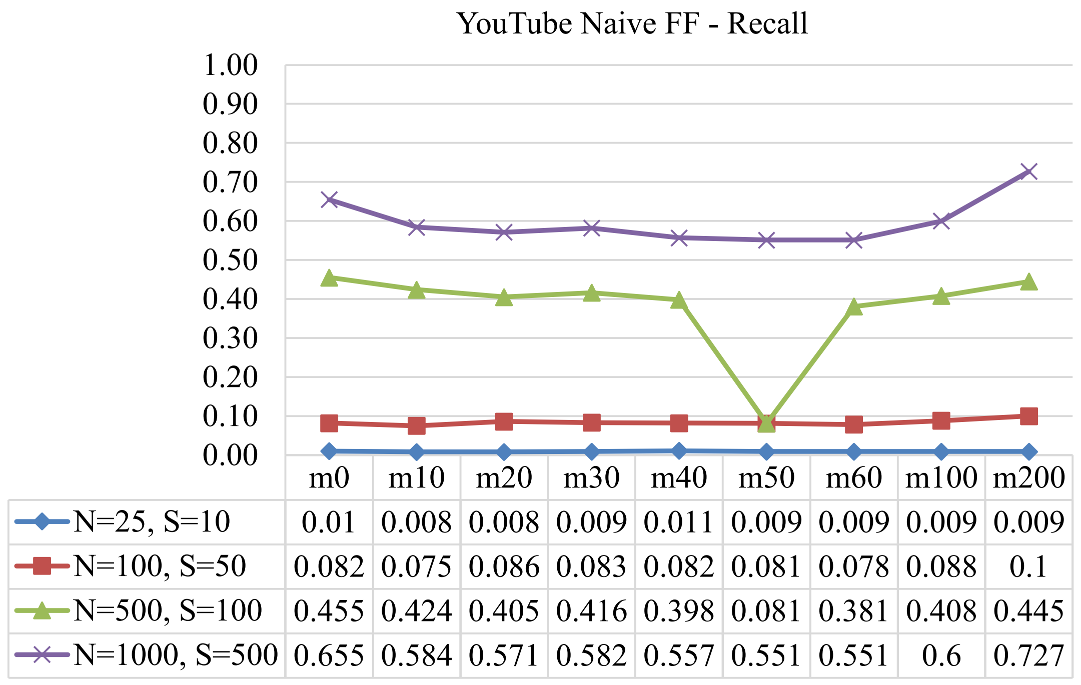
	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ n25,s10	0.672	0.668	0.668	0.675	0.683	0.651	0.653	0.682	0.679
■ n100,s50	0.848	0.734	0.718	0.724	0.72	0.757	0.729	0.742	0.739
▲ n500s200	0.83	0.827	0.811	0.791	0.792	0.812	0.769	0.891	0.779
× n1000,s500	0.881	0.885	0.917	0.894	0.895	0.822	0.822	0.896	0.889

Figure 10 Precision for Netflix J48 Full-flow Classifier Tested with Partial Flows

Naïve Bayes Full Flow Classifier Performance

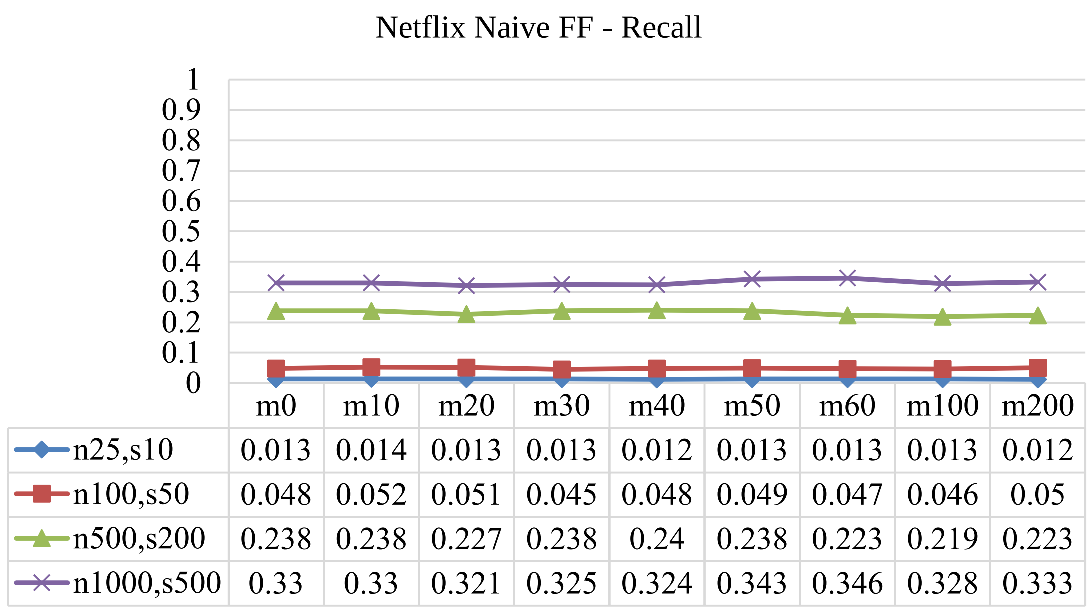
To develop the full flow Naïve Bayes classifier, the same approach was used. For Naïve Bayes a single configuration parameter was applied; specifically, “supervised discretization” was selected for the Naïve Bayes algorithm prior to training the classifier. Discretization is a method for transforming continuous values for variables, into discrete values, by creating intervals over the range of values for a specified variable ((Garcıa, Luengo, Sáez, López, & Herrera, 2013; H. Liu, Hussain, Tan, & Dash, 2002)). Research has shown that discretization may significantly improve the performance for certain machine learning algorithms, Naïve Bayes being a prominent example ((Al-Aidaroos, Bakar, & Othman, 2010; Y. Liu, Li, Guo, & Feng, 2008)). Certain variables for data used in these experiments had a broad range of values, e.g. 1 – 49,000, which impacted the performance of the Naïve Bayes machine learning algorithms. Weka's implementation of discretization was used for all experiments involving Naïve Bayes.

In Figure 11 and 12, recall for the Naïve Bayes is presented. Results were similar to that of J48 in terms performance, although poor performance in certain cases – where partial flows of 25 and 100 packets were tested against the full flow Naïve Bayes classifier – is more pronounced for Netflix traffic. Recall for YouTube sub-flow sizes of 1000 packets were well above 50% indicating the larger sub-flow sizes have a greater affiliation to statistics for full flows. Moreover, the performance for large YouTube sub-flow size was relatively consistent even when the number of missing packets increased sharply from 60 to 200.



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ N=25, S=10	0.01	0.008	0.008	0.009	0.011	0.009	0.009	0.009	0.009
■ N=100, S=50	0.082	0.075	0.086	0.083	0.082	0.081	0.078	0.088	0.1
▲ N=500, S=100	0.455	0.424	0.405	0.416	0.398	0.081	0.381	0.408	0.445
× N=1000, S=500	0.655	0.584	0.571	0.582	0.557	0.551	0.551	0.6	0.727

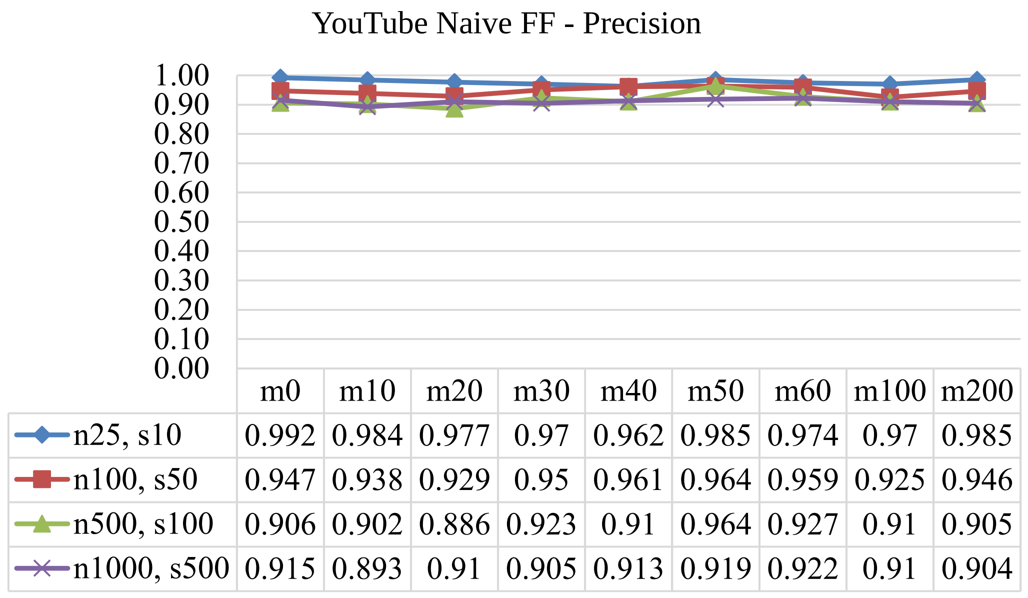
Figure 11 Recall for YouTube Naïve Bayes Full-flow Classifier Tested with Partial Flows



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ n25,s10	0.013	0.014	0.013	0.013	0.012	0.013	0.013	0.013	0.012
■ n100,s50	0.048	0.052	0.051	0.045	0.048	0.049	0.047	0.046	0.05
▲ n500,s200	0.238	0.238	0.227	0.238	0.24	0.238	0.223	0.219	0.223
× n1000,s500	0.33	0.33	0.321	0.325	0.324	0.343	0.346	0.328	0.333

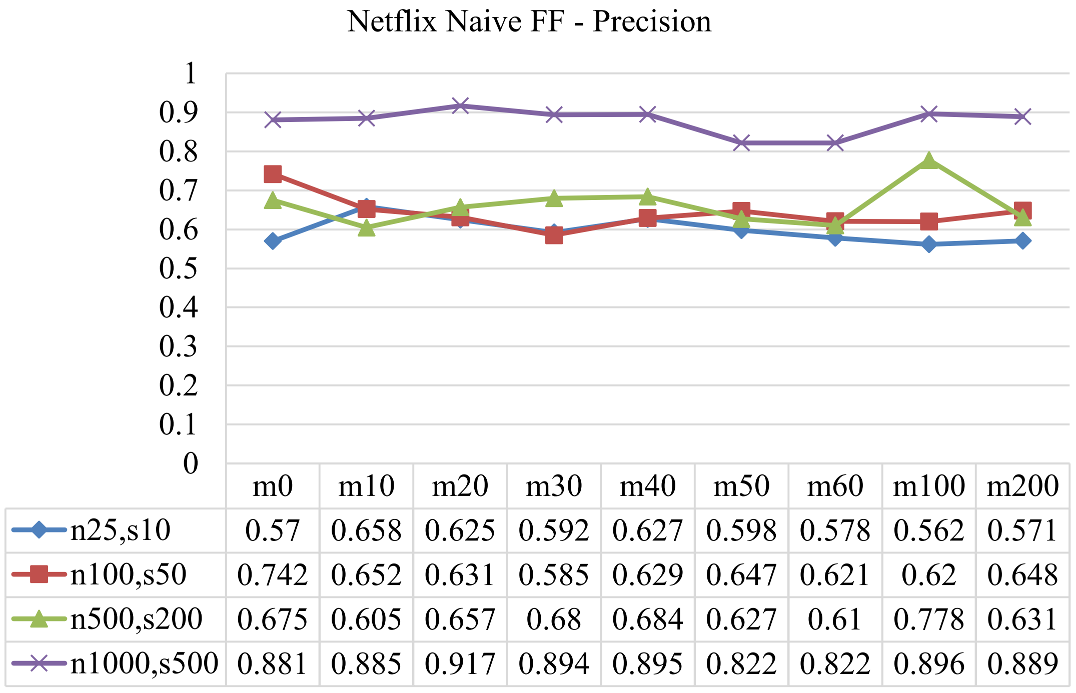
Figure 12 Recall for Netflix Naïve Bayes Full-flow Classifier Tested with Partial Flows

For precision, Figure 13 shows high values ranging from the low to high 90s for YouTube. Precision for Netflix attains values in the 90s as depicted in Figure 14. Although, performance for precision is reasonably high, recall is still relatively low for the Naïve Bayes full flow classifier.



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ n25, s10	0.992	0.984	0.977	0.97	0.962	0.985	0.974	0.97	0.985
■ n100, s50	0.947	0.938	0.929	0.95	0.961	0.964	0.959	0.925	0.946
▲ n500, s100	0.906	0.902	0.886	0.923	0.91	0.964	0.927	0.91	0.905
× n1000, s500	0.915	0.893	0.91	0.905	0.913	0.919	0.922	0.91	0.904

Figure 13 Precision for YouTube Naïve Bayes Full-flow Classifier Tested with Partial Flows



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ n25,s10	0.57	0.658	0.625	0.592	0.627	0.598	0.578	0.562	0.571
■ n100,s50	0.742	0.652	0.631	0.585	0.629	0.647	0.621	0.62	0.648
▲ n500,s200	0.675	0.605	0.657	0.68	0.684	0.627	0.61	0.778	0.631
× n1000,s500	0.881	0.885	0.917	0.894	0.895	0.822	0.822	0.896	0.889

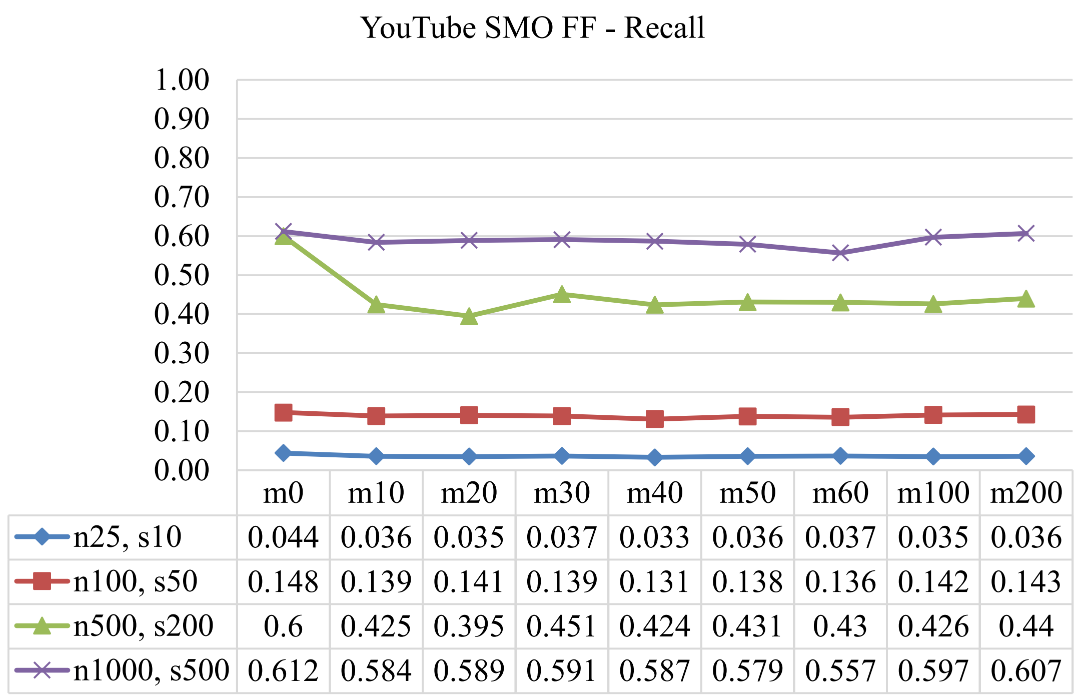
Figure 14 Precision for Netflix Naïve Bayes Full-flow Classifier Tested with Partial Flows

SVM Full Flow Classifier Performance

There are two separate implementations of SVM that can be executed using Weka: sequential minimal optimization (SMO) and libsvm. SMO was developed as part of the Weka platform and is available to users; whereas libsvm was developed by Yasser EL-Manzalawy and is not integrated as part of the weka distribution. However, Weka does provide a wrapper class to run libsvm.jar from the Weka user interface. The difference in performance between SMO and libSVM is not significant based on preliminary testing of both algorithms with the same dataset; therefore, to maintain consistency in using Weka implementation of machine learning algorithms, SMO was used for all experiments requiring SVM. Moreover, the terms SMO and SVM are interchangeable throughout the rest of this document.

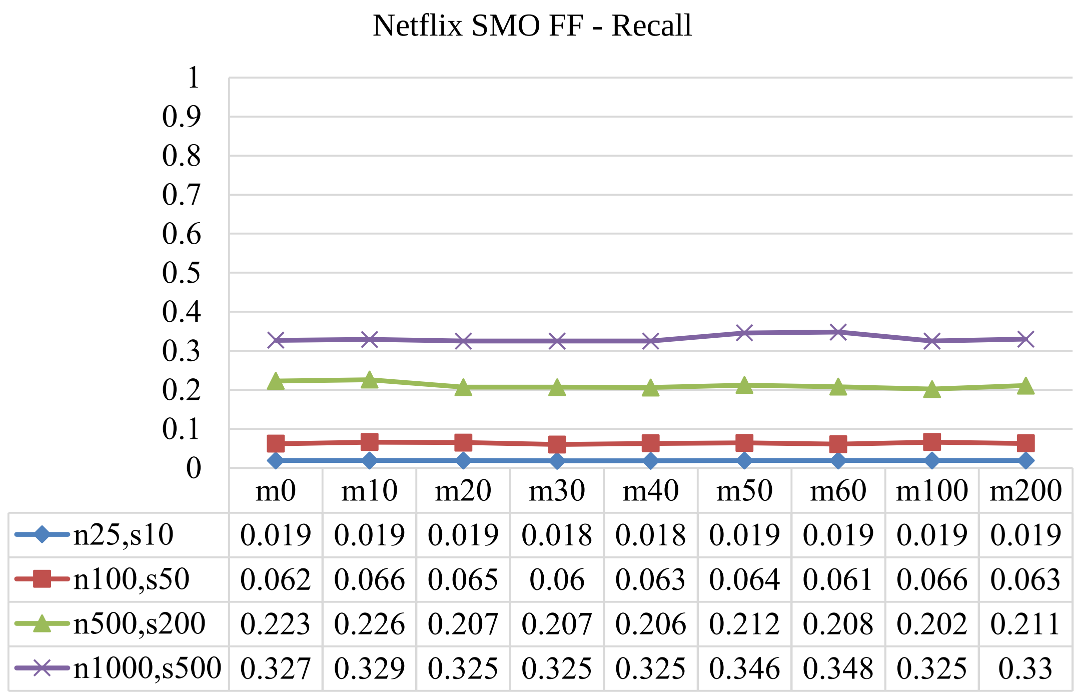
Configuring SMO entailed selecting normalization of training attributes and the use of a polykernel. The requirement for normalization was due to poor performance of SVM given the wide variance of values for the attributes used to train the algorithms. SVM may be negatively impacted by features that have a broad range of values, e.g. attribute-1 ranges from 1 – 4900, attribute-2 ranges 0 – 1 etc., across the total space of attributes ((Ben-Hur & Weston, 2010)). Normalization can, in some cases, reduce the effects of attribute variance ((W. Wang, Zhang, Gombault, & Knapskog, 2009)). Accordingly, SVM was trained with normalized values for all experiments for this research.

Figure 15 and 16 provides a summary of recall for the SVM model tested with various datasets of missing packets and varying window sizes: 25, 100, 500 and 1000 packets.



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ n25, s10	0.044	0.036	0.035	0.037	0.033	0.036	0.037	0.035	0.036
■ n100, s50	0.148	0.139	0.141	0.139	0.131	0.138	0.136	0.142	0.143
▲ n500, s200	0.6	0.425	0.395	0.451	0.424	0.431	0.43	0.426	0.44
× n1000, s500	0.612	0.584	0.589	0.591	0.587	0.579	0.557	0.597	0.607

Figure 15 Recall for YouTube SVM Full-flow Classifier Tested with Partial Flows

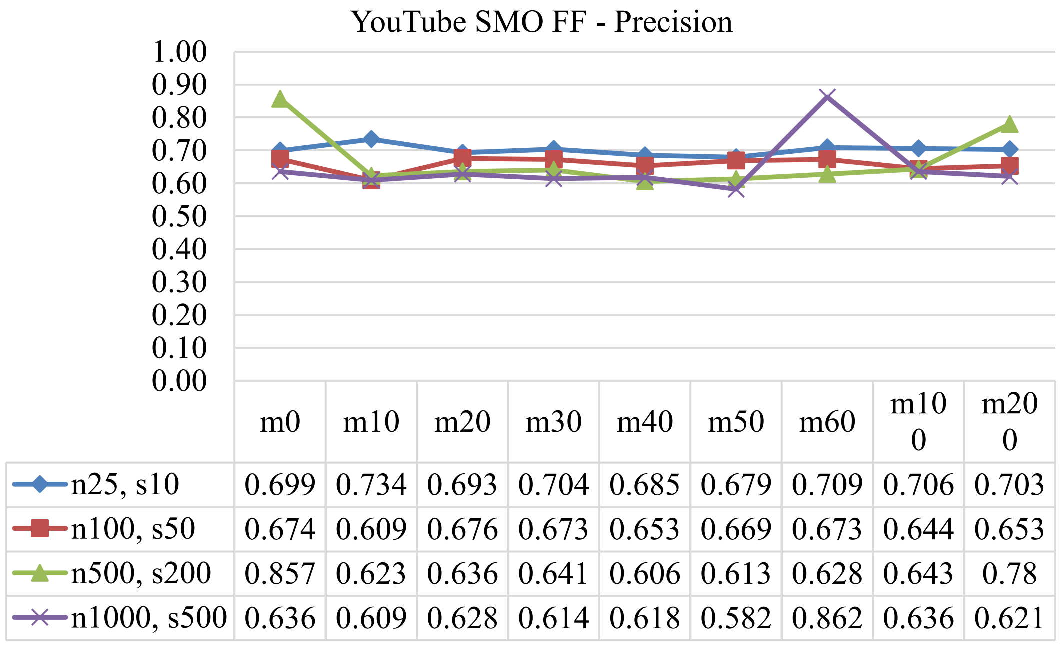


	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ n25,s10	0.019	0.019	0.019	0.018	0.018	0.019	0.019	0.019	0.019
■ n100,s50	0.062	0.066	0.065	0.06	0.063	0.064	0.061	0.066	0.063
▲ n500,s200	0.223	0.226	0.207	0.207	0.206	0.212	0.208	0.202	0.211
× n1000,s500	0.327	0.329	0.325	0.325	0.325	0.346	0.348	0.325	0.33

Figure 16 Recall for Netflix SVM Full-flow Classifier Tested with Partial Flows

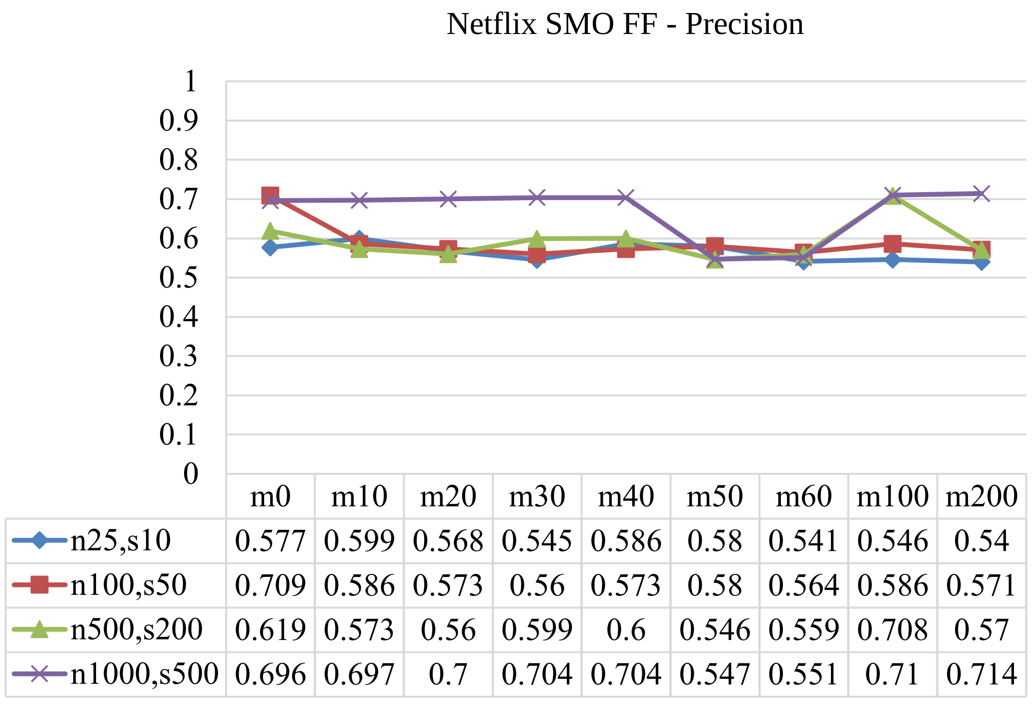
Results for SVM are low – the highest value is ~0.61 for YouTube traffic and even lower for Netflix traffic at ~0.35 -- although the values are reasonably consistent across the spectrum of datasets with missing packets from 0 – 200. Datasets containing window sizes of 1000 (n1000, s500) packets, provide the highest recall percentage relative to other datasets, indicating that larger sub-flow sizes provide better alignment to full-flow statistics for SVM. Worst performance is attributed to datasets with small sub-flow sizes, with "n25, s10", delivering the worst performance: ~0.035 for YouTube and ~0.019 for Netflix.

Precision for full flow SVM, as evidenced with previous full flow models, provides more consistent performance and higher performance levels. Figures 17 and 18 provide a summary of results for the full-flow trained SVM classifier tested with the same datasets.



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ n25, s10	0.699	0.734	0.693	0.704	0.685	0.679	0.709	0.706	0.703
■ n100, s50	0.674	0.609	0.676	0.673	0.653	0.669	0.673	0.644	0.653
▲ n500, s200	0.857	0.623	0.636	0.641	0.606	0.613	0.628	0.643	0.78
× n1000, s500	0.636	0.609	0.628	0.614	0.618	0.582	0.862	0.636	0.621

Figure 17 Precision for YouTube SVM Full-flow Classifier Tested with Partial Flows



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ n25,s10	0.577	0.599	0.568	0.545	0.586	0.58	0.541	0.546	0.54
■ n100,s50	0.709	0.586	0.573	0.56	0.573	0.58	0.564	0.586	0.571
▲ n500,s200	0.619	0.573	0.56	0.599	0.6	0.546	0.559	0.708	0.57
× n1000,s500	0.696	0.697	0.7	0.704	0.704	0.547	0.551	0.71	0.714

Figure 18 Precision for Netflix SVM Full-flow Classifier Tested with Partial Flows

In some cases, precision for YouTube reaches levels well above 0.80, although precision drops off as additional packets are deleted from the datasets. Netflix only reaches ~0.70, although uneven for certain values of “m”. Overall, precision results for SVM is similar to those of J48 and Naïve Bayes in terms of consistency and performance.

Summary

In this section testing confirmed that full flow classifiers have difficulty classifying traffic that contains partial flows. While in some cases, J48 C4.5 and Naïve Bayes, precision is high and consistent, better than 90%, recall is average at best (typically lower than ~0.70), and inconsistent. Classifiers with high precision are of little benefit when considering the large number of false negatives indicated by the low recall values, as this would lead to missing instances for target traffic when applied to real world implementations. Next, classifiers trained on sub-flows were evaluated with the same test datasets to determine if performance improves.

Sub-flow Trained Classifiers Applied to Partial Flows with Missing Packets

In the previous experiment, classifiers trained on full flows were tested against partial flows with packets missing, which resulted in average to poor performance for recall in addition to uneven results. For this part of the evaluation, classifiers trained on sub-flows were used to classify the same data sets containing partial flows with missing packets. Figure 19 provides an overview of the experimental process.

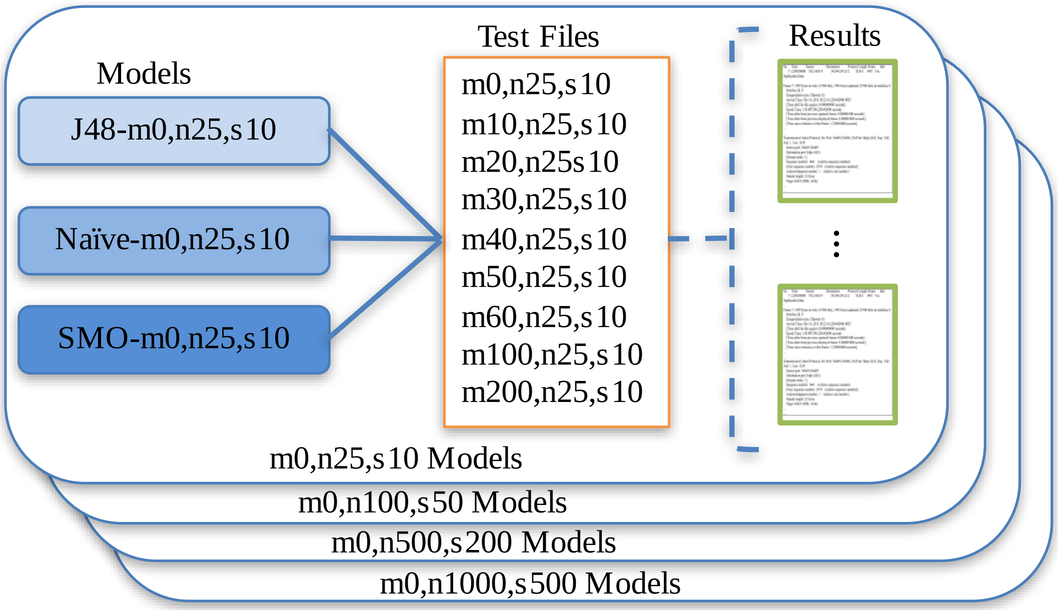


Figure 19 Process for Evaluating Sub-flow Models

To perform this evaluation, a J48 C4.5, Naïve Bayes and SMO SVM models were trained for each of four sub-flow sizes, “m0,n25,s10”, “m0,n100,s50”, “m0,n500s200” and “m0,n1000, s500” as depicted in Figure 19. The models were then tested against data sets of the same sub-flow size with missing packets from m0 to m200 (e.g. “m0,n25,s10” to “m200,n25,10”). This is an important distinction from the previous experiment that confirmed that full flow classifiers yielded both inconsistent and average to poor recall. In this evaluation, determining if sub-flow models perform better than full flow classifiers as well as which sub-flow model and associated machine learning algorithm performs best was the objective.

The process used for this experiment has real world application. In real world applications, collecting network traffic for a target sub-flow size for training a classifier, testing and evaluating new cases should be achievable relative to capturing the entire traffic flow from the beginning to end on real networks.

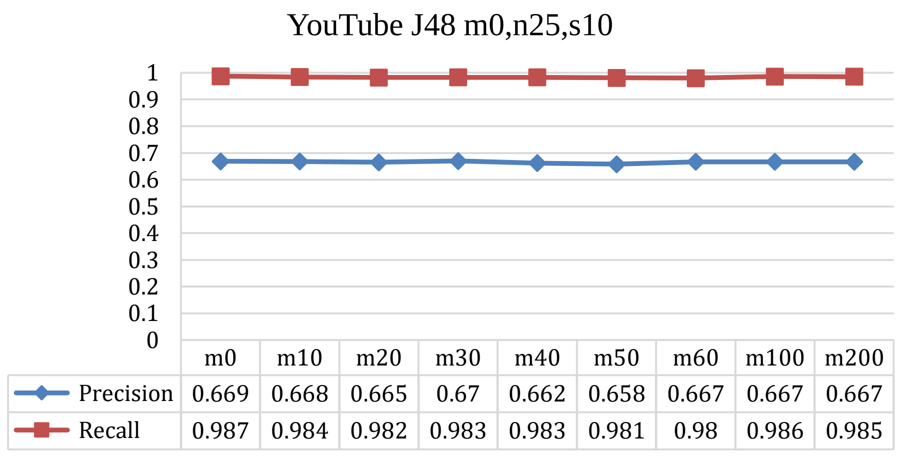
Sub-flow “m0,n25,s10” Model Evaluation

Table 7 provides a summary of results from testing a J48-m0,n25,s10, Naïve-m0,n25,s10 and SMO-m0,n25,s10 model for YouTube traffic. Each model was tested with data of the same sub-flow size with missing packets from m0 to m200, respectively. Results for J48 and Naïve are more consistent across the data set for recall and precision than the full flow test previously performed, even as the number of missing packets increases. While there are lower values for precision in this experiment, recall is now significantly higher than the full flow models tested against n25,s10 datasets previously. Note Table 7 also contains F-measure values along with precision and recall. F-measure, is the harmonic mean for precision and recall and is defined as: $2(Precision * Recall)/Precision + Recall$. A balanced F-measure was used for this research, and stipulated in the rest of this chapter to simplify comparison of results.

Table 7 YouTube m0,n25,s10 Model Results

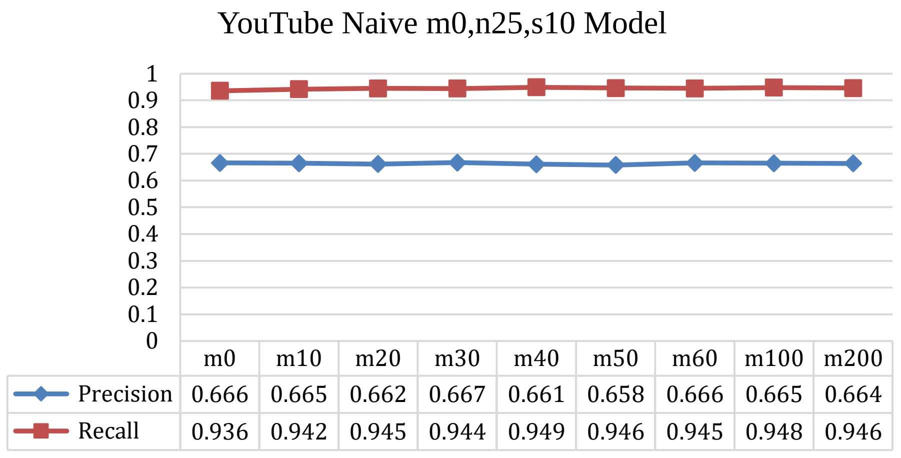
	J48-m0,n25,s10			Naïve-m0,n25,s10			SMO-m0,n25,s10		
YT	Prec.	Rec.	F-Mea	Prec.	Rec.	F-Mea	Prec.	Rec.	F-Mea
m0	0.669	0.987	0.798	0.666	0.778	0.936	0.64	0.974	0.773
m10	0.668	0.984	0.796	0.665	0.78	0.942	0.642	0.979	0.776
m20	0.665	0.982	0.793	0.662	0.779	0.945	0.637	0.98	0.772
m30	0.67	0.983	0.797	0.667	0.782	0.944	0.645	0.981	0.778
m40	0.662	0.983	0.791	0.661	0.779	0.949	0.637	0.98	0.772
m50	0.658	0.981	0.788	0.658	0.776	0.946	0.633	0.979	0.769
m60	0.667	0.98	0.794	0.666	0.781	0.945	0.641	0.981	0.776
m100	0.667	0.986	0.796	0.665	0.782	0.948	0.643	0.981	0.776
m200	0.667	0.985	0.795	0.664	0.78	0.946	0.641	0.978	0.774

In order to quickly assess which algorithm performs best for n25,s10 sub-flow size, Figure 20 – 22 provide a graphical representation of precision and recall for the results in Table 7. Both J48 and Naïve Bayes have similar plots for precision and recall, where recall stays relatively stable in the mid 0.90s, and precision hovers at ~0.70. While these results are an improvement over the full flow test, the implication is that a sub-flow size of n25,s10 may not be best suited for classifying YouTube traffic.



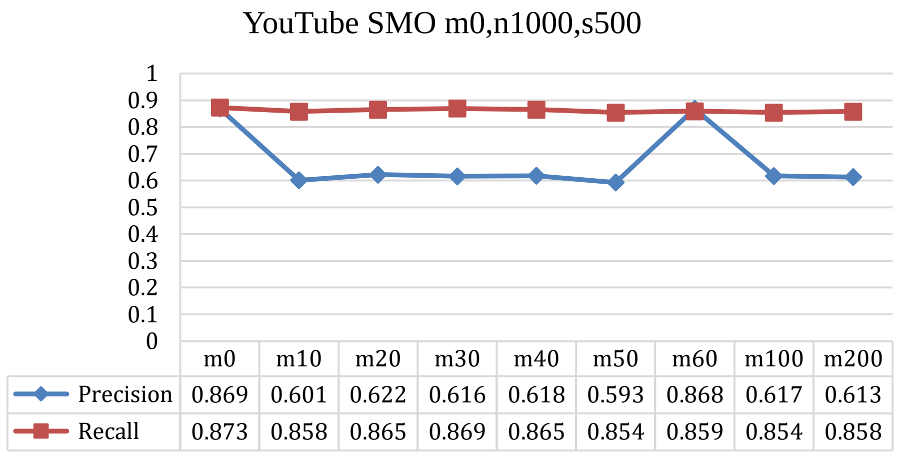
	m0	m10	m20	m30	m40	m50	m60	m100	m200
Precision	0.669	0.668	0.665	0.67	0.662	0.658	0.667	0.667	0.667
Recall	0.987	0.984	0.982	0.983	0.983	0.981	0.98	0.986	0.985

Figure 20 YouTube J48 m0,n25,s10 Model



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ Precision	0.666	0.665	0.662	0.667	0.661	0.658	0.666	0.665	0.664
■ Recall	0.936	0.942	0.945	0.944	0.949	0.946	0.945	0.948	0.946

Figure 21 YouTube Naive m0,n25,s10 Model



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ Precision	0.869	0.601	0.622	0.616	0.618	0.593	0.868	0.617	0.613
■ Recall	0.873	0.858	0.865	0.869	0.865	0.854	0.859	0.854	0.858

Figure 22 YouTube SMO m0,n25,s10 Model

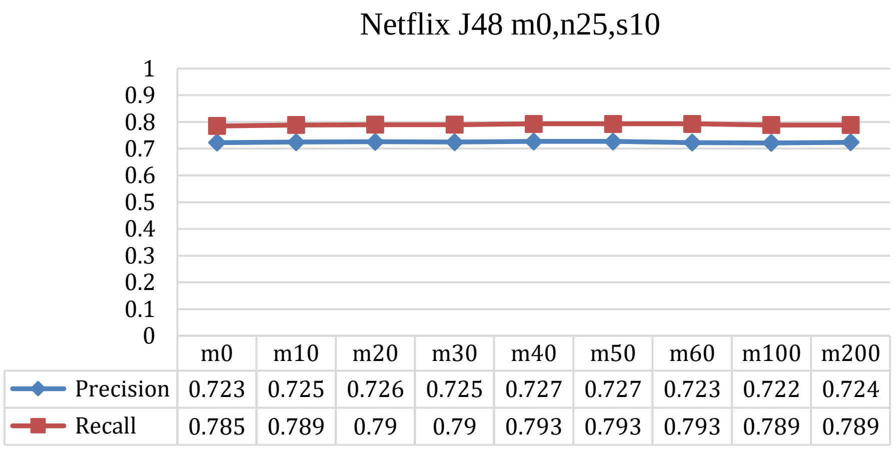
For Netflix traffic, Table 8 summarizes the results for testing J48, Naïve Bayes, and SMO SVM. Testing for Netflix traffic for “m0,n25,s10” models followed the same process as YouTube.

Table 8 Netflix m0,n25,s10 Model Results

	J48-m0,n25,s10			Naïve-m0,n25,s10			SMO-m0,n25,s10		
NF	Prec.	Rec.	F-Mea.	Prec.	Rec.	F-Mea.	Prec.	Rec.	F-Mea.
m0	0.723	0.785	0.753	0.643	0.971	0.773	0.601	0.049	0.091
m10	0.725	0.789	0.755	0.645	0.971	0.775	0.603	0.049	0.09
m20	0.726	0.79	0.757	0.646	0.971	0.776	0.61	0.05	0.092
m30	0.725	0.79	0.756	0.645	0.973	0.776	0.617	0.05	0.092

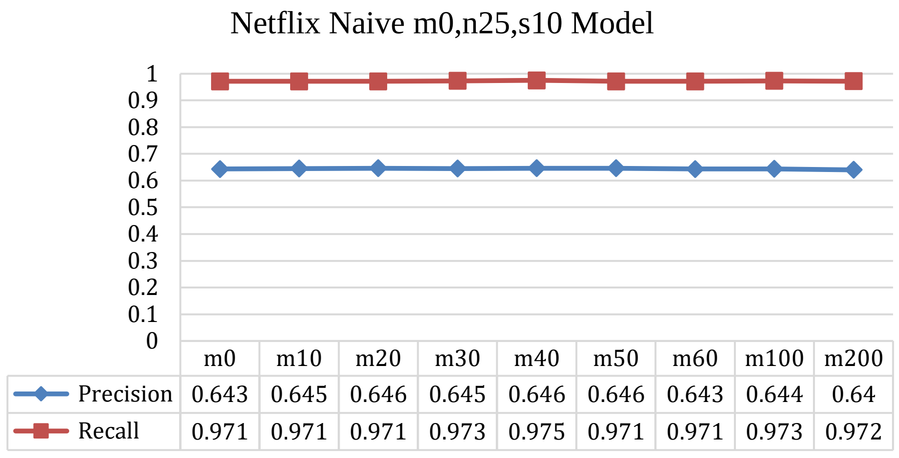
m40	0.727	0.793	0.759	0.646	0.975	0.777	0.613	0.047	0.087
m50	0.727	0.793	0.759	0.646	0.971	0.776	0.582	0.046	0.086
m60	0.723	0.793	0.757	0.643	0.971	0.774	0.574	0.047	0.087
m100	0.722	0.789	0.754	0.644	0.973	0.775	0.603	0.048	0.089
m200	0.724	0.789	0.755	0.64	0.972	0.772	0.612	0.049	0.091

Figures 23 - 25 provide a graphical representation of precision and recall in Table 8. As experienced previously with YouTube, the results are more consistent than full flow models; however, SVM performs particularly poorly for both precision and recall. J48 and Naïve Bayes performed similarly with average to poor results, with the best results at ~0.80 and ~0.72, respectively.



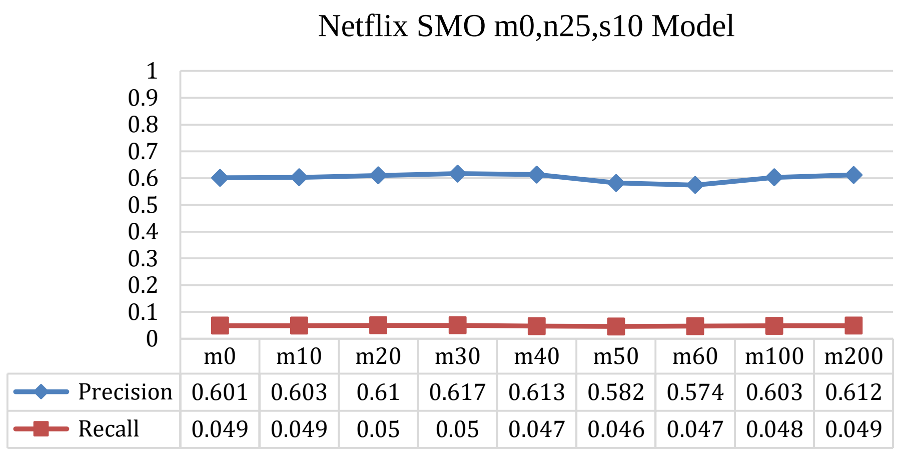
	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ Precision	0.723	0.725	0.726	0.725	0.727	0.727	0.723	0.722	0.724
■ Recall	0.785	0.789	0.79	0.79	0.793	0.793	0.793	0.789	0.789

Figure 23 Netflix J48 m0,n25,s10 Results



	m0	m10	m20	m30	m40	m50	m60	m100	m200
Precision	0.643	0.645	0.646	0.645	0.646	0.646	0.643	0.644	0.64
Recall	0.971	0.971	0.971	0.973	0.975	0.971	0.971	0.973	0.972

Figure 24 Netflix Naive m0,n25,s10 Results



	m0	m10	m20	m30	m40	m50	m60	m100	m200
Precision	0.601	0.603	0.61	0.617	0.613	0.582	0.574	0.603	0.612
Recall	0.049	0.049	0.05	0.05	0.047	0.046	0.047	0.048	0.049

Figure 25 Netflix SMO m0,n25,s10 Results

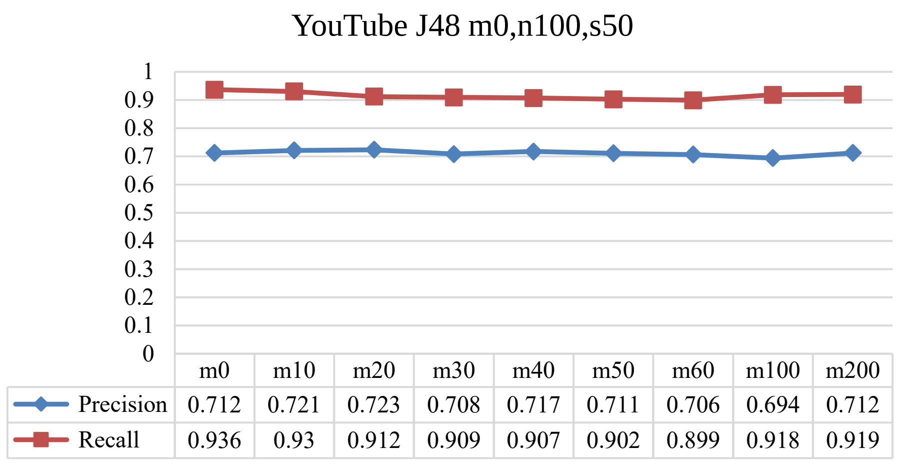
Sub-flow “m0,n100,s50” Model Evaluation

In this experiment results for the J48-m0,n100,s50, Naïve-m0,n100,s50 and SMO-m0,n100,s50 for YouTube and Netflix traffic classes are presented. In general, the results in Table 9 are similar to those in Table 8, m0,n25,s10, in terms of consistency and F-measure values for J48 and Naïve Bayes, although, SMO SVM improved significantly for this sub-flow size, n100,s50.

Table 9 YouTube m0,n100,s50 Model Results

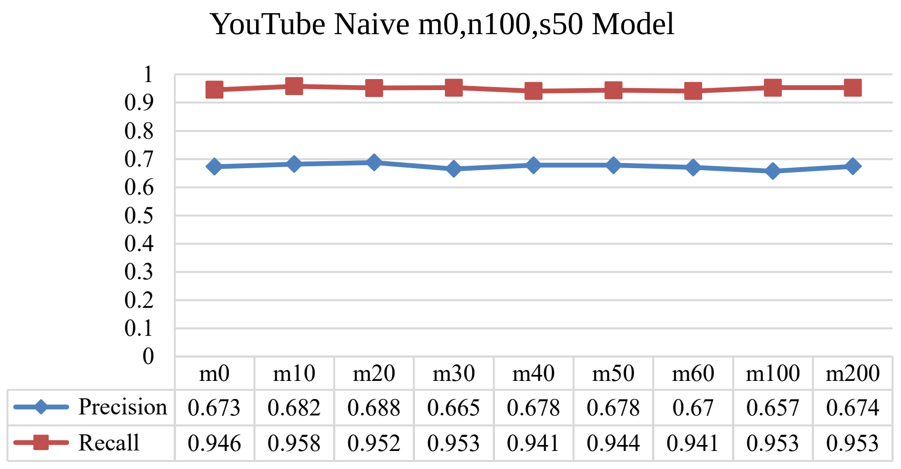
	J48-m0,n100,s50			Naïve-m0n100,s50			SMO-m0, n100,s50		
YT	Prec.	Rec.	F-Mea.	Prec.	Rec.	F-Mea.	Prec.	Rec.	F-Mea.
m0	0.712	0.936	0.809	0.673	0.946	0.787	0.662	0.891	0.76
m10	0.721	0.93	0.812	0.682	0.958	0.796	0.669	0.894	0.765
m20	0.723	0.912	0.807	0.688	0.952	0.799	0.674	0.887	0.766
m30	0.708	0.909	0.796	0.665	0.953	0.784	0.652	0.883	0.75
m40	0.717	0.907	0.801	0.678	0.941	0.788	0.665	0.881	0.758
m50	0.711	0.902	0.795	0.678	0.944	0.789	0.668	0.889	0.763
m60	0.706	0.899	0.791	0.67	0.941	0.783	0.66	0.885	0.756
m100	0.694	0.918	0.791	0.657	0.953	0.778	0.642	0.899	0.749
m200	0.712	0.919	0.802	0.674	0.953	0.79	0.662	0.898	0.762

Figures 26 – 28 provide graphical depictions of precision and recall. Note the improved precision for SVM in Figure 28: ~0.65 across the entire dataset. The plot also indicates all models are more consistent across the datasets. Overall J48,m100,s50 performs the best for YouTube traffic at this sub-flow size.



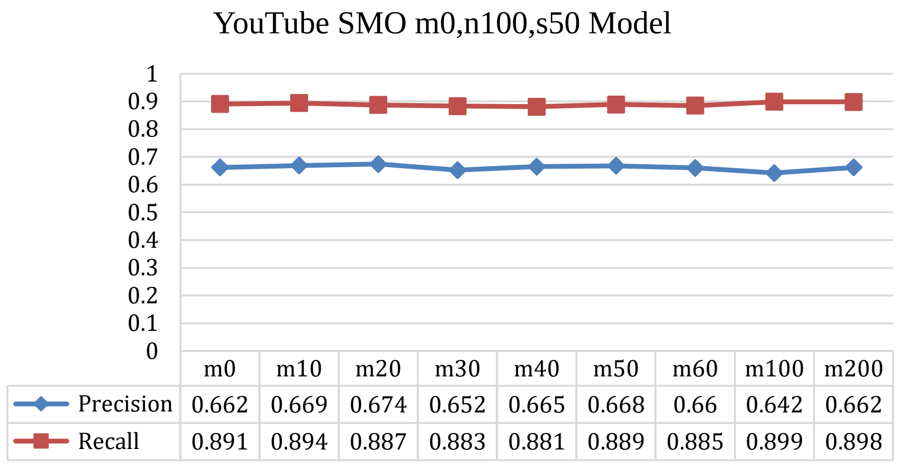
	m0	m10	m20	m30	m40	m50	m60	m100	m200
Precision	0.712	0.721	0.723	0.708	0.717	0.711	0.706	0.694	0.712
Recall	0.936	0.93	0.912	0.909	0.907	0.902	0.899	0.918	0.919

Figure 26 YouTube J48 m0,n100,s50 Results



	m0	m10	m20	m30	m40	m50	m60	m100	m200
Precision	0.673	0.682	0.688	0.665	0.678	0.678	0.67	0.657	0.674
Recall	0.946	0.958	0.952	0.953	0.941	0.944	0.941	0.953	0.953

Figure 27 YouTube Naive m0,n100,s50 Results



	m0	m10	m20	m30	m40	m50	m60	m100	m200
Precision	0.662	0.669	0.674	0.652	0.665	0.668	0.66	0.642	0.662
Recall	0.891	0.894	0.887	0.883	0.881	0.889	0.885	0.899	0.898

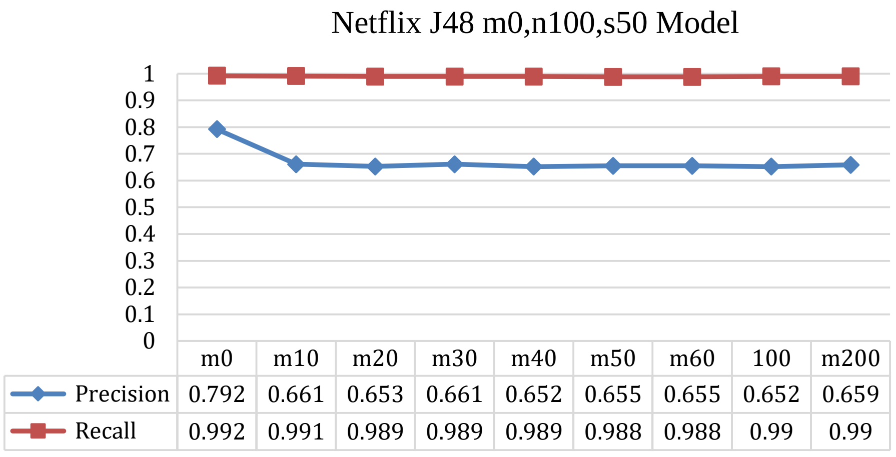
Figure 28 YouTube SMO m0,n100,s50 Results

Netflix results are detailed in Table 10. Of note is the recall of 1 for SMO model. A model that produces a 100% recall may indicate a problem with either the convergence of the algorithm or possibly over fitting. Since the data used for SMO is the same as J48 and Naïve Bayes, it is more likely that the model is suspect.

Table 10 Netflix m0,n100,s50 Model Results

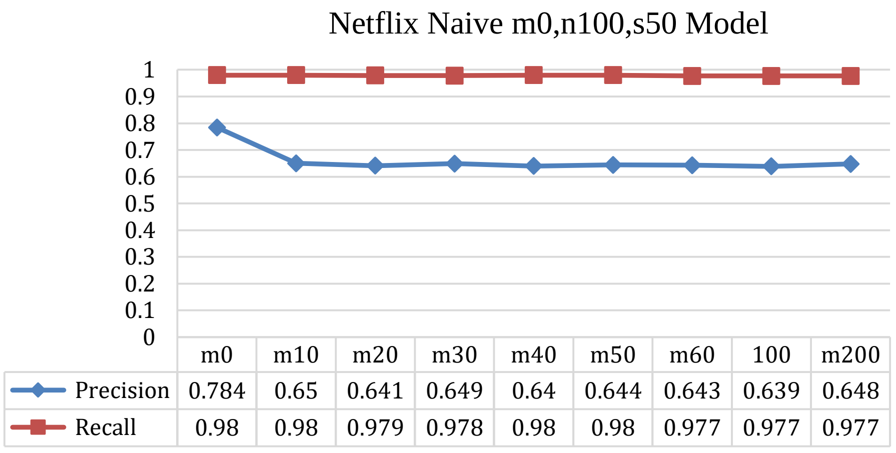
	J48-m0,n100,s50			Naive-m0,n100,s50			SMO-m0,n100,s50		
NT	Prec	Rec	F-Mea	Prec	Rec	F-Mea	597Prec	Rec	F-Mea
m0	0.792	0.992	0.88	0.784	0.98	0.871	0.604	1	0.753
m10	0.661	0.991	0.793	0.65	0.98	0.782	0.434	1	0.605
m20	0.653	0.989	0.786	0.641	0.979	0.775	0.429	1	0.601
m30	0.661	0.989	0.792	0.649	0.978	0.78	0.432	1	0.603
m40	0.652	0.989	0.786	0.64	0.98	0.774	0.426	1	0.
m50	0.655	0.988	0.788	0.644	0.98	0.777	0.427	1	0.599
m60	0.655	0.988	0.787	0.643	0.977	0.776	0.432	1	0.604
m100	0.652	0.99	0.786	0.639	0.977	0.773	0.427	1	0.598
m200	0.659	0.99	0.792	0.648	0.977	0.779	0.428	1	0.599

Figures 29 – 31 provide a graphical depiction of the precision and recall results detailed in Table 10. Visually, the patterns related to the consistency of the model track across model types; the performance is best for J48 for Netflix at this sub-flow size.



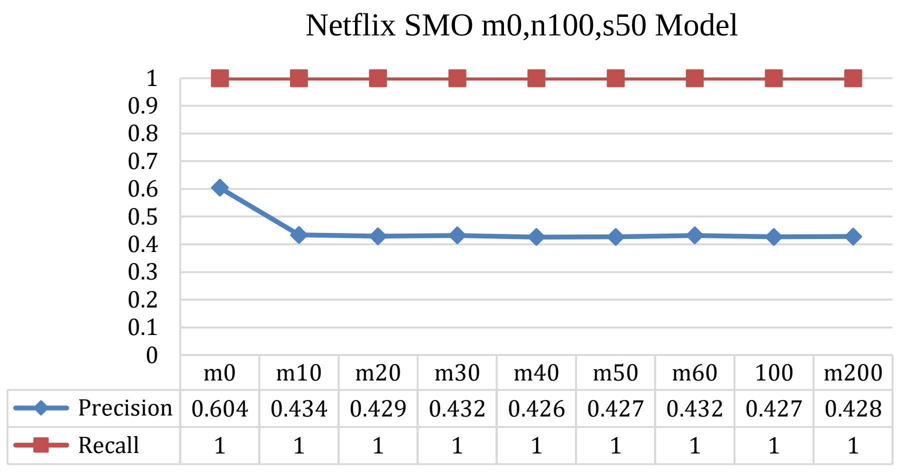
	m0	m10	m20	m30	m40	m50	m60	100	m200
Precision	0.792	0.661	0.653	0.661	0.652	0.655	0.655	0.652	0.659
Recall	0.992	0.991	0.989	0.989	0.989	0.988	0.988	0.99	0.99

Figure 29 Netflix J48 m0,n100,s50 Model



	m0	m10	m20	m30	m40	m50	m60	100	m200
Precision	0.784	0.65	0.641	0.649	0.64	0.644	0.643	0.639	0.648
Recall	0.98	0.98	0.979	0.978	0.98	0.98	0.977	0.977	0.977

Figure 30 Netflix Naïve m0,n100,s50 Model



	m0	m10	m20	m30	m40	m50	m60	100	m200
Precision	0.604	0.434	0.429	0.432	0.426	0.427	0.432	0.427	0.428
Recall	1	1	1	1	1	1	1	1	1

Figure 31 Netflix m0,n100,s50 Model

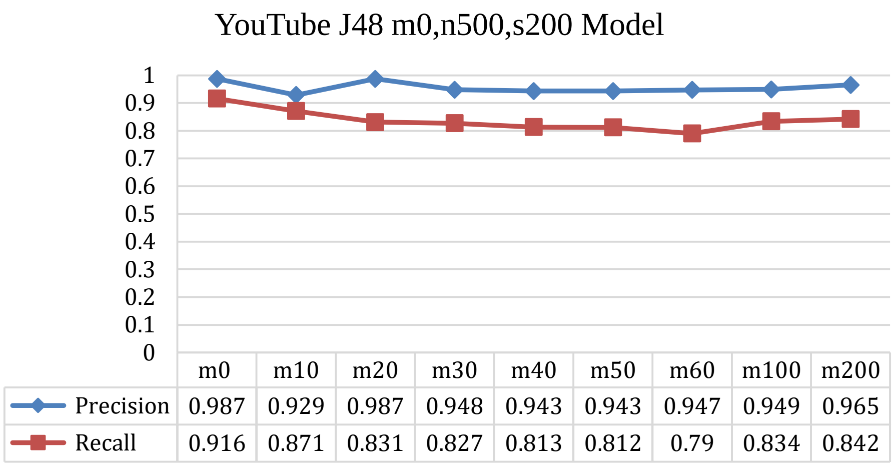
Sub-flow "m0,n500,s200" Model Evaluation

Table 11 provides results from testing J48, Naïve Bayes and SMO m0,n500,s200 models. Recall and precision has increased with values reaching into the mid 80s and 90s for J48. The F-measure for J48 has also increase well beyond full flow models tested previously. Naïve Bayes closely tracks to J48 in performance. While SMO improved over m0,n25,s10 and m0,n100,s50 models, the results are still below average.

Table 11 YouTube m0,n500,s200 Model Results

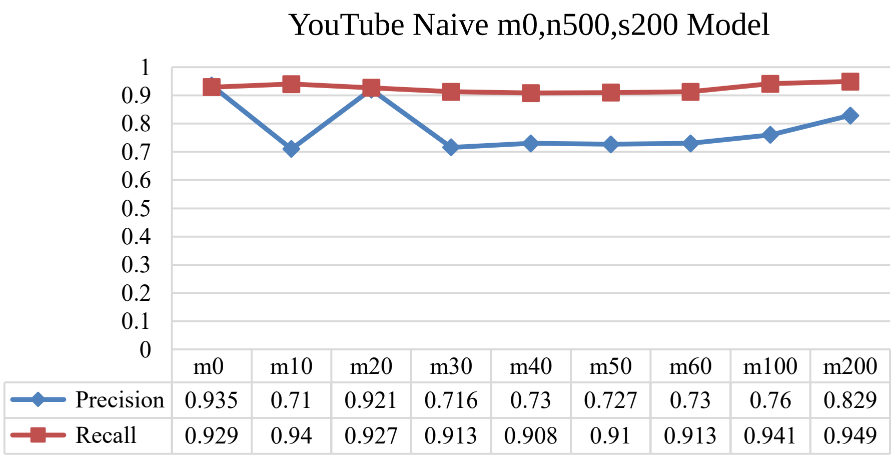
	J48-m0,n500,s200			Naive-m0,n500,s200			SMO-m0,n500,s200		
YT	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea
m0	0.987	0.916	0.95	0.935	0.929	0.932	0.93	0.632	0.753
m10	0.929	0.871	0.899	0.71	0.94	0.809	0.828	0.564	0.671
m20	0.987	0.831	0.902	0.921	0.927	0.924	0.955	0.54	0.69
m30	0.948	0.827	0.884	0.716	0.913	0.803	0.84	0.524	0.646
m40	0.943	0.813	0.873	0.73	0.908	0.809	0.83	0.521	0.64
m50	0.943	0.812	0.873	0.727	0.91	0.808	0.844	0.537	0.656
m60	0.947	0.79	0.861	0.73	0.913	0.812	0.879	0.56	0.684
m100	0.949	0.834	0.888	0.76	0.941	0.841	0.88	0.547	0.675
m200	0.965	0.842	0.899	0.829	0.949	0.885	0.92	0.569	0.703

Figure 32 provides a plot for the J48 “m0,n500,s200” model which illustrates good results for this model at this sub-flow size. There is a slight perturbation at m20; however, the performance is relatively consistent throughout. Naïve Bayes, Figure 32, also has good performance. SMO SVM is worst at this sub-flow size relative to J48 and Naïve Bayes. Indications are that a sub-flow size of n500,s200 may be suitable for classifying YouTube traffic.



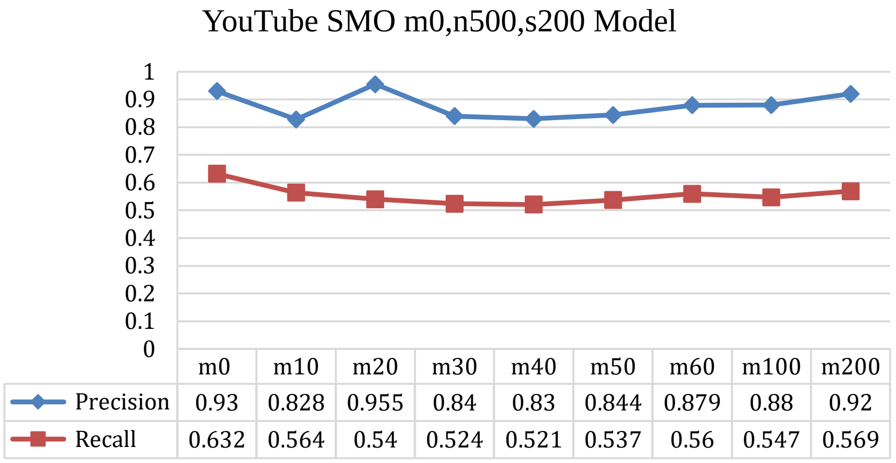
	m0	m10	m20	m30	m40	m50	m60	m100	m200
Precision	0.987	0.929	0.987	0.948	0.943	0.943	0.947	0.949	0.965
Recall	0.916	0.871	0.831	0.827	0.813	0.812	0.79	0.834	0.842

Figure 32 J48 m0,n500,s200 Model



	m0	m10	m20	m30	m40	m50	m60	m100	m200
Precision	0.935	0.71	0.921	0.716	0.73	0.727	0.73	0.76	0.829
Recall	0.929	0.94	0.927	0.913	0.908	0.91	0.913	0.941	0.949

Figure 33 Netflix Naïve m0,n500,s200 Model



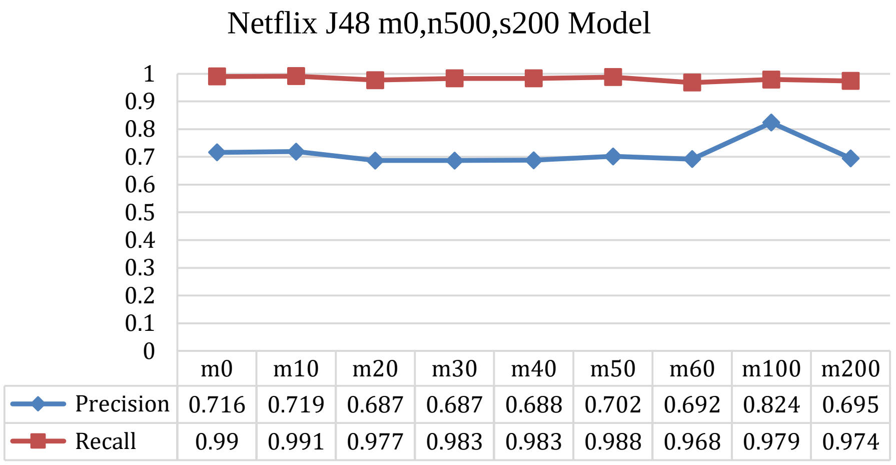
	m0	m10	m20	m30	m40	m50	m60	m100	m200
Precision	0.93	0.828	0.955	0.84	0.83	0.844	0.879	0.88	0.92
Recall	0.632	0.564	0.54	0.524	0.521	0.537	0.56	0.547	0.569

Figure 34 Netflix SMO m0,n500,s200 Model

Performance for J48, Naïve Bayes, and SMO for Netflix are detailed in Table 12. The results indicated that for a sub-flow size of "m0,n500,s200", all three models performed about average – ~0.70 for J48 and Naïve, and relatively poor for SMO. Figures 35 – 37 provide a graphical representation of precision and recall. In general results are consistent across the spectrum of datasets, with a slight uptick at m100.

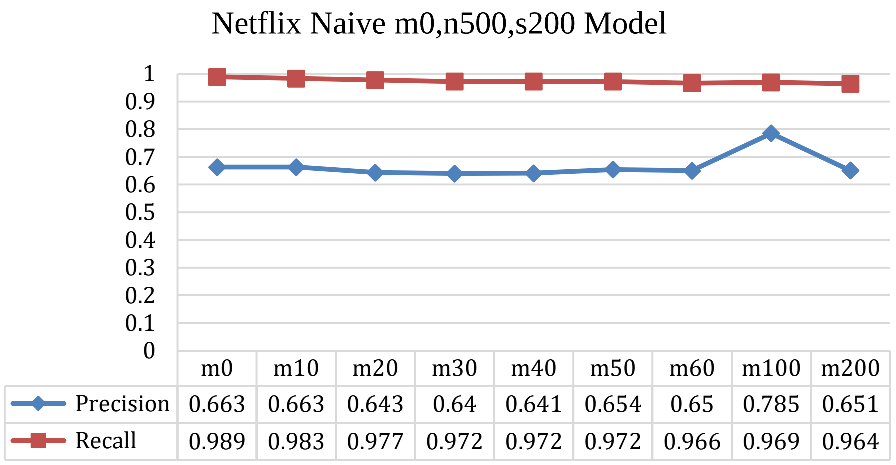
Table 12 Netflix m0,n500,s200 Model Results

	J48-m0,n500,s200			Naive-m0,n500,s200			SMO-m0,n500,s200		
NT	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea
m0	0.716	0.99	0.831	0.663	0.989	0.794	0.451	1	0.621
m10	0.719	0.991	0.834	0.663	0.983	0.792	0.441	1	0.612
m20	0.687	0.977	0.807	0.643	0.977	0.775	0.432	1	0.603
m30	0.687	0.983	0.809	0.64	0.972	0.772	0.425	0.999	0.596
m40	0.688	0.983	0.809	0.641	0.972	0.773	0.425	0.999	0.596
m50	0.702	0.988	0.821	0.654	0.972	0.782	0.424	0.999	0.595
m60	0.692	0.968	0.807	0.65	0.966	0.777	0.431	0.999	0.602
m100	0.824	0.979	0.895	0.785	0.969	0.867	0.597	1	0.748
m200	0.695	0.974	0.811	0.651	0.964	0.777	0.432	1	0.604



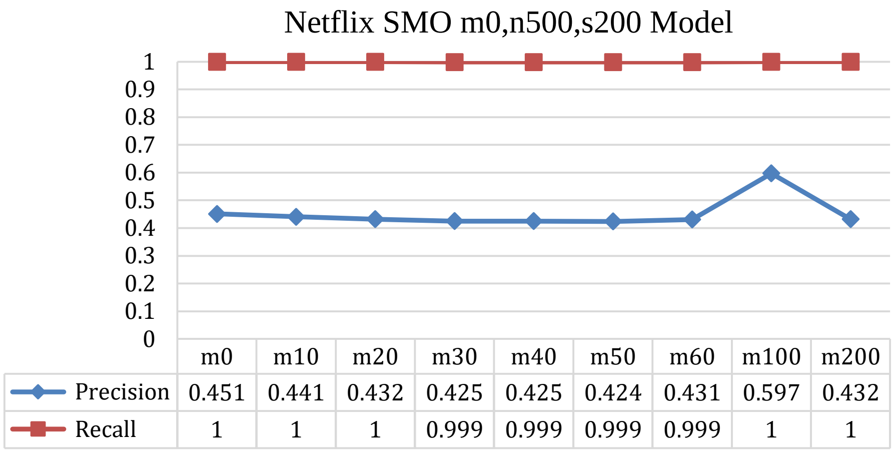
	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ Precision	0.716	0.719	0.687	0.687	0.688	0.702	0.692	0.824	0.695
■ Recall	0.99	0.991	0.977	0.983	0.983	0.988	0.968	0.979	0.974

Figure 35 Netflix J48 m0,n500,s200 Model



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ Precision	0.663	0.663	0.643	0.64	0.641	0.654	0.65	0.785	0.651
■ Recall	0.989	0.983	0.977	0.972	0.972	0.972	0.966	0.969	0.964

Figure 36 Netflix Naïve m0,n500,s200 Model



	m0	m10	m20	m30	m40	m50	m60	m100	m200
Precision	0.451	0.441	0.432	0.425	0.425	0.424	0.431	0.597	0.432
Recall	1	1	1	0.999	0.999	0.999	0.999	1	1

Figure 37 Netflix SMO m0,n500,s200 Model

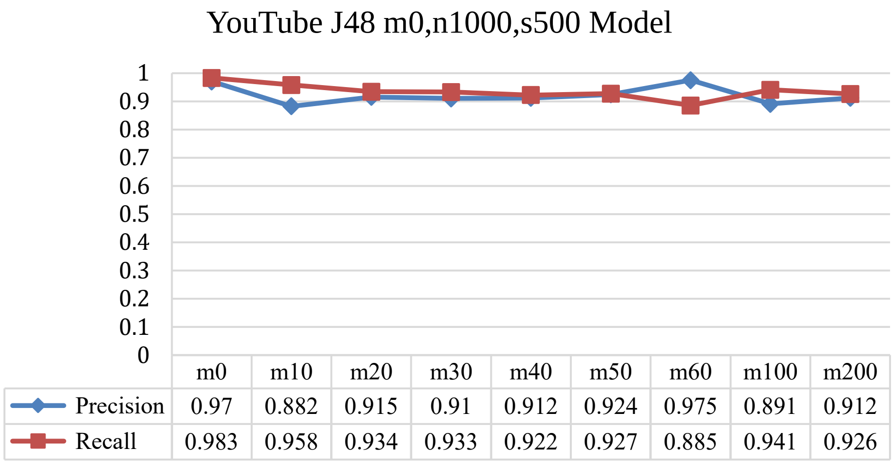
Sub-flow "m0,n1000,s500" Model Evaluation

Given the trend toward increased performance as sub-flow size increases, the expectation is that "m0,n1000,s500" for J48, Naïve Bayes and SMO models should continue to improve in terms of precision and recall. Table 13 indicates that performance has increased appreciably for the YouTube traffic class with precision and recall of ~0.90 and ~0.92, respectively. Moreover, the models performance is relatively stable across the 9 different test datasets with missing packets.

Table 13 YouTube m0,n1000,s500 Model Results

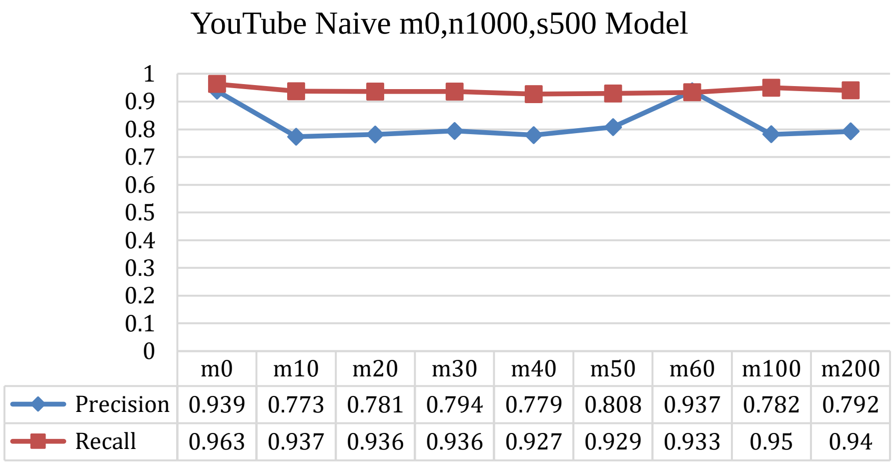
	J48-m0,n1000,n500			Naïve-m0,n1000,n500			SMO-m0,n1000,n500		
YT	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea
m0	0.97	0.983	0.976	0.939	0.963	0.951	0.869	0.873	0.871
m10	0.882	0.958	0.919	0.773	0.937	0.847	0.601	0.858	0.707
m20	0.915	0.934	0.925	0.781	0.936	0.852	0.622	0.865	0.723
m30	0.91	0.933	0.921	0.794	0.936	0.859	0.616	0.869	0.721
m40	0.912	0.922	0.917	0.779	0.927	0.847	0.618	0.865	0.721
m50	0.924	0.927	0.926	0.808	0.929	0.865	0.593	0.854	0.7
m60	0.975	0.885	0.928	0.937	0.933	0.935	0.868	0.859	0.863
m100	0.891	0.941	0.915	0.782	0.95	0.858	0.617	0.854	0.716
m200	0.912	0.926	0.919	0.792	0.94	0.86	0.613	0.858	0.715

As depicted in Figure 38, J48 has excellent and consistent performance overall. The implication being that J48-m0,n1000,s500 is well suited for classifying YouTube traffic with missing packets. Naïve Bayes, Figure 39, performed well with precision of ~0.80 and recall of ~0.90; SMO, Figure 40, performed below average with precision of ~0.60.



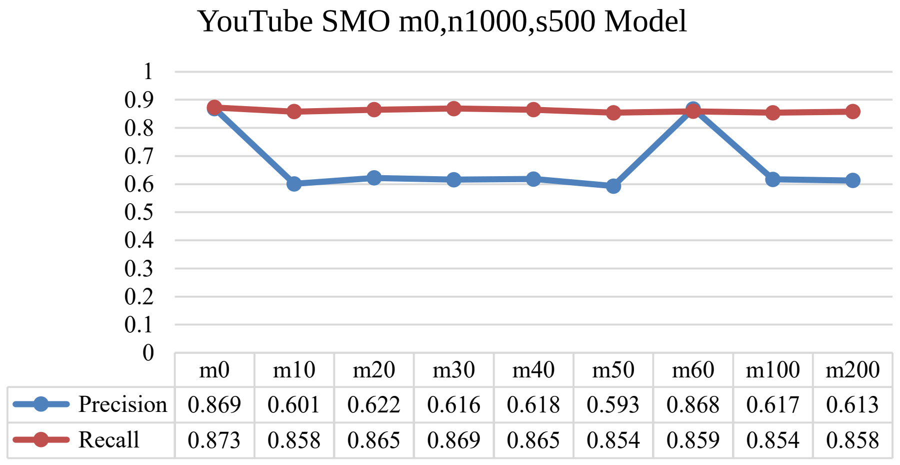
	m0	m10	m20	m30	m40	m50	m60	m100	m200
Precision	0.97	0.882	0.915	0.91	0.912	0.924	0.975	0.891	0.912
Recall	0.983	0.958	0.934	0.933	0.922	0.927	0.885	0.941	0.926

Figure 38 YouTube J48 m0,n1000,s500 Model



	m0	m10	m20	m30	m40	m50	m60	m100	m200
Precision	0.939	0.773	0.781	0.794	0.779	0.808	0.937	0.782	0.792
Recall	0.963	0.937	0.936	0.936	0.927	0.929	0.933	0.95	0.94

Figure 39 YouTube Naive m0,n1000,s500 Model



	m0	m10	m20	m30	m40	m50	m60	m100	m200
Precision	0.869	0.601	0.622	0.616	0.618	0.593	0.868	0.617	0.613
Recall	0.873	0.858	0.865	0.869	0.865	0.854	0.859	0.854	0.858

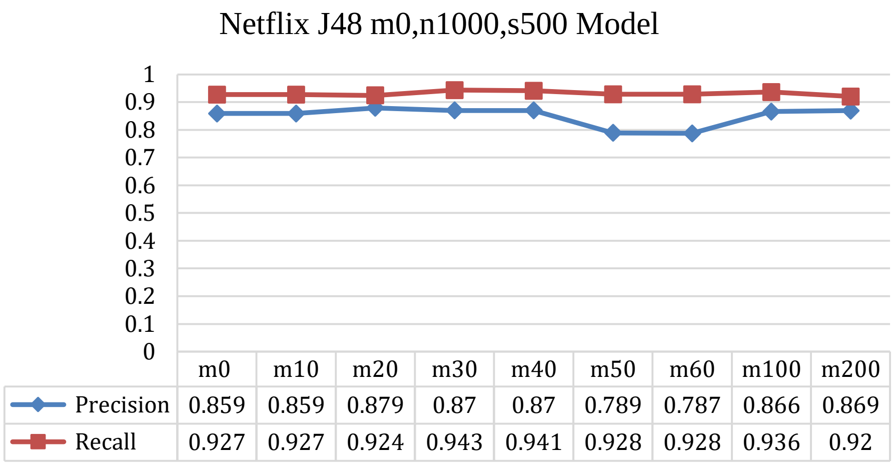
Figure 40 YouTube SMO m0,n1000,s500 Model

Table 14 details results for Netflix J48, Naïve Bayes and SMO for sub-flow m0,n1000,s500. J48 and Naïve Bayes performed very well with similar values for precision and recall: ~0.85 and ~0.92 across the datasets. F-measure values J48 and Naïve Bayes hover at ~0.88 and ~0.85, respectively, which suggest J48 performs slightly better that Naïve Bayes.

Table 14 Netflix m0,n1000,s500 Model Results

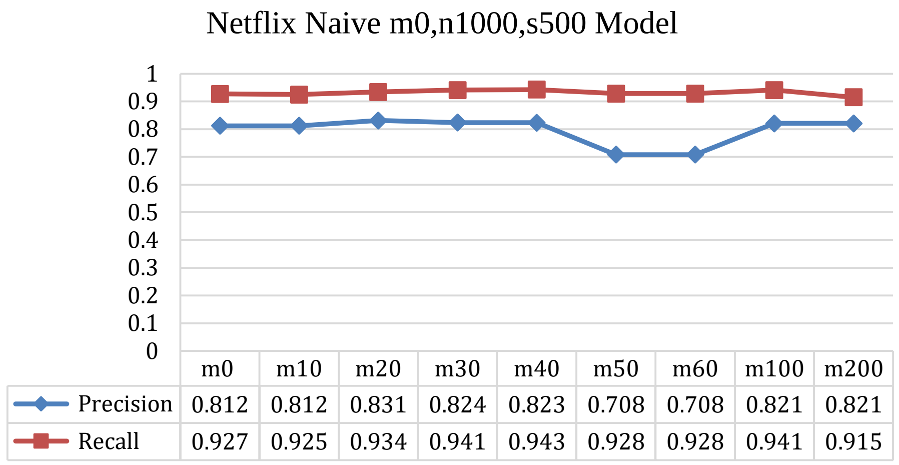
	J48-m0,n1000,s500			Naive-m0,n1000,s500			SMO-m0,n1000,s500		
NF	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea
m0	0.859	0.927	0.892	0.812	0.927	0.865	0.609	0.913	0.731
m10	0.859	0.927	0.892	0.812	0.925	0.865	0.61	0.913	0.731
m20	0.879	0.924	0.901	0.831	0.934	0.88	0.593	0.919	0.721
m30	0.87	0.943	0.905	0.824	0.941	0.879	0.611	0.911	0.732
m40	0.87	0.941	0.904	0.823	0.943	0.879	0.612	0.912	0.732
m50	0.789	0.928	0.853	0.708	0.928	0.803	0.448	0.888	0.595
m60	0.787	0.928	0.852	0.708	0.928	0.803	0.448	0.891	0.596
m100	0.866	0.936	0.9	0.821	0.941	0.877	0.612	0.911	0.732
m200	0.869	0.92	0.894	0.821	0.915	0.865	0.624	0.91	0.741

Figures 41 – 43 provide a graphical overview of precision and recall for each model. The graph illustrates how closely the plots from all 3 models track, although the magnitude of each value is quite different. All models experience a small depression when missing packets reach 50 and 60. The loss of packets m50 and m60 is affecting sub-flow statistics, which impacts the classification of both datasets. As the number of missing packets increases the effect is lessened because most of the initial packets that are used to sync communications are most likely outside the sub-flow window. Overall the plots indicate J48 provides more consistent and higher performance relative to other models for the n1000,s500 sub-flow size for the Netflix traffic class.



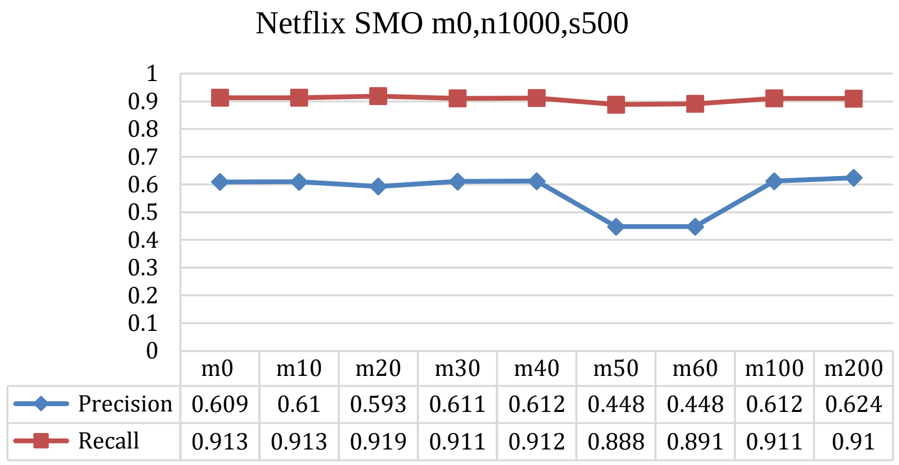
	m0	m10	m20	m30	m40	m50	m60	m100	m200
Precision	0.859	0.859	0.879	0.87	0.87	0.789	0.787	0.866	0.869
Recall	0.927	0.927	0.924	0.943	0.941	0.928	0.928	0.936	0.92

Figure 41 Netflix J48 m0,n1000,s500 Model



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ Precision	0.812	0.812	0.831	0.824	0.823	0.708	0.708	0.821	0.821
■ Recall	0.927	0.925	0.934	0.941	0.943	0.928	0.928	0.941	0.915

Figure 42 Netflix Naive m0,n1000,s500 Model



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ Precision	0.609	0.61	0.593	0.611	0.612	0.448	0.448	0.612	0.624
■ Recall	0.913	0.913	0.919	0.911	0.912	0.888	0.891	0.911	0.91

Figure 43 Netflix SMO m0,n1000,s500 Model

Summary

In this section, results were presented for sub-flow trained classifiers tested with datasets of the same sub-flow size with missing packets. Using models and datasets of the same sub-flow size was a direct outcome from evaluation of full flow models with different sub-flow sizes which returned poor results. Furthermore, testing of different machine learning algorithms (J48, Naïve Bayes, and SMO for 4 different sub-flow sizes, “n25,s10”, “n100,s50”, “n500,s200”, and “n1000,s500”), in order to identify the best sub-flow classifier, in terms of performance, for YouTube and Netflix was also performed. For both YouTube and Netflix traffic classes the “J48-m0,n1000,s500” model performed best.

The key outcome from this evaluation is that a sub-flow size along with an ML algorithm have been identified that provides very good (J48 for Netflix) and in some cases excellent (J48 for YouTube traffic) overall performance, while eliminating the need for collecting data for the entire network flow. In the next section, ensemble techniques were applied to each algorithm – J48, Naïve Bayes and SMO SVM – to determine if performance, precision, and recall, can be improved.

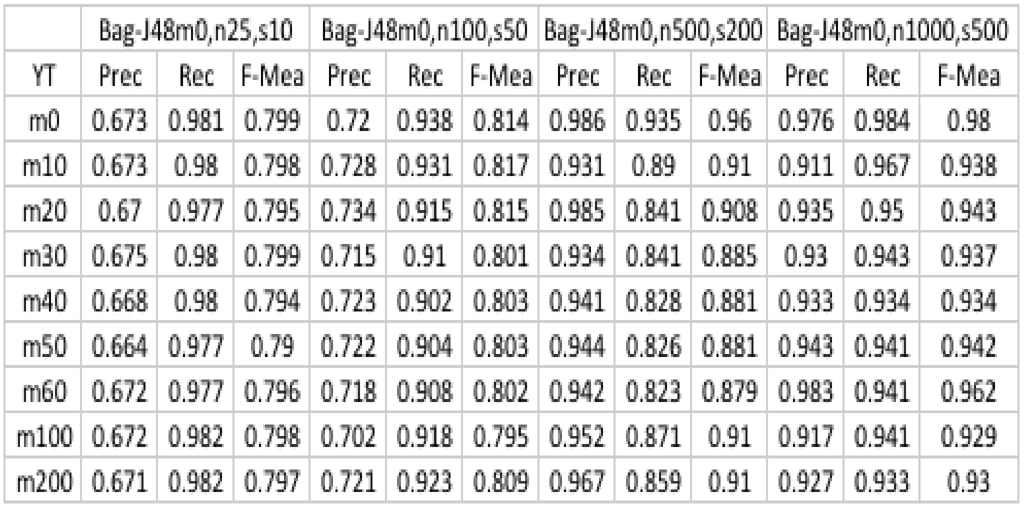
Evaluation of Ensemble Algorithms Applied to Sub-flow Classifiers

In the previous experiments, it was demonstrated that sub-flow classifiers performed substantially better than full flow classifiers on traffic with missing packets. Results presented in this section evaluated the effect of ensemble techniques on sub-flow classifiers, as exhibited through improved performance, precision and recall. Bagging and AdaBoost were the two ensemble techniques evaluated. Both Bagging and AdaBoost were applied to J48, Naïve Bayes, and SMO SVM for each traffic class and then tested with the same 9 datasets as the non-ensemble classifiers. The outcome of these experiments identified the best sub-flow classifier for YouTube and Netflix among all the sub-flow classifiers tested for this research.

YouTube Sub-flow Bagging Classifiers

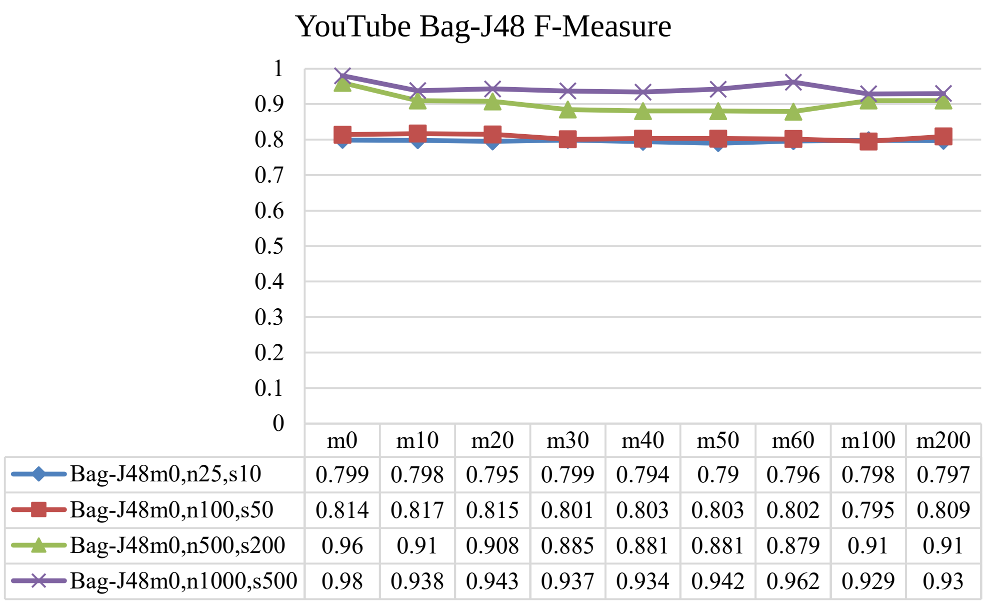
Table 15 and Figure 44 (F-Measure only) provides a tabular and graphical view of results for Bagging as applied to J48 decision tree algorithm. Plotting F-measure simplifies comparison across all Bagging models since the objective is identifying the best single model for YouTube and Netflix. Bagging applied to "m0,n1000,s500" provides excellent results with values in the mid-nineties (~.94) across each data set even when missing packets increases. These results are also higher than the non-ensemble model J48m0,n1000,s500 previously tested.

Table 15 YouTube Bagging-J48 Results



	Bag-J48m0,n25,s10			Bag-J48m0,n100,s50			Bag-J48m0,n500,s200			Bag-J48m0,n1000,s500		
YT	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea
m0	0.673	0.981	0.799	0.72	0.938	0.814	0.986	0.935	0.96	0.976	0.984	0.98
m10	0.673	0.98	0.798	0.728	0.931	0.817	0.931	0.89	0.91	0.911	0.967	0.938
m20	0.67	0.977	0.795	0.734	0.915	0.815	0.985	0.841	0.908	0.935	0.95	0.943
m30	0.675	0.98	0.799	0.715	0.91	0.801	0.934	0.841	0.885	0.93	0.943	0.937
m40	0.668	0.98	0.794	0.723	0.902	0.803	0.941	0.828	0.881	0.933	0.934	0.934
m50	0.664	0.977	0.79	0.722	0.904	0.803	0.944	0.826	0.881	0.943	0.941	0.942
m60	0.672	0.977	0.796	0.718	0.908	0.802	0.942	0.823	0.879	0.983	0.941	0.962
m100	0.672	0.982	0.798	0.702	0.918	0.795	0.952	0.871	0.91	0.917	0.941	0.929
m200	0.671	0.982	0.797	0.721	0.923	0.809	0.967	0.859	0.91	0.927	0.933	0.93

Figure 44 graphically depicts F-measure for the results found in Table 15. F-measure is consistently above 0.93 which confirms excellent results for the Bag-J48m0,n1000,s500 model. All Bagging J48 models perform consistently for all datasets of missing packets and sub-flow sizes.



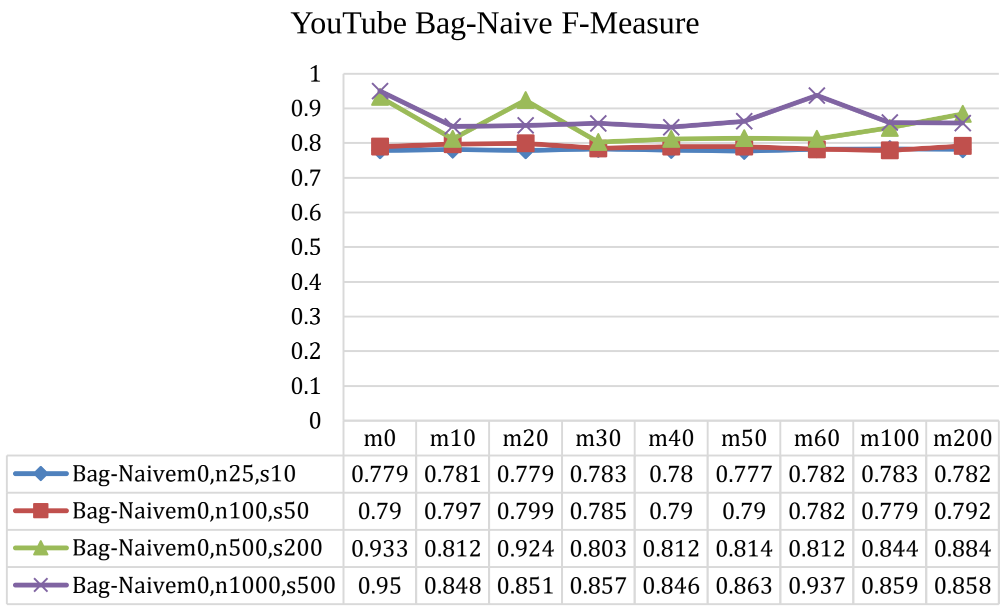
	m0	m10	m20	m30	m40	m50	m60	m100	m200
Bag-J48m0,n25,s10	0.799	0.798	0.795	0.799	0.794	0.79	0.796	0.798	0.797
Bag-J48m0,n100,s50	0.814	0.817	0.815	0.801	0.803	0.803	0.802	0.795	0.809
Bag-J48m0,n500,s200	0.96	0.91	0.908	0.885	0.881	0.881	0.879	0.91	0.91
Bag-J48m0,n1000,s500	0.98	0.938	0.943	0.937	0.934	0.942	0.962	0.929	0.93

Figure 44 YouTube Bagging-J48 F-Measure

Table 16 and Figure 45 provide results for Bagging applied to Naïve Bayes. Again results show improvement over the non-ensemble Naïve Bayes models tested previously. However, does not rise to the performance of the Bag-J48m0,n1000,s500 (Figure 44).

Table 16 YouTube Bagging-Naive Results

	Bag-Naivem0,n25,s10			Bag-Naivem0,n100,s50			Bag-Naivem0,n500,s200			Bag-Naivem0,n1000,s500		
YT	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea
m0	0.666	0.939	0.779	0.675	0.952	0.79	0.936	0.93	0.933	0.939	0.961	0.95
m10	0.666	0.944	0.781	0.682	0.958	0.797	0.712	0.943	0.812	0.773	0.939	0.848
m20	0.663	0.945	0.779	0.689	0.952	0.799	0.922	0.925	0.924	0.778	0.94	0.851
m30	0.668	0.946	0.783	0.666	0.955	0.785	0.715	0.915	0.803	0.79	0.936	0.857
m40	0.661	0.95	0.78	0.679	0.945	0.79	0.732	0.911	0.812	0.777	0.927	0.846
m50	0.658	0.947	0.777	0.678	0.946	0.79	0.731	0.918	0.814	0.805	0.931	0.863
m60	0.666	0.946	0.782	0.67	0.94	0.782	0.729	0.916	0.812	0.935	0.938	0.937
m100	0.666	0.949	0.783	0.657	0.957	0.779	0.761	0.948	0.844	0.782	0.953	0.859
m200	0.665	0.948	0.782	0.675	0.956	0.792	0.829	0.947	0.884	0.79	0.939	0.858



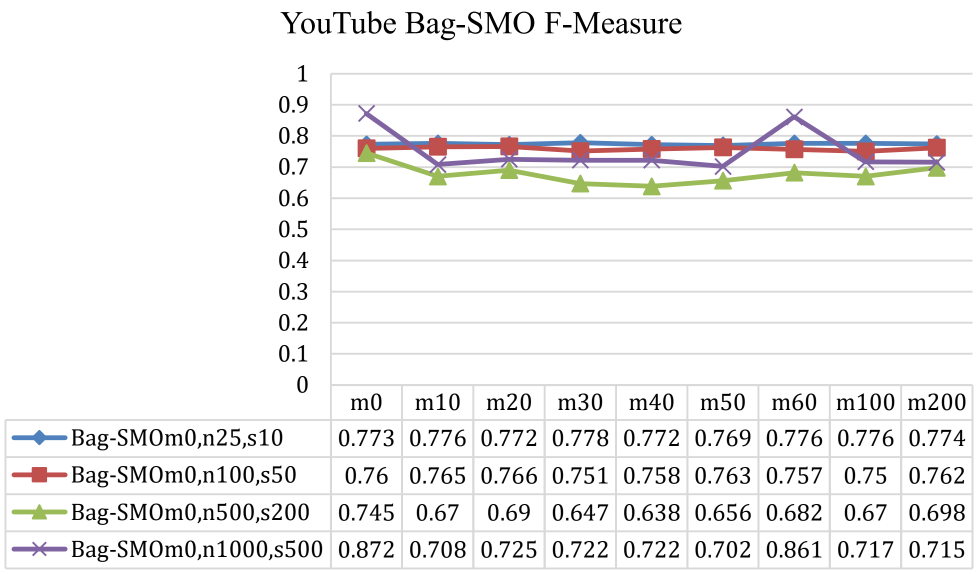
	m0	m10	m20	m30	m40	m50	m60	m100	m200
Bag-Naivem0,n25,s10	0.779	0.781	0.779	0.783	0.78	0.777	0.782	0.783	0.782
Bag-Naivem0,n100,s50	0.79	0.797	0.799	0.785	0.79	0.79	0.782	0.779	0.792
Bag-Naivem0,n500,s200	0.933	0.812	0.924	0.803	0.812	0.814	0.812	0.844	0.884
Bag-Naivem0,n1000,s500	0.95	0.848	0.851	0.857	0.846	0.863	0.937	0.859	0.858

Figure 45 YouTube Bagging-Naive F-Measure

Finally, Bagging is applied to SMO and results in the poorest performance of all Bagging models. Moreover, non-SMO models perform better overall, which indicates Bagging did not improve SMO precision and recall.

Table 17 YouTube Bagging-SMO Results

	Bag-SMOm0,n25,s10			Bag-SMOm0,n100,s50			Bag-SMOm0,n500,s200			Bag-SMOm0,n1000,s50		
YT	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea
m0	0.64	0.97	0.77	0.66	0.89	0.76	0.928	0.622	0.745	0.87	0.873	0.872
m10	0.64	0.98	0.78	0.67	0.89	0.765	0.829	0.562	0.67	0.602	0.858	0.708
m20	0.64	0.98	0.77	0.67	0.89	0.766	0.955	0.54	0.69	0.624	0.863	0.725
m30	0.65	0.98	0.78	0.65	0.88	0.751	0.84	0.526	0.647	0.618	0.867	0.722
m40	0.64	0.98	0.77	0.67	0.88	0.758	0.825	0.52	0.638	0.62	0.865	0.722
m50	0.63	0.98	0.77	0.67	0.89	0.763	0.844	0.537	0.656	0.595	0.854	0.702
m60	0.64	0.98	0.78	0.66	0.89	0.757	0.873	0.56	0.682	0.868	0.855	0.861
m100	0.64	0.98	0.78	0.64	0.9	0.75	0.876	0.543	0.67	0.619	0.852	0.717
m200	0.64	0.98	0.77	0.66	0.9	0.762	0.918	0.564	0.698	0.615	0.854	0.715



	m0	m10	m20	m30	m40	m50	m60	m100	m200
Bag-SMOm0,n25,s10	0.773	0.776	0.772	0.778	0.772	0.769	0.776	0.776	0.774
Bag-SMOm0,n100,s50	0.76	0.765	0.766	0.751	0.758	0.763	0.757	0.75	0.762
Bag-SMOm0,n500,s200	0.745	0.67	0.69	0.647	0.638	0.656	0.682	0.67	0.698
Bag-SMOm0,n1000,s500	0.872	0.708	0.725	0.722	0.722	0.702	0.861	0.717	0.715

Figure 46 YouTube Bagging-Naive F-Measure

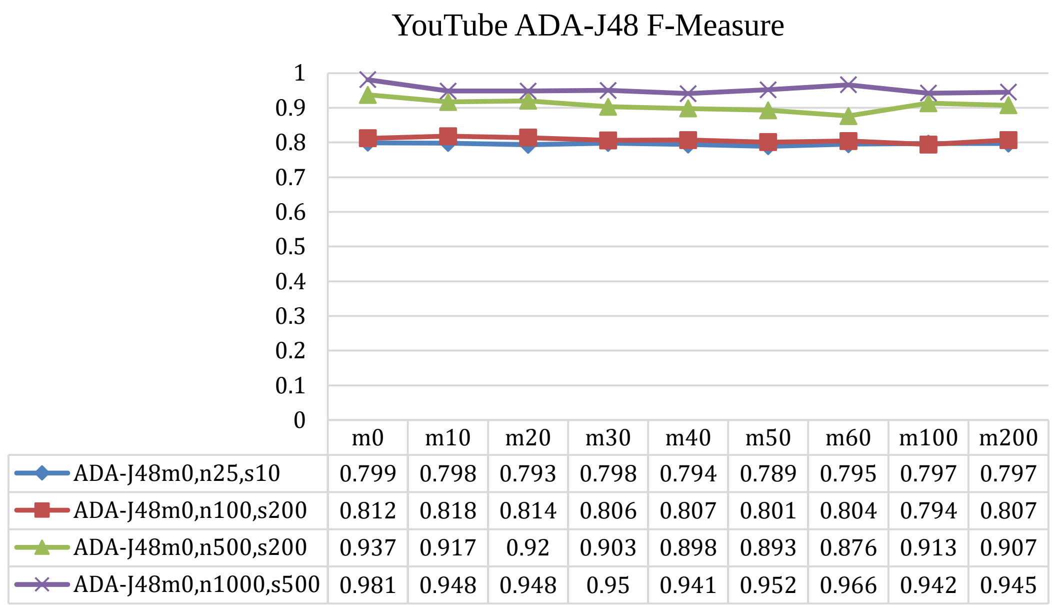
YouTube Sub-flow ADA Classifiers

Table 18 provides a tabular view, and Figure 46 the graphical view, of results from applying AdaBoost to J48 algorithm. Note ADA-J48m0,n1000,s500 model has the highest performance of all models for YouTube with F-measure values between 0.94 and 0.98 across the range of m0 – m200 datasets.

Table 18 YouTube ADA-J48 Results

	ADA-J48 m0,n25,s10			ADA-J48 m0,n100,s50			ADA-J48 m0,n500,s200			ADA-J48 m0,n1000,s500		
YT	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea
m0	0.67	0.99	0.8	0.71	0.96	0.81	0.98	0.9	0.94	0.98	0.98	0.98
m10	0.67	0.98	0.8	0.72	0.96	0.82	0.92	0.91	0.92	0.93	0.97	0.95
m20	0.67	0.98	0.79	0.72	0.94	0.81	0.98	0.86	0.92	0.96	0.94	0.95
m30	0.67	0.98	0.8	0.7	0.95	0.81	0.93	0.88	0.9	0.96	0.94	0.95
m40	0.67	0.98	0.79	0.71	0.93	0.81	0.93	0.87	0.9	0.95	0.93	0.94
m50	0.66	0.98	0.79	0.71	0.93	0.8	0.93	0.86	0.89	0.96	0.94	0.95
m60	0.67	0.98	0.8	0.71	0.93	0.8	0.92	0.84	0.88	0.99	0.95	0.97
m100	0.67	0.99	0.8	0.69	0.93	0.79	0.95	0.88	0.91	0.95	0.94	0.94
m200	0.67	0.99	0.8	0.71	0.94	0.81	0.96	0.86	0.91	0.95	0.94	0.95

Figure 47 confirms of the excellent performance and consistency of ADA-J48m0,n1000,s500 model in comparison to other ADA-J48 models of different sub-flow sizes. The performance of ADA-J48 are even higher than Bag-J48,n1000,s500 model previously tested.



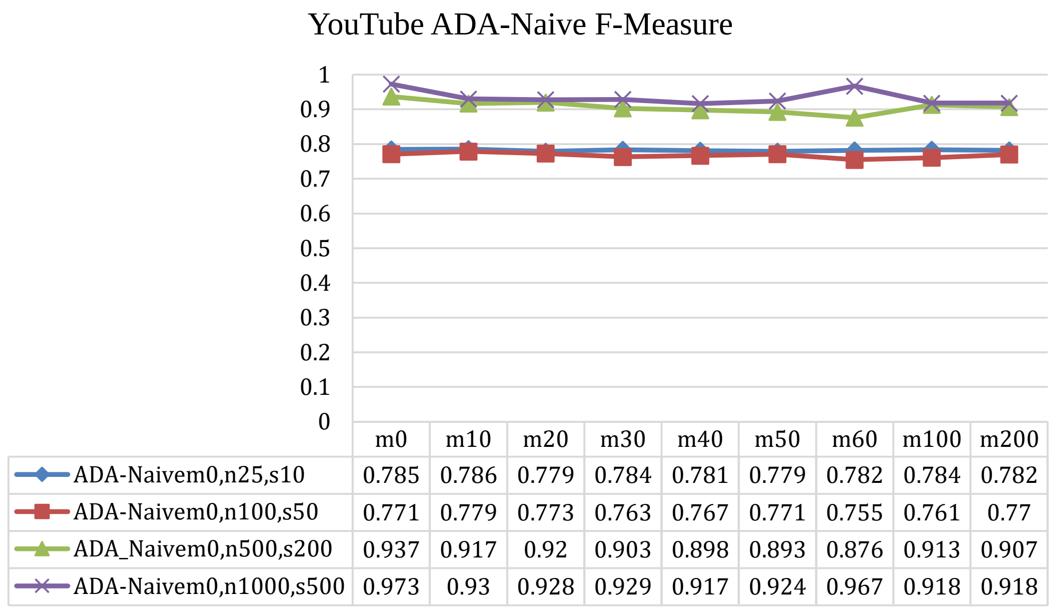
	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ ADA-J48m0,n25,s10	0.799	0.798	0.793	0.798	0.794	0.789	0.795	0.797	0.797
■ ADA-J48m0,n100,s200	0.812	0.818	0.814	0.806	0.807	0.801	0.804	0.794	0.807
▲ ADA-J48m0,n500,s200	0.937	0.917	0.92	0.903	0.898	0.893	0.876	0.913	0.907
× ADA-J48m0,n1000,s500	0.981	0.948	0.948	0.95	0.941	0.952	0.966	0.942	0.945

Figure 47 YouTube ADA-J48 F-Measure

Table 19 provides results from applying AdaBoost to Naïve Bayes for multiple sub-flows. Of significance is the performance of the ADA-Naïve m0,n1000,s500 model, which has F-measure values between ~0.92 and 0.97 across the m0 – m200 datasets. Similar to ADA-J48, ADA complements Naïve Bayes well, and is only slightly less effective than AdaBoost applied to J48. Figure 48 graphically confirms the findings for the ADA-Naïve m0,n1000,s500 model.

Table 19 YouTube ADA-Naive Results

	ADA-Naivem0,n25,s10			ADA-Naivem0,n100,s50			ADA-Naivem0,n500,s200			ADA-Naivem0,n1000,s500		
YT	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea
m0	0.677	0.934	0.785	0.74	0.805	0.771	0.983	0.896	0.937	0.958	0.988	0.973
m10	0.676	0.938	0.786	0.758	0.802	0.779	0.923	0.91	0.917	0.882	0.983	0.93
m20	0.67	0.931	0.779	0.754	0.794	0.773	0.984	0.864	0.92	0.888	0.972	0.928
m30	0.676	0.933	0.784	0.742	0.784	0.763	0.929	0.877	0.903	0.892	0.969	0.929
m40	0.669	0.937	0.781	0.75	0.785	0.767	0.926	0.871	0.898	0.879	0.957	0.917
m50	0.667	0.935	0.779	0.755	0.786	0.771	0.925	0.863	0.893	0.892	0.958	0.924
m60	0.674	0.933	0.782	0.74	0.771	0.755	0.922	0.835	0.876	0.97	0.964	0.967
m100	0.674	0.938	0.784	0.732	0.792	0.761	0.946	0.883	0.913	0.873	0.969	0.918
m200	0.672	0.933	0.782	0.752	0.788	0.77	0.957	0.863	0.907	0.869	0.974	0.918



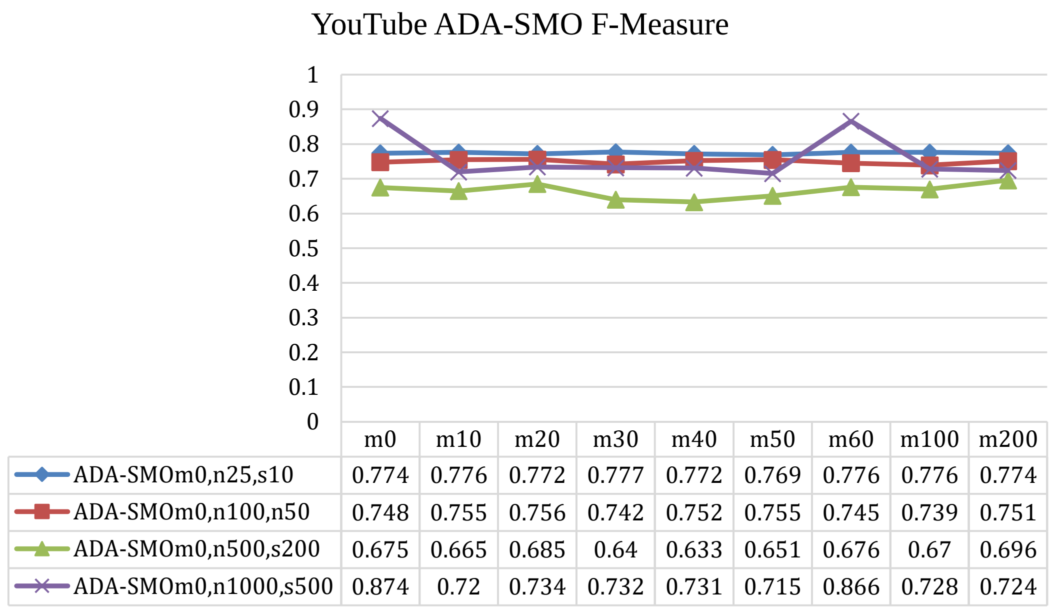
	m0	m10	m20	m30	m40	m50	m60	m100	m200
ADA-Naivem0,n25,s10	0.785	0.786	0.779	0.784	0.781	0.779	0.782	0.784	0.782
ADA-Naivem0,n100,s50	0.771	0.779	0.773	0.763	0.767	0.771	0.755	0.761	0.77
ADA_Naivem0,n500,s200	0.937	0.917	0.92	0.903	0.898	0.893	0.876	0.913	0.907
ADA-Naivem0,n1000,s500	0.973	0.93	0.928	0.929	0.917	0.924	0.967	0.918	0.918

Figure 48 YouTube ADA-Naive F-Measure

Table 20 list results for AdaBoost applied to SMO. Performance for ADA-SMO is relatively poor when compared to ADA-J48 and ADA-Naïve models. Moreover, AdaBoost only slightly improves SMO SVM relative to non-ensemble SVM models tested previously. Figure 49 graphically depicts these findings.

Table 20 YouTube ADA-SMO Results

	ADA-SMOm0,n25,s10			ADA-SMOm0,n100,s50			ADA-SMOm0,n500,s200			ADA-SMOm0,n1000,s500		
YT	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea
m0	0.645	0.967	0.774	0.667	0.851	0.748	0.929	0.53	0.675	0.877	0.872	0.874
m10	0.646	0.972	0.776	0.678	0.852	0.755	0.821	0.559	0.665	0.62	0.858	0.72
m20	0.641	0.971	0.772	0.683	0.847	0.756	0.953	0.535	0.685	0.638	0.865	0.734
m30	0.648	0.971	0.777	0.662	0.844	0.742	0.834	0.52	0.64	0.633	0.868	0.732
m40	0.641	0.972	0.772	0.677	0.846	0.752	0.825	0.514	0.633	0.633	0.863	0.731
m50	0.636	0.971	0.769	0.677	0.853	0.755	0.84	0.532	0.651	0.616	0.853	0.715
m60	0.645	0.972	0.776	0.669	0.842	0.745	0.871	0.553	0.676	0.875	0.857	0.866
m100	0.645	0.972	0.776	0.649	0.858	0.739	0.877	0.541	0.67	0.635	0.852	0.728
m200	0.644	0.969	0.774	0.671	0.852	0.751	0.92	0.56	0.696	0.627	0.858	0.724



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ ADA-SMOm0,n25,s10	0.774	0.776	0.772	0.777	0.772	0.769	0.776	0.776	0.774
■ ADA-SMOm0,n100,n50	0.748	0.755	0.756	0.742	0.752	0.755	0.745	0.739	0.751
▲ ADA-SMOm0,n500,s200	0.675	0.665	0.685	0.64	0.633	0.651	0.676	0.67	0.696
× ADA-SMOm0,n1000,s500	0.874	0.72	0.734	0.732	0.731	0.715	0.866	0.728	0.724

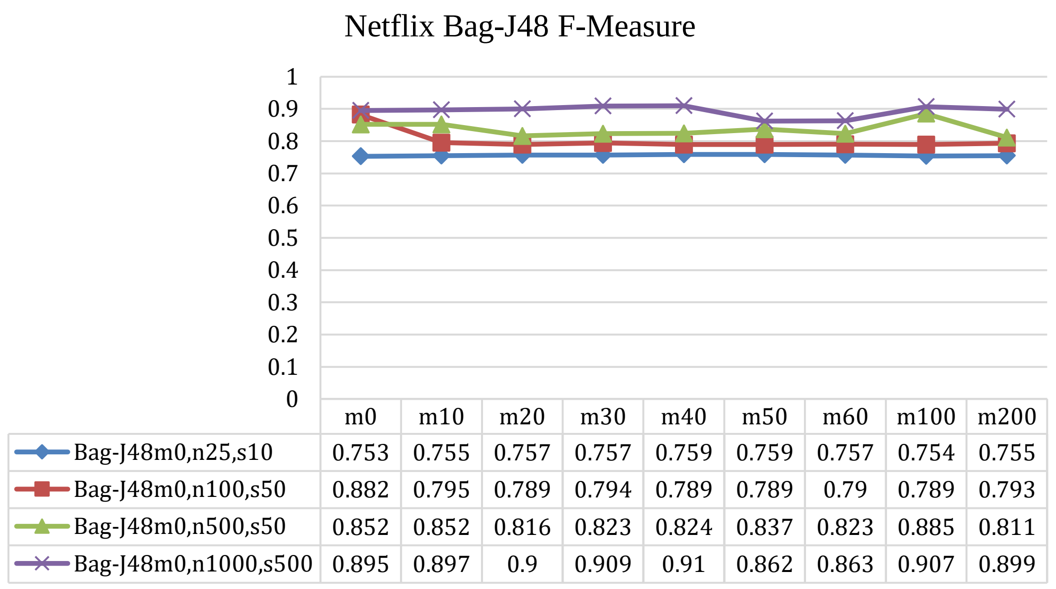
Figure 49 YouTube ADA-SMO F-Measure

Netflix Sub-flow Bagging Models

In this portion of the research, results from Bagging are presented to determine if ensemble techniques improved on previous findings for non-ensemble J48, Naïve Bayes and SMO models for Netflix traffic data. Table 21 details results for Bagging applied to J48 for Netflix traffic. Bag-J48-m0,n1000,s500 model’s performance is good relative to the other Bag J48 models; however, it is just slightly better than the non-ensemble J48 models tested previously with F-measure values between 0.86 - 0.90. Figure 50 graphically depicts F-measure for each Bagging model.

Table 21 Netflix Bag-J48 Results

	Bag-J48-m0,n25,s10			Bag-J48-m0,n100,s50			Bag-J48-m0,n500,s200			Bag-J48-m0,n1000,s500		
NF	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea
m0	0.723	0.785	0.753	0.793	0.993	0.882	0.805	0.905	0.852	0.859	0.933	0.895
m10	0.725	0.788	0.755	0.664	0.991	0.795	0.804	0.906	0.852	0.86	0.937	0.897
m20	0.726	0.791	0.757	0.657	0.988	0.789	0.754	0.89	0.816	0.876	0.926	0.9
m30	0.726	0.789	0.757	0.664	0.988	0.794	0.766	0.889	0.823	0.873	0.948	0.909
m40	0.728	0.793	0.759	0.656	0.989	0.789	0.767	0.891	0.824	0.873	0.949	0.91
m50	0.728	0.793	0.759	0.657	0.988	0.789	0.784	0.898	0.837	0.79	0.948	0.862
m60	0.724	0.793	0.757	0.658	0.988	0.79	0.774	0.879	0.823	0.792	0.948	0.863
m100	0.723	0.789	0.754	0.655	0.991	0.789	0.883	0.887	0.885	0.872	0.945	0.907
m200	0.724	0.788	0.755	0.662	0.99	0.793	0.75	0.883	0.811	0.872	0.927	0.899



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ Bag-J48m0,n25,s10	0.753	0.755	0.757	0.757	0.759	0.759	0.757	0.754	0.755
■ Bag-J48m0,n100,s50	0.882	0.795	0.789	0.794	0.789	0.789	0.79	0.789	0.793
▲ Bag-J48m0,n500,s50	0.852	0.852	0.816	0.823	0.824	0.837	0.823	0.885	0.811
× Bag-J48m0,n1000,s500	0.895	0.897	0.9	0.909	0.91	0.862	0.863	0.907	0.899

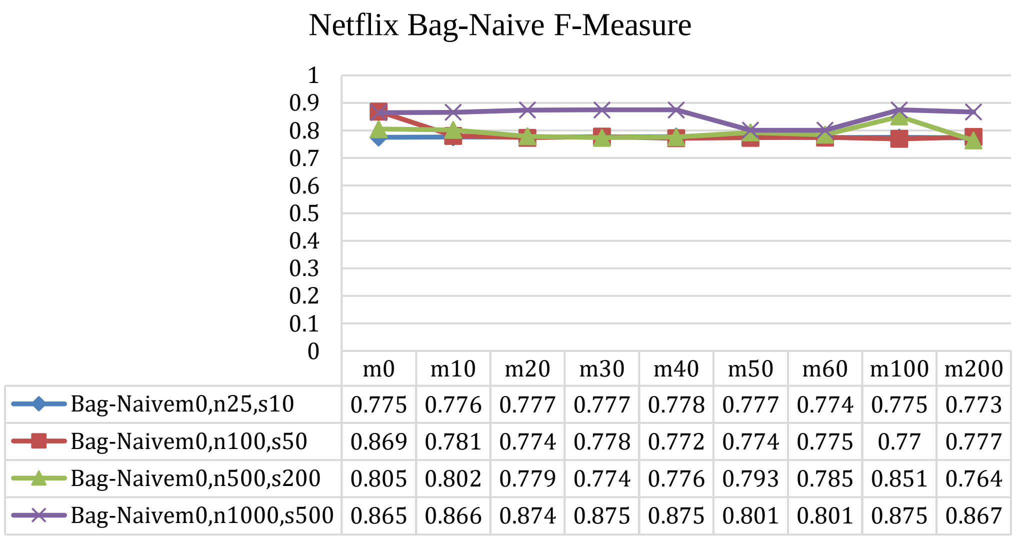
Figure 50 Netflix Bag-J48 F-Measure

Table 22 lists the results from applying Bagging to Naïve Bayes for Netflix traffic. Results indicate that Bag-Naïve-m0,n100,s500 model performs best relative to other Bagging Naïve Bayes models with F-measure scores ~0.87; however, the Netflix Bag-

J48-m0,n1000,s500 model, Figure 50, exhibits better performance overall with F-measure values ~0.90.

Table 22 Netflix Bag-Naive Results

	Bag-Naïve-m0,n25,s10			Bag-Naïve-m0,n100,s50			Bag-Naïve-m0,n500,s200			Bag-Naïve-m0,n1000,s500		
NF	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea
m0	0.65	0.97	0.78	0.78	0.98	0.87	0.73	0.9	0.81	0.81	0.93	0.87
m10	0.65	0.97	0.78	0.65	0.98	0.78	0.73	0.9	0.8	0.81	0.93	0.87
m20	0.65	0.97	0.78	0.64	0.98	0.77	0.7	0.88	0.78	0.83	0.93	0.87
m30	0.65	0.97	0.78	0.65	0.98	0.78	0.69	0.88	0.77	0.82	0.94	0.88
m40	0.65	0.97	0.78	0.64	0.98	0.77	0.69	0.88	0.78	0.82	0.94	0.88
m50	0.65	0.97	0.78	0.64	0.98	0.77	0.73	0.87	0.79	0.7	0.93	0.8
m60	0.65	0.97	0.77	0.64	0.98	0.78	0.72	0.87	0.79	0.7	0.93	0.8
m100	0.65	0.97	0.78	0.64	0.98	0.77	0.83	0.87	0.85	0.82	0.94	0.88
m200	0.64	0.97	0.77	0.65	0.98	0.78	0.7	0.85	0.76	0.82	0.92	0.87



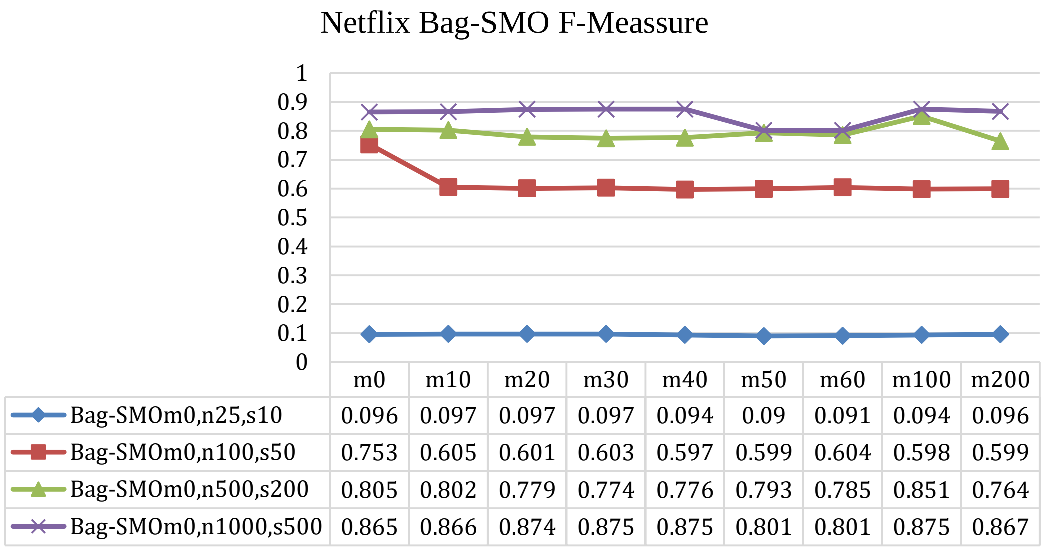
	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ Bag-Naivem0,n25,s10	0.775	0.776	0.777	0.777	0.778	0.777	0.774	0.775	0.773
■ Bag-Naivem0,n100,s50	0.869	0.781	0.774	0.778	0.772	0.774	0.775	0.77	0.777
▲ Bag-Naivem0,n500,s200	0.805	0.802	0.779	0.774	0.776	0.793	0.785	0.851	0.764
× Bag-Naivem0,n1000,s500	0.865	0.866	0.874	0.875	0.875	0.801	0.801	0.875	0.867

Figure 51 Netflix Bag-Naive F-Measure

Lastly, Table 23 lists the results from applying Bagging to SMO for Netflix traffic class. Performance is good for the Bag-SMO-m0,n1000,s500 model; however, the performance of Bag-J48-m0,n1000,s500 is still better in comparison. Of note, Bagging significantly improved SMO results over non-ensemble SMO previously tested for the Netflix traffic class.

Table 23 Netflix Bag-SMO Results

	Bag-SMO-m0,n1000,s500			Bag-SMO-m0,n100,s50			Bag-SMO-m0,n500,s200			Bag-SMO-m0,n1000,s500		
NF	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea
m0	0.598	0.052	0.096	0.6	1	0.75	0.73	0.9	0.81	0.809	0.929	0.865
m10	0.608	0.053	0.097	0.43	1	0.61	0.73	0.9	0.8	0.808	0.932	0.866
m20	0.604	0.052	0.097	0.43	1	0.6	0.7	0.88	0.78	0.827	0.928	0.874
m30	0.612	0.052	0.097	0.43	1	0.6	0.69	0.88	0.77	0.819	0.94	0.875
m40	0.614	0.051	0.094	0.43	1	0.6	0.69	0.88	0.78	0.818	0.941	0.875
m50	0.582	0.049	0.09	0.43	1	0.6	0.73	0.87	0.79	0.702	0.933	0.801
m60	0.573	0.049	0.091	0.43	1	0.6	0.72	0.87	0.79	0.702	0.933	0.801
m100	0.597	0.051	0.094	0.43	1	0.6	0.83	0.87	0.85	0.818	0.94	0.875
m200	0.61	0.052	0.096	0.43	1	0.6	0.7	0.85	0.76	0.817	0.923	0.867



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ Bag-SMOm0,n25,s10	0.096	0.097	0.097	0.097	0.094	0.09	0.091	0.094	0.096
■ Bag-SMOm0,n100,s50	0.753	0.605	0.601	0.603	0.597	0.599	0.604	0.598	0.599
▲ Bag-SMOm0,n500,s200	0.805	0.802	0.779	0.774	0.776	0.793	0.785	0.851	0.764
× Bag-SMOm0,n1000,s500	0.865	0.866	0.874	0.875	0.875	0.801	0.801	0.875	0.867

Figure 52 Netflix Bag-SMO F-Measure

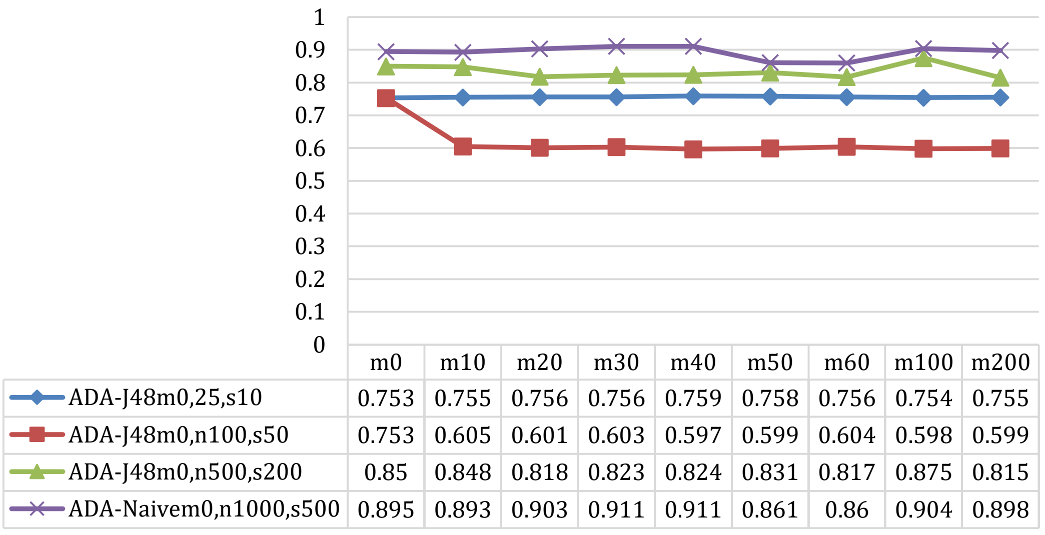
Netflix Sub-flow AdaBoost Classifiers

Now that Bagging has been evaluated, the results from testing AdaBoost on the same Netflix traffic data are presented. Table 24 provides results from applying AdaBoost to J48. The best performing model for ADA-J48 is ADA-J48-m0,n1000,s50. Additionally, results from the ADA-J48-m0,n1000,s500 model in comparison to Bag-J48-m0,n1000,s500 are essentially the same in terms of F-measure. Either model is an improvement over non-ensemble models previously tested. Figure 53 provides a graphical depiction of F-measure for the ADA J48 model for the four different sub-flow sizes.

Table 24 Netflix ADA-J48 Results

	ADA-J48-m0,n25,s10			ADA-J48-m0,n100,s50			ADA-J48-m0,n500,s200			ADA-J48-m0,n1000,s500		
NF	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea
m0	0.72	0.78	0.75	0.604	1	0.753	0.811	0.894	0.85	0.857	0.936	0.895
m10	0.73	0.79	0.76	0.434	1	0.605	0.806	0.894	0.848	0.858	0.932	0.893
m20	0.73	0.79	0.76	0.429	1	0.601	0.765	0.879	0.818	0.881	0.926	0.903
m30	0.73	0.79	0.76	0.432	1	0.603	0.773	0.881	0.823	0.875	0.95	0.911
m40	0.73	0.79	0.76	0.426	1	0.597	0.774	0.88	0.824	0.875	0.949	0.911
m50	0.73	0.79	0.76	0.427	1	0.599	0.788	0.878	0.831	0.794	0.94	0.861
m60	0.72	0.79	0.76	0.432	1	0.604	0.779	0.858	0.817	0.792	0.94	0.86
m100	0.72	0.79	0.75	0.427	1	0.598	0.885	0.865	0.875	0.869	0.941	0.904
m200	0.73	0.79	0.76	0.428	1	0.599	0.765	0.872	0.815	0.869	0.93	0.898

Netflix ADA-J48 F-Measure



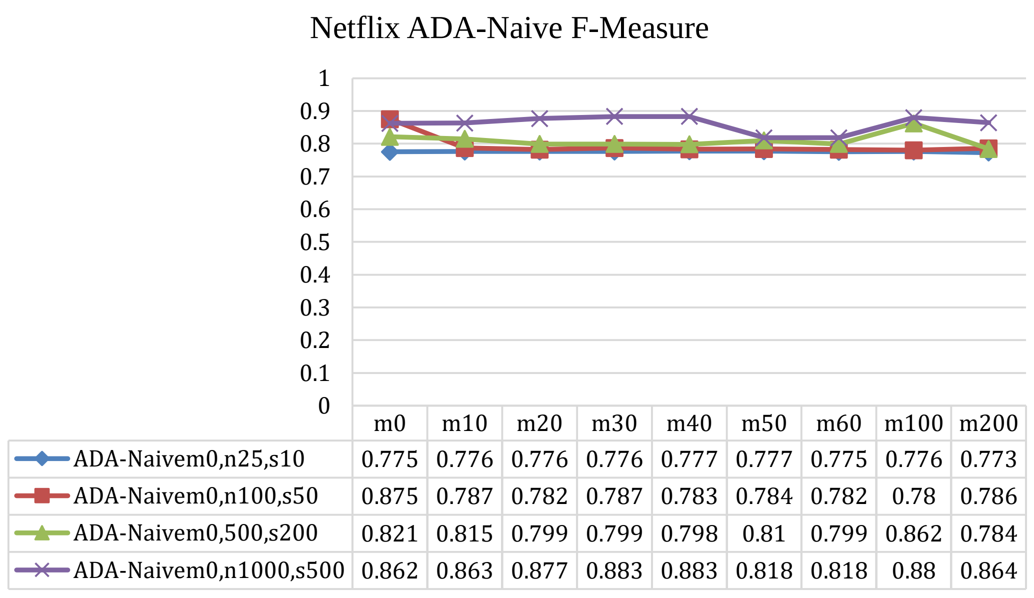
	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ ADA-J48m0,25,s10	0.753	0.755	0.756	0.756	0.759	0.758	0.756	0.754	0.755
■ ADA-J48m0,n100,s50	0.753	0.605	0.601	0.603	0.597	0.599	0.604	0.598	0.599
▲ ADA-J48m0,n500,s200	0.85	0.848	0.818	0.823	0.824	0.831	0.817	0.875	0.815
× ADA-Naivem0,n1000,s500	0.895	0.893	0.903	0.911	0.911	0.861	0.86	0.904	0.898

Figure 53 Netflix ADA-J48 F-Measure

Table 25 provides results from applying AdaBoost to Naïve Bayes. While ADA-Naïve-m0,n1000,s500 has the best results among the ADA Naïve Bayes models with F-measure values between 0.86 and 0.88, its performance is slightly less than ADA-J48-m0,n1000,s500, as depicted in Fig. 53. Figure 54 depicts F-measure for the ADA Naïve Bayes.

Table 25 Netflix ADA-Naive Results

	ADA-Naïve-m0,n25,s10			ADA-Naïve-m0,n100,s50			ADA-Naïve-m0,n500,s200			ADA-Naïve-m0,n1000,s500		
NF	Pre	Rec	F-Mea	Pre	Rec	F-Mea	Pre	Rec	F-Mea	Pre	Rec	F-Mea
m0	0.649	0.962	0.775	0.791	0.979	0.875	0.764	0.888	0.821	0.825	0.903	0.862
m10	0.65	0.961	0.776	0.659	0.979	0.787	0.756	0.883	0.815	0.827	0.901	0.863
m20	0.651	0.961	0.776	0.651	0.979	0.782	0.734	0.877	0.799	0.847	0.908	0.877
m30	0.65	0.963	0.776	0.658	0.978	0.787	0.733	0.877	0.799	0.85	0.918	0.883
m40	0.652	0.964	0.777	0.652	0.98	0.783	0.732	0.876	0.798	0.848	0.921	0.883
m50	0.652	0.961	0.777	0.653	0.979	0.784	0.755	0.873	0.81	0.745	0.908	0.818
m60	0.648	0.962	0.775	0.652	0.976	0.782	0.745	0.861	0.799	0.745	0.908	0.818
m100	0.65	0.962	0.776	0.649	0.977	0.78	0.861	0.864	0.862	0.846	0.917	0.88
m200	0.646	0.963	0.773	0.658	0.976	0.786	0.724	0.854	0.784	0.84	0.889	0.864



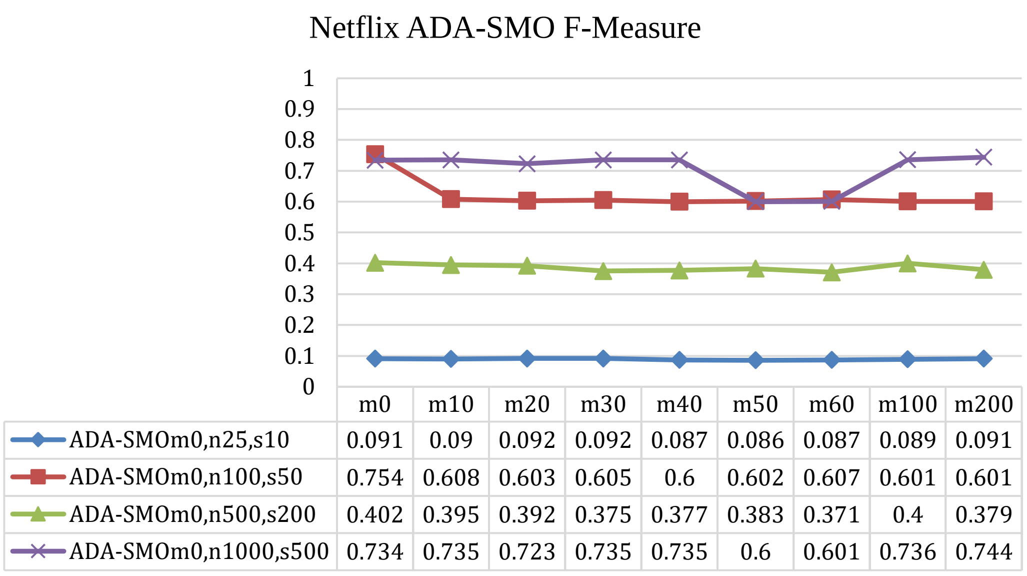
	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ ADA-Naivem0,n25,s10	0.775	0.776	0.776	0.776	0.777	0.777	0.775	0.776	0.773
■ ADA-Naivem0,n100,s50	0.875	0.787	0.782	0.787	0.783	0.784	0.782	0.78	0.786
▲ ADA-Naivem0,500,s200	0.821	0.815	0.799	0.799	0.798	0.81	0.799	0.862	0.784
× ADA-Naivem0,n1000,s500	0.862	0.863	0.877	0.883	0.883	0.818	0.818	0.88	0.864

Figure 54 Netflix ADA-Naive F-Measure

Table 26 list results from applying AdaBoost to SMO. Indications are that the ADA-SMO-m0,n1000,s500 model has the best performance among all other ADA-SMO models with values ~0.71 across the various datasets. Furthermore, there is an improvement to SMO when ADA is used in combination with the SVM algorithm. Although results for ADA-SMO-m0,n1000,s500 are good with F-measure values between 0.86 - 0.88, the Bag-J48-m0,n1000,s500 model, Figure 53, performs the best for Netflix traffic classification.

Table 26 Netflix ADA-SMO Results

	ADA-SMO-m0,n25,s10			ADA-SMO-m0,n100,s50			ADA-SMO-m0,n500,s200			ADA-SMO-m0,n1000,s500		
NF	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea	Prec	Rec	F-Mea
m0	0.6	0.05	0.091	0.61	0.996	0.754	0.68	0.29	0.402	0.61	0.92	0.734
m10	0.6	0.05	0.09	0.44	0.996	0.608	0.64	0.29	0.395	0.61	0.92	0.735
m20	0.61	0.05	0.092	0.43	0.997	0.603	0.65	0.28	0.392	0.59	0.92	0.723
m30	0.62	0.05	0.092	0.43	0.996	0.605	0.64	0.27	0.375	0.61	0.92	0.735
m40	0.61	0.05	0.087	0.43	0.998	0.6	0.65	0.27	0.377	0.61	0.92	0.735
m50	0.58	0.05	0.086	0.43	0.998	0.602	0.64	0.27	0.383	0.45	0.9	0.6
m60	0.57	0.05	0.087	0.44	0.998	0.607	0.6	0.27	0.371	0.45	0.9	0.601
m100	0.6	0.05	0.089	0.43	0.998	0.601	0.78	0.27	0.4	0.61	0.92	0.736
m200	0.61	0.05	0.091	0.43	0.996	0.601	0.63	0.27	0.379	0.63	0.92	0.744



	m0	m10	m20	m30	m40	m50	m60	m100	m200
◆ ADA-SMOm0,n25,s10	0.091	0.09	0.092	0.092	0.087	0.086	0.087	0.089	0.091
■ ADA-SMOm0,n100,s50	0.754	0.608	0.603	0.605	0.6	0.602	0.607	0.601	0.601
▲ ADA-SMOm0,n500,s200	0.402	0.395	0.392	0.375	0.377	0.383	0.371	0.4	0.379
× ADA-SMOm0,n1000,s500	0.734	0.735	0.723	0.735	0.735	0.6	0.601	0.736	0.744

Figure 55 Netflix ADA-SMO F-Measure

Summary

In this set of experiments, the effects of ensemble methodologies, Bagging and AdaBoost, were explored. The intent was to improve performance of sub-flow classifiers for J48, Naïve Bayes and SMO tested on the same partial flows as non-ensemble models. Generally, both Bagging and AdaBoost increased precision and recall for each sub-flow classifier tested. Moreover, the ADA-J48-m0,n1000,s500 model produced excellent performance with F-measures between 0.94 and 0.98 for the YouTube traffic class. For Netflix ADA-J48-m0,n1000,s500, there were slightly improved results with F-measure values from ~0.86 to 0.91. Overall, the results indicate that sub-flow classifiers using ensemble techniques in conjunction with J48 C4.5 are well suited for classification of YouTube and Netflix traffic.

Chapter 5

Conclusions, Implications, Recommendations, and Summary

Conclusion

This research focused on the evaluation of machine learning algorithms for classifying video streaming traffic. A key tenet of this research was the use of sub-flow based classifiers – ML models trained on statistics from a subset of packets instead of the entire traffic flow. Use of statistics derived from the entire flow, known as full flow, produced poor results when partial flows with missing packets are encountered. Specifically, full flow trained classifiers exhibited low recall and inconsistent performance as the number of missing packets increase. In contrast, classifiers trained on sub-flows exhibit higher and more consistent performance. Furthermore, ensemble techniques applied to the same ML algorithms improve performance substantively. To examine this supposition, 5 research questions were proposed and answered through experimentation and are listed below along with their associated findings:

1) What recall and precision can be attained using ML algorithms trained on multiple sub-flows in classifying video streaming traffic?
   Prior to examining the impact of sub-flow base classifiers, full flow classifiers were tested to confirm poor performance in terms of recall of ~0.70 for YouTube and 0.41 for Netflix. In contrast, sub-flow trained classifiers attained precision from 0.88 to 0.97 and recall of 0.88 to 0.98 for YouTube; for Netflix, values from ~0.80 to 0.82 for precision and ~0.92 to 0.94 for recall were attained. More importantly, ensemble based sub-flow classifiers produce excellent results for YouTube, and some improvement in performance for Netflix. For YouTube,

ADA-J48-m0,n1000,s500 (AdaBoost combined with C4.5) model produced precision values between ~0.93 and ~0.98 and recall values between ~0.94 and ~0.98; and for Netflix, ADA-J48-m0,n1000,s500 precision of ~0.80 to ~0.88 and recall of ~0.92 to ~0.95.

2) What sub-flow size used to train, test, and classify video traffic attained high recall and precision?

   Experiments indicate that a sub-flow size of 1000 packets results in very good performance for Netflix and excellent performance for YouTube traffic. While the experiments performed for this research were specific to Netflix and YouTube, results should be extensible to other video streaming applications. However, interactive video gaming systems, may respond differently to sub-flow techniques due to the number of changes in traffic patterns over the entire flow. Investigation of online large scale gaming traffic should be undertaken through future research efforts.

3) What features, sub-flow attributes, are required to enable classification of video traffic?

   A total of 19 features, including “class” of traffic, were identified and used for training and testing classifiers. Wireshark was used to capture and derive a number of statistics. Additionally, Wireshark was also used, in conjunction with manual inspection, to label flows correctly for training classifiers. Scripts were written to generate missing statistics and select the proper number features from Wireshark output. The list and description of features can be found in Chapter 3, Methodology. Preliminary tests were executed to ensure the relevance of selected features.

4) What is the effect of different sub-flow sizes, number of packets per sub-flow, on ML recall and precision?

   Results in Chapter 4 indicate precision and recall are impacted by sub-flow size; specifically, as sub-flow size increases, performance and recall also increase and become more consistent. To great extent this trending toward increase sub-flow size is understandable, considering video streaming traffic tends to be consistent and long lived. However, as sub-flows sizes get closer to full flows then precision and recall are reduced as experiments with full flow classification indicate. In general, there is a point at which larger sub-flows reflects the characteristics of full flow models and produces poor performance. Additionally, increasing sub-flow size is counter to the premise of this research in that it is generally difficult to ensure the capture of full flows in real world application of ML classifiers for video traffic.

5) How effective are ML algorithms trained on multiple sub-flows in classifying video streaming traffic from disparate data sets containing packets captured from different network environments?

   Traffic for YouTube and Netflix were captured from two different networks and stored as separate datasets for training and testing classifiers. Sub-flow classifiers were successful in classifying both types of traffic with solid performance results for both YouTube and Netflix. Moreover, ensemble techniques in concert with

C4.5 decision tree algorithm as detailed in Chapter 4, produced improved performance over non-ensemble classifiers.

Implications

Use of full flow classifiers in real world applications of machine learning should be questioned in terms of the practical application to classifying video streaming traffic with missing packets or partial flows. Testing of full flow classifiers performed for this research indicates that full flow classifiers had difficulty classifying video streaming traffic when partial flows were encountered. In the use cases examined in this research, J48-C4.5, Naïve Bayes and SVM performed poorly in terms of recall in comparison sub-flow classifiers tested with the same partial flow datasets. Furthermore, ensemble techniques paired with J48 C4.5, Naïve Bayes and SMO SVM sub-flow models performed significantly better than full flow classifiers. Therefore, use of full flow classifiers for classifying video streaming traffic is suspect when full flow capture cannot be assured due to volume, time, or network perturbations.

Recommendations

It is recommended based on the findings of this research that sub-flow classifiers offer significant benefits for classification of video streaming traffic with partial flows and missing packets. Moreover, ensemble techniques, specifically Bagging and AdaBoost applied to J48-C4.5 and Naïve Bayes can significantly improve performance. Accordingly, ensemble based sub-flow classifiers are recommended when classification of video streaming traffic is desired.

Summary

This research focused on the evaluation of ML classification models for video streaming traffic. An underlying premise is the use of sub-flow classifiers to classify partial traffic flows with missing packets. Three ML algorithms were used for experimentation: C4.5, Naïve Bayes and SVM. Moreover, ensemble techniques were applied to each of these models to evaluate if performance, precision, and recall could be improved. Experimentation proved that sub-flow classifiers were in fact more consistent and produced higher levels of performance overall. Specifically, ADA applied to Weka's implementation of C4.5 (J4.8) performed best for YouTube and Netflix traffic. Indications are that when implementing ML base classifiers in real world applications, consideration should be given to use of sub-flow base classifiers instead of full flow models.

Although this work was successful in addressing all research questions, limitations exist that should be examined in future research efforts:

- Applying Sub-flow Classifiers to Interactive On-line Video Games: While video streaming traffic is relatively consistent, interactive games played with thousands of users over the internet offer additional challenges. The characteristics of these types of interactive games may change meaningfully and continually over short intervals for the life of the traffic flow. Researchers should consider the application of ensemble base sub-flow classifiers to classification of interactive large scale internet games.
- Evaluation of other Ensemble Techniques: Only two ensemble techniques were tested for this research. In general, performance was improved. Other

ensemble techniques such as stacking, random forest, and Bayes Optimal Classifier may garner even better results.

- Automating Discovery of the Optimal Sub-flow Size: It may be possible to use clustering techniques to reduce number of choices related to the optimal sub-flow size. Clustering techniques may offer insights based on the groupings of packets. This may lead to reduced time to determine which sub-flow size provides optimum classification performance.
- Malware Command and Control (C2) Traffic: A key challenge for Cyber security is identifying malware that may be communicating with “home station” once an end-user system is compromised. Typically, this communication is intermittent and uses short duration flows. Since sub-flow methods take small samples of network traffic, it may be well suited for classifying this type of anomalous traffic.

As the expansion and use of the Internet continues, classification of network traffic to improve security, manage usage, and provide differentiated service will grow accordingly. Consequently, network administrators need techniques to classify traffic to make informed decisions related to use of network resources. This research and the associated findings build on previous work and provides additional insights on applying ML routines to real world classification problems.

References

Abu-Mostafa, Y. S., Magdon-Ismail, M., & Lin, H.-T. (2012). *Learning from data*: AMLBook.

Al-Aidaroos, K. M., Bakar, A. A., & Othman, Z. (2010, 17-18 March 2010). *Naive bayes variants in classification learning.* Paper presented at the Information Retrieval & Knowledge Management, (CAMP), 2010 International Conference on.

Alshammari, R., & Zincir-Heywood, A. N. (2008, 1-3 Oct. 2008). *Investigating Two Different Approaches for Encrypted Traffic Classification.* Paper presented at the Privacy, Security and Trust, 2008. PST '08. Sixth Annual Conference on.

Alshammari, R., & Zincir-Heywood, A. N. (2011). Can encrypted traffic be identified without port numbers, IP addresses and payload inspection? *Computer Networks, 55*(6), 1326-1350. doi:10.1016/j.comnet.2010.12.002

Baker, F., Foster, B., & Sharp, C. (2004). Cisco Architecture for Lawful Intercept in IP Networks.

Ben-Hur, A., & Weston, J. (2010). A user's guide to support vector machines. *Data mining techniques for the life sciences*, 223-239.

Bernaille, L., Teixeira, R., Akodkenou, I., Soule, A., & Salamatian, K. (2006). Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review, 36*(2), 23-26.

Blair, D. C., & Maron, M. (1985). An evaluation of retrieval effectiveness for a full-text document-retrieval system. *Communications of the ACM, 28*(3), 289-299.

Bouckaert, R. R., Frank, E., Hall, M., Kirkby, R., Reutemann, P., Seewald, A., & Scuse, D. (2013). WEKA Manual for Version 3-7-8: January.

Breiman, L. (1996). Bagging predictors. *Machine Learning, 24*(2), 123-140.

Breslow, L. A., & Aha, D. W. (1997). Simplifying decision trees: A survey. *The Knowledge Engineering Review, 12*(01), 1-40.

Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery, 2*(2), 121-167.

Caiyun, Z., Lizhi, P., Bo, Y., & Zhenxiang, C. (2012, 21-23 Sept. 2012). *Labeling the Network Traffic with Accurate Application Information.* Paper presented at the Wireless Communications, Networking and Mobile Computing (WiCOM), 2012 8th International Conference on.

Callado, A., Kamienski, C., Szabó, G., Gero, B., Kelner, J., Fernandes, S., & Sadok, D. (2009). A survey on internet traffic identification. *Communications Surveys & Tutorials, IEEE, 11*(3), 37-52.

Callado, A., Kelner, J., Sadok, D., Alberto Kamienski, C., & Fernandes, S. (2010). Better network traffic identification through the independent combination of techniques. *Journal of Network and Computer Applications, 33*(4), 433-446.

Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST), 2*(3), 27.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research, 16*, 321-357. doi:http://dx.doi.org/10.1613/jair.874

Chih-Wei, H., & Chih-Jen, L. (2002). A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on, 13*(2), 415-425. doi:10.1109/72.991427

Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. *Machine Learning, 20*, 273-297.

Dabir, A., & Matrawy, A. (2007). *Bottleneck analysis of traffic monitoring using wireshark.* Paper presented at the Innovations in Information Technology, 2007. IIT'07. 4th International Conference on.

Dainotti, A., Pescape, A., & Claffy, K. C. (2012). Issues and future directions in traffic classification. *Network, IEEE, 26*(1), 35-40. doi:10.1109/MNET.2012.6135854 de A Ribeiro, V. P., Filho, R. H., & Maia, J. E. B. (2011, 23-27 May 2011). *Online traffic classification based on sub-flows.* Paper presented at the Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on.

De Mántaras, R. L. (1991). A distance-based attribute selection measure for decision tree induction. *Machine Learning, 6*(1), 81-92.

Dehghani, F., Movahhedinia, N., Khayyambashi, M. R., & Kianian, S. (2010, 22-23 May 2010). *Real-Time Traffic Classification Based on Statistical and Payload Content Features.* Paper presented at the Intelligent Systems and Applications (ISA), 2010 2nd International Workshop on.

Dong, Y.-S., & Han, K.-S. (2005). *Boosting SVM classifiers by ensemble.* Paper presented at the Special interest tracks and posters of the 14th international conference on World Wide Web.

Elovici, Y., Shabtai, A., Moskovitch, R., Tahan, G., & Glezer, C. (2007). Applying machine learning techniques for detection of malicious code in network traffic. *KI 2007: Advances in Artificial Intelligence*, 44-50.

Erman, J., Arlitt, M., & Mahanti, A. (2006). *Traffic classification using clustering algorithms.* Paper presented at the Proceedings of the 2006 SIGCOMM workshop on Mining network data.

Erman, J., Mahanti, A., Arlitt, M., Cohen, I., & Williamson, C. (2007). *Semi-supervised network traffic classification.* Paper presented at the ACM SIGMETRICS Performance Evaluation Review.

Este, A., Gringoli, F., & Salgarelli, L. (2009). Support Vector Machines for TCP traffic classification. *Computer Networks, 53*(14), 2476-2490. doi:10.1016/j.comnet.2009.05.003

Fayyad, U. M., & Irani, K. B. (1992). On the handling of continuous-valued attributes in decision tree generation. *Machine Learning, 8*(1), 87-102.

Feily, M., Shahrestani, A., & Ramadass, S. (2009). *A Survey of Botnet and Botnet Detection*. New York: IEEE.

Fern, A., & Givan, R. (2003). Online Ensemble Learning: An Empirical Study. *Machine Learning, 53*(1-2), 71-109. doi:10.1023/A:1025619426553

Flach, P. (2012). *Machine learning: the art and science of algorithms that make sense of data*: Cambridge University Press.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences, 55*(1), 119-139.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning, 29*(2-3), 131-163.

Galar, M., Fernandez, A., Barrenechea, E., Bustince, H., & Herrera, F. (2012). A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 42*(4), 463-484.

Garcıa, S., Luengo, J., Sáez, J. A., López, V., & Herrera, F. (2013). A Survey of Discretization Techniques: Taxonomy and Empirical Analysis in Supervised Learning. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, 25*(4).

Grimaudo, L., Mellia, M., Baralis, E., & Keralapura, R. (2014). SeLeCT: Self-Learning Classifier for Internet Traffic. *Network and Service Management, IEEE Transactions on, PP*(99), 1-14. doi:10.1109/TNSM.2014.011714.130505

Grzymala-Busse, J. W., & Hu, M. (2001). A Comparison of Several Approaches to Missing Attribute Values in Data Mining. *Computer Science, 2005*, 378-385.

Haffner, P., Sen, S., Spatscheck, O., & Wang, D. (2005). *ACAS: automated construction of application signatures.* Paper presented at the Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data.

Hall, M., Witten, I., & Frank, E. (2011). Data mining: Practical machine learning tools and techniques. *Kaufmann, Burlington*.

Hand, D. J. (2009). Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Machine Learning, 77*(1), 103-123.

Howley, T., & Madden, M. G. (2005). The genetic kernel support vector machine: Description and evaluation. *Artificial Intelligence Review, 24*(3-4), 379-395.

Hu, Y., Chiu, D.-M., & Lui, J. C. S. (2009). Profiling and identification of P2P traffic. *Computer Networks, 53*(6), 849-863. doi:10.1016/j.comnet.2008.11.005

Jianli, X., & Yuncai, L. (2012, 16-19 Sept. 2012). *Traffic incident detection by multiple kernel support vector machine ensemble.* Paper presented at the Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on.

Jin, Y., Duffield, N., Erman, J., Haffner, P., Sen, S., & Zhang, Z.-L. (2012). A Modular Machine Learning System for Flow-Level Traffic Classification in Large Networks. *ACM Trans. Knowl. Discov. Data, 6*(1), 1-34. doi:10.1145/2133360.2133364

Karagiannis, T., Broido, A., Faloutsos, M., & claffy, K. (2004). *Transport layer identification of P2P traffic*. Paper presented at the Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, Taormina, Sicily, Italy. http://delivery.acm.org/10.1145/1030000/1028804/p121-karagiannis.pdf?ip=129.83.31.2&id=1028804&acc=ACTIVE%20SERVICE&key=A9ED11D7A520B19D%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=433333441&CFTOKEN=97218331&__acm__=1396838758_ef2138e596d590d0bed1f13233a14301

Karagiannis, T., Papagiannaki, K., & Faloutsos, M. (2005). BLINC: multilevel traffic classification in the dark. *SIGCOMM Comput. Commun. Rev., 35*(4), 229-240. doi:10.1145/1090191.1080119

Karam, M. J., & Tobagi, F. A. (2000). *On traffic types and service classes in the Internet.* Paper presented at the Global Telecommunications Conference, 2000. GLOBECOM'00. IEEE.

Keerthi, S. S., & Lin, C.-J. (2003). Asymptotic behaviors of support vector machines with Gaussian kernel. *Neural computation, 15*(7), 1667-1689.

Kim, H.-C., Pang, S., Je, H.-M., Kim, D., & Yang Bang, S. (2003). Constructing support vector machine ensemble. *Pattern recognition, 36*(12), 2757-2767.

Koc, L., Mazzuchi, T. A., & Sarkani, S. (2012). A network intrusion detection system based on a Hidden Naïve Bayes multiclass classifier. *Expert Systems with Applications, 39*(18), 13492-13500.

Korada, N. K., Kumar, N. S. P., & Deekshitulu, Y. (2012). Implementation of Naive Bayesian Classifier and Ada-Boost Algorithm Using Maize Expert System. *International Journal of Information, 2*(3).

Lamping, U., & Warnicke, E. (2004). Wireshark User's Guide. *Interface, 4,* 6. Retrieved from https://www.wireshark.org/download/docs/user-guide-us.pdf website:

Lewis, D. D. (1998). Naive (Bayes) at forty: The independence assumption in information retrieval *Machine learning: ECML-98* (pp. 4-15): Springer.

Li, B., Springer, J., Bebis, G., & Hadi Gunes, M. (2013). A survey of network flow applications. *Journal of Network and Computer Applications, 36*(2), 567-581. doi:http://dx.doi.org/10.1016/j.jnca.2012.12.020

Liu, H., Hussain, F., Tan, C. L., & Dash, M. (2002). Discretization: An Enabling Technique. *Data mining and knowledge discovery, 6,* 393-423.

Liu, Y., Li, Z., Guo, S., & Feng, T. (2008, 6-8 April 2008). *Efficient, Accurate Internet Traffic Classification using Discretization in Naive Bayes.* Paper presented at the Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference on.

Marnerides, A., Schaeffer-Filho, A., & Mauthe, A. (2014). Traffic Anomaly Diagnosis in Internet Backbone Networks: A Survey. *Computer Networks, 73,* 224-243. doi:DOI: 10.1016/j.comnet.2014.08.007

McCallum, A., & Nigam, K. (1998). *A comparison of event models for naive bayes text classification.* Paper presented at the AAAI-98 workshop on learning for text categorization.

McGregor, A., Hall, M., Lorier, P., & Brunskill, J. (2004). Flow clustering using machine learning techniques *Passive and Active Network Measurement* (pp. 205-214): Springer.

Meddeb, A. (2010). Internet QoS: Pieces of the puzzle. *Communications Magazine, IEEE, 48*(1), 86-94. doi:10.1109/MCOM.2010.5394035

Mitchell, T. M. (1997). *Machine Learning*: McGraw-Hill, Inc.

Moore, A. W., & Papagiannaki, K. (2005). Toward the accurate identification of network applications *Passive and Active Network Measurement* (pp. 41-54): Springer.

Moore, A. W., & Zuev, D. (2005). *Internet traffic classification using bayesian analysis techniques.* Paper presented at the ACM SIGMETRICS Performance Evaluation Review.

Mordelet, F., & Vert, J.-P. (2014). A bagging SVM to learn from positive and unlabeled examples. *Pattern Recognition Letters, 37*, 201-209.

Mu, X., & Wu, W. (2011). *A Parallelized Network Traffic Classification Based on Hidden Markov Model.* Paper presented at the Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 International Conference on.

Muller, K., Mika, S., Ratsch, G., Tsuda, K., & Scholkopf, B. (2001). An introduction to kernel-based learning algorithms. *Neural Networks, IEEE Transactions on, 12*(2), 181-201. doi:10.1109/72.914517

Ng, A. Y., & Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in Neural Information Processing Systems, 2*, 841-848.

Nguyen, T., & Armitage, G. (2006, 14-16 Nov. 2006). *Training on multiple sub-flows to optimise the use of Machine Learning classifiers in real-world IP networks.* Paper presented at the Local Computer Networks, Proceedings 2006 31st IEEE Conference on.

Nguyen, T., & Armitage, G. (2008). A Survey of Techniques for Internet Traffic Classification using Machine Learning. *IEEE Communications Surveys and Tutorials, 10*(4), 56-76. doi:10.1109/surv.2008.080406

Nguyen, T., Armitage, G., Branch, P., & Zander, S. (2012). Timely and Continuous Machine-Learning-Based Classification for Interactive IP Traffic. *Networking, IEEE/ACM Transactions on, 20*(6), 1880-1894. doi:10.1109/TNET.2012.2187305

Oza, N. C., & Tumer, K. (2008). Classifier ensembles: Select real-world applications. *Information Fusion, 9*(1), 4-20. doi:http://dx.doi.org/10.1016/j.inffus.2007.07.002

Pascoal, C., Rosario de Oliveira, M., Valadas, R., Filzmoser, P., Salvador, P., & Pacheco, A. (2012, 25-30 March 2012). *Robust feature selection and robust PCA for internet traffic anomaly detection.* Paper presented at the INFOCOM, 2012 Proceedings IEEE.

Peng, F., Schuurmans, D., & Wang, S. (2004). Augmenting naive Bayes classifiers with statistical language models. *Information Retrieval, 7*(3-4), 317-345.

Peng, L., Zhang, H., Yang, B., Chen, Y., & Wu, T. (2014). Traffic Labeller: Collecting Internet traffic samples with accurate application information. *Communications, China, 11*(1), 69-78. doi:10.1109/CC.2014.6821309

Piraisoody, G., Changcheng, H., Nandy, B., & Seddigh, N. (2013, 11-13 Nov. 2013). *Classification of applications in HTTP tunnels.* Paper presented at the Cloud Networking (CloudNet), 2013 IEEE 2nd International Conference on.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*(1), 81-106.

Quinlan, J. R. (1993). *C4. 5: programs for machine learning* (Vol. 1): Morgan kaufmann.

Quinlan, J. R. (1996). Improved Use of Continuous Attributes in C4. 5. *Journal of Artificial Intelligence Research, 4*, 77-90.

Raineri, F., & Verticale, G. (2009). *Early Internet Application Identification with Machine Learning Techniques*. Paper presented at the 2009 First International Conference on Evolving Internet, New York. <Go to ISI>://WOS:000289870300010
http://ieeexplore.ieee.org/ielx5/5277746/5277747/05277868.pdf?tp=&arnumber=5277868&isnumber=5277747

Rennie, J. D. (2001). *Improving multi-class text classification with naive Bayes.* Massachusetts Institute of Technology.

Reynolds, J., Postel, J., & Group, W. S.-N. W. (1994). RFC 1700. *Assigned Numbers. Oct*.

Rish, I. (2001). *An empirical study of the naive Bayes classifier.* Paper presented at the IJCAI 2001 workshop on empirical methods in artificial intelligence.

Rodríguez, J. J., & Maudes, J. (2008). Boosting recombined weak classifiers. *Pattern Recognition Letters, 29*(8), 1049-1059.

Rokach, L. (2010). Ensemble-based classifiers. *Artificial Intelligence Review, 33*(1-2), 1-39.

Roughan, M., Sen, S., Spatscheck, O., & Duffield, N. (2004). *Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification.* Paper presented at the Proceedings of the 4th ACM SIGCOMM conference on Internet measurement.

Sáez, J. A., Luengo, J., Stefanowski, J., & Herrera, F. (2015). SMOTE–IPF: Addressing the noisy and borderline examples problem in imbalanced classification by a re-sampling method with filtering. *Information Sciences, 291*, 184-203. doi:http://dx.doi.org/10.1016/j.ins.2014.08.051

Safavian, S. R., & Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics, 21*(3), 660-674.

Saleh, A. A. M., & Simmons, J. M. (2011). Technology and architecture to enable the explosive growth of the internet. *Communications Magazine, IEEE, 49*(1), 126-132. doi:10.1109/MCOM.2011.5681026

Schaffer, C. (1993). Overfitting Avoidance as Bias. *Machine Learning, 10*(2), 153-178.

Schneider, P. (1996). *TCP/IP traffic Classification Based on port numbers.* (2138).

Seeger, M. (2011). Pattern Classification and Machine Learning. *Machine Learning, 1*, 17.

Sen, S., Spatscheck, O., & Wang, D. (2004). *Accurate, scalable in-network identification of p2p traffic using application signatures*. Paper presented at the Proceedings of the 13th international conference on World Wide Web, New York, NY, USA. http://delivery.acm.org/10.1145/990000/988742/p512-sen.pdf?ip=129.83.31.2&id=988742&acc=ACTIVE%20SERVICE&key=A9ED11D7A520B19D%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=433333441&CFTOKEN=97218331&__acm__=1396839571_331034a2dab408962b34f518f108741e

Seni, G., & Elder, J. F. (2010). Ensemble methods in data mining: improving accuracy through combining predictions. *Synthesis Lectures on Data Mining and Knowledge Discovery, 2*(1), 1-126.

Shrivastav, A., & Tiwari, A. (2010). *Network Traffic Classification Using Semi-Supervised Approach.* Paper presented at the Machine Learning and Computing (ICMLC), 2010 Second International Conference on.

Singh, K., & Agrawal, S. (2011, 2011). *Comparative analysis of five machine learning algorithms for IP traffic classification.* Paper presented at the Emerging Trends in Networks and Computer Communications (ETNCC), 2011 International Conference on.

Singh, K., Agrawal, S., & Sohi, B. S. (2013). A Near Real-time IP Traffic Classification Using Machine Learning. *International Journal of Intelligent Systems and Applications, 5*(3), 83-93. doi:10.5815/ijisa.2013.03.09

Smola, A. J., & Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and computing, 14*(3), 199-222.

Soria, D., Garibaldi, J. M., Ambrogi, F., Biganzoli, E. M., & Ellis, I. O. (2011). A 'non-parametric'version of the naive Bayes classifier. *Knowledge-Based Systems, 24*(6), 775-784.

Soysal, M., & Schmidt, E. G. (2010). Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison. *Performance Evaluation, 67*(6), 451-467. doi:http://dx.doi.org/10.1016/j.peva.2010.01.001

Sutton, R. S., & Barto, A. G. (1998). *Introduction to reinforcement learning*: MIT Press.

Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y., & Singer, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research, 6*(9).

Wang, J., Xu, M., Wang, H., & Zhang, J. (2006, 16-20 2006). *Classification of Imbalanced Data by Using the SMOTE Algorithm and Locally Linear Embedding.* Paper presented at the Signal Processing, 2006 8th International Conference on.

Wang, W., Zhang, X., Gombault, S., & Knapskog, S. J. (2009, 14-16 Dec. 2009). *Attribute Normalization in Network Intrusion Detection.* Paper presented at the Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on.

Weston, J., & Watkins, C. (1998). *Multi-class support vector machines*. Retrieved from

Witten, I. H., Frank, E., Trigg, L. E., Hall, M. A., Holmes, G., & Cunningham, S. J. (1999). Weka: Practical machine learning tools and techniques with Java implementations.

Xipeng, X., & Ni, L. M. (1999). Internet QoS: a big picture. *Network, IEEE, 13*(2), 8-18. doi:10.1109/65.768484

Yibo, X., Dawei, W., & Luoshi, Z. (2013, 28-31 Jan. 2013). *Traffic classification: Issues and challenges.* Paper presented at the Computing, Networking and Communications (ICNC), 2013 International Conference on.

Yuan, R., Li, Z., Guan, X. H., & Xu, L. (2010). An SVM-based machine learning method for accurate internet traffic classification. *Information Systems Frontiers, 12*(2), 149-156. doi:10.1007/s10796-008-9131-2

Yuguang, H., & Lei, L. (2011, 15-17 Sept. 2011). *Naive Bayes classification algorithm based on small sample set.* Paper presented at the Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on.

Zander, S., Nguyen, T., & Armitage, G. (2005, 17-17 Nov. 2005). *Automated traffic classification and application identification using machine learning.* Paper presented at the Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on.

Zander, S., Nguyen, T., & Armitage, G. (2012, 22-25 Oct. 2012). *Sub-flow packet sampling for scalable ML classification of interactive traffic.* Paper presented at the Local Computer Networks (LCN), 2012 IEEE 37th Conference on.

Zhao, D., Traore, I., Sayed, B., Lu, W., Saad, S., Ghorbani, A., & Garant, D. (2013). Botnet detection based on traffic behavior analysis and flow intervals. *Computers & Security*(0). doi:http://dx.doi.org/10.1016/j.cose.2013.04.007